Proceedings of the 1997 IEEE
International Conference on Robotics and Automation
Albuquerque, New Mexico - April 1997

A Comparison of Direct and Model-Based Reinforcement Learning

Christopher G. Atkeson and Juan Carlos Santamaría
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280
cga@cc.gatech.edu, http://www.cc.gatech.edu/fac/Chris.Atkeson
carlos@cc.gatech.edu, http://www.cc.gatech.edu/ai/students/jcs

Abstract

This paper compares direct reinforcement learning (no explicit model) and model-based reinforcement learning on a simple task: pendulum swing up. We find that in this task model-based approaches support reinforcement learning from smaller amounts of training data and efficient handling of changing goals.

1 Introduction

Many proposed reinforcement learning algorithms require large amounts of training data before achieving acceptable performance. This paper explores the training data requirements of two kinds of reinforcement learning algorithms, direct (model-free) and indirect (model-based), when continuous actions are available. Direct reinforcement learning algorithms learn a policy or value function without explicitly representing a model of the controlled system (Sutton et al., 1992). Model-based approaches learn an explicit model of the system simultaneously with a value function and policy (Sutton, 1990, 1991a,b; Barto et al., 1995; Kaelbling et al., 1996). We find that in the pendulum swing up task model-based approaches support reinforcement learning from smaller amounts of training data from the actual controlled system, and efficient handling of changing goals. Bersini and Gorrini (1996) conduct a similar comparison study using a linear double integrator movement task, with similar conclusions.

There are at least two measures of efficiency of interest in reinforcement learning: data efficiency and computing efficiency. Data efficiency measures the amount of data used from the actual controlled system during learning. Computing efficiency measures the amount of computation the learning algorithm requires. In a model-based reinforcement learning system these two efficiency measures can be quite different. If a good or perfect model is available then the data efficiency can be extremely high, as little or no data from the physical system is required to "learn" optimal behavior. The actual system is only used to execute the final result of mental simulation and planning. On the other hand, the learning algorithm may compute for long periods of time before finding good policies, so the computing efficiency can be quite low. In a direct (model-free) reinforcement learning paradigm such as Q learning (Watkins and Dayan, 1992) applied directly to an actual robot, the robot must be moved for each time step of learning algorithm execution, so the computing efficiency in terms of number of robot movements considered is reflected directly in the data efficiency.

We focus on the case where continuous actions are available, and the reward function is smooth. Many researchers have focused on reinforcement learning problems where a finite set of discrete actions are available on each time step (discrete decision processes (DDP) or discrete choice problems). In some cases a continuous set of actions is available (continuous decision processes (CDP) or choice problems) (Rust, 1996). If, in addition, the one step reward function is smooth, the derivatives of the one step reward with respect to the actions exist. A number of researchers have focused on this case (Gullapalli, 1990; Jordan and Jacobs, 1990; Williams, 1990; Millington, 1991; Baird and Klopf, 1993; Bradtke, 1993; Bradtke et al., 1994; Doya, 1996). A related paper explores efficient reinforcement learning in the presence of discrete actions (Boone, 1997a). We also assume that the state space is continuous.

0-7803-3612-7-4/97 $5.00 © 1997 IEEE

We will only treat the discrete time case. A treatment of the continuous time case has recently been presented (Doya, 1996).

2 Pendulum Swing Up

This paper was inspired by the success of Sutton (1996) in using a function approximator (CMAC) in direct reinforcement learning of a double pendulum (acrobot) swing up maneuver. Sutton chose a task that required the double pendulum (actuated only at the elbow) to reach a certain tip height in minimum time. Boone (1997a) explores this task using bang-bang control and shows that a number of simple heuristics outperform both direct and model-based reinforcement learning in "pumping" energy into the double pendulum to cause it to attain the desired height.

