Using Temporal Neighborhoods to Adapt Function Approximators in Reinforcement Learning

R. Matthew Kretchmar
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
kretchma@cs.colostate.edu

Charles W. Anderson
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
anderson@cs.colostate.edu

Abstract

To avoid the curse of dimensionality, function approximators are used in reinforcement learning to learn value functions for individual states. In order to make better use of computational resources (basis functions) many researchers are investigating ways to adapt the basis functions during the learning process so that they better fit the value-function landscape. Here we introduce *temporal neighborhoods* as small groups of states that experience frequent intra-group transitions during on-line sampling. We then form basis functions along these temporal neighborhoods. Empirical evidence is provided which demonstrates the effectiveness of this scheme. We discuss a class of RL problems for which this method might be plausible.

1 Overview

In reinforcement learning an agent navigates an environment (a state space) by selecting various actions in each state. As the agent makes actions, it receives rewards indicating the "goodness" of the action. Reinforcement learning is a methodology which allows the agent to discover which actions to select in order to optimize the rewards in each state. The value of a state is the immediate reward an agent will receive from that state and the discounted sum of all future rewards encountered by the agent. Detailed reviews of reinforcement learning are available [2, 3].

On-line algorithms use the experience of the agent as it moves about the state space to learn the values of each state. Tables are often employed to "memorize" the value for each individual state. However, many RL problems involve very large state spaces, especially when the state space is multidimensional. The *curse of dimensionality* arises because state spaces grow too large to store all individual state values in a single table.

To lessen the curse of dimensionality, function approximators are commonly utilized: they require far fewer resources than a table look-up method, and they generalize over other parts of the state space so that learning experience can be shared among states. Function approximators commonly use fixed basis functions (such as CMACs and Radial Basis Functions) which have shown to be stable in both theory and in practice [8, 9]. Despite the proofs of convergence for fixed basis function approximators, these RL algorithms are often slow to converge in practice. Research indicates that different types of basis functions are better suited to different problems, and they often need to be "fine-tuned" to the particular task [4]. Fixed basis functions also tend to be somewhat wasteful of computational resources because they do not accommodate the pecularities of the value function landscape; one needs to be certain to employ enough fixed basis functions of adaquately fine resolution to learn a value function well.

There have been many attempts to adapt basis functions during learning to better fit the value function landscape. The most common methods perform gradient descent on an error metric but these techniques are generally slow to converge and are overly sensative to various parameters. Singh's *soft state aggregation* demonstrates success using a gradient descent technique to shape the basis functions [7]. There are also various other adaptive approaches. Anderson's *Hidden Restart Method* [1] relocates basis functions to regions of the state space which are not adequately modelled. Whitehead and Choate employ genetic algorithms to position and form basis functions [10]. Moore's *Parti-Game Algorithm* learns value functions by dynamically creating variable resolution "basis functions" [6].

Here we develop a novel approach in which the basis functions are adapted according to the perceived state transition probabilities. McCallum has shown successful results in using *Transitional Proximity* for a faster Q-value update scheme [5]. We use the same information in much different manner. Additionally, we provide a theoretical discussion regarding the types of RL tasks that would benefit from using state transitions. In Section 2 we define temporal neighborhoods and discuss their role in forming basis functions for function approximation. Section 3 shows a simple example illustrating the advantage of temporal neighborhoods. The details of the algorithm are presented in Section 4. In Section 5 we apply the algorithm to the more complex Mountain Car task. A summary and discussion of future work are presented in Section 6.

2 Temporal Neighborhoods

With most local function approximators, basis functions are created to span a small neighborhood of physically adjacent states. By "physically adjacent states" we mean states which are near each other in Euclidean distance. Although states may be physically adjacent, in many control problems it may be unlikely (or even impossible) that the agent can transition between them. A better notion of nearness is temporal adjacency. As the agent interacts with the environment, there tend to be pathways or trajectories through the state space which the agent uses with high frequency. The states which lie along these trajectories are temporaly adjacent.

> Two states are *temporally adjacent* if when the agent currently occupies one state, there exists a high probability that the agent will transition to the other state on the next move.

> A *temporal neighborhood* is a set of states which form along a common state space trajectory. When an agent is placed in one of these states, it often transitions from one state to the next within the set.

