Model-Based Reinforcement Learning with an Approximate, Learned Model

Leonid Kuvayev
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
kuvayev@cs.umass.edu

Richard S. Sutton
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
rich@cs.umass.edu

Abstract

Model-based reinforcement learning, in which a model of the environment's dynamics is learned and used to supplement direct learning from experience, has been proposed as a general approach to learning and planning. We present the first experiments with this idea in which the model of the environment's dynamics is both approximate and learned online. These experiments involve the Mountain Car task, which requires approximation of both value function and model because it has continuous state variables. We used models of the simplest possible form, state-aggregation or "grid" models, and CMACs to represent the value function. We find that model-based methods do indeed perform better than model-free reinforcement learning on this task, but only slightly.

1 INTRODUCTION

The most impressive successes of reinforcement learning so far have all used extensive offline experience with a model or simulation of the task in order to attain a high level of performance (Tesauro, 1992; Crites & Barto, 1996; Zhang & Dietterich, 1995). In these cases the model could be assumed completely known a priori, but models can also be useful even if they must be learned. Several researchers have demonstrated that reinforcement learning can be significantly accelerated if a model of the environment's dynamics is learned online and used to supplement direct learning from experience (Sutton, 1990, 1991; Moore & Atkeson, 1993; Peng & Williams, 1993). However, all prior work in which the model was learned used simple table-lookup methods to represent the model. These methods did not involve any generalization or transfer between states. Although they can be used for moderately large problems (Moore and Atkeson (1993) demonstrated their effectiveness for problems with tens of thousands of states), really large problems require the use of generalizing function approximators to represent the model.

The switch from table-lookup approaches to those based on function approximators has been found to be a significant one for model-free reinforcement learning (Boyan & Moore, 1995; Sutton, 1996; Tsitiklis & Van Roy, 1994). While a wide class of methods have been proven convergent for the table-lookup case, many of these, including Q-learning and dynamic programming methods, appear to be unstable when even simple function approximators are used (Baird, 1995; Gordon, 1995). Other methods, such as the Sarsa algorithm we use here (Rummery & Niranjan, 1994; Singh and Sutton, 1996) appear not to have these problems (Tsitsiklis & Van Roy, 1996).

It is possible that model-based reinforcement learning will generalize naturally and easily to the use of learned, approximate environmental models, but this is by no means certain. Model-free methods have the advantage that they are not affected by modeling errors. They always learn directly from real experience, which, however noisy or rare, is always a true sample of the real system. Whenever learning is done from an approximate model there arises the danger that modeling errors will permanently harm performance. In this and other respects the case of model-based learning in which the model is inherently approximate is a critical test of the idea of model-based learning.

In this paper we present the first experimental tests of model-based reinforcement learning with approximate learning models. We use a standard testbed problem, the Mountain Car task. Because this problem has continuous state variables, true table-lookup methods are impossible (there are an infinite number of states). The following sections describe the task and the model-free and model-based methods we apply to it.

2 THE MOUNTAIN CAR TASK

For our experiments we used the Mountain Car task (Moore, 1990). In this task a car starts at the bottom of a mountain and drives along a mountain track. The goal is to overpass the mountain top, but the motor is too weak to drive directly to the top. Instead, the car must first backup from the goal, then drive forward at full thrust. The version of this task that we used in these experiments are the same as that used by Boyan and Moore (1995). The details of the task are given below.

There are two continuous state variables, the position of the car, x_t, and the velocity of the car, v_t. The valid ranges are $-1 \leq x_t \leq 1$ and $-2 \leq v_t \leq 2$. The equations describing the system are:

$$q_t = \begin{cases} 2 \cdot x + 1 & \text{if } x < 0 \\ \frac{1}{(1+5x_t^2)^{3/2}} & \text{if } x \geq 0 \end{cases}$$

$$a_t = \frac{f_t}{m \cdot \sqrt{1+q_t^2}} - \frac{g \cdot q_t}{1+q_t^2}$$

$$x_{t+1} = x_t + v_t \cdot \Delta t + \frac{a \cdot \Delta t^2}{2}$$

$$v_{t+1} = v_t + a \cdot \Delta t$$

$$m = 1,\ g = 9.81, \text{ and } \Delta t = 0.03$$

The force, f_t, can take three distinct values -4, 0, or +4 corresponding to the three actions, reverse thrust, no thrust, or forward thrust. If x_{t+1} or v_{t+1} go out of range, then they are reset to the boundary value. The trial starts at the bottom of the mountain with $x_0 = -0.5$ and $v_0 = 0$. Reward is -1 on all time steps. The trial terminates with the first position value that exceeds $x_{t+1} > 0.5$.

