Representations for Learning Control Policies

Jeffrey Forbes FORBES@CS.DUKE.EDU
Department of Computer Science, Duke University, Durham, NC 27708-0129 USA

David Andre DANDRE@CS.BERKELEY.EDU
Computer Science Division, University of California, Berkeley, Berkeley, CA 94720-1776

Abstract

Representing the expected reward or cost for taking an action in a stochastic control problem, such as automated driving, is not trivial when the state and action spaces are continuous. Simple techniques can suffer from forgetting, where the lessons learned when the agent was doing poorly (e.g. how to recover when the car is headed off the road) can be forgotten when the agent has learned a policy that works in the majority of cases. This paper presents and demonstrates a method that effectively learns and maintains a Q function approximation utilizing stored instances of past observations. This instance-based reinforcement learning algorithm has the necessary extensions and optimizations required to learn in complex control domains. To make further use of the stored examples, our method learns a model of the environment and uses that model to improve the estimate of the value of taking actions in states. We explore several techniques for choosing how to use the model to efficiently improve the value function, and present an original algorithm based on generalized prioritized sweeping that outperforms the others on two example driving tasks.

In our experience, the task of learning to control an autonomous vehicle is best formulated as a stochastic optimal control problem. Reinforcement learning (RL) algorithms can learn optimal behavior for such problems from trial and error interactions with the environment. However, reinforcement learning algorithms often are unable to effectively learn policies for domains with certain properties: continuous state and action spaces, a need for real-time online operation, and continuous long-term operation.

Driving is a particularly challenging problem, since the task itself changes over time. A lane following agent may become proficient at negotiating curved roads and then go on a long straight stretch where it becomes even more proficient on straight roads. It should not, however, lose proficiency at the curved roads. The overall goal of staying in the center of the lane remains the same, but the kind of states with which the agent is faced changes when moving from curved roads to straight and back again. Many learning algorithms are vulnerable to *catastrophic interference* where, after experiencing numerous new examples in a different part of the state space, accuracy on older examples can decrease. This behavior is referred to as *forgetting*. As in real life, forgetting is obviously inadvisable for any learning control algorithm.

Instance-based learners are an example of non-parametric learners and thus avoid the problem of forgetting. Applying them directly to reinforcement learning, however, is not entirely trivial, as the reinforcement learning problem is not a supervised learning problem, but a delayed reinforcement problem. Furthermore, instance based techniques use a lot of memory, and are susceptible to the magnitudes of the inputs. This paper presents techniques that avoid the above difficulties by using a value-updating algorithm, instance averaging, and automatic dimension scaling.

Finally, reinforcement learning can require many runs in the environment to learn a successful policy. We mitigate this by using memory-based reinforcement learning, which learns a model of the environment and uses it to improve the value function without taking steps in the actual environment. Many methods have been suggested for how best to use a model to update the value function. We utilize a novel method for our representation based on generalized prioritized sweeping (Andre et al., 1997). We demonstrate our methods on two simulated driving tasks.

The structure of this paper is as follows. Section 1 describes the instance-based representation for value function approximation and how it can be used to learn control policies effectively from experience. This section also describes the extensions that were necessary for the representation to be practical for vehicle control. Section 2 shows how one can use a structured domain model to learn more efficiently and handle fundamental autonomous vehicle tasks. Section 3 presents some empirical results on learning to control a simulated vehicle to steer itself in the center of the lane. The paper ends with conclusions and acknowledgments.

1. Instance-based value reinforcement learning

In Q-learning, the agent learns a Q-function which gives the *value* of taking an action in a state (Watkins, 1989). The value of a state is the expected long term reward from that point. Parametric function approximators such as neural networks and linear combinations of basis functions are susceptible to forgetting in the presence of interference. Instance-based function approximators are more robust to interference because all experiences are stored and the effect of new experiences is local. Instance-based techniques benefit from being a local approximation method that gives accurate estimates of portions of the state space even when the majority of the overall space is unvisited. We developed a technique called instance-based reinforcement learning (IBRL) that is effective in learning control policies (Forbes & Andre, 2000a). Instead of just updating a global parametric approximation, IBRL records all experiences (state, action, and observed value) and predicts and generalizes for new unseen state-action pairs. Beyond just using an instance based function approximator, a number of extensions are needed to the representation to make it viable for control problems: bounding the number of stored examples using instance-averaging and automatic dimension weighting.