Why might temporally adjacent states be important in the formation of basis functions? We hypothesize that there exist a class of control problems in which the reward signal is bounded at each step (often it is constant at each step). We define this class of RL control problems:

> *Frequency Bounded Problems* are a class of control problems in which the reinforcement signal at each step has an upper bound. Therefore, the difference in the value function of any two temporally adjacent states is also bounded by this same quantity.

We refer to these collections of states as being frequency-bounded because the value function remains relatively constant across them. If the difference in state values of two temporally adjacent states is bounded by the reward signal r, then a series of k transitions among $k+1$ states implies that the maximal difference in state values is limitted to $k * r$.

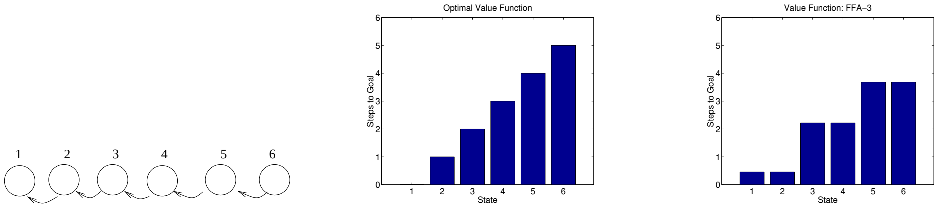
Notice the class of RL control problems commonly referred to as *steps-to-goal* problems are a subset of the class of frequency bounded problems. The steps-to-goal class includes Mountain Car, Puddle World, Maze World / Grid World, Acrobot, and many others. In these types of problems, the value function is a measure of how many steps remain before the agent reaches the goal state. Here the reinforcement signal is constant at each step. Also included in the class of frequency bounded problems are goal-avoidance problems such as the Pole Balancer.

In selecting a function approximator for RL tasks we desire to have each basis function cover states in which the optimal value functions are similar. This is advantageous because the value or weight of the basis function must generalize to all the states which it covers. We do not want to form a basis function that covers states whose optimal values differ widely. The method we present in this paper forms basis functions along temporal neighborhoods. We do this because we believe that the value functions of two temporally adjacent states are likely to be more similar than the value functions of two physically adjacent states. Thus, if we can form basis functions along temporal neighborhoods, we should expect to be able to better model the value function of the state space.

3 A Simple Markov Example

To facillitate discussion of this method, we will use a simple example to demonstrate how temporal neighborhoods work and their effectiveness in better modelling the value function. In Figure 1a we have a six state Markov Chain. State 1 is the goal state (absorbing); there is one action available from each state which transitions according to the arrows in the diagram. There is a reward signal of $+1$ for each step. We "sample" by starting randomly in one of the states and taking actions until we reach the goal state.

It is easy to compute the true optimal value function (steps-to-goal function) by inspection; the value function is shown in Figure 1b. We now apply traditional TD(0) [8] using three basis functions: ϕ_1, ϕ_2, ϕ_3 where ϕ_1 covers states 1 and 2, ϕ_2 covers states 3 and 4, and ϕ_3 covers states 5 and 6. By "covering" we mean that ϕ_1 is active when the agent is in state 1 or state 2: $\phi_1 = 1.0$ in these states while $\phi_2 = \phi_3 = 0$. This means that the weight associated with ϕ_1 will generalize to

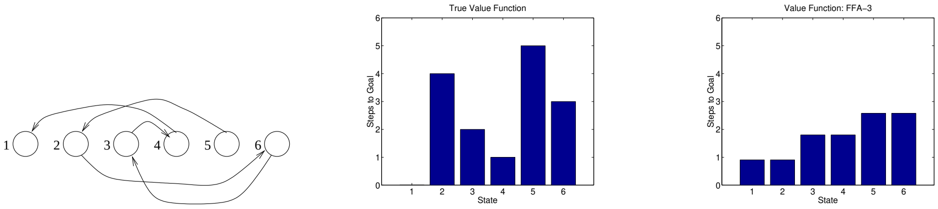


a. Markov Chain b. Optimal Value Function c. Approx Value Function

Figure 1: A six state Markov Chain

approximate the value functions for states 1 and 2. The weight associated with each basis function, W_i is trained by on-line sampling. We arrive at the approximated value function shown in Figure 1c. We can see that this function approximator represents the true value function with reasonable accuracy.

