TD Models: Modeling the World at a Mixture of Time Scales

Richard S. Sutton
Stow Research
sutton@gte.com

Abstract

Temporal-difference (TD) learning can be used not just to predict *rewards*, as is commonly done in reinforcement learning, but also to predict *states*, i.e., to learn a model of the world's dynamics. We present theory and algorithms for intermixing TD models of the world at different levels of temporal abstraction within a single structure. Such multi-scale TD models can be used in model-based reinforcement-learning architectures and dynamic programming methods in place of conventional Markov models. This enables planning at higher and varied levels of abstraction, and, as such, may prove useful in formulating methods for hierarchical or multi-level planning and reinforcement learning. In this paper we treat only the *prediction* problem—that of learning a model and value function for the case of fixed agent behavior. Within this context, we establish the theoretical foundations of multi-scale models and derive TD algorithms for learning them. Two small computational experiments are presented to test and illustrate the theory. This work is an extension and generalization of the work of Singh (1992), Dayan (1993), and Sutton & Pinette (1985).

1 Multi-Scale Planning and Modeling

Model-based reinforcement learning offers a potentially elegant solution to the problem of integrating planning into a real-time learning and decision-making agent (Sutton, 1990; Barto et al., 1995; Peng & Williams, 1993, Moore & Atkeson, 1994; Dean et al., in prep). However, most current reinforcement-learning systems assume a single, fixed time step: actions take one step to complete, and their immediate consequences become available after one step. This makes it difficult to learn and plan at different time scales. For example, commuting to work involves planning at a high level about which route to drive (or whether to take the train) and at a low level about how to steer, when to brake, etc. Planning is necessary at both levels in order to optimize precise low-level movements without becoming lost in a sea of detail when making decisions at a high level. Moreover, these levels cannot be kept totally distinct and separate. They must interrelate at least in the sense that the actions and plans at a high levels must be turned into actual, moment-by-moment decisions at the lowest level.

The need for hierarchical and abstract planning is a fundamental problem in AI whether or not one uses the reinforcement-learning framework (e.g., Fikes et al., 1972; Sacerdoti, 1977; Kuipers, 1979; Laird et al., 1986; Korf, 1985; Minton, 1988; Watkins, 1989; Drescher, 1991; Ring, 1991; Wixson, 1991; Schmidhuber, 1991; Tenenberg et al., 1992; Kaelbling, 1993; Lin, 1993; Dayan & Hinton, 1993; Dejong, 1994; Chrisman, 1994; Hansen, 1994; Dean & Lin, in prep). We do not propose to fully solve it in this paper. Rather, we develop an approach to multiple-time-scale modeling of the world that may eventually be useful in such a solution. Our approach is to extend temporal-difference (TD) methods, which are commonly used in reinforcement learning systems to learn value functions, such that they can be used to learn world models. When TD methods are used, the predictions of the models can naturally extend beyond a single time step. As we will show, they can even make predictions that are not specific to a single time scale, but intermix many such scales, with no loss of performance when the models are used. This approach is an extension of the ideas of Singh (1992), Dayan (1993), and Sutton & Pinette (1985).

Most prior work on multi-scale modeling has focused on state abstraction: Which sets of states can be treated as a group? What variables can be ignored? What is a good form of generalization between states? In this paper we instead focus exclusively on the relatively ignored *temporal* aspects of abstraction. In fact, we will assume each state is recognized and treated as

an individual, with no relationship to any other. This is an unrealistic extreme, of course, but is nevertheless useful in highlighting temporal issues: Does an abstract action always take the same length of time? Should its duration be explicitly represented? Is it committed to once, or redecided on each time step?

Two other limitations of the work reported in this paper should also be mentioned at the outset. One is that we restrict our attention to prediction rather than control. We ignore the question of action selection and instead focus on forming abstract models that *predict* (at multiple levels of temporal abstraction) the behavior of an autonomous world. A second simplification is that the specification of *what* is to be learned is assumed given. We will present algorithms for learning abstract multi-scale models, but not for specifying what should be modeled or at what scale it should be modeled. Both of these simplifying assumptions are meant to be lifted in immediately future work.

The rest of this paper is structured roughly as follows. First we introduce reinforcement learning and the prediction subproblem we focus on here. Then we present two kinds of multi-step models: n-step models and β-models. We present theoretical results and derive a TD(λ)-style learning algorithm for β-models. Finally, we describe computational experiments.

2 Reinforcement Learning

Reinforcement learning concerns a learning *agent* interacting with an *environment* at some discrete, lowest-level time scale, $t = 0, 1, 2, 3, \ldots$ At each time step, t, the environment is in some *state*, $s_t \in \{1, 2, \ldots, m\}$. The agent observes s_t and chooses an *action*, a_t, in response to which the environment changes state to s_{t+1} and emits a *reward*, r_{t+1}. The agent's (possibly stochastic) mapping from states to actions is called its *policy*, denoted π. The state-transition structure of the environment is described by a set of probabilities, $p_a(i, j)$, giving the probability of transition to state j when in state i and action a is selected by the agent. The reward structure is summarized by the expected value of the reward, $R_a(i)$, given the prior state, i, and action, a.

In general, the states and actions in reinforcement learning can be complex, multi-dimensional, real-valued, and incompletely-known. In this paper we treat the a particularly simple case described above just for conceptual clarity. Strictly speaking, our formal results apply only to this case, but the intent and expectation is that the ideas should carry over to the general case.