1.1 Value function approximation

Reinforcement learning can be applied to control by learning the value of taking actions in particular states, $Q(\boldsymbol{s}, \boldsymbol{a}) \rightarrow \Re$. Learning this function from trial and error experience is called Q learning. We assume no prior knowledge of the Q function, the state transition model ($\forall \boldsymbol{s}, \boldsymbol{s}' \in S, \forall a \in A, p(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$) or the reward model ($\forall \boldsymbol{s}, R(\boldsymbol{s})$).

A learning step in the IBRL algorithm is described in Figure 1.1.

```
function DoIBRLStep(state $\boldsymbol{s}_t$, reward $r_{t-1}$, time $t$)
returns action
  (1) Compute policy
      $\boldsymbol{a}_t \leftarrow \pi(\boldsymbol{s}_t) = \begin{cases} \text{argsup}_{\boldsymbol{a}} Q(\boldsymbol{s}_t, \boldsymbol{a}) & \text{w.p.} 1 - \epsilon(\text{t}) \\ \text{random action} & \text{w. p.} \epsilon(t) \end{cases}$
  (2) $\mathit{ObservedValue} \leftarrow r_{t-1} + \gamma Q(\boldsymbol{s}_t, \boldsymbol{a}_t)$
  (3) Compute prediction error
      $\mathcal{E} \leftarrow Q(\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}) - \mathit{ObservedValue}$
  (5) $\forall$ neighbors $i$ involved in prediction of
      $Q(\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$
      (6) Update examples
          $Q_i \leftarrow Q_i + \alpha(t) \cdot \mathcal{E} \cdot \nabla_{Q_i} Q(\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$
  (7) Store new example $Q_t \leftarrow \mathit{ObservedValue}$
  (8) Store estimate $Q(\boldsymbol{s}_t, \boldsymbol{a}_t)$ for next step
```

Figure 1. Pseudocode for the instance based learning algorithm. $\alpha(t)$ and $\epsilon(t)$ are the learning and exploration rates respectively at time t.

1.2 Maintaining Q Value Estimate

For every action at time step t, we can add a new example into the database:

$$Q_t = r_t + \gamma Q(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}). \tag{1}$$

The algorithm then updates each Q-value Q_i in our database according to a temporal-difference update:

$$Q_i \leftarrow Q_i + \alpha[r_t + \gamma Q(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) - Q(\boldsymbol{s}_t, \boldsymbol{a}_t)] \nabla_{Q_i} Q(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{2}$$

Given a distance metric and a weighting function, the predictions in instance-based learning can be computed in a number of ways. A reasonable prediction method is kernel regression where the fit is simply a weighted average of the outputs $y_i, i \in [1, n]$ of the neighbors: $\frac{\sum w_i y_i}{\sum w_i}$. The contribution of each instance can then be computed as:

$$\nabla_{Q_i} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) = \frac{w_i}{\sum_j w_j}.$$

Instead of updating all elements, we can update only the nearest neighbors. We want to credit those cases in memory which may have had a significant effect on the $Q(s_t, u_t)$ that were within the kernel width (τ_k). Those cases make up the nearest neighbors $NN(s_t, u_t)$ of the query point. By updating only the neighbors, we can bound the overall update time per step.

Given an accurate estimate of the Q function, one can calculate the optimal action a in any state s according to the policy π defined as $\pi(s) = \text{argsup}_a Q(s, a)$.

Note that we have an $\arg\sup_a$ to compute instead of an argmax_a, since we are dealing with a continuous action vector. Finding the maximum value for an arbitrary function can be a complex exercise in itself. Many algorithms discretize the action space, but for many control tasks, too coarse a discretization of actions can cause oscillations and unstable policies (Kumar & Varaiya, 1986). One simplifying assumption is that there is a unimodal distribution for the Q-values with respect to a particular state and scalar actions. Avoiding suboptimal actions in a continuous action space may be impossible, but by conducting a beam search where there are multiple parallel searches each returning a maximum, the optimization does well. The estimate of the global maximum is then the best of these searches. Each beam search uses a gradient ascent strategy to find a locally optimal action for our value function. As the number of beam searches approaches infinity, the maximum returned approaches the global maximum. In practice, of course, a large or unbounded number of searches cannot be performed on every policy lookup.