A different acrobot maneuver is to move from a stable hanging down configuration to an unstable inverted vertical configuration in which the robot has zero velocity and is actively balancing itself, a more difficult maneuver which has been extensively studied in control theory (Spong, 1995). We had a great deal of trouble implementing a direct reinforcement learning approach on the double pendulum swingup to vertical problem with continuous actions, so we are focusing on a simpler task, the single pendulum swingup, on which we can directly compare model-based and direct reinforcement learning. This task involves swinging up a single pendulum with a motor at the base from the hanging down position to the inverted position, and balancing in the up position (Atkeson, 1994; Doya, 1996). Note that this task is different from the cart-pole swingup, in which the base of the pendulum is moved horizontally and there is no motor applying torque directly to the pendulum (Standfuss and Eckmiller, 1994). We define the task by explicitly defining a time invariant one step reward (cost) criterion which is to be minimized:

$$r(\theta, \tau) = ((\theta - \theta_d)^2 + \tau^2)\Delta \tag{1}$$

where θ is the angle of the pendulum, θ_d is the desired angle for the inverted vertical state, τ is the motor torque, and Δ is the time step. This one step reward smoothly trades off a weighted distance to the goal with the size of the commands used to attain the goal, and can be thought of as a local quadratic model of a more complex global reward function. This formulation is also standard in optimal control theory. The task could also be formulated as a "delayed" reinforcement task in which reward is only given when one of a set of goal states is achieved. Moore and Atkeson (1995) discuss efficient planning and learning algorithms for this special case.

The equation of motion for the simulated single pendulum is $\ddot{\theta} = \tau - 9.81\cos(\theta)$. This equation is integrated with a step size of 0.05 seconds, but new control signals are only applied every 0.2 seconds, as in (Sutton, 1996). The starting state of each trial was $\theta = -\pi$ (the pendulum hanging straight down), and the angular velocity $\dot{\theta}$ was zero.

3 Direct Reinforcement Learning

Our implementation of direct reinforcement learning for the simulated single pendulum swingup task follows Sutton (1996) and Santamaría et al. (1996). We require the pendulum to reach and stay at the fully vertical configuration instead of just reaching a particular height. We use a CMAC as the representation for a learned Q function (with 12 3D tilings indexed by $(\sqrt{\theta}, \sqrt{\dot{\theta}}, \sqrt{\tau})$, 12 2D tilings indexed by $(\sqrt{\theta}, \sqrt{\dot{\theta}})$, 6 1D tilings indexed by $(\sqrt{\theta})$, and 6 1D tilings indexed by $(\sqrt{\dot{\theta}})$. The boundaries of the CMAC were $-2\pi < \theta < 2\pi$, $-10 < \dot{\theta} < 10$, and $-10 < \tau < 10$. In all cases the resolution of each dimension is 12, and the square root is used to increase resolution near the origin $(0, 0, 0)$. This resolution and fixed scaling of the inputs to the tilings was the best resolution and scaling we found (including using no scaling in a "uniform" CMAC). We perform no random non-optimal actions and follow a greedy policy, following Sutton (1996) and Santamaría et al. (1996). We use a continuous action version of the replace heuristic to update the eligibility traces. We learn online, without a model, and back up whatever states are encountered during 40 second long trials (200 steps) with the following equations (Sutton, 1996; Santamaría et al., 1996):

$$\begin{aligned} Q(\mathbf{x}_k, \mathbf{u}_k) = Q(\mathbf{x}_k, \mathbf{u}_k) + \\ \alpha[r(\mathbf{x}_k, \mathbf{u}_k) + \gamma Q(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}) - Q(\mathbf{x}_k, \mathbf{u}_k)] * \\ e(\mathbf{x}_k, \mathbf{u}_k) \end{aligned} \tag{2}$$

$\mathbf{x}$ is the state vector and $\mathbf{u}$ is the control vector. In a particular state $\mathbf{x}_1$ the optimal action is given by

$$\arg\min_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u}) \tag{3}$$

The value of $Q(\mathbf{x}, \mathbf{u})$ is the sum of the value for each activated tile in the 36 tilings. $\gamma = 0.99$ is the discount factor used to improve Q function learning, although γ was not used in computing the total cost per trial. $\alpha = 0.1$ is the aggregate learning rate, and updates are allocated equally to each tile with a learning rate of $\alpha/|\text{tilings}|$. $e(\mathbf{x}, \mathbf{u})$ is the corresponding eligibility value, and is made up of the sum of eligibilities e_{tile}