In reinforcement learning, the agent's objective is to find and follow a policy that maximizes the reward it receives from the environment. In particular, it seeks to maximize the expected, cumulative, discounted reward:

$$V^\pi(i) = E_\pi\left\{\sum_{t=0}^{\infty}\gamma^t r_{t+1} \;\middle|\; s_0 = i\right\},$$

where this expectation is conditional on the agent following a particular policy, π. The parameter γ, $0 \le \gamma < 1$, is called the *discount-rate*; it determines the relative merit of immediate as opposed to delayed reward. $V^\pi(i)$ is called the *value* of state i under policy π. Computing it is the key step in many planning and learning methods. Ideally, we seek an *optimal* policy, one at which $V^\pi(i)$ is maximized over all π simultaneously at all i.

3 The Prediction Problem

In this paper we focus on a subset of the full reinforcement learning problem, that of estimating the value function V^π for a fixed policy π. In fact, we will avoid all notion of actions altogether. Our learning agent will see only the states and rewards, $s_0, r_1, s_1, r_2, s_2 \ldots$, generated by the combined environment-policy system. This system constitutes a Markov chain. We denote its state transition probabilities as $p(i, j) = E\{s_{t+1}=j \mid s_t=i\}$, and the $m \times m$ matrix of such probabilities as P. Similarly, let R be the m-vector of expected rewards for each state. A vector notation for states will also be useful. Let x_t be the m-vector all components of which are 0 except that corresponding to s_t, which is 1. Then we can write

$$Px_t = E\{x_{t+1} \mid x_t\}$$

and

$$R^T x_t = E\{r_{t+1} \mid x_t\},$$

where all vectors are assumed to be column vectors unless explicitly transposed. We call P and R the *1-step model* of the process.

We also represent the value function as an m-vector V such that $V^T x_t$ is the value of s_t:

$$\begin{aligned} V^\pi(s_t) = V^T x_t &= E\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \;\middle|\; x_t\right\} \quad (1)\\ &= \sum_{k=0}^{\infty}\gamma^k R^T P^k x_t \end{aligned}$$

i.e.,

$$\begin{aligned} V &= \sum_{k=0}^{\infty}\gamma^k P^{T^k} R\\ &= R + \gamma P^T V. \quad (2) \end{aligned}$$

This last equation is a form of the *Bellman equation* for this prediction problem. It forms the basis for many of the update rules of dynamic programming and reinforcement learning.

4 A Generalized Bellman Equation

The Bellman equation is also key to determining the form of models that predict over many time steps. We seek multi-step models $\mathcal{P}$ and $\mathcal{R}$ that can take the place of γP and R in the Bellman equation (2), that is, that satisfy a *generalized Bellman equation*:

$$V = \mathcal{R} + \mathcal{P}^T V. \tag{3}$$

Any $\mathcal{P}$ and $\mathcal{R}$ that satisfy this equation, with $\lim_{k\to\infty} \mathcal{P}^k = 0$, are said to constitute a *valid* model. Any valid model can be used to update and improve an approximation V_t of V by *lookahead* or *backup* operations, e.g.,

$$V_{t+1} = \mathcal{R} + \mathcal{P}^T V_t. \tag{4}$$

As long as the model is valid, this process converges to

$$V_\infty = \sum_{k=0}^{\infty} {\mathcal{P}^T}^k \mathcal{R} = V.$$

This is the key insight of the present work, that any model satisfying the generalized Bellman equation is equally valid for the purpose of computing the value function. It turns out that there are many such models, some of which correspond to abstract, multi-step models and which may be more useful than conventional 1-step models. In considering such models, the generalized Bellman equation acts as a gold standard. It tells us which temporal details can be safely deleted in forming an abstract model, and which must be retained.

5 n-Step Models

A simple example of valid, multi-step models are the n-step models. For example, we can obtain a 2-step model by expanding the original Bellman equation once, and then rewriting it in the form of the generalized Bellman equation:

$$\begin{aligned} V &= R + \gamma P^T V \\ &= \underbrace{R + \gamma P^T R}_{\mathcal{R}} + \underbrace{(\gamma P^T)^2}_{\mathcal{P}^T} V. \end{aligned}$$

This $\mathcal{R}$ and $\mathcal{P}$ are called a 2-step model because $\mathcal{P}x_t$ predicts the discounted state two steps in the future and $\mathcal{R}^T x_t$ predicts the total discounted reward over the two steps. In general, for an n-step model:

$$\mathcal{P}^{(n)} = (\gamma P)^n, \tag{5}$$

i.e.,

$$\mathcal{P}^{(n)} x_t = E\{\gamma^n x_{t+n} \mid x_t\}$$

and

$$\mathcal{R}^{(n)} = \sum_{k=0}^{n-1} (\gamma P^T)^k R, \tag{6}$$

i.e.,

$${\mathcal{R}^{(n)}}^T x_t = E\left\{\mathbf{r}_t^{(n)} \;\middle|\; x_t\right\},$$

where $\mathbf{r}_t^{(n)} = \sum_{k=1}^{n} \gamma^{k-1} r_{t+k}$ is the n-step truncated return starting from state x_t. The precise form of these equations is determined by the requirement that the resulting $\mathcal{P}^{(n)}$ and $\mathcal{R}^{(n)}$ satisfy the multi-step Bellman equation (3). We will prove later that they do.

n-step models achieve some of our goals for multi-step models. They can be used for lookahead to find V just like the base model, while requiring significantly fewer steps. One step of lookahead (4) with a 10-step model is exactly equivalent to 10 steps of lookahead with a 1-step model. Moreover, lookahead with different n-step models are completely compatible. One could do lookahead with a 10-step model followed by lookahead with a 20-step model. The end result—convergence to V—is unaffected.