The basic idea of using instance-based methods for value function approximation is critical to being able to effectively learn controllers, but extensions are necessary to maintain a value function estimate in continual operation in domains. Simple instance-based learning alone will not be practical for the control of autonomous vehicles because the number of stored examples would become unmanageable as the agent continued to learn. Also, nearest neighbor techniques are sensitive to the relative values and variances of the different state vector dimensions. For proper generalization, extensions have to be be made to adapt to the relative importance and range of the different features.

1.3 Managing the number of stored examples

For the applications that have been discussed, the system may need to continue to operate and learn for an arbitrary length of time. While there do exist some well-practiced techniques for pruning "useless" examples in training (Bradshaw, 1985; Kibler & Aha, 1988; Kohonen, 1990), learning in this setting poses a few more difficulties. One method of editing the set of examples is to determine if an example contributes to a decision boundary. If one of its neighbors has a different value then it remains in the set, otherwise it is removed. This algorithm can reduce the number of examples with no effect on the algorithm's accuracy. However, for a number of reasons, this method does not work for value function approximation. First, this task is not one of classification but of regression, so there are no decision boundaries. Instead, there are gradations of value in regions, and it is unknown how finely-grained those gradations may be. Also this editing method requires knowledge of all the training data ahead of time. Once all examples have been seen, it is possible to rule out that a particular region, no matter how small, could have a completely different value than the points around it. An online algorithm has to determine the complexity of value space as new states and rewards are experienced.

Finally, the stored examples are only estimates, so it is not possible to know whether an example is an important outlier or just wrong. If the example bound is set too low, the estimates may not be good enough to warrant trying to determine which ones are most essential. Premature pruning is particularly harmful for reinforcement learning where it may take a number of trials before a goal is even reached and any useful feedback is gained. While it may not be possible to find a pruning algorithm that reduces space and time requirements without compromising accuracy, we can pick examples to prune that appear to have the least effect on the estimate.

A first step in managing the size of the examples set is to limits its growth; redundant examples should not be added. There is a density parameter, τ_d, that limits the density of examples stored. For every new example, $\boldsymbol{x}_q$, and given a distance metric, $D(\cdot,\cdot)$, if $\forall \boldsymbol{x} D(\boldsymbol{x}_q, \boldsymbol{x}) > \tau_d$, then $\boldsymbol{x}_q$ is added to the set of examples. Since the neighboring examples will be updated, the effect of the new experience will still be incorporated into the overall estimate. Another optimization is to add only those examples that cannot be predicted within some parameter ϵ. Both τ_d and ϵ can be adjusted down to increase accuracy or up to decrease memory requirements. While these optimizations can have both direct and indirect effects on the learning process, they are useful in practice in limiting the growth of the example set. Being selective in adding examples may slow growth; when the number of examples is greater than the maximum number allowed, some example must be removed. A first approach would be to remove the oldest example. Another possibility would be to remove the least recently "used" example, the example that has not been a neighbor in a query for the longest time. There are problems encountered with these methods that are similar to those using a parametric function approximator, namely in the forgetting experiences from the past. This forgetting can be particularly problematic if some important experiences only occur rarely, though, in some cases, forgetting early experiences can be beneficial, since early estimates often provide no useful information and can be misleading. Forgetting the most temporally distant examples does not differentiate between incorrect examples and relevant old examples.

