Model-Based Reinforcement Learning

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

Learnig on-line can be very expensive in the real world. We propose to learn the model of the environment while obtaining on-line experience and then use this model to facilitate learning and avoid costly actual experiences. The model can be learned by different value approximation schemes. In this paper the results are shown for the most straight forward implementation. Nevertheless the improvement in the convergence speed is significant.

1 Introduction

On-line methods proved their usefulness in hands of numerous researchers. This paper first discusses the Mountain Car that has been proven a useful testbed task. In the next section it discusses the methods to approximate state values and describes the CMAC approach. In the Primitive Learning section we provide well known Sarsa algorithm that will serve as a basis for comparison with a new algorithm for model-based learning. The details of this algorithm as well as the experiments are shown in the Model-Based section. Finally the direction for further research is discussed.

2 MOUNTAIN CAR TASK

For our experiments we used Mountain Car task first studied by Boyan and Moore (1995). 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.

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 state transitions conform to the following simplified physics model obtained from Justin

Boyan at CMU:

$$m = 1,\ g = 9.81,\ \Delta t = 0.03$$

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

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

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

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

Force, f_t, can take three distinct values -4, 0, or +4 corresponding to 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 $x_{t+1} > 0.5$.

The steepness of the mountain prevents the car from reaching the top by moving forward from the initial state. The optimal solution is to move away from the mountain top initially in order to gain enough momentum, and then thrust forward until the top is reached.

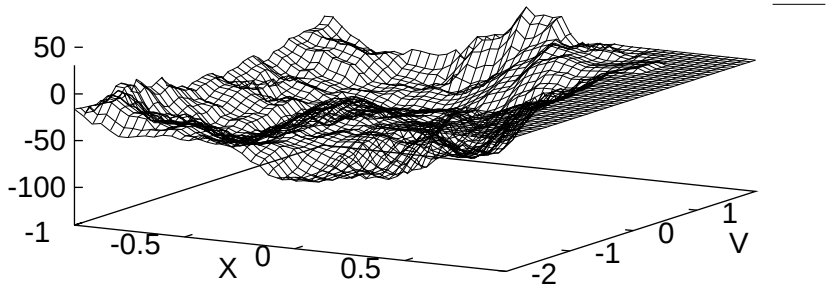

Figure 1: Surface of the value function after 500 trials.

3 REINFORCEMENT LEARNING AND FUNCTION APPROXIMATION

To obtain a good policy we learn the state value function that represents remaining cost to go. The greedy policy with respect to such function will produce the optimal solution.

Several ways were considered to approximate the state value function, including lookup table, CMAC networks and neural networks. Lookup table approach is the most straight forward method but has severe limitations. The first one is the prohibitive memory demand. The other limitation is the poor solution quality due to unavoidable discretisation error. The CMAC networks require less memory and possess excellent convergence speed and solution quality. Their limitation, the dimensionality curse, did not produce any obstacles for the Mountain Car problem with 2-dimensional state space. Neural networks were significantly slower than CMAC during initial stage of learning. It was necessary to spend one or two orders of magnitude more trials compare to CMAC networks before learning curve reached the "knee" point. Neural networks are more suitable for a complex non-linear task where the state space dimensions prohibit the use of the first two methods.

The CMAC network that was used in the experiments consisted of 10 layers with 10 by 10 grid in each layer. The total number of boxes is $10 \cdot 10 \cdot 10 = 1000$. The offsets for each layer were evenly spaced and ranged from 0 to 9/10 of the dimension of the cell. We also experimented with 5-layer CMAC and obtained similar results with a slight loss of stability.

4 PRIMITIVE LEARNING

As a basis for comparison with a new model-based algorithm we selected the Sarsa algorithm, (Sutton, 88). It performed slightly better than the other well know algorithm, Q-Learning introduced by (Watkins, 89). Sarsa algorithm is an on-line learning algorithm were the state values are learned from the temporal differences. The details of the algorithm are the following:

1. Initially: $Q(s,a) := 0$, $\forall a \in \mathit{Actions}$, $\forall s \in \mathit{States}$
2. Start of Trial: $s = s_0$, $a := \mathit{exploration\text{-}policy}(s)$
3. Take action a; observe resultant reward, r, and next state s'
4. $a' := \mathit{exploration\text{-}policy}(s)$
5. Learn: $Q(s,a) := Q(s,a) + \alpha[r + Q(s',a') - Q(s,a)]$
6. Loop: $a := a'$; $s := s'$; if s' is the terminal state, go to 2, else go to 3

The first objective was to determine the best learning rate that can be used for all further experiments. The result is shown in Figure 2. For this experiment we ran the algorithm for 20 trials, calculated the average number of steps and then repeated it 30 times. The mean and standard error of the sample are shown for different values of learning rate, α. The global minimum is reached in the neighborhood of $\alpha = 1.6$. This learning rate was used in further experiments. The unusually high value for learning rate is explained by how the individual cell values are updated. The rule is $w(s,l) := w(s,l) + \frac{\alpha}{L} \cdot \epsilon$ for each layer l and error ϵ, where L is the number of layers. In our study the effective learning rate for each box is $\alpha/L = 1.6/10 = 0.16$.