However, n-step models are also limited in important ways, and we have introduced them here only to give a simple example of what is theoretically possible. One limitation of n-step models is that they insist on a single, fixed time interval. Each n-step model has a fixed n. It cannot say that some result will occur without specifying the exact time step on which it will occur. A second problem with n-step models is that they may be computationally difficult to learn, particularly for large n. In the next two section we gradually generalize n-step models to remove these limitations.

6 Intermixing Time Scales

If we are predicting many steps into the future, then it is impractical and usually unnecessary to learn precise n-step models. Which of us can predict events 100 seconds into the future to a precision of one second? Which of us needs to? It would be preferable, if possible, to make temporally approximate predictions, e.g., that are weighted averages of n-step predictions. For example, instead of trying to predict precisely x_{t+100} we might prefer to predict a weighted average of states in the neighborhood of x_{100}, perhaps from x_{t+90} to x_{t+110}. In general, let w_n denote the weight given in the weighted average to the n-step prediction, where $\sum_{n=1}^{\infty} w_n = 1$ (e.g., $w_n = \frac{1}{20}$ for $90 < n \leq 110$, $w_n = 0$ otherwise). Then the desired multi-scale model is

$$\mathcal{P} = \sum_{n=1}^{\infty} w_n \mathcal{P}^{(n)} \qquad \text{and} \qquad \mathcal{R} = \sum_{n=1}^{\infty} w_n \mathcal{R}^{(n)}. \tag{7}$$

Figure 1a shows the weighting sequence corresponding to an n-step model, and Figure 1b shows a weighting sequence for a "blurred" n-step model, in which the states predicted are an average of those occurring *approximately* n steps later. We prove later that any unitary mixture of n-step models is a valid model. This means that a single matrix can accurately hold

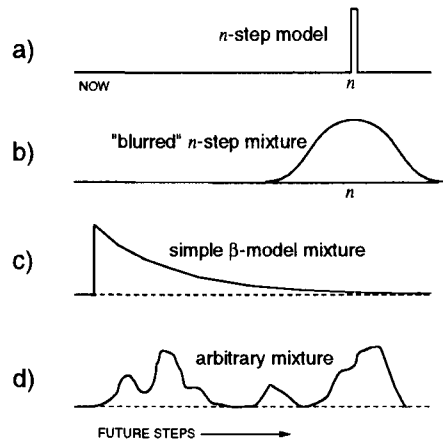

Figure 1: Four different weighting sequences, w_n, of future states to be predicted: **a)** an n-step model puts all weight on the nth time step; **b)** better is to put weight on and thus average all future time steps near n; **c)** a simple β-model puts less and less weight on more and more distant events; and **d)** an arbitrary mixture over future steps is still valid. All weightings must sum to one.

a mixture of models from many different time scales! For example, a single $\mathcal{P}$ can make predictions that are half 10-step predictions and half 20-step predictions. The predictions of the different time scales are linearly mixed and yet still they can be used in backup operations without altering convergence to V. As long as it is done in this way, the time scales of models can be mixed and lengthened arbitrarily without any degradation of the results of planning.

7 β-Models

One mixture of n-step models that may be of particular interest is that in which the weighting falls off exponentially with delay (see Figure 1c), i.e., in which the n-step prediction is given weight

$$w_n = (1-\beta)\beta^{n-1},$$

for some parameter β, $0 \leq \beta < 1$, that determines the time constant of the exponential weighting. The $\mathcal{P}$ with this weighting contains the "state-transition probabilities" of a form of multi-step model that we call a *simple β-model*. The (i,j)-th entry of this matrix gives the expected β-discounted occupancy of state j over the time period following observation of state i. The "expected reward" vector, $\mathcal{R}$, for a β-model can be interpreted as the return received during this time period.

The main topic of this paper are what we call *full β-models*. In these, β is allowed to vary from state to state. Let $\beta(i)$ denote the β for state i, and let $\beta_n = \beta(x_n)$ denote the β applicable after n steps. A full β-model uses the weighting

$$w_n = (1-\beta_n)\prod_{i=1}^{n-1}\beta_i.$$

This weighting sequence does not follow a simple pattern such as in Figure 1b or 1c, but can have an arbitrary weighting profile, as in Figure 1d. Moreover, the weighting profile is not fixed, but is dependent on the sequence of states observed. For different state sequences, a totally different weighting may apply. Whatever the sequence, the total weight given to all the states in it must sum to one. The quantity $1-\beta_n$ is a measure of how much weight is given to the nth state. If it is zero (i.e., if $\beta_n = 1$) then no weight is given, whereas if it is 1 ($\beta_n = 0$), then the state is given all the remaining weight. The product $\prod_{i=1}^{n}\beta_i$ is a measure of the remaining weight, of how much weight has not yet been given to preceding states. If they all had β's of 1, then all the weight remains, but if even one of them had a β of 0, then no weight remains.