Instead of removing old examples, the examples that should be removed are those that contribute the least to the overall approximation. The example that is classified the best by its neighbors, and where the neighbors can still be classified well or possibly better without the example, should be chosen for averaging. The score for each exemplar, Q_i can be computed as follows:

$$\text{score}_i = |Q_i - Q^{-i}(s_i, a_i)| + \frac{1}{K}\sum_{j \neq i} |Q(s_j, a_j) - Q^{-i}(s_j, a_j)|$$

where $Q^{-i}(\boldsymbol{s}_i, \boldsymbol{a}_i)$ is the the evaluation of the Q function at $(\boldsymbol{s}_i, \boldsymbol{a}_i)$ while not using Q_i in the evaluation. The scores can be updated incrementally in the time it takes for a Q evaluation. Instead of removing the example with lowest score, a better method is instance-averaging – where two neighbors are reduced into one exemplar, which is located at the midpoint of the two inputs and its value is the mean of the two values. Averaging reduces the bias in comparison to just removing an example. A similar approach was proposed by Salzberg where instead of averaging instances, he generalizes instances into nested hyperrectangles (Salzberg, 1991). While partitioning the space into hyperrectangles can work well for discrete classification tasks, it has flaws for regression. Most importantly, all of the points within a region almost never have the exact same value.

1.4 Optimizations

Each estimate is computed from the values of its neighbors. The estimate is sensitive to distances between examples in the input space and to how much each of the examples is weighted. The kernel size and the number of neighbors determine the size of the relevant neighborhood. The different dimensions in a input vector tend to have different levels of importance. It is often desirable to scale and skew the input space so that more attention is paid to certain dimensions in the distance metric. This effect is achieved by adding a covariance matrix to the Gaussian weighting function. This effect is achieved by adding a covariance matrix to the Gaussian weighting function. Thus, for two points x, y in $\langle$state,action$\rangle$ space, we use

$$w(x, y) = e^{-[x-y]^T \Sigma [x-y]}$$

Σ^{-1} is a matrix that is essentially the inverse covariance matrix of the Gaussian. Note that a single Σ^{-1} is used for all the data. Thus, when Q is updated, Σ^{-1} must also be updated. This update is shown below:

$$\Sigma^{-1}_{i,j} += \alpha[r_{t+1} + \gamma Q(s', a') - Q(s, a)] \nabla_{\Sigma^{-1}_{i}{}_j} Q(s, a)$$

Beyond dimension weighting, some dimensions may be irrelevant altogether.

As this work is motivated by working on autonomous vehicles, the instance-based reinforcement learning algorithm must be as efficient as possible. Furthermore, it is important that the learning steps can be done in close to real time. A previous paper discusses the reinforcement learning system in terms of its abilities to be a real-time autonomous system (Forbes & Andre, 2000b). As with most learning tasks, the goal is to reach a certain level of performance in as few training steps as possible. I have already mentioned in Section 1.3 that bounding the number of examples limits the time of the value computation and updating steps will take. However, efficiency of the system allows for maximization of the number of examples that can be stored and the speed with which the system learns. Determining the k nearest neighbors of an example can be achieved more efficiently using kd-trees (Moore, 1991).

1.5 Related Work

There has been significant research in using instance-based learning for control. Most of the work focuses on learning direct input-output mappings for control tasks. Moore, Atkeson, and Schaal and show that techniques that store examples are ideally suited for control (Moore et al., 1997).

Instance-based learning was used for function approximation in reinforcement learning by Santamaria, Sutton, and Ram (Santamaria et al., 1998). The contribution this paper makes is in presenting the general update rule for instance-based approximators presented in Equation 2. Furthermore, this paper addresses many of the practical problems that come up when applying reinforcement learning to more complex problems such as vehicle control. In particular, our work addresses limited space and time resources Another similar line of research is that of Ormoneit and Sen (Ormoneit & Sen, 1999). They present an algorithm called kernel-based reinforcement learning which has both practical and theoretical promise. An interesting result is that kernel regression when used as function approximation technique performs like an averager which does converge to a unique fixed point (Gordon, 1995). The algorithm relies on there being a finite number of actions in order for it to be feasible, since the value is estimated by averaging the values of estimates of previous times when the action was executed in similar states. In control domains, there is an infinite number of actions that necessitate an incremental approach. A similar method to the one in this paper is that of Smart and Kaelbling (Smart & Kaelbling, 2000). Our approaches were developed independently, and both approaches were motivated by the goal of making reinforcement learning practical for dynamic control tasks.