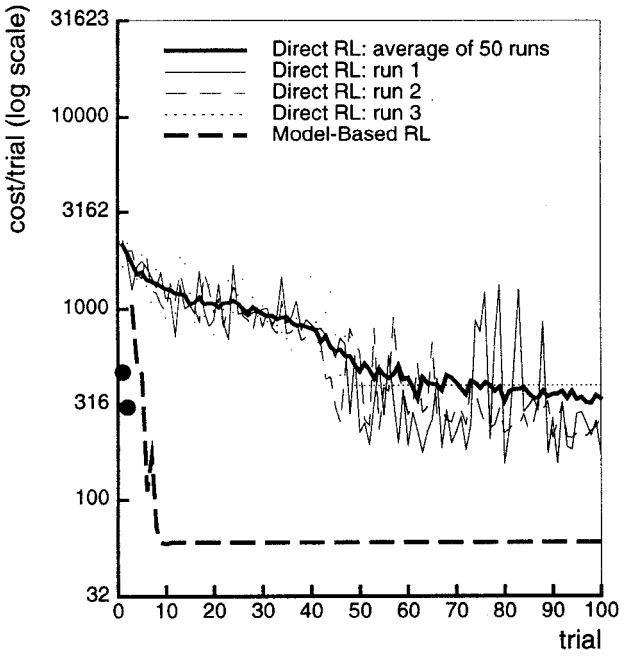

Figure 1: Typical learning curves for direct and model-based reinforcement learning. Note the log scale for the cost/trial axis. The thick solid line is the average of learning curves from 50 direct reinforcement learning runs. The thin lines are the first three individual learning curves of the 50 runs, to illustrate typical learning curves for a single experimental run. The thick dashed line is the cost/trial for a typical run of model-based reinforcement learning. The two large dots are the cost/trial for two initial random trials for model-based learning of 10 time steps total.

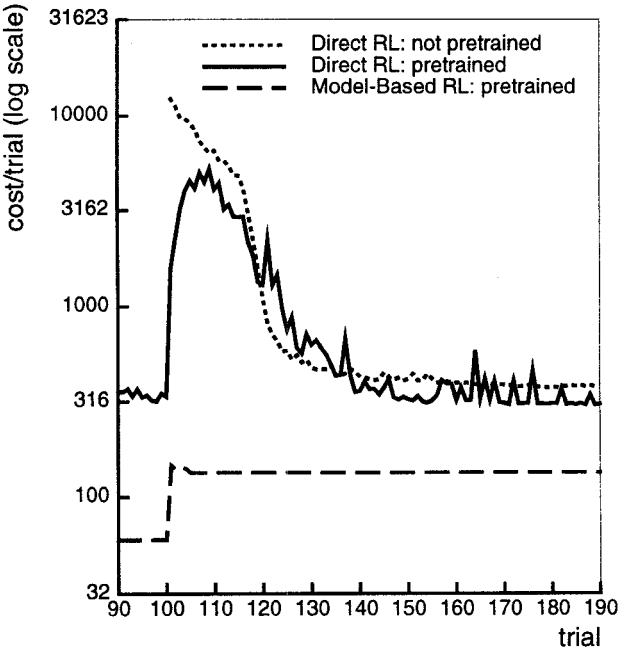

Figure 2: Transfer learning curves for direct and model-based reinforcement learning. Note the log scale for the cost/trial axis. After the 100th trial, the one step cost $r()$ is changed. This causes a large increase in cost/trial for direct reinforcement learning (solid line) that is later reduced through further learning. The dashed line shows the corresponding curve for model-based reinforcement learning. The dotted line indicates the learning curve for direct reinforcement learning if no pretraining had taken place.

for each activated tile. The eligibilities are updated according to:

$$e_{\text{tile}} = \begin{cases} 1 & \text{if the tile includes } (\mathbf{x}, \mathbf{u}) \\ \lambda\gamma e_{\text{tile}} & \text{otherwise} \end{cases} \quad (4)$$

The eligibility decay rate is $\lambda = 0.5$.