More formally, given a set of β's, the corresponding β-model $\mathcal{P}$ and $\mathcal{R}$ are defined by

$$\mathcal{P}x_0 = E\left[\sum_{t=1}^{\infty}\gamma^t(1-\beta_t)\prod_{i=1}^{t-1}\beta_i\, x_t\right] \qquad (8)$$

and

$$\mathcal{R}^T x_0 = E\left[\sum_{t=1}^{\infty}\gamma^{t-1}\prod_{i=1}^{t-1}\beta_i\, r_t\right], \qquad (9)$$

for all possible starting states, x_0 (the expected values are all conditional on x_0). Here $\beta_t = \beta(s_t)$ is the β that applies at time t. Let B be the diagonal matrix with the $\beta(i)$ as diagonal entries. Then the $\mathcal{P}$ and $\mathcal{R}$ of a β-model can alternatively be specified in matrix-vector form as:

$$\mathcal{P} = \gamma(I-B)P\sum_{k=0}^{\infty}(\gamma BP)^k$$

and

$$\mathcal{R} = \sum_{k=0}^{\infty}(\gamma P^T B)^k R.$$

Full β-models can represent variable mixtures of time scales dependent on the state trajectory taken. For example, the abstract action "look for your keys" might end very quickly or might take a long time. It can be implemented in a β-model by setting $\beta = 1$ for states in which one is still looking, and $\beta = 0$ for states in which the keys have been found.

Full β-models are a very general form of abstract, multi-scale model for stochastic worlds. In the rest of this paper we develop learning algorithms for β-models and illustrate their properties and abilities through computational examples.

8 Theoretical Results

So far we have described models as consisting of two parts, $\mathcal{P}$ and $\mathcal{R}$. For some purposes it is useful to combine these into one structure, an $(m+1)\times(m+1)$ matrix:

$$\mathcal{M} = \left[\begin{array}{c|c} 1 & -\mathcal{R}^T- \\ \hline 0 & \mathcal{P} \end{array}\right]$$

We call this putting the model in *homogeneous coordinates*. If the value vector V is also augmented by adding an initial component whose value is always 1 (we will continue to refer to the augmented vector simply as V) then the generalized Bellman equation (3) can be written

$$V = \mathcal{M}^T V = \mathcal{M}^{T^\infty} V. \tag{10}$$

As before, we consider a model $\mathcal{M}$ to be *valid* if and only if it satisfies (10). In homogeneous coordinates the following theorems become evident:

THEOREM 1: *Closure under composition.* For any valid models $\mathcal{M}_1$ and $\mathcal{M}_2$, the composed model $\mathcal{M}_1\mathcal{M}_2$ is also valid.

PROOF: $(\mathcal{M}_1\mathcal{M}_2)^T V = \mathcal{M}_2^T\mathcal{M}_1^T V = \mathcal{M}_2^T V = V$. □

Note that $\mathcal{M}$ has been constructed such that it is valid only if the corresponding $\mathcal{P}$ and $\mathcal{R}$ are valid. Thus, the composition theorem proves the validity of the n-step models, (5) and (6).

THEOREM 2: *Closure under averaging.* For any set of valid models $\{\mathcal{M}_i\}$ and corresponding weights $\{w_i\}$ such that $\sum_i w_i = 1$, the weighted model $\mathcal{M} = \sum_i w_i\mathcal{M}_i$ is also valid.

PROOF: $\left(\sum_i w_i\mathcal{M}_i\right)^T V = \sum_i w_i\mathcal{M}_i^T V = \sum_i w_i V = V\sum_i w_i = V$. □

The averaging theorem proves the validity of the mixture models (7) and thus of simple β-models.

The generalized Bellman equation (10) is really a requirement on each *column* of $\mathcal{M}$, corresponding to all the predictions *from* a single state. Let c be the j-th column of $\mathcal{M}$. We define c to be a *valid j-th column* if and only if $V(j) = c^T V$, where $V(j)$ is the j-th component of V. A model $\mathcal{M}$ is valid if and only if all its columns are valid. We can then extend the averaging theorem to a column by column form:

THEOREM 3: *Closure under columnwise averaging.* For any set of valid j-th columns $\{c_i\}$ and corresponding weights $\{w_i\}$ such that $\sum_i w_i = 1$, the weighted vector $c = \sum_i w_i c_i$ is also a valid j-th column.

PROOF: $\left(\sum_i w_i c_i\right)^T V = \sum_i w_i c_i^T V = \sum_i w_i V(j) = V(j)\sum_i w_i = V(j)$. □

One application of this theorem is as a justification of learning methods. To update the model for a certain starting state one looks ahead from it using a model or a sampling of state transitions to obtain an expected distribution of result states. By the composition theorem this is a valid column (at least in expected value), and thus averaging it in with the original column for the starting state (which is what learning rules do) will maintain the validity of the original column.

Unfortunately, even the columnwise averaging theorem does not apply directly to full β-models. A separate theorem is needed:

THEOREM 4: *Validity of β-models.* Any β-model, $\mathcal{R}$, $\mathcal{P}$, and B, satisfying (8) and (9), with $0 \le \beta(i) \le 1$, is valid.