3 REINFORCEMENT LEARNING AND FUNCTION APPROXIMATION

We used a simple CMAC network to represent the approximate value function, as described by Sutton (1996). CMAC networks require less memory than table lookup approaches and possess excellent convergence speed and solution quality (e.g., see Albus, 1981; Miller et al., 1990). We have also obtained satisfactory results with backpropagation networks, but they always learn much slower than CMAC networks, particularly in the initial stages of training.

In this paper's experiments we used a CMAC consisting of 10 tilings, each a simple 10 by 10 grid. The total number of tiles was $10 \cdot 10 \cdot 10 = 1000$. Each tiling was offset from the previous by one-tenth of the width of a tile in each direction, giving a uniform spacing. We also experimented with a 5-tiling CMAC network and obtained similar results with some loss of stability.

4 MODEL-FREE LEARNING

In these experiments we used the Sarsa model-free algorithm both as a basis for comparison and as the underlying algorithm for the model-based method. Although closely related to Q-learning (Watkins, 1989), Sarsa was preferred here because of its better convergence assurances in the case of approximate value functions (Tsitsiklis & Van Roy, 1996). Empirically, we also found sarsa to perform slightly better than Q-learning (consistent with the larger study by Rummery, 1995). We used the sarsa algorithm exactly as given in (Sutton, 1996) and (Singh & Sutton, 1996), except that actions were selected not according to an

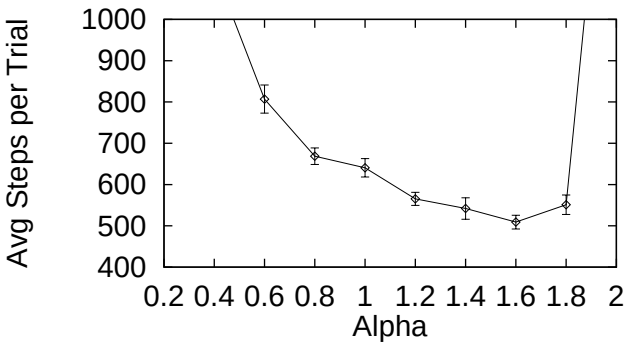

Figure 1: Convergence speed at various step sizes.

ϵ-greedy policy, but according to the Boltzmann distribution. The probability of selecting action a in state s was determined from its action value, $Q(s,a)$ as

$$\frac{e^{Q(s,a)/T}}{\sum_{b \in Actions} e^{Q(s,b)/T}}$$

where T is a "temperature" parameter, controlling the degree of exploration. In all of our experiments we used $T = 0.2$.

The first experiment evaluated the performance of this model-free method as a function of the step-size parameter, α. For each value of α we allowed learning to occur over 20 trials and measured the average steps per trial over the run. Results averaged over 30 runs are shown in Figure 1. The mean and standard error are shown for each values of α. Best performance is attained in the neighborhood of $\alpha = 1.6$. This value was used in all further experiments for both model-free and model-based methods.

5 MODEL-BASED LEARNING

We propose to learn the model of the environment while obtaining on-line experience and then use this model to facilitate learning. Given a state and action the model of the environment predicts the next state. The mapping can be learned from observing actual state transitions. In the case of discrete state space the transitions can be stored in the lookup table. However in our study the state space is continuous, hence the mapping of a state and action to a next state needs to be approximated. The simplest approach using state aggregation was implemented. That is, a fine grid was laid over the 2-dimensional state space, and each real state represented by the grid box within which it fell. For any given box, the observed next states may fall in several different boxes. The model stores all observed next boxes along with their frequencies of occurrence. When the model is used to generate hypothetical next states, one of the previously observed boxes is drawn with probabilities according to the observed frequencies. For the purposes of computing the action (Q) values of the new box, it was taken to be the continuous state at the lower corner of the cell. The complete

model-based Sarsa algorithm is as follows:

1. Initially: $w(t) := 0$, $\forall t \in Tiles, s_{sim} = s_0$, $a_{sim} = policy(s_{sim})$
2. Start of Trial: $s = s_0$, $a := policy(s)$
3. Take action a; observe reward, r, and next state s'
4. $a' := policy(s)$
5. Learn:
   $\epsilon := r + \sum_{t' \in Tiles(s',a')} w(t') - \sum_{t \in Tiles(s,a)} w(t)$
   $w(t) := w(t) + \frac{\alpha}{L} \cdot \epsilon, \forall t \in Tiles(s,a)$
6. Update Model: Add a new observation s' to a list of past observations kept in the hash table entry $m(s,a)$. If s' is already in the table then increment the number of times, s' has been observed, by 1
7. Sample Model:
   Repeat K times
   take action a_{sim};
   use model to compute the predicted next state, s'_{sim}, and reward, r';
   if s'_{sim} is the terminal state
   set $s_{sim} = s_0, a_{sim} = policy(s_{sim})$
   go to the beginning of the loop
   $a'_{sim} := policy(s_{sim})$;
   learn: $\epsilon := r + \sum_{t' \in Tiles(s'_{sim}, a'_{sim})} w(t') - \sum_{t \in Tiles(s_{sim}, a_{sim})} w(t)$
   $w(t) := w(t) + \frac{\alpha}{L} \cdot \epsilon, \forall t \in Tiles(s_{sim}, a_{sim})$
8. Loop: $a := a'; s := s'$; if s' is the terminal state, go to 2, else go to 3

