Locally Weighted Learning for Control

Christopher G. Atkeson[*], **Andrew W. Moore**[†], **and Stefan Schaal**[‡]
College of Computing, Georgia Institute of Technology[*‡]
801 Atlantic Drive, Atlanta, GA 30332-0280
cga@cc.gatech.edu, sschaal@cc.gatech.edu
http://www.cc.gatech.edu/fac/Chris.Atkeson
http://www.cc.gatech.edu/fac/Stefan.Schaal
404-894-1076, fax: 404-853-9378
Carnegie Mellon University[†]
5000 Forbes Ave, Pittsburgh, PA 15213
awm@cs.cmu.edu
http://www.cs.cmu.edu/~awm/hp.html
ATR Human Information Processing Research Laboratories[‡]
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

June 13, 1996 at 9:05

Abstract

Lazy learning methods provide useful representations and training algorithms for learning about complex phenomena during autonomous adaptive control of complex systems. This paper surveys ways in which locally weighted learning, a type of lazy learning, has been applied by us to control tasks. We explain various forms that control tasks can take, and how this affects the choice of learning paradigm. The discussion section explores the interesting impact that explicitly remembering all previous experiences has on the problem of learning to control.

Keywords: locally weighted regression, LOESS, LWR, lazy learning, memory-based learning, least commitment learning, forward models, inverse models, linear quadratic regulation (LQR), shifting setpoint algorithm, dynamic programming.

1 Introduction

The necessity for self improvement in control systems is becoming more apparent as fields such as robotics, factory automation, and autonomous vehicles become impeded by the complexity of inventing and programming satisfactory control laws. Learned models of complex tasks can aid the design of appropriate control laws for these tasks, which often involve decisions based on streams of information from sensors and actuators, where data is relatively plentiful. The tasks may change over time, or multiple tasks may need to be performed. Lazy learning methods provide an approach to learning models of complex phenomena, dealing with large amounts of data, training quickly, and avoiding interference between multiple tasks during control of complex systems (Atkeson et al., 1995). This paper describes five ways in which lazy learning techniques have been applied by us to control tasks.

In learning control, there is an important distinction between *representational tools*, such as lookup tables, neural networks, databases of experiences, or structured representations, and what we will call *learning paradigms,* which define what the representation is used for, where training data comes from, how the training data is used to modify the representation, whether exploratory actions are performed, and other related issues. It is difficult to evaluate a representational tool independently of the paradigm in which it is used, and vice versa. A successful robot learning algorithm typically is composed of sophisticated representational tools and learning paradigms. We will describe using the same representational tool, *locally weighted learning* (Atkeson et al., 1995), in different tasks with different learning paradigms and with different results.

In defining paradigms for learning to control complex systems it is useful to identify three separate components of an indirect (model-based) adaptive control system: modeling, exploration, and policy design. The first component, *modeling,* is the process of forming explicit models of the task and the environment. All of the approaches we will describe will form explicit world models. Moore and Atkeson (1993) explore some of the advantages and disadvantages of approaches that form explicit models versus those that avoid forming models. Often the modeling process is equated with *function approximation*, in which a representational tool is used to fit a training data set. Focusing only on the modeling component leaves several important questions unanswered. For example, "where does the training data come from?" and "what new training data should be collected?" are addressed by the *exploration* component. The question "how should the identified model be used to select actions?" is addressed by the *policy design* or *control law design* component.

The aim of this paper is to survey the implications of using locally weighted regression, a lazy learning technique, as the modeling component of our three part control system. Lazy modeling techniques cannot be implemented or discussed without exploring related issues in exploration and policy design. Although the policy design and exploration components are not "lazy" in the same sense as the modeling component, they should exploit the capabilities of lazy modeling, and make a lazy modeler's job easier.

1.1 Why Focus on Lazy Learning For Learning to Control?

We will not review lazy learning here, but expect that our reader has already read the companion paper in this collection (Atkeson et al., 1995), from which we will borrow both terminology and notation. In the form of lazy learning we will focus on, *locally weighted learning,* experiences are explicitly remembered, and predictions and generalizations are performed in real time by building a local model to answer any particular query (an input for which the function's output is desired). The motivation for focussing on locally weighted learning was not that it is a more accurate function approximator than other methods such as multi-layer sigmoidal neural networks, radial basis functions, regression trees, projection pursuit regression, other statistical nonparametric regression techniques, and global regression techniques, but that lazy learning techniques avoid negative interference. One of the primary characteristics of learning to control a robot is that data comes in continuously, and the distribution of the data changes as the robot learns and changes its performance task. Locally weighted learning easily learns in real time from the continuous stream of training data. It also avoids the negative interference exhibited by other modeling approaches, because locally weighted learning retains all the training data, as do many lazy learning methods (Atkeson et al., 1995).

Our approach to modeling the complex functions found in typical task or process dynamics is to use a collection of simple local models. One benefit of local modeling is that it avoids the difficult problem of finding an appropriate structure for a global model. A key idea in lazy learning is to form a training set for the local model after a query is given. This approach allows us to select from the training set only relevant experiences (nearby samples) and to weight those experiences according to their relevance to the query. We form a local model of the function at the query point, much as a Taylor series models a function in the neighborhood of a point. This local model is then used to predict the output of the function for that query. After answering the query, the local model is discarded. A new local model is created to answer each query. This leads to another benefit of lazy modeling for control: we can delay the choice of local model structure and structural parameters until a query must be answered, and we can make different choices for subsequent queries (Atkeson et al., 1995).

Locally weighted learning can represent nonlinear functions, yet has simple training rules with a single global optimum for building a local model in response to a query. This allows complex nonlinear models to be identified (trained) quickly. Currently we are using polynomials as the local models. Since the polynomial local models are linear in the parameters to be estimated, we can calculate these parameters using a linear regression. Fast training makes continuous learning from a stream of new input data possible. It is true that lazy learning transfers the computational load onto the lookup process, but our experience is that the linear parameter estimation process during lookup in locally weighted learning is still fast enough for real time robot learning (Atkeson et al., 1995).

We use cross validation to choose an appropriate distance metric and weighting function, and to help find irrelevant input variables and terms in the local model. In fact, performing one cross validation evaluation in lazy learning is no more expensive than processing a single query (Atkeson et al., 1995). Cheap cross validation makes search for model parameters routine, and we have explored procedures that take advantage of this (Atkeson et al., 1995; Maron and Moore, 1994; Moore et al., 1992; Moore and Lee, 1994).

We have extended the locally weighted learning approach to give information about the reliability of the predictions and local linearizations generated, based on the local density and distribution of the data and an estimate of the local variance (Atkeson et al., 1995; Schaal and Atkeson, 1994a,b). This allows a robot to monitor its own skill level, protect itself from its ignorance by designing robust policies, and guide its exploratory behavior.

Another attractive feature of locally weighted learning is flexibility. There are explicit parameters to control smoothing, outlier rejection, forgetting, and other processes. The modeling process is easy to understand, and therefore easy to adjust or control (Atkeson et al., 1995).

We will see how the explicit representation of specific memories can speed up convergence and improve the robustness and autonomy of optimization and control algorithms (Atkeson et al., 1995; Moore and Schneider, 1995). It is frustrating to watch a robot repeat its mistakes, with only a slight improvement on each attempt. The goal of the learning algorithms described here is to improve performance as rapidly as possible, using as little training data as possible (data efficiency).

1.2 Related Work

Locally weighted learning is being increasingly used in control. Zografski has explored the use of locally weighted regression in robot control and modeling time series, and also compared LWR to neural networks and other methods (Zografski, 1989, 1991, 1992; Zografski and Durrani, 1995). Gorinevsky and Connolly (1994) compared several different approximation schemes (neural nets, Kohonen maps, radial basis functions, and local polynomical fits) on simulated robot inverse kinematics with added noise, and showed that local polynomial fits were more accurate than all other methods. van der Smagt et al. (1994) learned robot kinematics using local linear models at the leaves of a tree data structure. Tadepalli and Ok (1996) apply local linear regression to reinforcement learning. Baird and Klopf (1993) apply nearest neighbor techniques and weighted averaging to reinforcement learning and Thrun (1996) and Thrun and O'Sullivan (1996) apply similar techniques to robot learning. Connell and Utgoff (1987) interpolated a value function using locally weighted averaging to balance an inverted pendulum (a pole) on a moving cart. Peng (1995) performed the cart pole task using locally weighted regression to interpolate a value function. Aha and Salzberg (1993) explored nearest neighbor and locally weighted learning approaches to a tracking task in which a robot pursued and caught a ball. McCallum (1995) explored the use of lazy learning techniques in situations where states were not completely measured. Farmer and Sidorowich (1987, 1988a,b) apply locally weighted regression to modeling and prediction of chaotic dynamic systems. Huang (1996) uses nearest neighbor and weighted averaging techniques to cache simulation results and accelerate a movement planner.

1.3 Outline

This article is organized by types of control tasks, and in the next sections we will examine a progression of control tasks of increasing complexity. We have chosen these tasks because we have implemented lazy learning as part of a learning controller for each of them. For each type of task we will show how lazy learning of models interacts with other parts of the learning control paradigm being described. For several tasks we also provide implementation details. The progression of control tasks is outlined in Table 1. Temporally independent tasks include many forms of setpoint based process control, and are of economic importance. We describe several versions of temporally dependent tasks, which include trajectory following tasks such as process control transients and vehicle maneuvers. We conclude with a discussion of some of the benefits and drawbacks of lazy learning in this context.

Table 1: The control tasks explored in this paper. Symbols and mathematics described in some of the entries will be explained in the corresponding sections.