The exploration was controlled through the temperature parameter. An action was chosen with respect to Boltzman distribution. During all experiments the temperature was kept at the low value of 0.2. It guaranteed the near greedy policy and prevented an agent from being stuck.

5 MODEL-BASED LEARNING

Various on-line methods such as Sarsa and Q-Learning already proved their effectiveness in hands of numerous researchers. The convergence is guaranteed in on-line

cases if value approximators are sufficiently accurate (Ref ???). The opportunities for researchers exist in improving the speed of convergence. Learning on-line is expensive in the real world, compare to a simulation running in the computer memory.

We propose to learn the model of the environment while obtaining on-line experience and then use this model to facilitate learning. The model can be learned by different value approximation schemes, however the emphasis should be put on the achieving the maximum accuracy in order to rely on the model-based experiences.

Given a state and action the model of the environment predicts the next state that an agent will get to. The mapping can be learned from observing actual state transitions. In case of discreet 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 an action to a next state need to be approximated. The simplest approach using a hash table was implemented. The new algorithm is based on Sarsa described in the previous section with the addition of model update step and sampling step. The complete model-based Sarsa algorithm is as follows:

1. Initially: $w(t) := 0$, $\forall t \in Tiles, s_{sim} = s_0, a_{sim} =$*expl-policy*$(s_{sim})$
2. Start of Trial: $s = s_0$, $a :=$*expl-policy*(s)
3. Take action a; observe reward, r, and next state s'
4. $a' :=$ *expl-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 $s_{sim} = s_0, a_{sim} =$ *expl-policy*(s_{sim})
   go to the beginning of the loop
   $a'_{sim} :=$*expl-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

The observed next states can be different and still hash to the same entry. We store all observed next states with their frequencies. When the next state is looked up in the table, one of the observed values is drawn according to its frequency.

To gain confidence in the model-based approach and to compare the effect of the grid resolution for the hash table we designed the following experiment. We ran the algorithm for 20 trials and computed the averaged number of steps. The model is learned during the run but not expected to improve the convergence. Then another run of 20 trials is performed using the model from the previous run. This time the model should already be sufficiently learned to speed up the convergence. The procedure is repeated for 50 runs each time using an up-to-date model. In Figure 3 we compared two grid sizes for the hash table. The high resolution 100×100 grid is learned slower but shows more accurate performance. The low resolution 30×30

grid can be learned during the first run and provides the foundation for the next experiment.

Our goal is to find the way a model can be used during on-line learning. In this experiment we showed that a model can indeed provide a significant improvement over model-free learning. The same experiment was run with different number of model steps. The first data point with $K = 0$ corresponds to a model-free Sarsa algorithm. With $K > 0$ the average number of steps is going down.

The improvement during the first 20 trials is slight. However a model-based learning provides not only faster initial convergence but also more stable learning under high learning rates.

6 FUTURE WORK

One of the drawbacks of the described model-based method is significant memory consumption. For the relatively small task the hash table contained 1000 different states where each state would typically have several predicted next states. This approach is also inefficient since the next state probabilities have to be updated on each real step. The possible solution is the use of CMAC or neural networks to learn the mappings of states and actions to the next states. It however poses immediate challenges due to initial disturbance the inaccurate model brings into the learning of state values.

Acknowledgements

Thanks to Professor Richard Sutton, and to the members of the Adaptive Networks Laboratory. This work is supported by National Science Foundation under grant ECS-9511805 to Professor Barto and Professor Sutton.

References

Singh, S.P. & Sutton, R.S. (1996) Reinforcement Learning with Replacing Eligibility Traces. *Machine Learning* 3:123-158.

Watkins, C.J.C.H. (1989) *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, England.

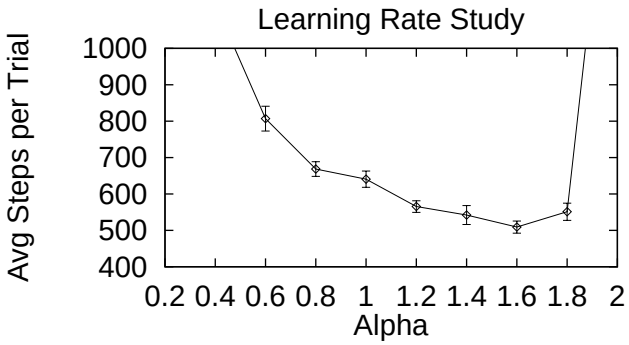

Figure 2: Convergence speed at various learning rates.