To gain confidence in the model-based approach and to compare the effect of the grid resolution for the model we designed the following experiment. We ran the algorithm for repeated epochs each of 20 trials. The model continued to improve across epochs, but the value functions was reset to zero at the beginning of each epoch. The improvement in average performance over epochs shown in Figure 2 thus shows the ability of a better and better model to induce better and better performance. Results are shown for models of two different grid resolutions. The high resolution 100×100 model is learned slower but shows more accurate performance. The low resolution 30×30 model is learned much more quickly but can not model the system as accurately. In either case it is clear that a good model enables much better performance than is possible without a model (cf. Figure 1).

Finally, Figure 3 compares the performance of the model-free and the model-based methods. For the model-free method, we used the optimal parameters found in the earlier study. We varied the number of model steps, K, from 0 to 10. Performance is shown averaged over the first 20 and first 500 trials. These data are averages over 30 runs.

The first data point, with $K = 0$, corresponds to a model-free Sarsa algorithm. Performance increases

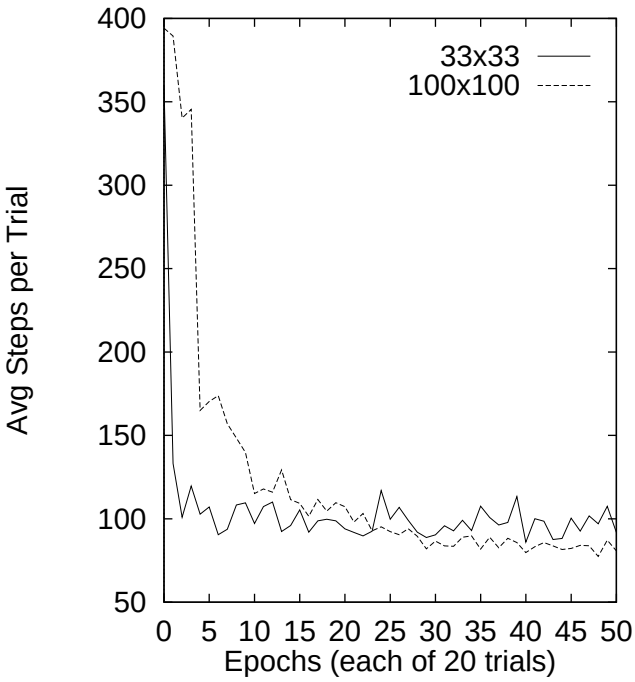

Figure 2: Learning during each epoch is better and better as the model improves over epochs, with the higher-resolution model improving more slowly but ultimately being better. Note that all these performances are better than those attainable without a model (i.e., best performance from Figure 1 is about 500 steps per trial). For this experiment, $K = 10$.

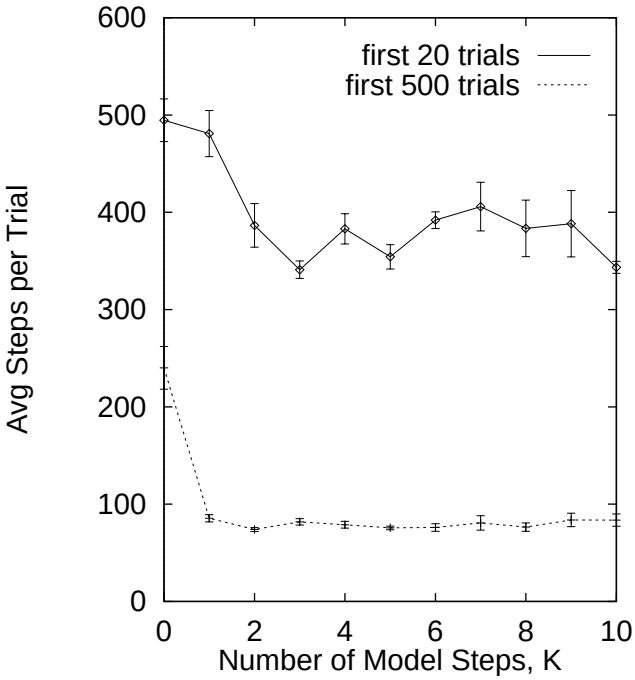

Figure 3: Performance versus K, where $K = 0$ corresponds to model-free learning.