Task	Task Specification	Goal	Example	Sec.
Temporally Independent	$\mathbf{y}_\mathrm{d}$: the desired output	Choose $\mathbf{u}$ such that $\mathrm{E}[\mathbf{y}] = \mathbf{y}_\mathrm{d}$	Billiards	2
Deadbeat Control	$\mathbf{x}_\mathrm{d}$ or trajectory $\{\mathbf{x}_\mathrm{d}(t)\}$	Choose $\mathbf{u}(t)$ such that $\mathrm{E}[\mathbf{x}(t+1)] = \mathbf{x}_\mathrm{d}(t+1)$	Devil Sticking I	3.1
Dynamic Regulation	$\mathbf{x}_\mathrm{d}$ and matrices $\mathbf{Q}$ and $\mathbf{R}$	Minimize future cost $C = \sum_{t=0}^{\infty}\left(\boldsymbol{\delta}\mathbf{x}(t)^\mathrm{T}\mathbf{Q}\boldsymbol{\delta}\mathbf{x}(t) + \mathbf{u}(t)^\mathrm{T}\mathbf{R}\mathbf{u}(t)\right)$	Devil Sticking II	3.2
Dynamic Regulation, unspecified setpoint	$\mathbf{Q}$ and $\mathbf{R}$	Choose setpoint to minimize future cost C	Devil Sticking III	3.4
Nonlinear Optimal Control	Cost function $G(\mathbf{x}(t), \mathbf{u}(t), t)$	Find a control policy to minimize the sum of future costs	Puck	3.6

2 Temporally Independent Tasks

In the simplest class of tasks we will consider, the environment provides an outcome represented with a vector $\mathbf{y}$ as a function of an action vector $\mathbf{u}$, which we can choose, a state vector $\mathbf{x}$, which we can observe but not choose, and random noise.

$$\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{u}) + \text{noise} \tag{1}$$

The task is to choose $\mathbf{u}$ so that the expected outcome $\mathbf{y}$ is $\mathbf{y}_\mathrm{d}$: $\mathrm{E}\left[\mathbf{y}\right] = \mathbf{y}_\mathrm{d}$, where E is the expectation operator from probability theory. The function $\mathbf{f}()$ is not known at the beginning of the task. Section 2.2 will describe how lazy learning can be used to learn a model of $\mathbf{f}()$: $\widehat{\mathbf{f}}()$.

Several relationships could be modeled using lazy learning techniques including forward models, inverse models, policies, and value functions. We will discuss policies and value functions in the context of temporally dependent tasks in later sections. The next sections describe inverse and forward models.

2.1 Control Using Inverse Models

An *inverse model* uses states and outcomes to predict the necessary action (Atkeson, 1990; Miller, 1989):

$$\mathbf{u} = \widehat{\mathbf{f}}^{-1}(\mathbf{x}, \mathbf{y}) \tag{2}$$

This function specifies directly what action to take in each state, but does not specify what would happen given a state and an action. A lazy learner can represent an inverse model using a database of experiences, arranged so that the input vectors of each experience are the concatenation of state and outcome vectors (Figure 1). The corresponding output is the action needed to produce the given outcome from the given state. The database is trained by adding new observed states, actions, and outcomes: $(\mathbf{x}, \mathbf{u}, \mathbf{y})$.

A learned inverse model can provide a conceptually simple controller for temporally independent tasks. An action is chosen by using the current state and desired outcome as an

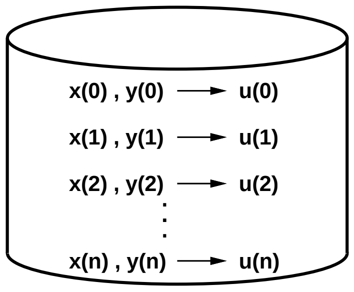


Figure 1: A database implementing an inverse model.

index into the database. The closest match in the database can be found or an interpolation of nearby experiences (i.e., a weighted average or locally weighted regression approach) can be used. If there are no stored experiences close enough to the current situation, another method, such as choosing actions randomly, can be used to select an action. This distance threshold is task dependent and can be set by the user.

The strength of an inverse model controller in conjunction with lazy learning is that the learning is *aggressive*: during repeated attempts to achieve the same goal the action that is applied is not an incrementally adjusted version of the previous action, but is instead the action that the lazy learner predicts will directly achieve the required outcome. Given a monotonic relationship between $\mathbf{u}$ and $\mathbf{y}$, the sequence of actions that are chosen are closely related to the Secant method (Conte and De Boor, 1980) for numerically finding the zero of a function. See (Ortega and Rheinboldt, 1970) for a good discussion of the multidimensional generalization of the Secant method. An inverse model, represented using locally weighted regression and trained initially with a feedback learner, has been used by Atkeson (1990).

A commonly observed problem with the inverse model is that, if the vector space of actions has a different dimensionality than that of outcomes, then the inverse model is not well defined. Problems also result if the mapping is not one to one, or if there are misleading noisy observations. Learning can become stuck in permanent pockets of inaccuracy that are not reduced with experience. Figure 2 illustrates a problem where a non-monotonic relation between actions and outcomes is misinterpreted by the inverse model. Even if the inverse model had interpreted the data correctly, any locally weighted averaging on $\mathbf{u}$ would have led to incorrect actions (Moore, 1991a; Jordan and Rumelhart, 1992). In subsequent sections on temporally dependent tasks, we will discuss how sometimes the action selected by the inverse function is too aggressive.

2.2 Control Using Forward Models

The *forward model* uses states and actions to predict outcomes (Miller, 1989; Mel, 1989; Moore, 1990; Jordan and Rumelhart, 1992):

$$\mathbf{y} = \widehat{\mathbf{f}}(\mathbf{x}, \mathbf{u}) \tag{3}$$

This allows prediction of the effects of various actions (mental simulation) but does not prescribe the correct action to take.

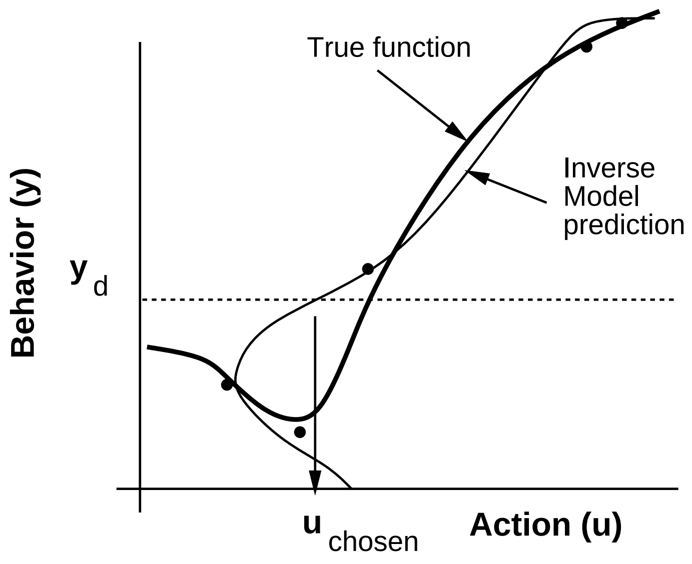


Figure 2: The true relation (shown as the thick black line) is non-monotonic. When an outcome is desired at the shown value y_d, the action that is suggested produces an outcome that differs from the desired one. Worse, the new data point that is added (at the intersection of the thick black line and the vertical arrow) will not change the inverse model near $\mathbf{y}_\mathrm{d}$, and the same mistake will be repeated indefinitely.

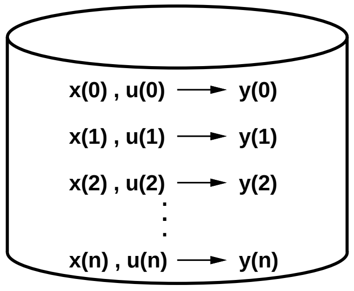


Figure 3: A database implementing a forward model.

We now arrange the memory-base so that the input vectors of each data point are the concatenation of state and action vectors (Figure 3). The corresponding output is the actual outcome that was observed when the state-action pair was executed in the real world. The forward model can be trained from observations of states, actions, and outcomes: $(\mathbf{x}, \mathbf{u}, \mathbf{y})$.

To use this model for control requires more than a single lookup. Actions are chosen by on–line numerical inversion of the forward model, that requires searching a set of actions to find one that is predicted to achieve the desired output. This computation is identical to numerical root finding over the empirical model. A number of root-finding schemes are applicable, with desirability depending on the dimensionality of the actions, the complexity of the function and the amount of time available in which to perform the search:

- *Grid Search:* Generate all available actions sampled from a uniform grid over action space. Take the action that is predicted to produce the closest outcome to $\mathbf{y}_\mathrm{d}$.
- *Random Search:* Generate random actions, and again use the action which is predicted to produce the closest outcome to $\mathbf{y}_\mathrm{d}$.
- *First Order Gradient Search:* Perform a steepest-ascent search from an initial candidate action toward an action that will give the desired output (Press et al., 1988).