Next we change the example slightly by re-aligning the transition probabilities according to the arrows in Figure 2a. This example purposely contrasts physical adjancency (the states are lined up 1 through 6) with temporal adjacency (the states transition in the order $5 \mapsto 2 \mapsto 6 \mapsto 3 \mapsto 4 \mapsto 1$).



a. Markov Chain b. Optimal Value Function c. Approx Value Function

Figure 2: Six State Temporal Markov Chain

The optimal value function for this new chain is shown in Figure 2b. We also apply TD(0) with the same basis functions in the previous problem (ϕ_1 covers states 1 and 2, ϕ_2 covers states 3 and 4, and ϕ_3 covers states 5 and 6). The approximated value function is shown in Figure 2c. As can be seen, the approximated value function does not accurately represent the true value function. Here is a classical case of how a poor choice of function approximator can drastically affect the performance of the learning algorithm. At this point, traditional research methods would typically attempt to overcome this problem by adding more basis functions. An alternative would be to re-arrange the basis function coverage to better suit the true value function. Because this problem is relatively simple it is easy to see that $\phi_1 = \{1, 4\}$, $\phi_2 = \{3, 6\}$, and $\phi_3 = \{2, 5\}$ would be a better choice for basis functions. However this realization requires that we know the transition probabilities a priori.

What is required is an algorithm which aligns basis functions along these trajectories by sampling the transition probabilities.

4 Temporal Neighborhoods Algorithm

In this section we sketch the major elements of an algorithm which aligns basis fuctions according to temporal neighborhoods. We have a state space with N states. We will cover this state space with a function approximation matrix Φ composed of

K orthogonal basis functions $(\phi_1, \ldots, \phi_K)$. Each basis function is an N dimensional column vector where each entry indicates the "activation level" (amount of coverage) of the basis function for that state.

$$\Phi = \begin{pmatrix} | & | & & | \\ \phi_1 & \phi_2 & \vdots & \phi_K \\ | & | & & | \end{pmatrix}$$

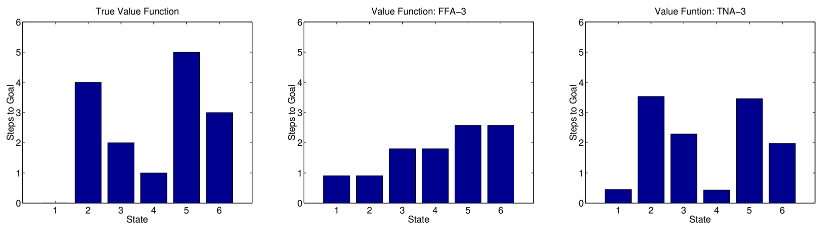
There are a set of K weights stored in a column vector W. The weights are trained via TD(0) to arrive at the best approximation which is computed as $Value = \Phi * W$. In order to keep basis functions local and to ensure even distribution throughout the state space we use a normalization procedure. Here is a sketch of how the algorithm operates:

1. The agent starts in state i, selects an action and transitions to state j.
2. We then find the basis function k which currently has the largest activation for states i and j combined: $k = argmax\{\phi_k(i) + \phi_k(j)\}$.
3. We then increase the activation for basis function k on states i and j: $\phi_k(i) += \Delta$, $\phi_k(j) += \Delta$.
4. The basis functions are then normalized to ensure that each remains local and that the state space is adequately covered.

We alternate periods of TD(0) with iterations of the above temporal neighborhoods algorithm to alternately train the weights and shape the basis vectors.

In the previous six state Markov Chain we show the fixed function approximator below on the left. After applying the temporal neighborhoods algorithm, the basis vectors have re-aligned to those shown on the right. In Figure 3a is the optimal value function for this Markov Chain. Figure 3b shows the results of TD(0) on the fixed function approximator and Figure 3c is the value function for the basis functions formed by the temporal neighborhoods method. As can be seen, the temporal neighborhoods basis functions clearly better approximate the optimal value function.

$$\Phi_{fixed} = \begin{pmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \\ 0 & 0 & 0.5 \end{pmatrix} \qquad \Phi_{Temp.\ Neigh.} = \begin{pmatrix} 0 & 0.57 & 0 \\ 0 & 0 & 0.51 \\ 0.56 & 0 & 0 \\ 0 & 0.54 & 0 \\ 0 & 0 & 0.49 \\ 0.47 & 0 & 0 \end{pmatrix}$$