PROOF: For any β-model and initial state vector x_0:

$$\begin{aligned}
&\mathcal{R}^T x_0 + V^T\mathcal{P}x_0 \\
&\quad = E\left[\sum_{t=1}^{\infty}\gamma^{t-1}\prod_{i=1}^{t-1}\beta_i\, r_t + \gamma^t(1-\beta_t)\prod_{i=1}^{t-1}\beta_i\, V^T x_t\right] \\
&\text{(by (8) and (9))} \\
&\quad = E\left[\sum_{t=1}^{\infty}\gamma^{t-1}\prod_{i=1}^{t-1}\beta_i\left(r_t + \gamma(1-\beta_t)\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\right)\right],
\end{aligned}$$

by applying (1) as the definition of $V^T x_t$. Careful inspection of these terms reveals that they can be grouped by r_t and written as

$$\begin{aligned}
&\mathcal{R}^T x_0 + V^T\mathcal{P}x_0 \\
&\quad = E\left[\sum_{t=1}^{\infty}\gamma^{t-1} r_t\left(\prod_{i=1}^{t-1}\beta_i + \sum_{k=1}^{t-1}(1-\beta_k)\prod_{i=1}^{k-1}\beta_i\right)\right] \\
&\quad = E\left[\sum_{t=1}^{\infty}\gamma^{t-1} r_t\right] \\
&\quad = V^T x_0.
\end{aligned}$$

Since this holds for all x_0, it follows that $\mathcal{R}^T + V^T\mathcal{P} = V^T$, and thus that $\mathcal{R}$ and $\mathcal{P}$ form a valid model. □

Finally, note that models in homogeneous coordinates can be used not just in backups, as in (10), but also in the forward direction, for multi-step predictions. For this we augment the state vectors, x_t, by adding an initial component with the value 0. For example, for the n-step homogeneous model $\mathcal{M}^{(n)}$, we can write

$$\mathcal{M}^{(n)}\left[\begin{array}{c} 0 \\ x_t \end{array}\right] = E\left\{\left[\begin{array}{c} \mathbf{r}_t^{(n)} \\ \gamma^n x_{t+n} \end{array}\right] \middle| x_t\right\}.$$

where $\mathbf{r}_t^{(n)}$ is the n-step truncated return from x_t.

9 TD(λ) Learning of β-models

In principle, β-models and n-step models could be learned in any of a variety ways. However, temporal-difference (TD) methods offer great advantages in terms of incremental computation and, potentially, in learning rate if significant state information is available. In this section we derive a TD(λ)-like learning algorithm (Sutton, 1988) for β-models. Our derivation follows that of TD(λ) in Watkins (1989) and Jaakkola, Jordan & Singh (1994). The derivation is very similar for $\mathcal{P}$ and $\mathcal{R}$, and the $\mathcal{R}$ derivation is closest to the existing analyses for TD(λ), so here we present only the derivation for $\mathcal{P}$ (and even that in an abbreviated form).

The main idea is to construct a TD target for our "next state" prediction, $\mathcal{P}_t x_t$ (given a set of β's). Based on (8) we can write the ideal prediction, $y_t = \mathcal{P} x_t$, as

$$y_t = \mathcal{P} x_t = E\left[\sum_{k=1}^{\infty} \gamma^k (1-\beta_{t+k}) \prod_{i=1}^{k-1} \beta_{t+i}\, x_{t+k}\right] =$$

$$E\left[\sum_{k=1}^{n} \gamma^k (1-\beta_{t+k}) \prod_{i=1}^{k-1} \beta_{t+i}\, x_{t+k} + \gamma^n \prod_{i=1}^{n} \beta_{t+i}\, \mathcal{P} x_{t+n}\right]$$

Although we can't know y_t itself, we can take advantage of the observed sequence, $x_t, x_{t+1}, \ldots, x_{t+n}$, to form an n-step TD approximation to it. We do this by replacing each expected state with the observed state, and the final $\mathcal{P}$ with the current estimate $\mathcal{P}_t$. This yields an n-step TD target for the "next state" of a β-model:

$$y_t^{(n)} = \sum_{k=1}^{n} \gamma^k (1-\beta_{t+k}) \prod_{i=1}^{k-1} \beta_{t+i}\, x_{t+k} + \gamma^n \prod_{i=1}^{n} \beta_{t+i}\, \mathcal{P}_t x_{t+n}.$$

To form a TD(λ)-style method we use as target an exponential combination of the n-step targets. The exponential combination is parameterized by λ, $0 \leq \lambda \leq 1$, and denoted y_t^λ:

$$y_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} y_t^{(n)}.$$

Ideally, we would like to use y_t^λ as the target in an update rule such as

$$\Delta \mathcal{P}_t = \alpha \left[y_t^\lambda - \mathcal{P}_t x_t\right] x_t^T,$$

where α is a positive step-size parameter. However, the target, y_t^λ, is not actually available at time t when it is needed. This is where standard tricks for obtaining an incremental method are used. These are based on the assumption that the estimate being updated—$\mathcal{P}_t$ in this case—does not change greatly from time step to time step (or, equivalently, that the updates are accumulated offline until the end of a trial). In this case we can define the TD error

$$\varepsilon_t = (1-\beta_{t+1})\gamma x_{t+1} + \gamma \beta_{t+1} \mathcal{P}_t x_{t+1} - \mathcal{P}_t x_t$$

and express the overall error term as[1]

$$y_t^\lambda - \mathcal{P}_t x_t = \varepsilon_t + (\gamma\lambda)\beta_t \varepsilon_{t+1} + (\gamma\lambda)^2 \beta_t \beta_{t+1} \varepsilon_{t+2} + \cdots$$

to obtain the learning rule

$$\Delta \mathcal{P}_t = \alpha \varepsilon_t \sum_{k=0}^{\infty} (\gamma\lambda)^k \prod_{i=0}^{k-1} \beta_{t-i}\, x_{t-k}^T. \qquad (11)$$

A similar analysis for $\mathcal{R}$ yields the following update rule for $\mathcal{R}_t$:

$$\Delta \mathcal{R}_t = \alpha \varepsilon_t^{\mathcal{R}} \sum_{k=0}^{\infty} (\gamma\lambda)^k \prod_{i=0}^{k-1} \beta_{t-i}\, x_{t-k},$$

where

$$\varepsilon_t^{\mathcal{R}} = r_{t+1} + \gamma \beta_{t+1} \mathcal{R}_t^T x_{t+1} - \mathcal{R}_t^T x_t.$$

Both of these learning rules can be implemented completely incrementally by implementing the sums as an accumulating eligibility trace in the usual way.