Finding the local gradient of the empirical model is easy if locally weighted regression is used (Atkeson et al., 1995). Part of the computation of the locally weighted regression model forms the local linear map, so it is already available. We may write the prediction local to $\mathbf{x}$ and $\mathbf{u}$ as

$$\widehat{\mathbf{f}}(\mathbf{x} + \boldsymbol{\delta}\mathbf{x}, \mathbf{u} + \boldsymbol{\delta}\mathbf{u}) \approx \mathbf{c} + \mathbf{A}\boldsymbol{\delta}\mathbf{x} + \mathbf{B}\boldsymbol{\delta}\mathbf{u} + \text{2nd order terms} \tag{4}$$

where $\mathbf{c}$ is a vector and $\mathbf{A}$ and $\mathbf{B}$ are matrices obtained from the regression, such that

$$\mathbf{c} = \widehat{\mathbf{f}}(\mathbf{x}, \mathbf{u}) \quad \mathbf{A}_{ij} = \frac{\partial \widehat{\mathbf{f}}_i}{\partial \mathbf{x}_j} \quad \mathbf{B}_{ij} = \frac{\partial \widehat{\mathbf{f}}_i}{\partial \mathbf{u}_j} \tag{5}$$

The gradient ascent iteration is:

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{B}^{\mathrm{T}}(\mathbf{y}_{\mathrm{d}} - \mathbf{c}) \tag{6}$$

with $\mathbf{B}$ and $\mathbf{c}$ as defined in Equation 5. This approach may become stuck in local minima, so an initial grid search or random search may provide a set of good starting points for gradient searches.

- *Second Order Gradient Search:* Use Newton's method to iterate towards an action with the desired output (Press et al., 1988). If $\mathbf{u}_k$ is an approximate solution, Newton's method gives $\mathbf{u}_{k+1}$ as a better solution where

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{B}^{-1}(\mathbf{y}_{\mathrm{d}} - \mathbf{c}) \tag{7}$$

with $\mathbf{B}$ and $\mathbf{c}$ as defined in Equation 5. Newton's method is less stable than first order gradient search, but if a good approximate solution is available, perhaps from one of the other search methods, and the local linear model structure is correct in a region including the current action and the best action, it produces a good estimate of the best action in only two or three iterations.

If the partial derivative matrix $\mathbf{B}$ is singular, or the action space and state space differ in dimensionality, then robust matrix techniques based on the pseudo-inverse can be applied to invert $\mathbf{B}$ (Press et al., 1988). The forward model can be used to minimize a criterion C that penalizes large commands as well as errors, which also makes this search more robust:

$$C = (\mathbf{y}_{\mathrm{d}} - \mathbf{c})^{\mathrm{T}}\mathbf{Q}(\mathbf{y}_{\mathrm{d}} - \mathbf{c}) + \mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u} \tag{8}$$

The matrices $\mathbf{Q}$ and $\mathbf{R}$ allow the user to control which components of the error are most important.

2.3 Combining Forward and Inverse models

The inverse model can provide a good initial starting point for a search using the forward model:

$$\mathbf{u}_0 = \widehat{\mathbf{f}}^{-1}(\mathbf{x}, \mathbf{y}_{\mathrm{d}})$$

$\mathbf{u}_0$ can be evaluated using a lazy forward model with the same data:

$$\hat{\mathbf{y}} = \hat{\mathbf{f}}(\mathbf{x}, \mathbf{u}_0)$$

Provided $\hat{\mathbf{y}}$ is close to $\mathbf{y}_\mathrm{d}$, Newton's method can then be used for further refinement. If $\hat{\mathbf{y}}$ is not close to $\mathbf{y}_\mathrm{d}$, the local linear model may not be a good fit, and the aggressive Newton step may move away from the goal.

2.4 Exploration in Temporally Independent Learning

A nice feature of the approaches described so far is that in normal operation they perform their own exploration, reducing the need for human supervision or external guidance. The experiments are chosen greedily at the exact points where the desired output is predicted to be, which for the forward model is guaranteed to provide useful data. If an action is wrongly predicted to succeed, the resulting new data point will change the prediction of the forward model for that state and action, helping to prevent the error from being repeated.

In the early stages of learning, however, there may be no action that is predicted to give the desired outcome. A simple experiment design strategy is to choose actions at random. It is more effective to choose data points which, given the uncertainty inherent in the prediction, are considered most likely to achieve the desired outcome. This can considerably reduce the exploration required (Moore, 1991a; Cohn et al., 1995).

2.5 A Temporally Independent Task: Billiards

In order to explore the efficacy of lazy learning methods for the control of temporally independent tasks, the previously described approaches were implemented on the billiards robot shown in Figure 4 (Moore, 1992; Moore et al., 1992). The equipment consists of a small ($1.5m \times 0.75m$) pool table, a spring actuated cue with a rotary joint under the control of a stepper motor, and two cameras attached to a Datacube image processing system. All sensing is visual: one camera looks along the cue stick and the other looks down at the table. The cue stick swivels around the cue ball, which, in this implementation, has to start each shot at the same position. A shot proceeds as follows:

1. At the start of each attempt the *object ball* (i.e., the ball we want to sink in a pocket) is placed at a random position in the half of the table opposite the cue stick. This random position is selected by the computer to avoid human bias.

2. The camera above the table obtains the centroid image coordinates of the object ball ($x^{\mathrm{above}}_{\mathrm{object}}, y^{\mathrm{above}}_{\mathrm{object}}$), which constitute the state $\mathbf{x}$.

3. The controller then uses an inverse model followed by search over a forward model to find an action, $\mathbf{u}$, that is predicted to sink the object ball into the nearer of the two pockets at the far end of the table. The action is specified by what we wish the view from the cue to be just prior to shooting. Figure 5 shows a view from the cue camera during this process. The cue swivels until the centroid of the object ball's image (shown by the vertical line) coincides with the chosen action, $x^{\mathrm{cue}}_{\mathrm{object}}$, shown by the cross.

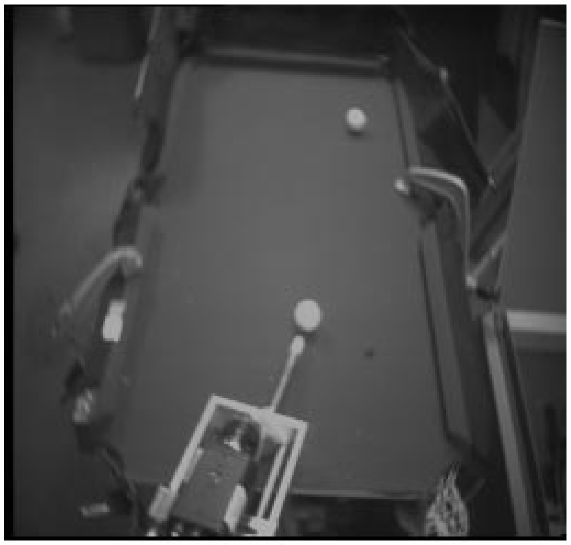


Figure 4: The billiards robot. In the foreground is the cue stick, which attempts to sink balls in the far pockets.

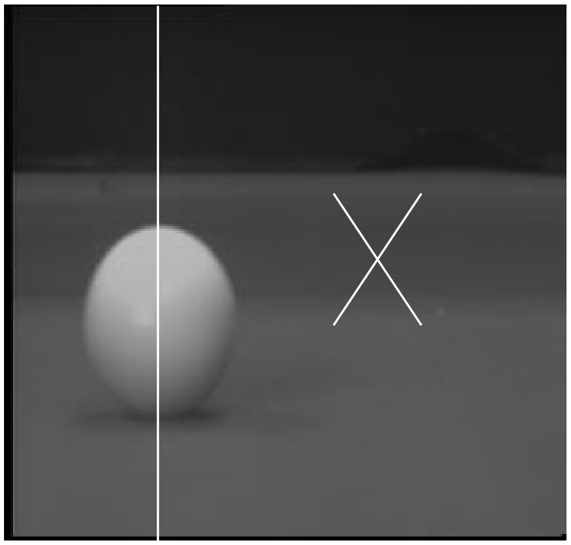


Figure 5: The view from the cue camera during aiming. The cue swivels until the centroid of the object ball's image (shown by the vertical line) coincides with the chosen action, $x^{\text{cue}}_{\text{object}}$, shown by the cross.

4. The shot is then performed and observed by the overhead camera. The image after a shot, overlaid with the tracking of both balls, is shown in Figure 6. The outcome is defined as the cushion and position on the cushion where the object ball first collides. In Figure 6 it is the point b.

5. Independent of success or failure, the memory-base is updated with the new observation $(x^{\text{above}}_{\text{object}}, y^{\text{above}}_{\text{object}}, x^{\text{cue}}_{\text{object}}) \to b$.

As time progresses, the database of experiences increases, hopefully converging to expertise in the two-dimensional manifold of state-space corresponding to sinking balls placed in arbitrary positions. Before learning begins there is no explicit knowledge or calibration of the robot, pool table, or cameras, beyond having the object ball in view of the overhead camera, and the assumption that the relationship between state, action and outcome is reasonably repeatable.

In this implementation the representation used for both forward and inverse models was locally weighted regression using outlier removal and cross validation for choosing the kernel width (Atkeson et al., 1995). Inverse and forward models were used together; the forward model was searched with steepest ascent. Early shots (when no success was predicted) were uncertainty-based (Moore, 1991a). After 100 shots, control choice running on a Sun-4 was taking 0.8 seconds.

This implementation demonstrates several important points. The first is the precision required of the modeling component. The cue-action must be extremely precise for success. Locally weighted regression provided the needed precision. A graph of the number of successes against trial number (Figure 7) shows the performance of the robot against time.

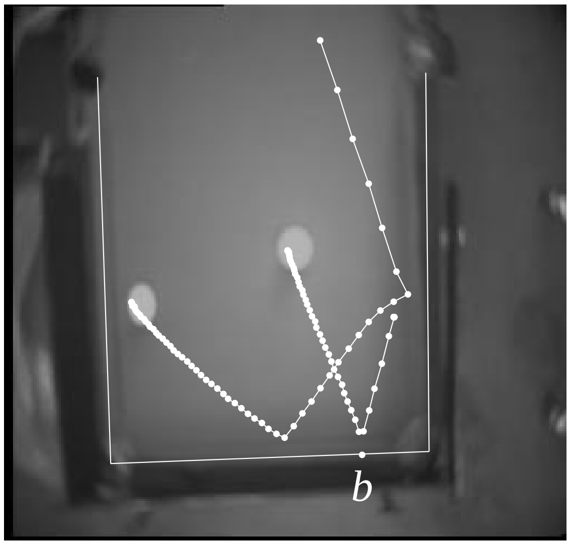


Figure 6: The trajectory of both balls is tracked using the overhead camera. b indicates the cushion and position on the cushion where the object ball first collides. In this shot the pocket was missed.