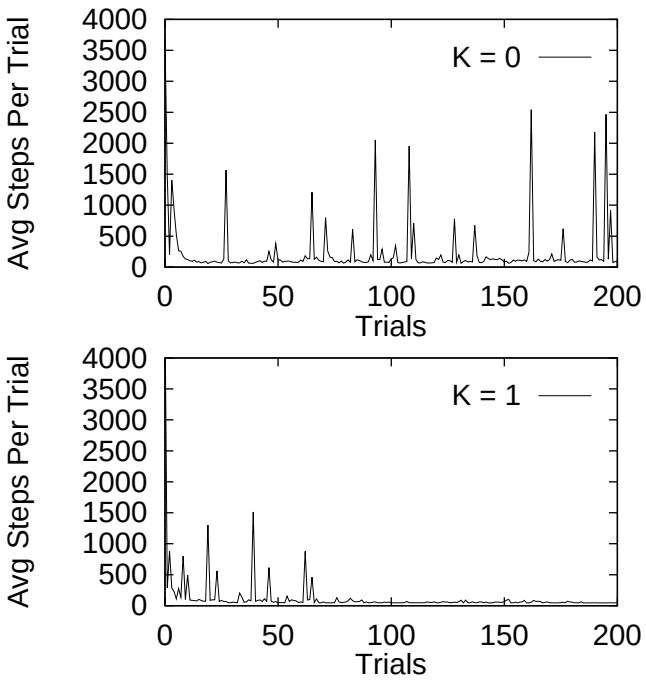

Figure 4: The learning curves for model-free (top) and model-based (bottom) algorithms.

significantly for the model-based methods. In addition, the model-based methods appeared to be more stable, as shown in the learning curves in Figure 4.

6 CONCLUSIONS AND FUTURE WORK

These experiments are just the first steps in evaluating the effectiveness of model-based methods in reinforcement learning with approximate, learned models. On the one hand, we have found significant performance improvements with the model based methods. On the other, we have been impressed by the resilience of the model-free methods. It was not easy to beat them on this task, and they are much simpler. In particular, model-based methods tend to consume a great deal of memory. This is exacerbated by the grid-like models we used in these experiments. Our plans for the immediate future are to experiment with more sophisticated models, using CMAC networks or local-weighted regression methods, and to experiment with other tasks. In principle, the use of models should produce dramatic, easy, and immediate performance improvements. We would like to find a test problem and modeling methodology which clearly demonstrates this potential.

Acknowledgements

We thank Andrew Moore, for suggesting this direction of research, and all the members of the Adaptive Networks Group at the University of Massachusetts for helpful feedback and discussions. This work was supported by the National Science Foundation under grant ECS-9511805 to Andrew Barto and Richard Sutton.

References

Albus, J. S. (1981). *Brain, Behavior, and Robotics*, chapter 6, pages 139–179. Byte Books.

Baird, L. C. (1995). Residual Algorithms: Reinforcement Learning with Function Approximation. *Proc. ML95*. Morgan Kaufman, San Francisco, CA.

Boyan, J. A. & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. *NIPS-7*. San Mateo, CA: Morgan Kaufmann.

Gordon, G. (1995). Stable function approximation in dynamic programming. *Proc. ML95*.

Crites, R. H. & Barto, A. G. (1995). Improving elevator performance using reinforcement learning. Presented at NIPS'95. To appear in *Proceeding NIPS-8*.

Miller, W. T., Glanz, F. H., & Kraft, L. G. (1990). CMAC: An associative neural network alternative to backpropagation. *Proc. of the IEEE*, *78*, 1561–1567.

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.

Rummery, G.A. (1994). Problem Solving with Reinforcement Learning. Cambridge University PhD Thesis.

Rummery, G. A. & Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Dept.

Singh, S. P. & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*.

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

Sutton, R. (1991). Dyna, an integrated architecture for learning, planning and reacting," *Working Notes of the 1991 AAAI Spring Symposium on Integrated Intelligent Architectures* and *SIGART Bulletin*, *2*, pp. 160-163.

Sutton, R. (1996). Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. *Advances in Neural Information Processing Systems 8*, MIT Press.

Tesauro, G. J. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*(3/4), 257–277.

Tsitsiklis, J. N. & Van Roy, B. (1994). Feature-based methods for large-scale dynamic programming. Techical Report LIDS-P2277, MIT, Cambridge, MA 02139.

Tsitsiklis, J. N. & Van Roy, B. (1996). An analysis of temporal-difference learning with function approximation. Techical Report LIDS-P2322, MIT, Cambridge, MA 02139.

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

Zhang, W. & Dietterich, T. G., (1995). A reinforcement learning approach to job-shop scheduling." *Proc. IJCAI95*.