2. Model-based reinforcement learning

Despite significant progress in theoretical and practical results for reinforcement learning, RL algorithms have not been applied on-line to solve many complex problems. One reason is that the algorithms are weak learning algorithms in an AI sense. They use a *tabula rasa* approach with little or no prior domain knowledge and will generally scale poorly to complex tasks (Mataric, 1991). However, there are model-based approaches which can incorporate domain knowledge into directing the search of a RL agent. The algorithms use experience to build a model, use the experience to adjust the estimated utility of states, and then use that model and the utility function to update the policy.

As is well known in the reinforcement learning community, a model of the environment can be used to perform extra simulated steps in the environment, allowing for additional planning steps (Sutton, 1990). In the extreme, the MDP in question could be solved as well as possible given the current model at each step. For the problems of interest here, this approach is intractable, and certainly undesirable when the model is not very accurate. Thus, most applications benefit from a middle ground of doing some extra planning steps after each actual step in the environment.

There are several different methods of choosing the state and action pairs for which to perform simulations. One possibility is to take actions from randomly selected states. This was the approach pursued in the DYNA (Sutton, 1990) framework. A second possibility is to search forward from the current state-action pair $\boldsymbol{x}$, doing simulations of the N next steps in the world. Values can then be backed up along the trajectory, with those Q-values furthest from $\boldsymbol{x}$ being backed up first. This form of lookahead search is potentially useful as it focuses attention on those states of the world that are likely to be encountered in the agent's near future. A third possibility is to search backwards from the current state finding the likely predecessors and updating those states. Since one of the challenges in reinforcement learning is that of delayed reward, updating previous steps can more quickly update those states that may have been responsible for a later penalty or reward.

The technique of Prioritized Sweeping (Moore & Atkeson, 1993) chooses the states to update based on a priority metric that approximates the expected size of the update for each state. Those state-action pairs expected to have the highest changes in value, i.e. the priorities, should be updated. The motivation for this idea is straightforward. Those Q-values that are the most incorrect are exactly those that contribute the most to the policy loss. To implement prioritized sweeping, there must be an efficient estimate of the amount that the value of a state-action pair will change given an other update. When will updating a state-action pair make a difference? When the value of the likely outcome states has changed, and when the transition model has changed, changing the likely output states.

In this work, the principles of generalized prioritized sweeping (Andre et al., 1997) are applied to a continuous state and action space using an instance-based value-function and a dynamic Bayesian network (DBN) representation of the model. A potential disadvantage of using a model is that it must be learned, which can be difficult in a complex environment. In a discrete state space, it is possible to store the model simply as a table of transition probabilities. Clearly, in a continuous domain, other forms of representation are required. By using a simple DBN for the model, one can take advantage of prior knowledge about the structure of the environment, and is left only with the problem of learning the parameters of the DBN (Russell et al., 1994). For our linear models, learning is then an application of multivariate linear regression.

The general model-based algorithm is described in Figure 2. The algorithm for prioritized sweeping is similar to the general model-based case. The key difference is that priorities are used to determine which simulations are performed. After each change to the model or the value function, update-priorities is called. The state-action pair with the highest priority is chosen to be simulated at each iteration of the "while time is available" loop.

3. Results

The vehicle control algorithms were tested in the BAT project simulator which is a two-dimensional microscopic highway traffic simulator. Individual vehicles are simulated traveling along networks that may include highways with varying numbers of lanes as well as highways interconnected with onramps and offramps. The simulation proceeds according to a sim-

```
procedure DoModelBasedRL ()
  (1) loop
    (2) perform an action $\boldsymbol{a}$ in the
        environment from state $\boldsymbol{s}$,
        end up in state $\boldsymbol{s}'$
    (3) update the model;
        let $\Delta_{\Theta_M}$ be the change in the model
    (4) perform value-propagation for $\boldsymbol{x} = \langle s, a \rangle$,
        let $\Delta_{\Theta_V}$ be the change in the Q function
    (5) while there is available computation time
      (6) choose a state-action pair, $\boldsymbol{x}_i$
      (7) perform value-propagation for $\boldsymbol{x}_i$,
          update $\Delta_{\Theta_V}$
```