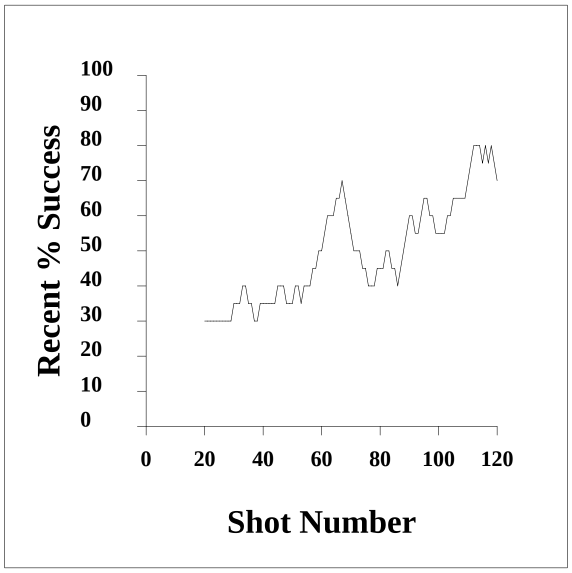


Figure 7: Frequency of successes versus control cycle for the billiards task. The number of successes, averaged over the twenty previous shots, is shown.

Sinking the ball requires better than 1% accuracy in the choice of action, the world contains discontinuities and there are random outliers in the data due to visual tracking errors, and so it is encouraging that within less than 100 experiences the robot had reached a 75% success rate. An informal assessment of this performance is that its success rate is as high as possible (given that the ball is placed at random positions, some of which are virtually impossibly difficult). Unfortunately, the only evidence for this is anecdotal: the students who built the robot (one of whom was an MIT billiards champion) could not do any better.

A second point is the non-uniformity of the training data distribution due to the implicit exploration process. Although the function being learned is only 3 inputs $\to$ 1 output, it is perhaps surprising that it achieved sufficient accuracy in only 100 data points. The reason is the aggressive non-uniformity of the training data distribution—almost all the training data was clustered around state-action pairs which get the ball in or close to a pocket. The lazy learner did not expend many resources on exploring or representing how to make bad shots.

2.6 Optimizing a Performance Criterion

Often a goal in temporally independent learning is to optimize a particular criterion, rather than achieve a particular outcome. Lazy learning can be used to represent the cost function directly and to speed the search for maxima or minima (Moore and Schneider, 1995). A linear local model can be used to estimate the first derivatives (gradient) and a quadratic local model can be used to estimate the second derivatives (Hessian) of the cost function at the current point in the optimization procedure. These estimates can be used in first order gradient search, or in a Newton search that uses estimates of second derivatives. Constraints on the output can be included in this optimization process.

2.7 Temporal Dependence in Temporally Independent Tasks

It is considerably easier to choose actions for temporally independent than temporally dependent tasks because the choice of action has no effect on future states. There is no need to consider the effects of the current action on future states and indirectly on future performance. In Section 3 we will consider temporally dependent tasks where there is an opportunity to choose suboptimal actions in the short-term to obtain more desirable states and thereby improve performance in the long-term.

However, temporally independent tasks do provide an opportunity to increase the knowledge available to the controller in order to improve future performance. They differ from batch learning tasks, because new training data becomes available after each action, and the choice of action, which depends on inferences from earlier training data, affects the training data available to future decisions. Modifying actions to increase knowledge rather than greedily pursue a desired outcome is the responsibility of the exploration component of the controller.

3 Temporally Dependent Tasks

A more complex class of learning control tasks occur when the assumption of temporal independence is removed: $\mathbf{x}(t+1)$ may now be influenced by $\mathbf{x}(t)$. A useful case to explore is when the outcome is the next state:

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{9}$$

The task may be to regulate the state to a predefined desired value called a *setpoint* $\mathbf{x}_\mathrm{d}$ or to a sequence or trajectory of states: $\mathbf{x}_\mathrm{d}(1), \mathbf{x}_\mathrm{d}(2), \mathbf{x}_\mathrm{d}(3) \ldots$

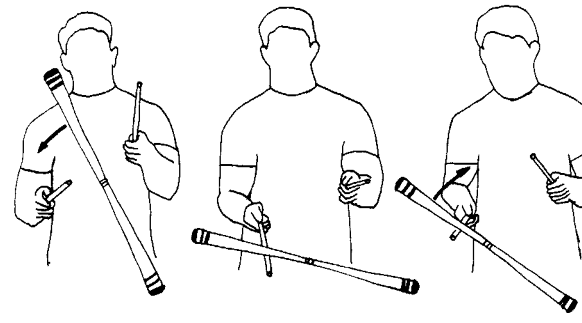
3.1 Deadbeat Control

One approach to performing temporally dependent tasks is to use the successful techniques from the previous section, and ignore the temporal dependence. One-step *deadbeat* control chooses actions to (in expectation) cause the immediate next state to be the desired next state (Stengel, 1986). Assuming the next state is always attainable in one step, the action may be chosen without paying attention to future states, decisions, or performance.

3.1.1 An Implementation of Deadbeat Control: Devil Sticking I

Deadbeat control using lazy learning models was explored by implementing it for a juggling task known as *devil sticking* (Schaal and Atkeson, 1994a,b). A center stick is batted back and forth between two handsticks. Figure 8 shows a sketch of our devil sticking robot. The juggling robot uses its top two joints to perform planar devil sticking. Hand sticks are mounted on the robot with springs and dampers. This implements a passive catch. The center stick does not bounce when it hits the hand stick, and therefore requires an active

(a)



(b)

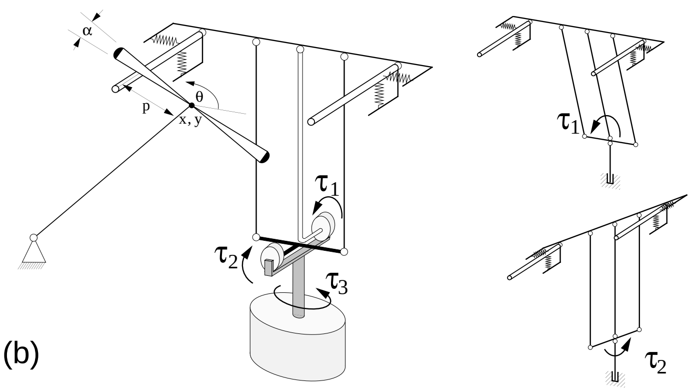


Figure 8: (a) An illustration of devil sticking, (b) A sketch of our devil sticking robot. A position change due to movement of joint 1 and 2, respectively, is indicated in the small sketches.

throwing motion by the robot. To simplify the problem the center stick is constrained by a boom to move on the surface of a sphere. For small movements the center stick movements are approximately planar. The boom also provides a way to measure the current state of the center stick. The task state is the predicted location of the center stick when it hits the hand stick held in a nominal position. Standard ballistics equations for the flight of the center stick are used to map flight trajectory measurements into a task state. The dynamics of throwing the devil stick are parameterized by five state and five action variables, resulting in a 10/5-dimensional input/output model for each hand.

Every time the robot catches and throws the devil stick it generates an experience of the form $(\mathbf{x}_k, \mathbf{u}_k, \mathbf{x}_{k+1})$ where $\mathbf{x}_k$ is the current state, $\mathbf{u}_k$ is the action performed by the robot, and $\mathbf{x}_{k+1}$ is the state of the center stick that results.

Initially we explored learning an inverse model of the task, using deadbeat control to attempt to eliminate all error on each hit. Each hand had its own inverse model of the form:

$$\hat{\mathbf{u}}_k = \hat{\mathbf{f}}^{-1}(\mathbf{x}_k, \mathbf{x}_{k+1}) \tag{10}$$

Before each hit the system looked up a command with the predicted nominal impact state and the desired result state $\mathbf{x}_d$:

$$\hat{\mathbf{u}}_k = \hat{\mathbf{f}}^{-1}(\mathbf{x}_k, \mathbf{x}_d) \tag{11}$$

Inverse model learning using lazy learning (locally weighted regression) was successfully used to train the system to perform the devil sticking task. Juggling runs up to 100 hits were achieved. The system incorporated new data in real time, and used databases of several hundred hits. Lookups took less than 15 milliseconds, and therefore several lookups could be performed before the end of the flight of the center stick (the flight duration was approximately 0.4s). Later queries incorporated more measurements of the flight of the center stick and therefore more accurate predictions of the state of the task.

However, the system required substantial structure in the initial training to achieve this performance. The system was started with a manually generated command that was appropriate for open loop performance of the task. Each control parameter was varied systematically to explore the space near the default command. A global linear model was made of this initial data, and a linear controller based on this model was used to generate an initial training set for the locally weighted system (of approximately 100 hits). Learning with small amounts of initial data was not possible. Furthermore, learning based on just an inverse model was prone to get stuck at poor levels of performance and to repeat the same mistakes for reasons discussed in the previous section.

To eliminate these problems, we also experimented with learning based on both inverse and forward models. After a command is generated by the inverse model, it can be evaluated using a forward model based on the same data.