A major difference between our implementation and (Sutton, 1996) is that our trials are terminated after a fixed time. This modification was necessary due to the lack of a finite goal region. Our system never actually reaches the goal, which is an infinitely small point. Also, terminating trials when a "goal" is reached artificially simplifies the task if it is non-trivial to maintain the system at the goal, as it is in the inverted pendulum case where the pendulum must be actively balanced near the goal state. Exploration is forced by initializing the Q function to zero and having a one step cost $r()$ that is always positive.

Figure 1 shows an average learning curve and some learning curves from individual runs. After around 50 trials the performance reaches a cost plateau of between 400 and 300. This would have been after 33 minutes of continuous movement if it had been implemented on an actual robot.

In order to explore the effect of changing the goal during learning and to assess transfer from one learned task to another, we changed the one step reward function after trial 100 to (Figure 2):

$$r(\theta, \tau) = (10(\theta - \theta_d)^2 + \tau^2)\Delta \quad (5)$$

This reward function gives relatively more priority to reducing the distance to the goal than to reducing the size of the command, and the robot will apply larger torques to reduce the distance to the goal more quickly. The solid line shows the increase in average cost due to the change of goal followed by a decrease as a new policy is learned. The dotted line shows the average of 50 learning curves where no pretraining on the original reward function had occurred. There is some positive transfer between the initial learning and performance with the new reward function: the initial cost is lower

and the ultimate performance is slightly better with pretraining.

4 Model-Based Reinforcement Learning

This section describes a simple implementation of a model-based reinforcement learning algorithm. The algorithm uses an approximated (learned) model of the system dynamics but knows the reward function perfectly. The algorithm iteratively 1) updates the learned model, 2) plans a policy that optimizes the given criterion given the current learned model and updates the corresponding value function, and 3) executes the policy. The task is the same as with the direct reinforcement learning implementation. Each trial lasts for 100 seconds or until the state exceeded preset state limits ($-2.25\pi < \theta < 1.5\pi, -10 < \dot{\theta} < 10$). We use locally weighted learning to learn a discrete time forward model of the pendulum dynamics $(\theta_{k+1}, \dot{\theta}_{k+1}) = \mathbf{f}(\theta_k, \dot{\theta}_k, \tau_k)$ using all the data available (Atkeson et al., 1996a). Before storing new data we test to see if it can already be predicted by the model. If it can, we don't add that point to the database. We use state increment dynamic programming to find a policy and represent the value function using a bilinearly interpolated 20x20 grid (Larson, 1968). We run this planner at the beginning of each trial and after 10 new points are added to the data base. The learned model is initialized by executing random actions until the database contains 10 data points. At this point the first value function is computed.

This approach learns very quickly (Figure 1). After less than 10 trials near optimal performance is attained. Also, the cost achieved (60) is much smaller than that achieved by the direct reinforcement learning implementation (more than 300). Small numbers of data points are stored for use in locally weighted learning (usually less than 200). Other data points are already predicted by the database, and are not stored. In addition, when the reward function is changed to Equation 5 the system converges in a few trials to a new trajectory (and associated cost which reflects the higher cost of distance to the goal) without a large cost transient (Figure 2). Learning the new task is greatly speeded up compared to learning from scratch and compared to the direct reinforcement learning of the new task. Note that the change in reward function does not require storing many more points in the locally weighted regression model of the pendulum dynamics, as the pendulum dynamics did not change.

5 Discussion

This paper compares direct reinforcement learning (no explicit model) and model-based reinforcement learning on a simple task. In our tests we have seen that:

- Model-based reinforcement learning is more data efficient than direct reinforcement learning.
- Model-based reinforcement learning finds better trajectories, plans, and policies.
- Model-based reinforcement learning handles changing goals more efficiently.

Bersini and Gorrini (1996) explore the first two points and find similar results. It seems that it is more efficient to use the data collected from a system to build a model than to learn the value function directly, even when the model is only an approximation of the real system. The rest of this discussion attempts to put these results in a broader context.