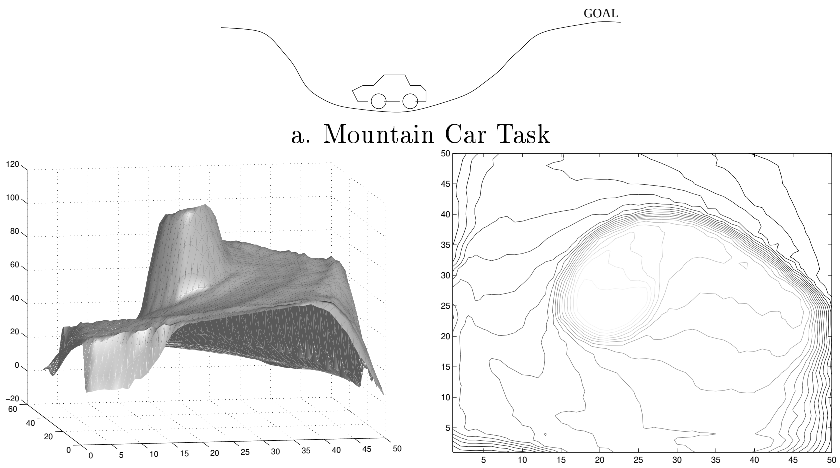
a. Value Function b. w/ Fixed Bas Fxn c. w/ Temp Neigh

Figure 3: Value Functions for Temporal Markov Chain

The temporal neighborhoods algorithm succeeded in redistributing the basis functions according to transition probabilities. The improved basis functions did indeed better approximate the value function for this task. This task is purposely trivial so that one can follow closely the details of the algorithm. Next we present a more complex task, the mountain car, to further demonstrate the effectiveness of temporal neighborhoods.

5 Moutain Car Problem

There is a small car positioned in a valley (see Figure 4a). The goal is to drive the car out of the valley to the hill on the right. However, the car's engine does not have enough power to drive straight up the hill; first the car must rock backward to gain momentum and then drive to the top of the hill. For reference, the optimal value is shown in Figure 4b. Along the x-axis (the bottom right) is the car's position. The y-axis (bottom left) indicates the car's velocity. The z-axis is the value function at each (x,y) state – this represents the number of steps to the goal from that state. A contour plot of the value function is shown below in Figure 4c.



a. Mountain Car Task

b. Optimal Value Function

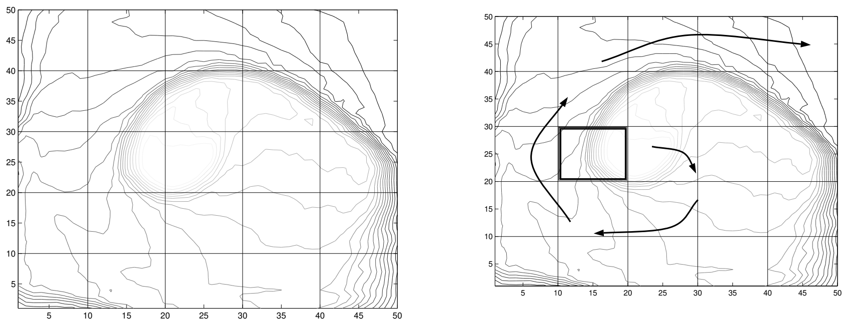
c. Contour Plot of Value Function

Figure 4: Mountain Car Task

We use 25 discrete basis functions to learn the value function at this coarse level using TD(0). The basis functions are initially deployed in a 5x5 non-overlapping grid (orthogonal)on the two dimensional state space. In Figure 5a the grid is visible on the contour plot. Each "box" is a basis function. The weight of the basis function will learn to approximate the values of the states inside the box.

When we apply TD(0) each basis function will approximate the average value function the states in its "box". Notice from the contour plot of the value function, that several basis functions span states which have very different values. In particular the basis function highlighted with the dark border in Figure 5b sits right on the edge of a "steep value-function cliff"; this is not a good position for this basis function because it must average the low state values on one side and the high state values on the other side.

The arrows in in Figure 5b show a typical trajectory as the mountain car moves through the state space. The car starts from rest at the bottom of the hill (the



a. 5x5 Grid of Basis Fxns   b. Typical MtnCar Trajectory

Figure 5: Contour Plot of MtnCar Task

center of the contour plot), it rocks backward (spirals clockwise toward the bottom right) and then drives forward to the top of the goal hill. This trajectory spirals clockwise out from the center to the upper right-hand corner.

We then apply the Temporal Neighborhoods Algorithm on the Mountain Car problem which causes the basis functions align themselves along this common trajectory. As in the simple Markov Chain example, we alternate episodes of TD(0) with Temporal Neighborhoods. In Figure 6 we can see the result of the re-alignment of basis functions. Many of the basis functions have aligned themselves along the clockwise spiral trajectory that dominates the mountain car state space.