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k, \hat{\mathbf{u}}_k) \tag{12}$$

Because it produces a local linear model, the locally weighted regression procedure will produce estimates of the derivatives of the forward model with respect to the commands as part of the estimated parameter vector. These derivatives can be used to find a correction to the command vector that reduces errors in the predicted outcome based on the forward model.

$$\frac{\partial \hat{\mathbf{f}}}{\partial \mathbf{u}} \Delta \hat{\mathbf{u}}_k = \hat{\mathbf{x}}_{k+1} - \mathbf{x}_d \tag{13}$$

This process of command refinement can be repeated until the forward model no longer produces accurate predictions of the outcome, which will happen when the query to the forward model requires significant extrapolation from the current database. The distance to the nearest stored data point can be used as a crude measure of the validity of the forward model estimate.

We investigated this method for incremental learning of devil sticking in simulations. However, the outcome did not meet expectations: without sufficient initial data around the setpoint, the algorithm did not work. We see two reasons for this. First, similar to the pure inverse model approach, the inverse-forward model acts as a one-step deadbeat controller in that it tries to eliminate all error in one time step. One-step deadbeat control applies large commands to correct for deviations from the setpoint, especially in the presence of state measurement errors. The workspace bounds and command bounds of our devil sticking robot limit the size of allowable commands. Large control actions may also be less accurate or robust. This was the case in devil sticking, where a large control action tended to cause the center stick to fly in a random direction, and nothing was learned from that hit. Second, the ten dimensional input space is large, and even if experiences are uniformly randomly distributed in the space there is often not enough data near a particular point to make a robust inverse or forward model.

Thus, two ingredients had to be added to the devil sticking controller. First, the controller should not be deadbeat. It should plan to attain the goal using multiple control actions. We discuss control approaches that keep commands small in the next section. Second, the control must increase the data density in the current region of the state-action space in order to arrive at the desired goal state. We discuss control approaches that are more tightly coupled to exploration in a Section 3.4.

3.2 Dynamic Regulation

In this section we discuss a reformulation of temporally dependent control tasks to avoid the problems encountered by the first implementation of a lazy learner for robot control, which used deadbeat control. From a theoretical point of view, it is often not possible to return to the desired setpoint or trajectory in one step: an attempt to do so would require actions of infinite magnitude or cause the size of the required actions to grow without limit. One step deadbeat control will fail on some non-minimum phase systems, of which pole balancing is one example (Cannon, 1967). In these systems, one must move away from the goal to approach it later. In the case of the cart-pole system the cart must initially move away from the target position so that the pole leans in the direction of future cart motion towards the target. This maneuvering avoids having the pole fall backwards as the cart moves toward the target.

A controller can perform more robustly if it uses smaller magnitude actions and returns to the correct state or trajectory in a larger number of steps. This idea is posed precisely in the language of *linear quadratic regulation* (LQR), in which a long term quadratic cost criterion C is minimized that penalizes both state-errors and action magnitudes (Stengel, 1986):

$$C = \sum_{t=0}^{\infty} \Big((\mathbf{x}(t) - \mathbf{x}_{\mathrm{d}})^{\mathrm{T}} \mathbf{Q} (\mathbf{x}(t) - \mathbf{x}_{\mathrm{d}}) + \mathbf{u}^{\mathrm{T}}(t) \mathbf{R} \mathbf{u}(t) \Big) = \sum_{t=0}^{\infty} \Big(\boldsymbol{\delta}\mathbf{x}^{\mathrm{T}}(t) \mathbf{Q} \boldsymbol{\delta}\mathbf{x}(t) + \mathbf{u}^{\mathrm{T}}(t) \mathbf{R} \mathbf{u}(t) \Big) \tag{14}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are matrices whose elements set the tradeoff between the size of the action components and the error components. If, for example, $\mathbf{Q}$ and $\mathbf{R}$ were identity matrices, then the sum of squared state errors and the sum of the squared action components would be minimized.

Not using deadbeat control laws implies some amount of lookahead. LQR control assumes a time invariant task and performs an infinite amount of lookahead. *Predictive* or *Receding Horizon* control design techniques look N steps ahead every time an action is chosen. All of these techniques will allow larger state errors to reduce the size of the control signals, when compared to deadbeat methods.

The *Linear* part of the LQR approach is a local linearization of the forward dynamics of the task. We can take advantage of the locally linear state-transition function provided by locally weighted regression (Equation 4):

$$\mathbf{x}(t+1) = \mathbf{x}_{\mathrm{d}} + \boldsymbol{\delta}\mathbf{x}(t+1) \approx \widehat{\mathbf{f}}(\mathbf{x}_{\mathrm{d}} + \boldsymbol{\delta}\mathbf{x}(t), \mathbf{u}(t)) \approx \widehat{\mathbf{f}}(\mathbf{x}_{\mathrm{d}}, 0) + \mathbf{A}\boldsymbol{\delta}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{15}$$

We will assume that $(\mathbf{x}_{\mathrm{d}}, 0)$ is an equilibrium point, so $\mathbf{x}_{\mathrm{d}} = \widehat{\mathbf{f}}(\mathbf{x}_{\mathrm{d}}, 0)$, and we have the following linear dynamics:

$$\boldsymbol{\delta}\mathbf{x}(t+1) = \mathbf{A}\boldsymbol{\delta}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{16}$$

The optimal action with respect to the criteria in Equation 14 and linear dynamics in Equation 16 can be obtained by solution of a matrix equation called the *Ricatti equation* (Stengel, 1986). Assuming the locally linear model provided by the locally weighted regression is correct, the optimal action $\mathbf{u}$ is

$$\mathbf{u} = -(\mathbf{R} + \mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{A}\boldsymbol{\delta}\mathbf{x} \tag{17}$$

where $\mathbf{P}$ is obtained by initialliy setting $\mathbf{P} := \mathbf{Q}$ and then running the following iteration to convergence:

$$\mathbf{P} := \mathbf{Q} + \mathbf{A}^{\mathrm{T}}\mathbf{P}[\mathbf{I} - \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{P}]^{-1}\mathbf{A} \tag{18}$$

This rather inscrutable result is not obvious from visual inspection but follows from reasonably elementary algebra and calculus that can be found in almost any introductory controls text. We recommend (Stengel, 1986). We also provide a very simplified self-contained derivation in Appendix A. The long term cost starting from state $\mathbf{x}_{\mathrm{d}} + \boldsymbol{\delta}\mathbf{x}$ turns out to be $\boldsymbol{\delta}\mathbf{x}^{\mathrm{T}}\mathbf{P}\boldsymbol{\delta}\mathbf{x}$. Note that $\mathbf{u}$ is a linear function of the state $\mathbf{x}$ in Equation 17:

$$\mathbf{u} = -\mathbf{K}\boldsymbol{\delta}\mathbf{x} \tag{19}$$

Linear quadratic regulation has useful robustness when compared to deadbeat controllers even if the underlying linear models are imprecise (Stengel, 1986).

3.3 Implementation of Dynamic Regulation: Devil Sticking II

Linear quadratic regulation controller design permitted successful devil sticking. It did require manual generation of training data to estimate the matrices of the local linear model: $\mathbf{A}$ and $\mathbf{B}$. However, once the local linear model was reliable the robot had a complete *policy* (i.e., a control law) for the vicinity of the local linear model. The aggressiveness of the control law could be controlled by choosing $\mathbf{Q}$ and $\mathbf{R}$. These matrices were set once by us, and then not adjusted during learning.

One drawback of our LQR implementation was the need for the manual search for an equilibrium point. The robot needed to be told a nominal hit that would actually send the devil stick to the other hand. There is a continuum of reasonable equilibrium points, but our formulation required the arbitrary selection of only one. Furthermore, the experimenter did not know in advance where the set of equilibrium points were for the actual machine, so manual search for equilibrium points was a difficult task, given the five dimensional action space. The next section describes a new procedure to search for equilibrium points.

3.4 Dynamic Regulation With An Unspecified Setpoint

The learning task is considerably harder if the desired setpoint is not known in advance, and instead must itself be optimized to achieve some higher level task description. However, the setpoint of the task can be manipulated during learning to improve exploration. This is done by the *shifting setpoint algorithm* (SSA) (Schaal and Atkeson, 1994a).

SSA attempts to decompose the control problem into two separate control tasks on different time scales. At the fast time scale, it acts as a dynamic regulator by trying to keep the controlled system at a chosen setpoint. On a slower time scale, the setpoint is shifted to accomplish a desired goal. SSA uses local models from lazy learning and can be viewed as an approach to exploration in these regulation tasks, based on information on the quality of predictions provided by lazy learning.

3.4.1 Experiment Design with Shifting Setpoints

The major ingredient of the SSA is a statistical self-monitoring process. Whenever the current location in input space has obtained a sufficient amount of experience such that a measure of confidence rises above a threshold, the setpoint is shifted in the direction of the goal until the confidence falls below a minimum confidence level. At this new setpoint location, the learning system collects new experiences. The shifting process is repeated until the goal is reached. In this way, the SSA builds a narrow tube of data support in which it knows the world. This data builds the basis for the first success of the regulator controller. Subsequently, the learned model can be used for more sophisticated control algorithms, for planning, or for further exploration.

3.5 Dynamic Regulation With An Unspecified Setpoint: Devil Sticking III

The SSA method was tested on the devil sticking juggling task (Schaal and Atkeson, 1994a,b). In this case it had the following steps.

1. Regardless of the poor juggling quality of the robot (i.e., at most two or three hits per trial), the SSA made the robot repeat these initial actions with small random perturbations until a cloud of data was collected somewhere in the state-action space for each hand. An abstract illustration for this is given in Figure 9a.