5.1 How Sensitive Are These Results To Parameters Choices?

In our experience the performance of the model-based learner is relatively insensitive to parameter choices, and there are fewer parameter choices to make (there are no learning rates, for example). However, the direct reinforcement learner does seem more sensitive to parameter choices, including choosing learning rates, trial duration, variable ranges, and CMAC resolution. Bersini and Gorrini (1996) found a similar parameter sensitivity for direct reinforcement learning. We are presenting the best performance gotten so far in our simulations of direct reinforcement learning, and the performance presented here is comparable to the performance presented in Sutton (1996) and Santamaría et al. (1996). We hope to perform a more exhaustive survey on parameter settings and on a variety of tasks in the near future.

5.2 Why Don't The Direct And Model-Based Implementations Use The Same Representations?

In this comparison we have used the representations for each method that work best for that method. The model-based method used locally weighted regression (Atkeson et al., 1996a) to build a model and a grid with bilinear interpolation to represent the value function. The direct reinforcement learning implementation did not learn a model, and used a CMAC to represent the Q function, a form of value function. Note

that the value function computed by the model-based approach should be related to the ideal Q function by:

$$V(\mathbf{x}) = \min_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) \quad (6)$$

Locally weighted regression does a better job in learning a model using supervised learning than tables (Atkeson et al., 1996b) and worked better than a CMAC for model learning in our own tests. Value and Q function learning are very different, as, unlike supervised learning, values learned for a particular state change during learning. In this case representations that adjust stored values (including tables and CMACs) work better than locally weighted regression for learning a Q function. A table with bilinear interpolation worked best for the computational of the value function $V(\mathbf{x})$. In order to assess the best performance of each method we have implemented the method with the most appropriate representations, and used different representations for the different methods.

5.3 Simple Dynamics Favor Model-Based Approaches

One concern is that the simplicity of the dynamics of the pendulum favors model based approaches in this comparison. In our view this indicates a benefit of model-based approaches. There is a large class of problems for which there is underlying simplicity, even in real implementations (Atkeson and Schaal, 1997), and model-based approaches can take advantage of that simplicity. The real world successes of reinforcement learning that we are aware of have been in problems where good or perfect models were available, and in fact the results were often only demonstrated on simulations using those models (Tesauro, 1995; Zhang and Dietterich, 1995; Crites and Barto, 1996). It is likely that there are complex problems where possible policies are extremely limited (a binary choice, perhaps) where simply trying out policies is the most efficient reinforcement learning approach. However, if the possible policies are complex then we hypothesize that model-based reinforcement learning will be more data efficient.

The model-based learning implementation uses a non-parametric model, which requires the same a priori knowledge as the direct reinforcement learning implementation: the elements of the state and action vector. Model based approaches can take advantage of a priori knowledge of the dynamics of the system, such as its structure. Atkeson and Schaal (1997) describe an implementation of model-based reinforcement learning of a pendulum swing up on an actual robot that identifies a parametric pendulum model. Both model-based and direct reinforcement learning approaches can take advantage of a priori knowledge about the form of the optimal value function or policy for a system (for example, a quadratic value function for linear systems with quadratic criteria). It may be the case that a priori knowledge about system dynamics is more readily available than a priori knowledge about optimal value functions or policies.

5.4 Different Exploration Strategies

Exploration is forced with the direct reinforcement learner, even when it is greedily following the current policy, as in this implementation. By initializing the Q function to zero and having a one step cost $r()$ that is always positive, untried actions have a lower cost than actions that have been tried. It is not clear how to circumvent this effect and still have the Q learner produce good policies quickly. The model based reinforcement learning implementation also greedily followed the current policy, and did not perform any significant exploration with the simulated robot except for initial random trials.

5.5 The Cost of Representing A Value Function

Both approaches represent a value function with similar storage costs. In fact, the learned Q function should be closely related to the computed value function, although it has added dimensionality due to the actions being added to the inputs (Equation 6).

5.6 The Cost of Planning

Updating all the parameters in the representation of the value function used in the state increment dynamic programming (a "sweep") takes less than a second ($0.79s$) on a SPARCstation LX. In an actual implementation on a robot, this computation could be run on a dedicated processor and redone each second with the updated model, as is currently done with inverse kinematics and dynamics (Atkeson and Schaal, 1997). Thus, there is no "extra" cost of planning or re-planning, as it is done in parallel with robot execution. As computing power is getting cheaper and more compact, it will get even easier to implement real time planning.

