Model-Based Reinforcement Learning with an Approximate, Learned Model

Leonid Kuvayev **Rich Sutton**

Department of Computer Science
University of Massachusetts
Amherst, MA 01003

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.
Keywords: Reinforcement learning, planning, model-based learning, function approximation, CMAC networks.

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), (Moore & Atkeson 1993), (Peng & Williams 1992). 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 & 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), (Tsitsiklis & 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 simple function approximators are used (Baird 1995), (Gordon 1995). Other methods, such as the Sarsa algorithm we use here (Rummery & Niranjan 1994), (Singh & Sutton 1996) appear not to have these problems (Tsitsiklis & 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 present a new algorithm for learning with both exact and approximate models and study the intricacies of using approximate models.

2 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 & Roy 1996). Empirically, we also found sarsa to perform slightly better than Q-learning (consistent with the larger study by (Rummery & Niranjan 1994). We used the sarsa algorithm exactly as given in (Sutton 1996) and (Singh & Sutton 1996), except that actions were selected not according to an 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 $T = 1.0$.

3 Learning a Model

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 we also experimented with the continuous state space task, 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.

4 Using a Model

The simple way of using the model is implemented in Dyna algorithm (Sutton 1990). The model updates are selected randomly among the updates already experienced in reality. To implement this method we choose state-action pairs randomly among all legal pairs and then discard the ones that have not yet been seen. The method does not distinguish between more and less relevant updates, all updates have an equal chance to be executed.

There are other methods for choosing updates more effectively, e.g. Prioritized Sweeping (Moore & Atkeson 1993) and Focused Dyna (Peng & Williams 1992) where the updates are selected based on the prediction difference and the closeness to the starting state. All updates are gathered in the queue and the ones that have the biggest difference or the ones that are nearest to the starting state are updated first. These methods were more effective than Dyna on grid world tasks.

4.1 Trajectory Model Updates

We propose an intuitive method called *Trajectory Model Updates* that also outperforms Dyna. The idea is to select updates by generating hypothetical trajectories through the state space. For every real step we take k hypothetical steps. The start and goal states of the hypothetical trajectory are the same as of the real one. If after a hypo step the goal is reached then the next step proceeds from the start state. The rewards and the Boltzmann action selection are the same for hypothetical and real experiences. Thus, we can think of them as of two parallel processes where one is the real exploration and the other one is a hypo exploration. The latter has k times more updates than a real exploration. The difference between there processes is that when a hypothetical exploration attempts a transition that is not in the model the hypo trial is reset to the start state. Also if the model is not accurate the hypothetical exploration may make transitions that are not possible in the real world.

The complete *Trajectory Model Updates* 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

This algorithm is similar to Experience Replay algorithm (Lin 1993) only instead of replaying a past experience we generate a new one using a model of the world and the same action selection mechanism that we use for the real experiences. Experience replay idea is fruitful when model is unavailable and cannot be learned. In this case the accumulated past experiences provide a surrogate model. However, if the model could be learned, the hypothetical experiences should better cover the state space and thus provide more efficient learning.

5 Grid World

To evaluate the potential of these model-based learning methods we experimented with a simple maze task taken from (Peng & Williams 1992). The maze is shown on Figure 1. A trial begins at the Start state and terminates at the Goal state. Each move has penalty of -1. Hitting the wall does not produce a movement. The learning rate, 0.5, was the same as in Peng and Williams' paper.

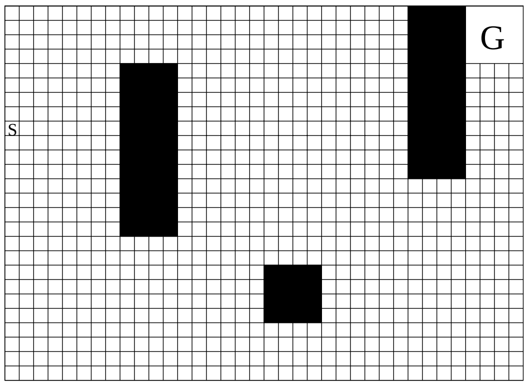

Figure 1: A maze navigation task. Each step bears a penalty of -1. Reaching the goal state terminates the trial. Hitting a wall does not produce a movement.