10 A Wall-Following Example

An abstract action that might be useful to a mobile robot is that of *wall-following* (e.g., Lin, 1993; Mataric, 1990). Wall-following is not a single action or sequence of actions, but a complete closed-loop policy for moving forward while staying close to a wall on one side. Such an abstract action might be useful in navigating about an office environment.

Figure 2 shows a gridworld version of wall-following. The robot starts at one of the three grid cells at the bottom and travels up, one row per time step. The robot's wall-following policy tries to keep the robot in the center column, neither too close to the wall, which risks a collision, nor too far from it. If the robot strays too far, into the "open space" three or more cells from the wall, then it loses sensory contact with the wall and wall-following terminates. A third way for wall-following to terminate is for the robot to reach the region labeled "exit", as shown in the sample trajectory in Figure 2. In detail, the robot's stochastic wall-following policy is given in Table 1.

We do not seek to change the robot's wall-following policy, only to learn a predictive model for it that abstracts away the low-level details and just predicts the likely outcomes of wall-following. (A full control treatment would presumably involve comparing these predictions with those for other abstract actions.) We

[1] If you wish to verify this step, first show that y_t^λ can be written as

$$\sum_{k=1}^{\infty} \lambda^{k-1} \gamma^k \prod_{i=1}^{k-1} \beta_{t+i} \left[(1-\beta_{t+k}) x_{t+k} + (1-\lambda)\beta_{t+k} \mathcal{P}_t x_{t+k}\right]$$

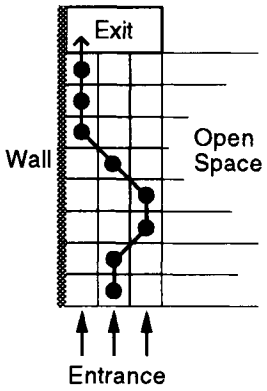

Figure 2: A Sample Trajectory in the Wall-Following Task. The robot starts in one of the bottom three states, then works its way up one row at a time following the policy given in Table 1. The trial terminates if the robot runs into the wall, wanders off into the "open space", or successfully reaches the exit. The task is to learn a model of the world at the level of these three ultimate outcomes.

Table 1: Built-in Policy of Wall-Following Robot. The upper-left outcome is a collision and the lower-right outcome is a loss of contact.

Distance from wall	Probability of Movement		
	Forward & Left	Directly Forward	Forward & Right
1	1/6	1/3	1/2
2	1/4	1/2	1/4
3	1/2	1/3	1/6

applied the β-model learning equation (11). Each wall-following attempt was treated as a separate trial, with traces set to zero at the beginning of each trial. Each of the 24 grid cells was treated as a distinct state, plus three more states for the three outcomes: colliding, losing-contact, and exiting. So that learning would be only about these outcomes, their β were set to 0 whereas the β of all the other states were 1. The discount parameter was $\gamma = 0.9$.

Inspection of the learned prediction matrix $\mathcal{P}_t$ after 5000 trials with $\alpha = 0.01$ revealed that the predictions of all grid-cell states were indeed 0. The predictions of the outcome states were as shown in Figure 3a. For comparison, Figure 3b shows the ideal predictions for this γ and β's, as given by (7). The similarity of these two shows that the learning algorithm is working as intended, providing a rudimentary check on its derivation.

The results in Figure 3c are the easiest to interpret. These are the learned predictions for the case of $\gamma = 1$ (possible here because this is a trial-based task). These

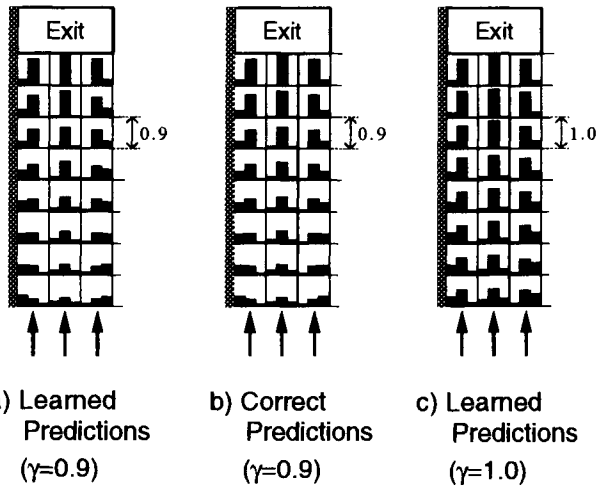

Figure 3: Predictions of a β-Model in the Wall-Following task. Inside each state, the three bars indicate the predictions of running into the wall, reaching the exit, and wandering into the open space, respectively.

predictions can be directly interpreted as the probability of the three possible outcomes given that the robot is in each cell. From all cells, exiting is predicted to be the most likely outcome, ranging from a 100% probability for the last center cell to about 50% for an early off-center cell. For example, the lower-left-most cell has a 48% probability of exiting, a 38% probability of colliding, and a 14% probability of losing contact. The algorithm has successfully learned a high-level model of the ultimate consequences of the wall-following abstract action. In this task, all rewards were zero, so in fact there is no particular optimal behavior (and the learned reward model is all zeros). But if certain of the outcomes did become desirable or undesirable, then this multi-scale model might be very useful for quickly computing the effect on received reward of taking or not taking the wall-following action.