5.7 The Cost of Exploration

For model-based reinforcement learning methods there are two kinds of exploration: exploring to improve the model and exploring to improve the policy. A model-based reinforcement learning method can stop

or reduce exploration with the physical robot when it has sufficient confidence in the accuracy of the learned model. Exploration to improve the policy can take place in mental simulation. In order to plan the model-based learner mentally considered all actions within the range of allowable states. In direct reinforcement learning, exploration to improve the policy must take place on the actual system, leading to a corresponding performance decrement. In both cases, if the policy exploration is not adequate, some regions of the policy may be incorrect. If a function approximator is used to learn the policy, value, or Q function inadequate exploration may lead to interference during learning, so correct portions of the policy are actually degraded during learning.

The mental exploration in model-based learning is clearly more data efficient than the actual exploration performed in direct reinforcement learning. In many cases model queries have low computational requirements and the learner can have mental experiences faster than actual experiences, allowing more mental than actual exploration to be done. Using a model to compute a value function or policy enables systematic exploration of the value function, since the model can be queried for any point in the state space and any pattern of queries can be used. Exploration with the actual controlled system is much more limited, as exploration can only occur along physically possible trajectories, and the robot cannot be directed to a particular location in state space without already having achieved successful control.

5.8 Scaling Up to Harder Problems

Comparisons of direct reinforcement learning and model-based reinforcement learning on more complex tasks would be desirable, but this must await more powerful and sophisticated versions of both types of algorithms. Both direct RL and model-based RL face serious challenges in handling high dimensional problems. It is not yet clear which is worse: using function approximation to learn high dimensional models, using function approximation to learn high dimensional value functions, or using function approximation to learn high dimensional Q functions. It is also not clear which approach will better adapt to the limited exploration possible in high dimensional spaces.

5.9 Inaccurate Models and Inaccurate Q Functions

A serious challenge for model-based approaches is overcoming inaccurate modeling. The model-based planning process finds good policies for the learned model, not the actual system. It may be the case that a plan based on a learned but still inaccurate model does not perform acceptably. This did not happen in our tests, but has happened in actual implementations (Atkeson and Schaal, 1997). Depending on the characteristics of the controlled system, small modeling errors can lead to large policy errors.

A serious challenge for direct reinforcement learning is overcoming inaccurate value or Q functions. It may be the case that the policy represented by a learned Q function does not perform acceptably. In our tests the direct reinforcement learner did not find as good policies as the model based learner. Additional experience did not improve the policies past a certain point. Depending on the characteristics of the controlled system, small Q function errors can lead to large policy or performance errors.

There are at least three approaches to address these challenges.

1) Adaptive representations can be developed for models and value functions. These representations use information from experience to adapt the representational structure. The representations used in this study did not change their structure during learning. Locally weighted learning could adapt architectural parameters such as a distance metric (Atkeson et al., 1996a). Tables and CMACs can adapt their resolution in different parts of the space during learning. However, each of these structural adaptations typically requires large amount of training data, which means the improvements due to structural refinements may come relatively late in the learning process.

2) Developing explicit exploration algorithms is an important challenge for model-based and direct reinforcement learning. Model-based approaches need to explore the controlled system to correct important modeling errors. Direct reinforcement learning uses exploration to correct errors in the Q function.

3) In the case of model-based learning the planner can compensate for modeling error by building robust plans and by taking into account previous task outcomes in adjusting the plan independently of model updates (Atkeson and Schaal, 1997).

There is much more to do both in the practical development of algorithms and in the theoretical analysis of them. At this point it is only a hope rather than a guarantee that a policy based on the imperfect model (Q function) will lead to experiences that correct the model's (Q function's) flaws.

5.10 Is The Reward Function Known?

In the case of model-based reinforcement learning we assume that a (possibly time-varying) one step reward function $r()$ is known or learned, and is used to program the desired behavior of the controlled system. In

the case where the reward function is provided only be a sensor reading, a model-based approach would have to model the reward function as well as the system dynamics. Direct reinforcement learning does not require explicit knowledge or learning of the reward function. A benefit of knowing or learning the reward function explicitly is the ability to recompute policies when the one step reward function $r()$ changes by allowing reuse of the model (Figure 2). Direct reinforcement learning algorithms must exercise the actual physical system in order to learn a new policy.