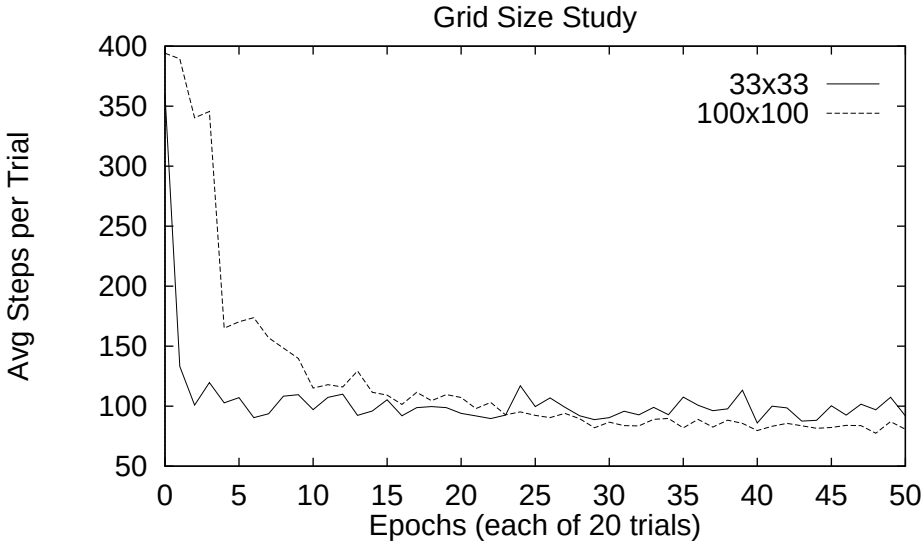

Figure 3: Convergence speeds up when using a model. Note the difference between high and low resolution grids

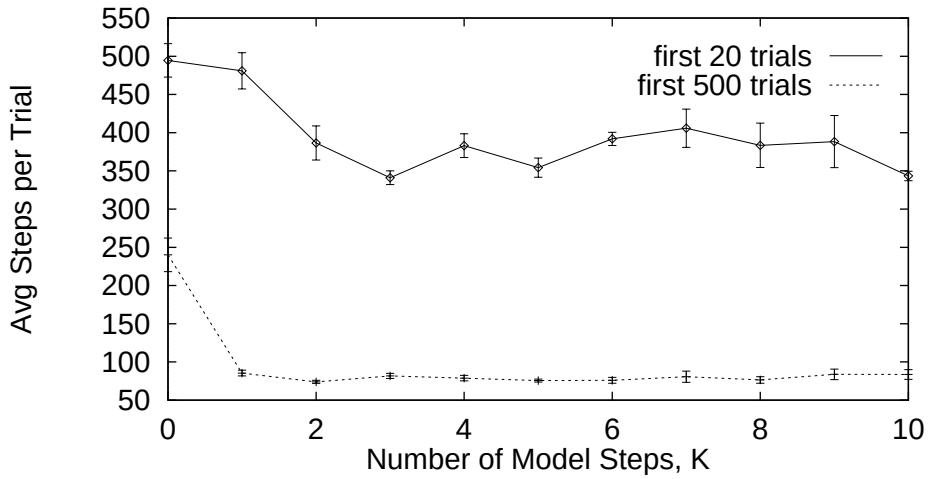

Figure 4: Convergence speeds up when using a model. K = 0 corresponds to model-free Sarsa algorithm.

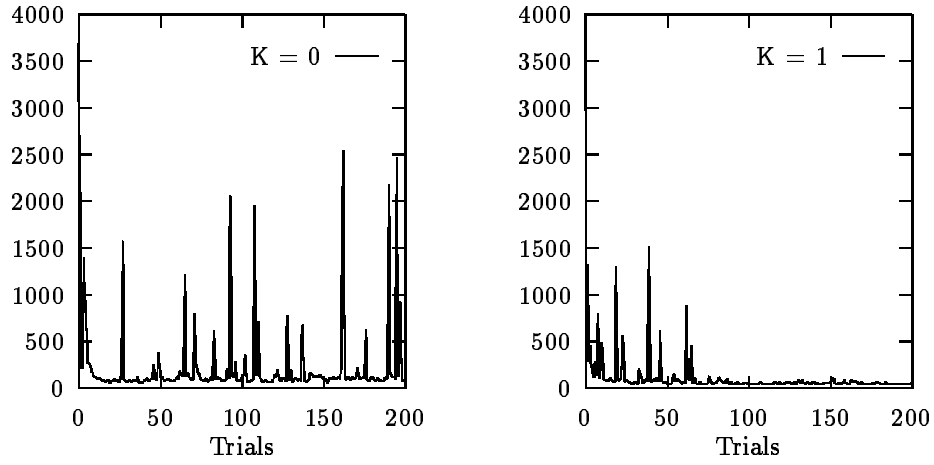

Figure 5: The learning curves for model-free (left) and model-based (right) algorithms. The averaged number of steps per trial versus is plotted for the first 200 trials.