We compared two model-based methods and a model-free sarsa algorithm. For each real step the model-based methods executed $k = 10$ hypothetical steps. A model for the maze world was easy to learn since all transitions were deterministic and a model constituted a table lookup. Both model-based methods did 11 times more updates and outperformed model-free method nearly 10 times (cf. Figure 2). After 100 trials a model was learned perfectly. Out of the two model-based methods, *Trajectory Updates* slightly beat *Random Updates*. The first algorithm used more focused updates and thus achieved a faster convergence than a latter one. Once converged, both methods exhibited the stable performance due to the accurate model and table lookup function. We will see a more significant difference in the case with an approximate model.

6 The Mountain Car Task and an Approximate Model

For the experiments with an approximate model 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

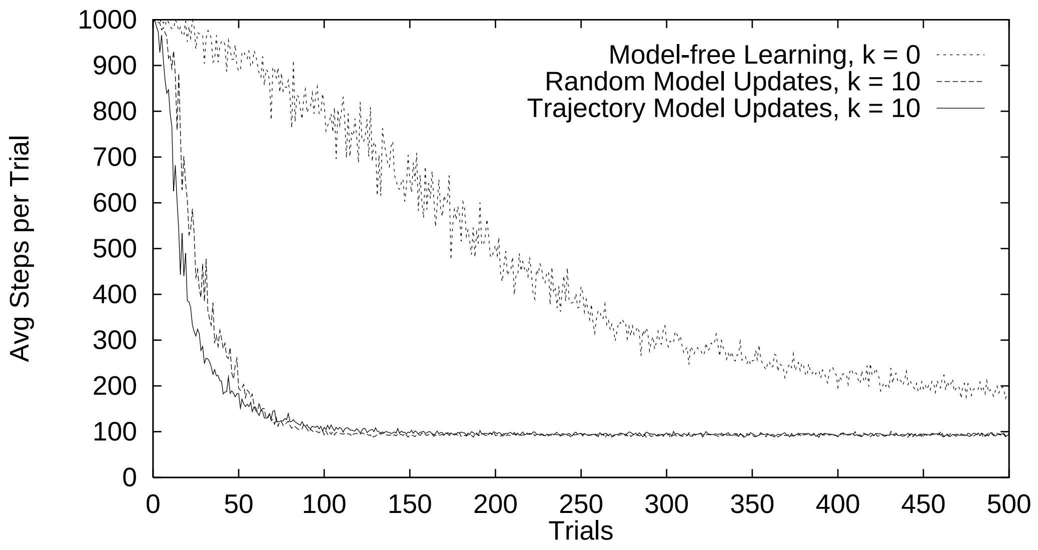

Figure 2: Learning curves of model-free and model-based algorithms for a maze task averaged over 30 trials. k is the number of model-based updates for one real update. A model of the maze is learned quickly and soon becomes the perfect model. Both model-based methods outperformed model-free method nearly 10 times.

this task that we used in these experiments are the same as that used by (Boyan & 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:

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

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

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

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 $x_{t+1} > 0.5$.

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, Glanz, & Kraft 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 5 tilings, each a simple 9 by 9 grid. The total number of tiles was $9 \cdot 9 \cdot 5 = 405$. Each tiling was offset from the previous by one-tenth of the width of a tile in each direction, giving a uniform spacing.

6.1 The Experiments

First we repeated the same experiment as we did for a maze task. Only this time the model was never perfect due to the continuity of the state space. Results are shown on Figure 3. Model-based methods did not have as much advantage as in the case with the exact model. Despite 6 times more updates, model-based methods converged only twice as fast as the model-free