5.11 Limitations Of The Approaches

Reinforcement learning algorithms are not limited to the applications and implementation details described in this paper. Direct reinforcement learning and model-based approaches can be applied to stochastic tasks. Tesauro (1995) gives an example of direct reinforcement learning applied to a stochastic system, and Moore and Atkeson (1993) give an example of model-based learning in a stochastic context. This paper discusses a quadratic reward function, but the same approaches can easily be applied to general smooth reward functions. Sutton (1996) discusses direct approaches and Boone (1997a,b) discusses model based approaches appropriate for other types of reward functions. This paper used non-parametric models (Atkeson et al., 1996a) in the model-based implementation, which require the same information needed by direct reinforcement learning. The components of the state and action vector need to be determined by human programmers or other approaches, although irrelevant components can be identified and ignored. Often, model-based control has been applied where the model is not learned and the planner does not re-plan during operation of the system. In model-based reinforcement learning the model is updated and the planner re-plans constantly. This avoids problems in handling changes in dynamics over time. However, learning about changes in system properties often involves a delicate interplay of a priori beliefs about the system and about the reliability of the training data.

5.12 The Role of Function Approximation in Reinforcement Learning

In assessing the role of function approximation in reinforcement learning it is important to decide what should be approximated. Direct reinforcement learning approximates the Q function. Model-based function approximation uses separate representations for a model and for a value function. The model representation is learned from data, and the value function representation is computed. It may be the case that learning models is easier than learning Q functions, as models can be learned in a supervised manner and may be smoother or less complex than Q functions. Once a model or partially correct model is in hand, computing a value function using specialized techniques and representations such as state increment dynamic programming (Larson, 1968) or adaptive grid techniques (Atkeson, 1994) may also be easier than learning a Q function.

It is important to distinguish Q learning from direct reinforcement learning. It may be the case that Q learning is an efficient planning algorithm, using a model and mental simulation to plan good policies in model based reinforcement learning (Sutton, 1990, 1991a,b). Q learning could certainly have been used as the planner in our model-based learner, although it is not as computationally efficient as state increment dynamic programming.

5.13 Some Historical Context

The debate on the proper roles for models in intelligence has a long history. we would like to end with a 50 year old quote:

> If the organism carries a "small-scale model" of external reality and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of past events in dealing with the present and future, and in every way react in a much fuller, safer, and more competent manner to the emergencies which face it. (Craik, 1943)

6 Conclusions

This paper compares direct reinforcement learning (no explicit model) and model-based reinforcement learning on a simple task. In our tests we have seen that model-based reinforcement learning is more data efficient than direct reinforcement learning, model-based reinforcement learning finds better trajectories, plans, and policies, and model-based reinforcement learning handles changing goals more efficiently.

Acknowledgments

Support for C. Atkeson was provided under Air Force Office of Scientific Research grant F49-6209410362, by the ATR Human Information Processing Research Laboratories, and by a National Science Foundation Presidential Young Investigator Award. Support for J.C. Santamaria was provided under Army Research

Laboratories grant number DAKF11-91-D-0004-0051 to Ashwin Ram.

References

Atkeson, C. G. (1994). Using local trajectory optimizers to speed up global optimization in dynamic programming. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 663–670. Morgan Kaufmann, San Mateo, CA.

Atkeson, C. G., Moore, A. W., and Schaal, S. (1996a). Locally weighted learning. *Artificial Intelligence Review.* in press.

Atkeson, C. G., Moore, A. W., and Schaal, S. (1996b). Locally weighted learning for control. *Artificial Intelligence Review.* in press.

Atkeson, C. G. and Schaal, S. (1997). Learning tasks from a single demonstration. ICRA 97.

Baird, L. C. and Klopf, A. H. (1993). Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base Ohio. http://kirk.usafa.af.mil/ baird/papers/index.html.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81—138.

Bersini, H. and Gorrini, V. (1996). Three connectionist implementations of dynamic programming for optimal control: A preliminary comparative analysis. To appear in the proceedings of Nicrosp'96.