Figure 2. Model-based reinforcement learning algorithm

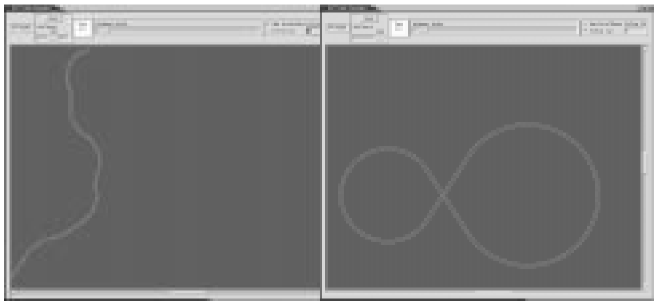

Figure 3. Two highway networks for training: a figure-eight network and a winding highway with a number of tight curves

ulated clock. At every time step, each vehicle receives sensor information about its environment and chooses steering, throttle, and braking actions based on its control strategy.

The controller controls the acceleration a and the steering angle α. where W is the wheelbase or length of the vehicle. In lane following, the object is to have the vehicle to control the steering so that the vehicle follows the center of a predetermined as closely as possible. Speed and highway networks are varied. The evaluation highway networks consisted of a figure-eight track, an oval, and a road with a variety of sharp curves (see Figure 3).

The vehicle dynamics for the simple idealized case are:

$$\begin{aligned} \dot{V_a} &= a \\ \dot{\psi} &= \frac{V_a \sin \alpha}{W} \end{aligned}$$

where W is the wheelbase or the length of the vehicle and ψ is the change in the vehicle's heading. For this model, one can derive near-optimal hand-coded controllers. For a curve with a radius of curvature ρ, the correct tire angle $\alpha = \arcsin -W/\rho$ and a PD controller can be used to correct the vehicle within the lane. Maintaining the speed can be done by simply setting a to zero. Tracking a vehicle is done by using a PD controller where the gains have been fine tuned by hill climbing.

The results for lane following with simple dynamics are in Figure 4. Model-free IBRL and prioritized sweeping are compared to a derived controller for lane following. The speeds were varied between 15 and 35 m/s. Neither of the learned controllers exceeded the performance of the analytically-derived control, but they did both come rather close. Both the model-free and model-based methods were able to successfully navigate all of the tracks at all of the various speeds.

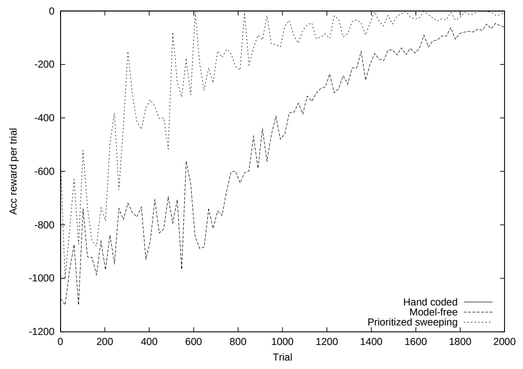

Figure 4. Lane following (Q, PS, Control-theoretic): This figure shows the results for lane following with simple vehicle dynamics. The analytically-derived hand-coded controller performance is -0.72 reward/trial, while the prioritized sweeping and model-free controllers reached -2 and -8 respectively after 2500 trials. The learning controller only controls the steering angle. Each controller was run on three highway networks for a maximum of 200 seconds.

A more realistic set of vehicle dynamics is the combined lateral and longitudinal model developed by PATH researchers (Tomizuka et al., 1995; Hedrick et al., 1993). It models the longitudinal dynamics of a Lincoln Town Car and the lateral dynamics of a Toyota Celica. This model is among the most accurate vehicle models available, although even this incredibly detailed system does not perfectly represent the complexity of a real vehicle. For example, the model assumes that there is no slip at the wheels and that there are no time delays in combustion dynamics. Nevertheless, the system is decidedly difficult to control. As shown in Table 1, the learned actions do significantly outperform a painstakingly developed hand-coded policy. Moreover, the learned controllers were able to achieve greater functionality. On the test track with curves, the hand-coded controller was not able to navigate the entire route at $45m/s$ without driving off the road. The learned controller was able to make it through the entire track. The same structure was used for models as in the simple dynamics. In the case of the complex dynamics, it was not possible to learn the model exactly since it no longer exactly fit the true system, but using the mean of the various state variables from the model was still quite effective.