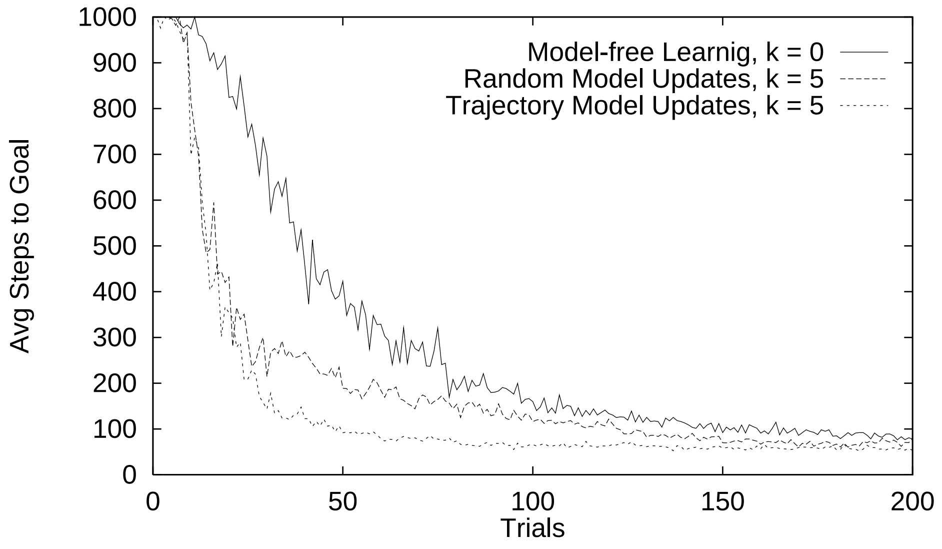

Figure 3: The learning curves for model-free and model-based algorithms. The trials are averaged over 30 runs. Trajectory model updates are the most efficient way to use an approximate model.

method. Again *Trajectory Model Updates* outperformed *Random Updates.* Convergence speed was approximately the same but the first method converged to a more optimal solution.

Next we describe the experiments using *Trajectory Model Updates* that will shed some light on the intricacies of the learning with an approximate model. 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 4 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 33×33 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 5 compares the performance of the model-free and the

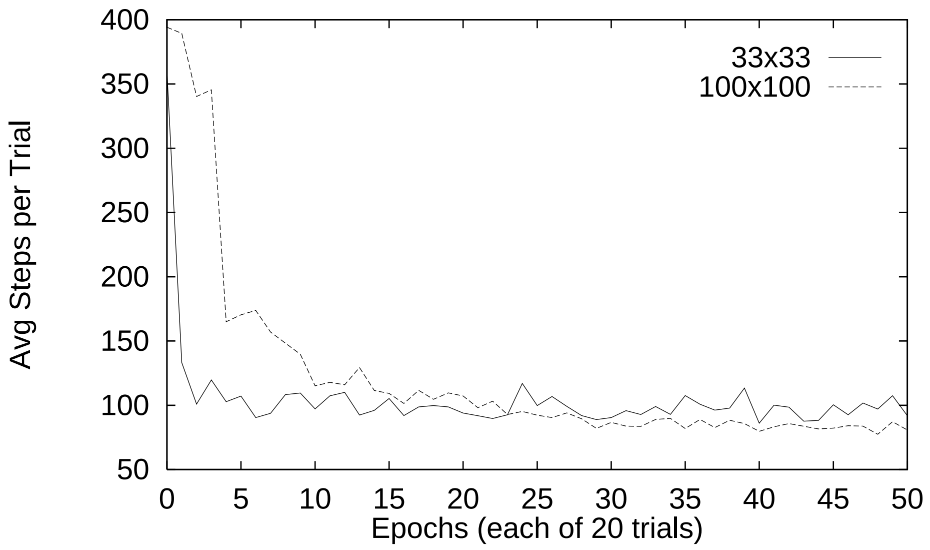

Figure 4: Learning during each epoch is better and better as the model improves over epochs, with the higher-resolution model improving more slowly but ultimately being better. Note that all these performances are better than those attainable without a model (i.e., best performance from Figure 1 is about 500 steps per trial). For this experiment, $K = 10$.