Boone, G. (1997a). Efficient reinforcement learning: Model-based acrobot control. ICRA 97.

Boone, G. (1997b). Minimum–time control of the acrobot. ICRA 97.

Bradtke, S. J. (1993). Reinforcement learning applied to linear quadratic regulation. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 295–302. Morgan Kaufmann, San Mateo, CA.

Bradtke, S. J., Ydstie, B. E., and Barto, A. G. (1994). Adaptive linear quadratic control using policy iteration. Technical Report CMPSCI Technical Report 94-49, Department of Computer Science, University of Massachusetts, Amherst. ftp://ftp.cs.umass.edu/pub/techrept/techreport/1994, also published in the Proceedings of the 8th Yale Workshop on Adaptive and Learning Systems. Yale University, 1994, pp. 85-96.

Craik, K. J. W. (1943). *The Nature of Explanation.* Cambridge University Press, Cambridge, UK.

Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8.* MIT Press, Cambridge, MA.

Doya, K. (1996). Temporal difference learning in continuous time and space. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8.* MIT Press, Cambridge, MA. http://www.hip.atr.co.jp/ doya/papers/nips8.ps.Z.

Gullapalli, V. (1990). A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3:671–692.

Jordan, M. I. and Jacobs, R. A. (1990). Learning to control an unstable system with forward modeling. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 324–331. Morgan Kaufmann, San Mateo, CA.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Larson, R. E. (1968). *State Increment Dynamic Programming.* Elsevier, NY.

Millington, P. J. (1991). *Associative Reinforcement Learning For Optimal Control.* MS thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics. also published as Technical Report CSDL-T-1070, The Charles Start Draper Laboratory, Cambridge, MA.

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

Moore, A. W. and Atkeson, C. G. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. *Machine Learning*, 21:199–233.

Rust, J. (1996). Numerical dynamic programming in economics. In Amman, H., Kendrick, D., and Rust, J., editors, *Handbook of Computational Economics.* North-Holland. http://thor.econ.wisc.edu.

Santamaría, J. C., Sutton, R., and Ram, A. (1996). Experiments with reinforcement learning in problems with continuous states and action spaces. COINS Technical Report 96-88, Dept. of Computer Science, University of Massachusetts.

Spong, M. W. (1995). The swing up control problem for the acrobot. *IEEE Control Systems Magazine*, 15(1):49–55.

Standfuss, A. and Eckmiller, R. (1994). To swing up an inverted pendulum using stochastic real-valued reinforcement learning. In *4th International Conference on Artificial Neural Networks (ICANN '94)*, pages 655–658, Sorrento, Italy. Springer Verlag. http://marvin.nero.uni-bonn.de/veroeffentl-en.html.

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8.* http://envy.cs.umass.edu/People/sutton/publications.html.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Seventh International Machine Learning Workshop*, pages 216–224. Morgan Kaufmann, San Mateo, CA. http://envy.cs.umass.edu/People/sutton/publications.html.

Sutton, R. S. (1991a). Dyna, an integrated architecture for learning, planning and reacting. http://envy.cs.umass.edu/People/sutton/publications.html, Working Notes of the 1991 AAAI Spring Symposium on Integrated Intelligent Architectures pp. 151–155 and SIGART Bulletin 2, pp. 160-163.

Sutton, R. S. (1991b). Planning by incremental dynamic programming. In *Eighth International Machine Learning Workshop*, pages 353–357. Morgan Kaufmann, San Mateo, CA. http://envy.cs.umass.edu/People/sutton/publications.html.

Sutton, R. S., Barto, A. G., and Williams, R. J. (1992). Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12:19—22.

Tesauro, G. (1995). Temporal difference learning and TD-gammon. *Communications of the ACM*, pages 58—67.

Watkins, C. J. C. H. and Dayan, P. (1992). Q–learning. *Machine Learning*, 8:279—292.

Williams, R. J. (1990). Simple statistical gradient–following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Zhang, W. and Dietterich, T. (1995). A reinforcement learning approach to job–shop scheduling. In *Proceedings of the Fourteenth IJCAI.* Morgan Kaufmann, San Mateo, CA.