2. Each point in the data cloud of each hand was used as a candidate for a setpoint of the corresponding hand by trying to predict its output from its input with locally weighted regression. The point achieving the narrowest local confidence interval became the setpoint of the hand and a linear quadratic regulator was calculated for its local linear model, estimated using locally weighted regression. By means of these controllers, the amount of data around the setpoints could quickly be increased until the quality of the local models exceeded a statistical threshold (Figure 9b) (Atkeson et al., 1995).

3. At this point, the setpoints were gradually shifted towards the goal setpoints until the statistical confidence in the predictions made by the local model again fell below a threshold (Figure 9c).

4. The SSA iterated by collecting data in the new regions of the workspace until the setpoints could be shifted again. The procedure terminated when the goal was reached, leaving a ridge of data in the state-action space (Figure 9d).

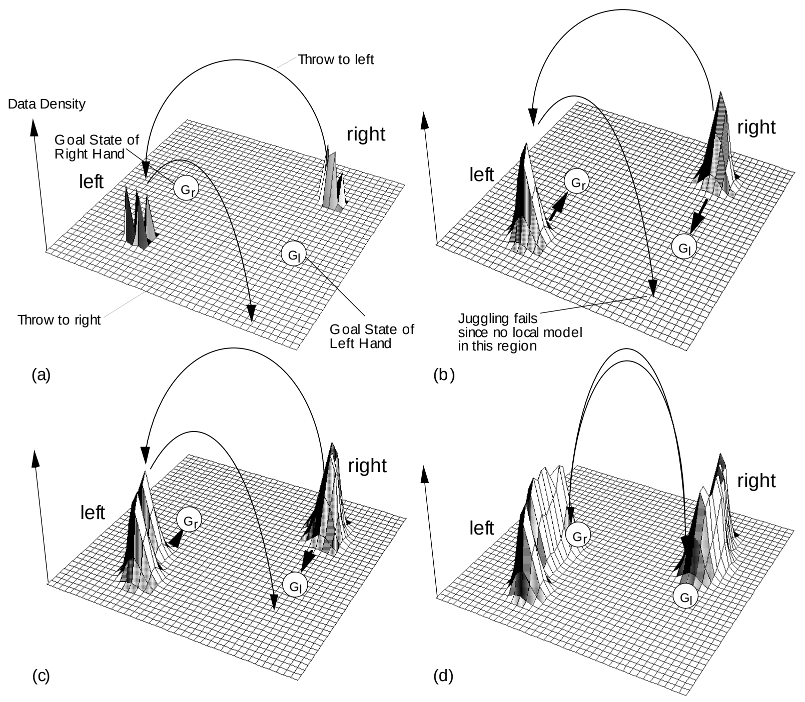


Figure 9: Abstract illustration on how the SSA algorithm collects data in space: (a) sparse data after the first few hits; (b) high local data density due to local control in this region; (c) increased data density on the way to the goals due to shifting the setpoints; (d) ridge of data density after the goal was reached.

The SSA was tested in a noise corrupted simulation and on the real robot. Each attempt to juggle the devil stick is called a *trial,* which consists of a series of left and right handed hits. Each series of trials that begins with the lazy learning system in its initial state is referred to as a *run.* Our measure of performance is the number of hits per trial. In the simulation it takes on average 40 trials before the setpoint of each hand has moved close enough to the other hand's setpoint. This is slightly better performance than with the real robot.

At that point, a breakthrough occurs and, afterwards the simulated robot rarely drops the devilstick. At this time, about 400 data points (hits) have been collected in memory. The real robot's learning performance is qualitatively the same as that of the simulated robot. Due to stronger nonlinearities and unknown noise sources the actual robot takes more trials to accomplish a steady juggling pattern. We show three typical learning runs for the actual robot in Figure 10. We do not show averages of these learning runs because averaged runs show a gradual increase in performance, which is unlike any individual learning run, which show sudden increases in performance. Peak performance of the robot was more than 2000 consecutive hits (15 minutes of continuous juggling).

3.5.1 Limits For Linear Quadratic Regulation

Control laws based on linear quadratic regulator designs are not useful if the task requires operation outside a locally linear region. The LQR controller may actually be unstable. For

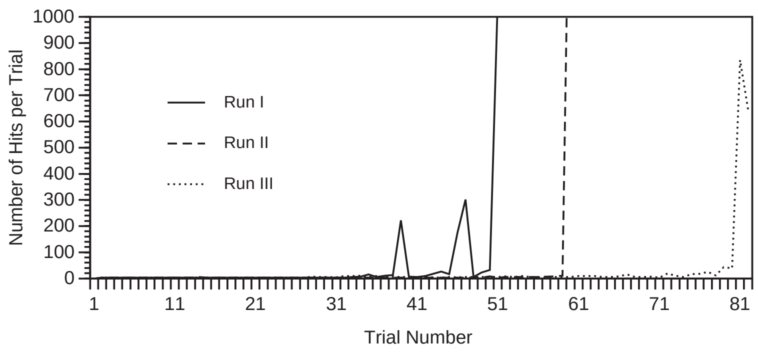


Figure 10: Learning curves of devil sticking for three runs.

example, the following one dimensional system with a one dimensional action

$$x_{k+1} = 2x_k + u_k + x_k^2 u_k \tag{20}$$

has a local linear model at the origin ($x = 0$) of $\mathbf{A} = 2$ and $\mathbf{B} = 1$ (all matrices are 1×1 for this one dimensional problem). For the optimization criteria $\mathbf{Q} = 1$ and $\mathbf{R} = 1$, and the Ricatti equation (Equations 17 and 19) gives $\mathbf{K} = 1.618$. For a goal of moving to the origin ($\mathbf{x}_{\mathrm{d}} = 0$), this linear control law is unstable for x larger than 0.95, because the actions $\mathbf{u}$ are too large. This means that the LQR "optimal" action actually increases the error $\boldsymbol{\delta}\mathbf{x}$ if the error is already larger than 0.95. This limitation of linear quadratic regulation motivates us to explore full dynamic programming based policy design approaches, which are described in the next section. Figure 11 compares the LQR based control law and the control law based on full dynamic programming using the same model and optimization criteria. Note that the shifting setpoint algorithm can provide the initial training data for these more complex approaches.

3.6 Nonlinear Optimal Control

In more general control design we must accommodate a more general formulation of the cost function or criterion to optimize and also move from local control laws based on a small number of local models to more global control laws based on many local models. We now need to learn not just a local model of the task, but many local models of the task distributed throughout the task space. We will first discuss a more general formulation of cost functions.

We are given a cost function for each step, which is known by the controller:

$$g(t) = G(\mathbf{x}(t), \mathbf{u}(t), t) \tag{21}$$

The task is to minimize one of the following expressions:

$$\sum_{t=0}^{\infty} g(t) \quad \text{or} \quad \sum_{t=0}^{t_{\max}} g(t) \quad \text{or} \quad \sum_{t=0}^{\infty} \gamma^t g(t) \text{ where } 0 < \gamma < 1 \quad \text{or} \quad \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} g(t)$$

The attractive aspect of these formulations is their generality. All of the previously described control formulations are special cases of at least one of these. For example, the quadratic one step cost defined by $\mathbf{Q}$ and $\mathbf{R}$ can be viewed as a local quadratic model of $g(t)$.

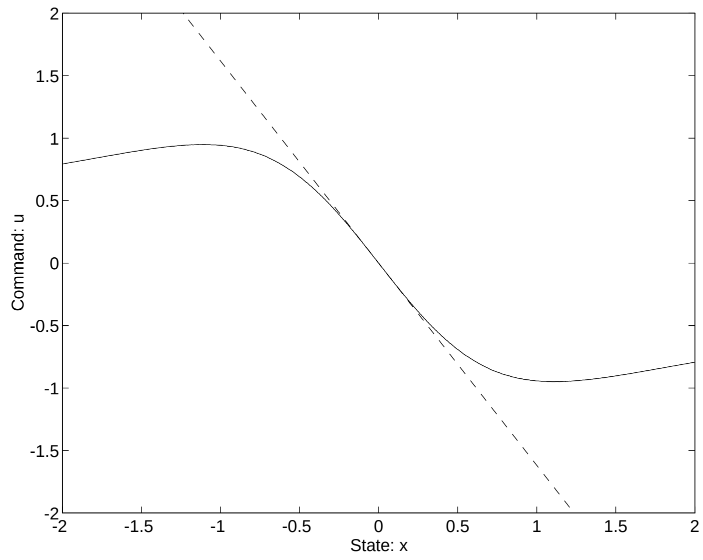


Figure 11: Solid line: optimal action based on dynamic programming (DP) using the nonlinear model; dashed line: optimal command based on a LQR design using a single linear forward model at the origin. Although in both cases the optimization criterion is the same and the LQR and DP-based control laws agree for small x, the LQR control law is linear and does not take into account the nonlinear dynamics of the task for large x

The delayed rewards nature of these tasks means that actions we choose at time t do not only affect the quality of the immediate reward but also affect the next, and all subsequent states, and in so doing affect the future rewards attainable. This leads to computational difficulties in the general case. A large literature on such learning control problems has sprung up in recent years, with the general name of *reinforcement learning*. Overviews may be found in (Sutton, 1988; Barto et al., 1990; Watkins, 1989; Barto et al., 1995; Moore and Atkeson, 1993). In this paper we will restrict discussion to the applications of lazy learning to these problems.

Again, we proceed by learning an empirical forward model $\widehat{\mathbf{x}}_{k+1} = \widehat{\mathbf{f}}(\mathbf{x}_k, \widehat{\mathbf{u}}_k)$. A general-purpose solution can be obtained by discretizing state-space into a multidimensional array of small cells, and performing a dynamic programming method (Bellman, 1957; Bertsekas and Tsitsiklis, 1989) such as value iteration or policy iteration to produce two things:

1. A *value function*, $V(\mathbf{x})$, mapping cells onto the minimum possible sum of future costs if one starts in that cell.