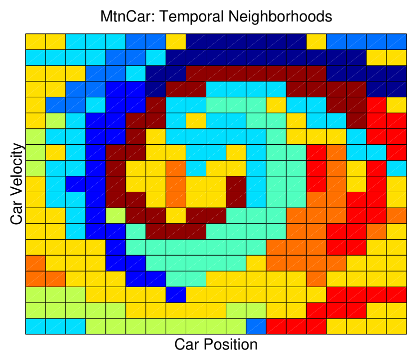


Figure 6: Temp Neigh Basis Functions

Basis Function	**Average Approx Error**
Fixed 5x5 Grid	25.4
Temporal Neigh.	17.6

Because these new basis functions are aligned along common trajectories (temporal neighborhoods), fewer of the basis functions cover drastically steep gradients in the value function. These basis functions can therefore better approximate the mountain car steps-to-goal value function. In the table below, we summarize the average approximation error (mean squared error as measured by the optimal value function) for both the standard 5x5 basis functions and the basis functions shaped by temporal neighborhoods. The temporal neighborhoods algorithm reduced the approximation error by 31

6 Summary and Future Work

Here we introduce the notion of temporal neighborhoods which are small sets of states that experience frequent intra-set transitions. We present an algorithm which aligns basis functions along temporal neighborhoods. We have discussed why this might be an improvement for selecting basis functions over the grid method: namely we believe that for certain classes of problems (frequency bounded problems) the value function is more similar for temporally adjacent states than it is for pysically adjacent states. Thus we should expect that basis functions aligned along temporal trajectories should be better able to model the optimal value function. We have supported our theory with empirical evidence. In a very simple Markov Chain we see a dramatic difference in function approximation performance using the temporal neighborhoods algorithm to align basis functions. In the more complex Mountain Car task, again we see the benefits of using temporal sequences to form basis functions.

The primary reason for approximating the value function is that the curse of dimensionality makes it intractable to use a table method in which we maintain the value function for each individual state. There are simply too many states to create such a table. The work in this paper sidesteps this issue; namely we have built basis functions using large state vectors. This is impracticle for large and highly dimensional state spaces. A more plausible scheme is to use CMACs, radial basis functions, or some other standard approximation method. This work does indicate the utility of re-alignment of basis functions along temporal trajectories; an obvious next step for this work is to combine the temporal neighborhoods algorithm with these more tractable function approximation schemes.

Re-aligning basis functions may not be the most powerful use of temporal perceptions. Recently there has been breakthrough work in Reinforcement Learning at multiple temporal scales. This work shows immense potential at bridging the gap between symbolic-level planning and subsymbolic-level AI. A key concept in these multiple time scale algorithms is the formation of options (aka meta-actions, subgoals) in the state space. The early work in multiple time scale RL has "hand crafted" these subgoals in various regions of the state space. Temporal neighborhoods provide a way to construct options incrementally on-line. We have preliminary results indicating the successful application of temporal neighborhoods in multiple time scale Reinforcement Learning.

References

[1] Charles W. Anderson. Q-Learning with Hidden-Unit Restarting. *Advances in Neural Information Processing Systems*, volume 5, S. J. Hanson, J. D. Cowan, and C. L. Giles, eds., Morgan Kaufmann Publishers, San Mateo, CA, pp. 81–88.

[2] Andrew G. Barto, and Richard S. Sutton. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA. 1988.

[3] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*: 4, May 1996.

[4] R. Matthew Kretchmar and Charles W. Anderson. Comparison of CMACs and Radial Basis Functions for Local Function Approximators in Reinforcement Learning. *Proceedings of the International Conference on Neural Networks*. 1997.

[5] R. Andrew McCallum. Using Transitional Proximity for Faster Reinforcement Learning. International Conference on Machine Learning, 1992.

[6] Andrew W. Moore. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces. *Machine Learning*: 21, 1995.

[7] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement Learning with Soft State Aggregation. *NIPS94*, 1994.

[8] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*:3, 1988.

[9] John N. Tsitsiklis, and Benjamin Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, Vol. 42, No. 5, May 1997.

[10] Bruce A. Whitehead, and Timothy D. Choate. Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction. *IEEE Transactions on Neural Networks*, Vol 7, No 4. July 1996.