Algorithm	Reward/Trial	Trials required
Hand-coded	-126.44	n/a
Prioritized Sweeping	-9.24	21k

Table 1. Lane following performance with complex vehicle dynamics (PS, Hand-coded). The hand-coded controller always failed at high speeds on the curves highway.

4. Conclusion

Motivated by the need to learn controllers for autonomous vehicles, we found that some of the crucial design decisions and contributions involved choosing the proper representation. Using instance-based function approximation and a structured model yielded very good results. We were able to learn controllers for a number of driving tasks and other control domains achieving better performance in many cases than the explicit hand-crafted controllers.

For the details of these algorithms and their results please see the first author's doctoral dissertation (Forbes, 2002).

5. Acknowledgments

This work was supported in part by the following two grants: ONR MURI N00014-00-1-0637 "Decision Making Under Uncertainty" and NSF ECS-9873474 "Complex Motor Learning." The second author was supported by the generosity of the Fannie and John Hertz Foundation. We would also like to thank Stuart Russell, Ron Parr, Dirk Ormoneit, and Chris Atkeson for their ideas and useful conversations on this work.

References

Andre, D., Friedman, N., & Parr, R. (1997). Generalized prioritized sweeping. *Advances in Neural Information Processing Systems.*

Bradshaw, G. (1985). *Learning to recognize speech sounds: A theory and model.* Doctoral dissertation, Carnegie Mellon University.

Forbes, J., & Andre, D. (2000a). *Practical reinforcement learning in continuous domains* (Technical Report UCB/CSD-00-1109). UC Berkeley Computer Science Division, Berkeley, California.

Forbes, J., & Andre, D. (2000b). Real-time reinforcement learning in continuous domains. *AAAI Spring Symposium on Real-Time Autonomous Systems.*

Forbes, J. R. N. (2002). *Reinforcement learning for vehicle control.* Doctoral dissertation, University of California, Berkeley.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. *Twelth International Conference on Machine Learning* (pp. 290–299). San Francisco: Morgan Kaufmann.

Hedrick, J., McMahon, D., & Swaroop, D. (1993). *Vehicle modeling and control for automated highway systems* (Technical Report UCB-ITS-PRR-93-24). California PATH/UC Berkeley.

Kibler, D., & Aha, D. W. (1988). Comparing instance-averaging with instance-filtering algorithms. *Third European Working Session on Learning* (pp. 63–80). Glasgow, Scotland: Pitman.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, *78*, 1464–1480.

Kumar, P., & Varaiya, P. (1986). *Stochastic systems: Estimation, identification, and adaptive control.* Englewood Cliffs, New Jersey: Prentice-Hall.

Mataric, M. J. (1991). *A comparative analysis of reinforcement learning methods* (Technical Report 1322). M.I.T. AI Lab.

Moore, A. W. (1991). *An introductory tutorial on kd-trees* (Technical Report 209). Computer Laboratory, University of Cambridge. Extract from A. W. Moore's Phd. thesis: Efficient Memory-based Learning for Robot Control.

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

Moore, A. W., Atkeson, C. G., & Schaal, S. A. (1997). Locally weighted learning for control. *AI Review*, *11*, 75–113.

Ormoneit, D., & Sen, S. (1999). *Kernel-based reinforcement learning* (Technical Report 1999-8). Department of Statistics, Stanford University.

Russell, S. J., Binder, J., & Koller, D. (1994). *Adaptive probablistic networks* (Technical Report UCB/CSD-94-824). Computer Science Division, University of California, Berkeley.

Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine Learning*, *6*, 251–276.

Santamaria, J. C., Sutton, R. C., & Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, *6*.

Smart, W. D., & Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. *Seventeenth International Conference on Machine Learning.*

Sutton, R. S. (1990). First results with DYNA, an integrated architecture for learning, planning, and reacting. *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90).* Boston, Massachusetts: MIT Press.

Tomizuka, M., Hedrick, J., & Pham, H. (1995). *Integrated manuevering control for automated highway system based on a magnetic reference/sensing system* (Technical Report UCB-ITS-PRR-95-12). California PATH/UC Berkeley.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* Doctoral dissertation, King's College, Cambridge, UK.