2. A *policy*, $\mathbf{u}(\mathbf{x})$, mapping cells onto the optimal action to take in that cell.

Value iteration can be used in conjunction with learning a world model. However, it is extremely computationally expensive. For a fixed quantization level, the cost is exponential in the dimensionality of the state variables. For a D dimensional state space and action space, and a grid resolution of R for both states and actions, one value iteration pass would require R^{2D} evaluations of the forward model. The most computationally intensive version would perform several cycles of value iteration after every update of the memory base. Less expensive forms of dynamic programming would normally perform value iteration only at the end of each trial (as we do in the example in Section 3.6.1), or as an incremental parallel process (Sutton, 1990; Moore and Atkeson, 1993; Peng and Williams, 1993).

3.6.1 A Simulation Example: The Puck

We illustrate this form of learning by means of a simple simulated example. Figure 12 depicts a frictionless puck on a bumpy surface, whose objective is to drive itself up the hill to a goal region in the minimum number of time steps. The state, $\mathbf{x} = (x, \dot{x})$, is two-dimensional and must lie in the region $-1 \leq x \leq 1$, $-2 \leq \dot{x} \leq 2$. x denotes the horizontal position of the puck in Figure 12. The action $\mathbf{u} = a$ is one-dimensional and represents the horizontal force applied to the puck. Actions are constrained such that $-4 \leq a \leq 4$. The goal region is the rectangle $0.5 \leq x \leq 0.7$, $-0.1 \leq \dot{x} \leq 0.1$. The surface upon which the puck slides has the following height as a function of x:

$$H(x) = \begin{cases} x^2 + x & \text{if } x < 0 \\ x/\sqrt{1+5x^2} & \text{if } x \geq 0 \end{cases} \tag{22}$$

The puck's dynamics are given by:

$$\ddot{x} = \frac{a}{M\sqrt{1+(H'(x))^2}} - \frac{gH'(x)}{1+(H'(x))^2} \tag{23}$$

where $M = 1$ and $g = 9.81$. This equation is integrated using:

$$\begin{aligned} x(t+1) &= x(t) + h\dot{x}(t) + \tfrac{1}{2}h^2\ddot{x}(t) \\ \dot{x}(t+1) &= \dot{x}(t) + h\ddot{x}(t) \end{aligned} \tag{24}$$

where $h = 0.01$ is the simulation time step.

Because of gravity, there is a region near the center of the hill at which the maximum rightward thrust is insufficient to accelerate up the slope. If the goal region is at the hill-top, a strategy that proceeded by greedily choosing actions to thrust towards the goal would get stuck. This is made clearer in Figure 13, a *state transition diagram.* The puck's state has two components, the position and velocity. The hairs show the next state of the puck if it were to thrust rightwards with the maximum legal force of 4 Newtons for 0.01s. At the center of state-space, even when this thrust is applied, the puck velocity decreases and it eventually slides leftwards. The optimal solution for the puck task, depicted in Figure 14, is to initially thrust away from the goal, gaining negative velocity, until it is on the far left of the diagram. Then it thrusts hard right, to build up sufficient energy to reach the top of the hill.

We explored two implementations of adaptive controllers, one of which used lazy learning techniques.

- **Implementation 1 (Grid Based): Conventional Discretization.** This used the conventional reinforcement learning strategy of discretizing state space into a grid of 60×60 cells for the forward model and value function. The reinforcement learning

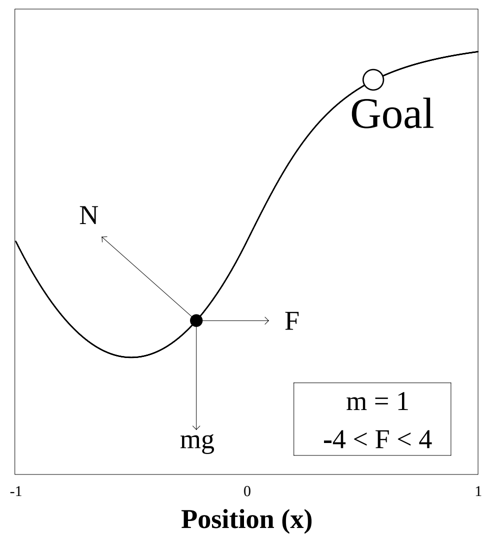


Figure 12: A frictionless puck acted on by gravity and a horizontal thruster. The puck must get to the goal as quickly as possible. There are bounds on the maximum thrust.

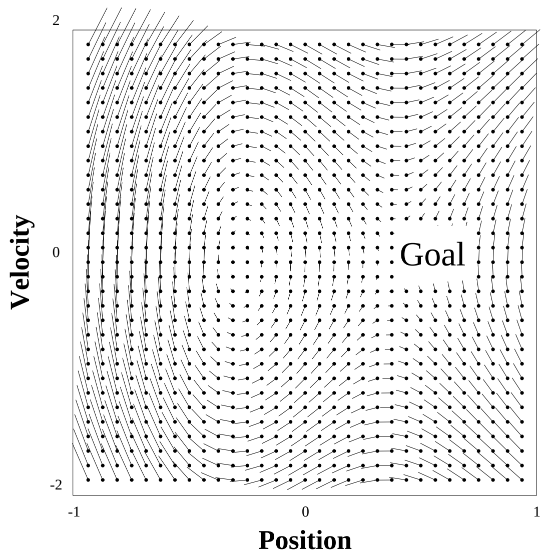


Figure 13: The state transition diagram for a puck that constantly thrusts right with maximum thrust.

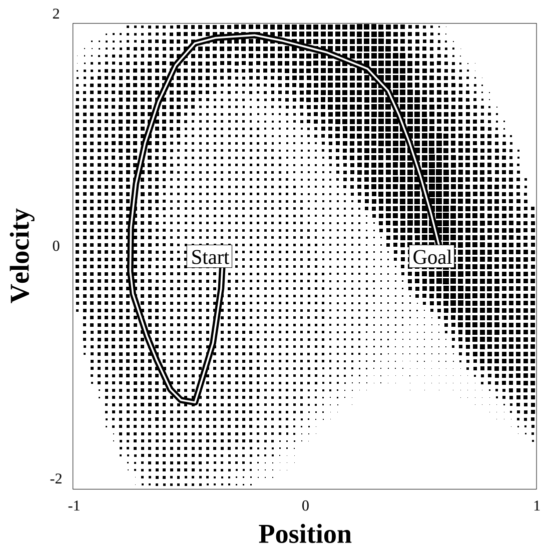


Figure 14: The minimum-time path from start to goal for the puck on the hill. The optimal value function is shown by the background dots. The shorter the time to goal, the larger the black dot. Notice the discontinuity at the escape velocity.

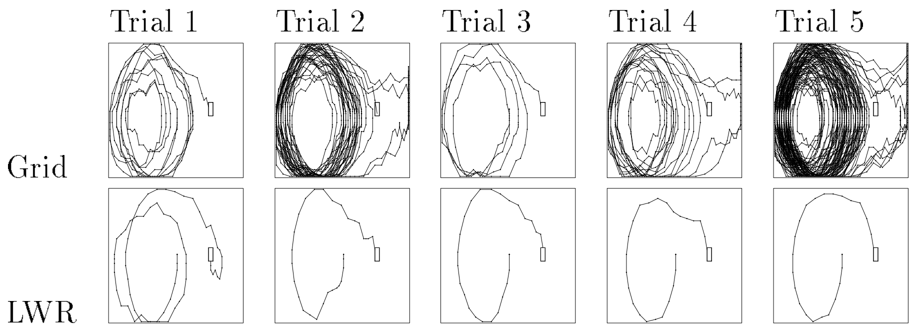


Figure 15: The first five trials for both implementations of the puck controller.

algorithm was chosen to be as efficient as possible (i.e., in terms of data needed for convergence) given that we were working with a fixed discretization. All transitions between cells experienced by the system were remembered in a discrete state transition model. A learning algorithm similar to Dyna (Sutton, 1990) was used with full value iteration carried out on the discrete model every time-step. Exploration was achieved by assuming any unvisited state had a future cost of zero. The action, which is one-dimensional, was discretized to five levels: $\{-4N, -2N, 0N, 2N, 4N\}$.

- **Implementation 2 (LWR): Lazy Forward Model.** The second implementation was the same as the first, except that transitions between cells were filled in by predictions from a locally weighted regression forward model $\mathbf{x}(t+1) = \widehat{\mathbf{f}}(\mathbf{x}(t), \mathbf{u}(t))$. Thus, unlike implementation 1, many discrete transitions that had not been physically experienced were stored in the transition table by extrapolation from the actual experiences. Also, the lazy model supported a higher resolution representation in areas where many experiences had been collected. The value function was represented by a table in both implementations.

The experimental domain is a simple one, but its empirical behavior demonstrates an important point. A lazy forward model in combination with value iteration can dramatically reduce the amount of actual data needed during learning. The graphs of the first five trajectories of the two experiments are shown in Figure 15. The steps per trial for both implementations are shown in Figure 16. The best possible number of steps per trial is 23. The implementation using the locally weighted regression forward model learns much faster in terms of trials than the implementation using the grid. The lazy model based implementation also requires approximately two orders of magnitude fewer steps in order to reach optimal performance. For example, after trial 150 the grid based implementation has executed 26297 total steps more than the optimal required when all trials are combined, while the lazy forward model based implementation has executed only 260 suboptimal steps.