11 A Hidden-State Example

Although we have not emphasized it in this paper, multi-scale models are also useful in overcoming the problems of hidden state (e.g., see Sutton & Pinette, 1985). Figure 4 shows one example. States 6 and 7 are confounded, both appearing to the learning agent as State 6.

Any system that learned a 1-step model of this world would be unable to do any useful planning about the difference between States 2 and 3. It would model both as always leading to State 6. A simple β-model, on the other hand, learns something more. The multi-scale model learned by a simple β-model applied in a

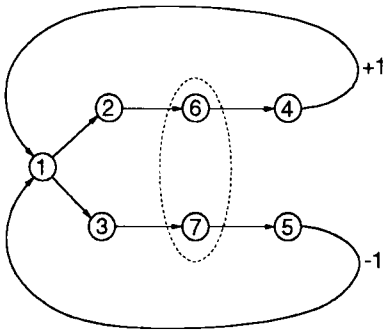

Figure 4: World with Hidden State. States 6 and 7 are ambiguous. The two outcomes from State 1 each occur with 50% probability.

simulation of the world in Figure 4 was:

$$\mathcal{P}_{5000} = \begin{pmatrix} .04 & .10 & .10 & .47 & .47 & .21 \\ .24 & .02 & .02 & .11 & .11 & .05 \\ .23 & .02 & .02 & .1 & .10 & .05 \\ .05 & .21 & .01 & .02 & .02 & .25 \\ .05 & .00 & .21 & .02 & .02 & .22 \\ .21 & .47 & .47 & .10 & .10 & .04 \end{pmatrix}$$

where $\beta = 0.5$, $\gamma = 0.9$, and $\alpha = .01$. The second and third columns represent the predictions from States 2 and 3. Note that the prediction from State 2 is larger at the fourth component whereas the prediction from State 3 is larger at the fifth component. The β-model learned that, in a long-term sense, State 2 is followed more often by State 4, whereas State 3 is followed more often by State 5. Thus even a simple β-model is able to learn a model that would be useful in making rapid changes in policy if, for example, the rewards out of States 4 and 5 changed.

If the apparent State 6 could be recognized as ambiguous and given a β of 0 (in a full β-model), then in fact a perfect multi-scale model could be learned for this problem.

12 Adding Actions (Future Work)

Before closing, let us speculate on how this work could be extended beyond the pure prediction problem to a full reinforcement-learning problem, including actions. A natural approach is to have a great many β-models, each corresponding to a different policy. Thus we would have one set of predictions about what would happen if we choose to drive to work, another for if we choose to take the train, or to stay home. Even for the same policy, we could have many different models with different sets of β's, e.g., one to predict how long it would take to drive, another to predict the stressfulness of the journey, another the probability of an accident, etc. If seems appropriate to think of each such β-model as an abstract action.

Much of the substance in this paper has been driven by the Bellman equation for the prediction problem (2). As actions are added and we consider a control problem, it is natural to consider the full Bellman equation, including the maximization over actions. In matrix-vector form it might look something like this:

$$V^* = \max_{a \in \mathcal{A}} \mathcal{R}_a + \mathcal{P}_a^T V^* \qquad (12)$$

where V^* is the optimal value function, and $\mathcal{A}$ is the set of *all* actions, both primitive and abstract. For the primitive actions, $\mathcal{R}_a$ and $\mathcal{P}_a$ would be simply the 1-step expected rewards and next-state probabilities (times γ) for each action. The abstract actions could be an arbitrary set of β-models, each with an $\mathcal{R}_a$, $\mathcal{P}_a$, and B_a. It is not hard to show that including any set of valid β-models in this equation does not change the value at which the maximum is achieved.

V^* can then be estimated by appropriate backup processes. Starting from any initial state, any of the β-models could be applied, in any order and mixture, to predict the consequences of abstract actions or sequences of abstract actions. The resultant distribution of outcomes could then be used to update the value function for the initial state, or to update the predictions of the β-models for the initial state. Actual outcomes, or actual outcomes combined with β-model predictions, could also be used in these updates.

There is great flexibility here and it is not clear exactly how the choices should best be made. The computation of V^* could be dramatically sped up if the abstract actions were well chosen and selectively applied. On the other hand it could also be slowed down if they were poorly chosen and applied. Classical issues of "macro utility" arise here, though the goals and tradeoffs may be somewhat different. In this context a useful β-model might be defined as one that often obtains the maximum in (12). This could be used to distinguish good β-models from bad, or even to direct the alteration of the β's or policies of individual β-models so as to make them more likely to be good. In other words, it might lead to a basis for sculpting and creating good β-models.

13 Conclusions

We have introduced an approach to multi-level modeling and planning with several attractive features. The framework applies to a broad class of stochastic environments (specified as Markov decision processes) and to arbitrary, closed-loop, stochastic policies. Although predictions are made over different time scales, there is no need to separate them into distinct levels. The mathematical framework we have developed here means that each high-level model has a clear, definite semantics. Among other benefits, this leads directly to the learning algorithms we have presented.