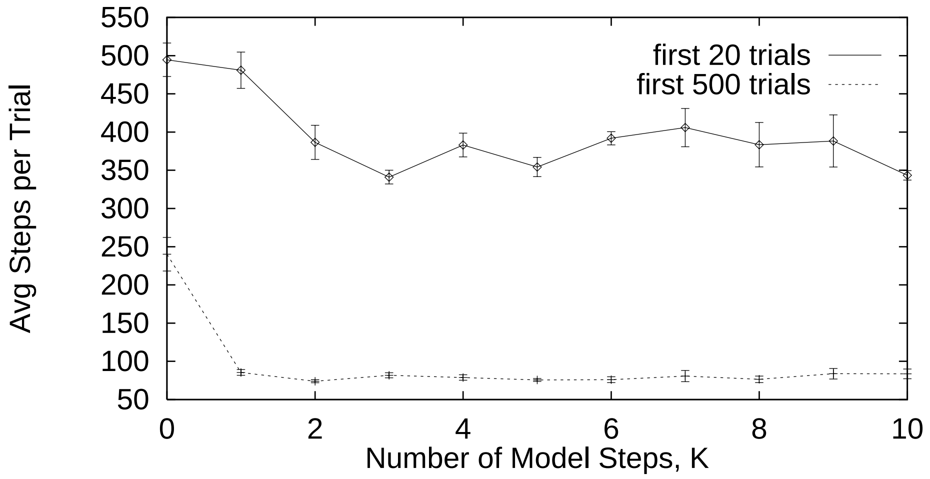

Figure 5: Performance versus K, where $K = 0$ corresponds to model-free learning. Performance improves slightly as the number of model steps is increased.

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 improves significantly after adding only a few model steps. The improvement stops with additional model steps since the model is scarce and inaccurate in the beginning and does not supply a useful information.

7 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. We have found significant performance improvements with the model based methods. 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. However, the speedup of several times during the convergence well justifies the extra memory costs.

We proposed a *Trajectory Model Updates* algorithm and compared it against the other well-known model-based algorithm Dyna. The proposed method showed a faster and more optimal convergence on tasks with exact and approximate models.

This study could be augmented with additional comparison with algorithms, such as Prioritized Sweeping (Moore & Atkeson 1993), Focused Dyna (Peng & Williams 1992), and Experience Replay (Lin 1993) on a bigger continuous task. 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.

References

Albus, J. 1981. *Brain, Behavior, and Robotics.* Byte Books.

Baird, L. C. 1995. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Machine Learning Conference.* San Francisco, CA: Morgan Kaufman.

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

Crites, R., and Barto, A. 1996. Improving elevator performance using reinforcement learning. In DS Touretzky, M. M., and Hasselmo, M., eds., *Advances in Neural Information Processing Systems 8.* Cambridge, MA: MIT Press.

Gordon, G. 1995. Stable function approximation in dynamic programming. In *Proceedings of the Machine Learning Conference.*

Lin, L.-J. 1993. Scaling up reinforcement learning for robot control. In *Proceedings of the Tenth International Conference on Machine Learning*, 182–189.

Miller, W.; Glanz, F.; and Kraft, L. 1990. Cmac: An associative neural network alternative to backpropagation. In *Proceedings of the IEEE*, 78, 1561–1567.

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

Moore, A. 1990. Acquisition of dynamic control knowledge for a robotic manipulator. In *Proceedings of the 7th International Conference on Machine Learning.* Morgan Kaufmann.

Peng, J., and Williams, R. 1992. Efficient learning and planning within the dyna framework. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 281–290.

Rummery, G., and Niranjan, M. 1994. On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University.

Singh, S., and Sutton, R. 1996. Reinforcement learning with replacing eligibility traces. *Machine Learning* 22:123–158.

Sutton, R. 1990. Integrated architecture for learning, planning, and reacting based on approximating dynamic programming. In *Seventh International Conference on Machine Learning.* Morgan-Kaufmann.

Sutton, R. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8.* MIT Press.

Tesauro, G. 1992. Practical issues in temporal difference learning. *Machine Learning* 8:257–277.

Tsitsiklis, J., and Roy, B. V. 1994. Feature-based methods for large-scale dynamic programming. Technical Report LIDS-P2277, MIT, Cambridge, MA.

Tsitsiklis, J., and Roy, B. V. 1996. An analysis of temporal-difference learning with function approximation. Technical Report LIDS-P2322, MIT, Cambridge, MA.

Watkins, C. 1989. *Learning with delayed rewards.* Ph.D. Dissertation, Cambridge University.

Zhang, W., and Dietterich, T. 1995. A reinforcement learning approach to job-shop scheduling. In *Proceedings of IJCAI-95.*