The abstract models we have introduced are based on how they will be used, that is, on backups and the Bellman equation. Any abstraction collapses and loses some detail, some of the information that would be represented in the finest possible level of modeling. The trick is in knowing what detail to omit. Knowing that the model is to be used in backups based on the Bellman equation provides critical information about what can be safely omitted without degrading the ultimate result.

Much more work is needed before this can be considered a full approach to multi-scale planning. Nevertheless, the ideas presented here already add significant new dimensions of flexibility to planning and learning methods for Markov decision processes, while retaining their mathematical foundations. Our multi-scale methods are explicitly designed to learn models of the world that behave like Markov processes—that have observable state and that obey a Bellman equation. Ultimately, this goal may be key to creating models of the world that are useful for learning and planning.

Perhaps most important of all, this work provides a clear semantics for multi-time-scale models that permits their reliable learning and use. This provides a firmer foundation for addressing the questions of model creation and the modeling of abstract actions.

Acknowledgments

The author gratefully acknowledges the substantial assistance and encouragement he has received from Satinder Singh, Peter Dayan, Andy Barto, Chris Watkins, and Lonnie Chrisman in developing the ideas presented in this paper.

References

Barto, A.G., Bradtke, S.J., Singh, S.P. (1995) "Learning to act using real-time dynamic programming," *Artificial Intelligence.*

Chrisman, L. (1994) "Reasoning about probabilistic actions at multiple levels of granularity," *AAAI Spring Symposium: Decision-Theoretic Planning*, Stanford University.

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

Dayan, P., Hinton, G.E. (1993) "Feudal reinforcement learning". In C.L. Giles, S.J. Hanson, J.D. Cowan, Editors, *Advances in Neural Information Processing Systems, 5*, 271–278. San Mateo, CA: Morgan Kaufmann.

Dean, T., Kaelbling, L.P., Kirman, J., Nicholson, A. (in preparation) "Planning under time constraints in stochastic domains."

Dean, T., Lin, S.-H. (in preparation) "Decomposition techniques for planning in stochastic domains."

Drescher, G.L. (1991) *Made Up Minds: A Constructivist Approach to Artificial Intelligence.* MIT Press.

Fikes, R.E., Hart, P.E., Nilsson, N.J. (1972) "Learning and executing generalized robot plans," *Artificial Intelligence 3*, 251–288.

Hansen, E. (1994) "Cost-effective sensing during plan execution," *Proc. AAAI-94*, 1029–1035.

Kuipers, B.J. (1979) "Commonsense Knowledge of Space: Learning from Experience," *Proc. IJCAI-79*, 499–501.

Kaelbling, L.P. (1993) "Hierarchical learning in stochastic domains: Preliminary results," *Proc. of the Tenth Int. Conf. on Machine Learning*, 167–173, Morgan Kaufmann.

Korf, R.E. (1985) *Learning to Solve Problems by Searching for Macro-Operators.* Boston: Pitman Publishers.

Laird, J.E., Rosenbloom, P.S., Newell, A. (1986) "Chunking in SOAR: The anatomy of a general learning mechanism," *Machine Learning 1*, 11–46.

Lin, L.-J. (1993) *Reinforcement Learning for Robots Using Neural Networks.* PhD thesis, Carnegie Mellon University. Technical Report CMU-CS-93-103.

Mataric, M.J. (1990) *A Model for Distributed Mobile Robot Environment Learning and Navigation.* MIT Masters thesis, Electrical Engineering and Computer Science.

Minton, S. (1988) *Learning Search Control Knowledge: An Explanation-based Approach.* Kluwer Academic.

Moore, A.W., Atkeson, C.G. (1993) "Prioritized sweeping: Reinforcement learning with less data and less real time," *Machine Learning 13*, 103–130.

Peng, J., Williams, R.J. (1993) "Efficient learning and planning within the Dyna framework," *Adaptive Behavior 1*, 437–454.

Ring, M. (1991) "Incremental development of complex behaviors through automatic construction of sensory-motor hierarchies," *Proceedings of the Eighth International Conference on Machine Learning*, 343–347, Morgan Kaufmann.

Sacerdoti, E.D. (1977) *A Structure for Plans and Behavior.* New York: Elsevier.

Schmidhuber, J. (1991) "Neural Sequence Chunkers." Technische Universitat Munchen TR FKI-148-91.

Singh, S.P. (1992) "Reinforcement learning with a hierarchy of abstract models," *Proceedings of the Tenth National Conference on Artificial Intelligence*, 202–207. MIT/AAAI Press.

Singh, S.P. (1992) "Scaling reinforcement learning by learning variable temporal resolution models," *Proceedings of the Ninth International Conference on Machine Learning*, 406–415, Morgan Kaufmann.

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

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

Sutton, R.S., Pinette, B. (1985) "The learning of world models by connectionist networks," *Proc. of the Seventh Annual Conf. of the Cognitive Science Society*, 54-64.

Tenenberg, J. Karlsson, J., & Whitehead, S. (1992) "Learning via task decomposition," *Proc. Second Int. Conf. on the Simulation of Adaptive Behavior.* MIT Press.

Thrun, T., Schwartz, A. (1995) "Finding Structure in Reinforcement Learning," in *Advances in Neural Information Processing Systems, 7.* San Mateo: Morgan Kaufmann.

Watkins, C.J.C.H. (1989) *Learning with Delayed Rewards.* PhD thesis, Cambridge University.

Wixson, L.E. (1991) "Scaling reinforcement learning techniques via modularity," *Proc. Eighth Int. Conf. on Machine Learning*, 368–372, Morgan Kaufmann.