Since we did not include any random noise in this simulation these numbers are deterministic. The spikes in Figure 16 are due to the severe nonlinearity of this problem, where small errors in the policy may lead to the puck failing to have enough energy to get to the goal. In this case the puck slides back down and must perform another "orbit" of the start point in state space before reaching the goal. The lack of random sensor or actuator noise

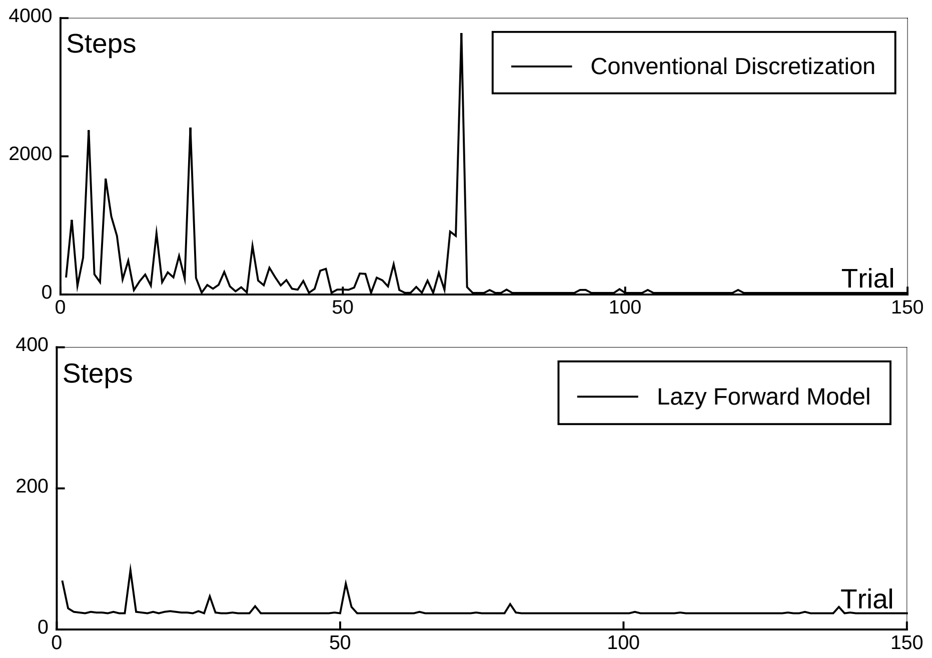


Figure 16: Top: Steps per trial for a grid based forward model. Bottom: Steps per trial for an LWR based forward model. Note the difference in vertical scales.

makes the problem unrealistically easy for both approaches. We expect the benefits of a lazy model over the standard grid model to carry over to the stochastic case.

The computational costs of this kind of control are considerable. Although it is not necessary to gather data from every part of the state space when generalization occurs with a model, the simple form of value iteration requires a multidimensional discretization for computing the value function. Several researchers are investigating methods for reducing the cost of value iteration when a model has been learned (e.g. (Moore, 1991b; Mahadevan, 1992; Atkeson, 1994)).

3.6.2 Exploration

The approach we have described does not explicitly explore. If the learned model contains serious errors, a part of state space that wrongly looks unrewarding will never be visited by the real system, so the model will never be updated. On the other hand, we do not want the system to explore every part of state space explicitly—the supposed advantage of lazy learning based function approximation is the ability to generalize parts of the model without explicitly performing an action. To resolve this dilemma, a number of useful exploration heuristics can be used, all based on the idea that it is worth exploring only where there is little confidence in the empirical model (Sutton, 1990; Kaelbling, 1993; Moore and Atkeson, 1993; Cohn et al., 1995).

4 Lazy Learning of Models: Pros and Cons

Lazy learning of models leads to new forms of autonomous control. The control algorithms explicitly perform empirical nonlinear modeling as well as simultaneously designing policies, without a strong commitment to a model structure or controller structure in advance. Parametric modeling approaches, such as polynomial regression, multi-layer sigmoidal neural networks, and projection pursuit regression, all make a strong commitment to a model structure, and new training data has a global effect on the learned function. Locally weighted learning only assumes local smoothness. This section discusses the strengths and weaknesses of a local and lazy modeling approach in the context of control. Stanfill and Waltz (1986) provide a similar discussion for lazy approaches to classification.

4.1 Benefits of Lazy Learning of Models

- **Automatic, empirical, local linear models.** Locally weighted linear regression returns a local linear map. It performs the job of an engineer who is trying to empirically linearize the system around a region of interest. It is not difficult for neural net representations to provide a local linear map too, but other approximators such as straightforward nearest neighbor or the original version of CMAC (Albus, 1981; Miller, 1989) are less reliable in their estimation of local gradients because predicted surfaces are not smooth. Additionally, if the input data distribution is not too non-uniform, it can be shown that the linearizations returned by locally weighted learning accomplish a low-bias estimate of the true gradient with fewer data points than required for a low-bias prediction of a query (Hastie and Loader, 1993).

- **Automatic confidence estimations.** Locally weighted regression can also be modified to return a confidence interval along with its prediction. This can be done heuristically with the local density of the data providing an uncertainty estimate (Moore, 1991a) or by making sensible statistical assumptions (Schaal and Atkeson, 1994b; Cohn et al., 1995). In either case, this has been shown empirically to dramatically reduce the amount of exploration needed when the uncertainty estimates guide the experiment design. The cost of estimating uncertainty with locally weighted methods is small. Nonlinear parametric representations such as multi-layer sigmoidal neural networks can also be adapted to return confidence intervals (MacKay, 1992; Pomerleau, 1994), but approximations are required, and the computational cost is larger. Worse, parametric models (e.g., global polynomial regression) that predict confidence statistically are typically assuming that the true world can be perfectly modeled by at least one set of parameter values. If this assumption is violated, then the confidence intervals are difficult to interpret.

- **Adding new data to a lazy model is cheap.** For a lazy model adding a new data point means simply inserting it into the data base.

- **One-shot learning.** Lazy models do not need to be repeatedly exposed to the same data to learn it. A consequence of this rapid learning is that errors are not repeated and can be eliminated much more quickly than approaches that incrementally update parameters. Nonlinear parametric models can be trained by 1) exposing the model to a new data point only once (e.g., (Jordan and Jacobs, 1990; Kuperstein, 1988)), or 2) by storing the data in a database and cycling through the training data repeatedly. In case 1, much more data must be collected, since the training effect of each data point is small. This leads to slower learning, since real robot movements take time, and to increased wear-and-tear on the robot or industrial process that is to be controlled. In case 2, a lazy learning approach has been adopted, and one must then evaluate the relative benefits of complex and simple local models.

- **Non-linear, yet no danger of local minima in function approximation.** Locally weighted regression can fit a wide range of complex non-linear functions, and finds the best fit directly, without requiring any gradient descent. There are no dangers of the model learner becoming stuck in a local optimum. In contrast, training nonlinear parametric models can get stuck in local minima.

  However, some of the control law design algorithms we have surveyed can become stuck (Moore, 1992; Jordan and Rumelhart, 1992). The inverse-model method can become stuck with non-monotonic or highly noisy systems. The shifting setpoint algorithm can become stuck in principle, although this has not yet occurred in practice.

- **Avoids interference.** Lazy modeling is insensitive to what task it is currently learning or if the data distribution changes. In contrast, nonlinear parametric models trained incrementally with gradient descent eventually forget old experiences and concentrate representational power on new experiences.

4.2 Drawbacks of Lazy Learning of Models

Here we consider the disadvantages of lazy learning that may be encountered under some circumstances, and we also point out promising directions for addressing them.

- **Lookup costs increase with the amount of training data.** Memory and computation costs increase with the amount of data. Memory costs increase linearly with the amount of data, and are not generally a problem. Any algorithm that avoids storing redundant data would greatly reduce the amount of memory needed, and one can also discard data, perhaps selected according to predictive usefulness, redundancy, or age (Atkeson et al., 1995).

  Computational costs are more serious. For a fixed amount of computation, a single processor can process a limited number of training data points. There are several solutions to this problem (Atkeson et al., 1995): The database can be structured so that the most relevant data points are accessed first, or so that close approximations to the output predicted by locally weighted regression can be obtained without explicitly visiting every point in the database. There are a surprisingly large number of algorithms available for doing this, mostly based on kd-trees (Preparata and Shamos, 1985; Omohundro, 1987; Moore, 1990; Grosse, 1989; Quinlan, 1993; Omohundro, 1991; Deng and Moore, 1995).

- **Is the curse of dimensionality a problem for lazy learning for control?** The curse of dimensionality is the exponential dependence of needed resources on dimensionality found in many learning and planning approaches. The methods we have discussed so far can handle a wide class of problems. On the other hand, it is well known that, without strong constraints on the class of functions being approximated, learning with many input dimensions will not successfully approximate a particular function over the entire space of potential inputs unless the data set is unrealistically large.

  This is an apparently serious problem for multivariate control using locally weighted learning, and raises the question as to why the examples given in this paper worked. Happily, it is actually quite difficult to think of useful tasks that *require* the system to have an accurate model over the entire input space (Albus, 1981). Indeed, for a robot of more than, say, eight degrees of freedom, it will not be possible for it to get into every significantly different configuration even once in its entire lifetime.

  Many tasks require high accuracy only in low-dimensional manifolds of input space or thin slices around those manifolds. In some cases these may be clumps around the desired goal value of stationary tasks. For example, in devil sticking the robot needs to gain highly accurate expertise only in the vicinity of stable juggling patterns. Another common task involves the system spending most of its life traveling along a number of important trajectories, "highways", through state space, in which case expertise need only be clustered in these regions. In general, the curse of dimensionality may not be dangerous for *tasks* whose solution lies in a low-dimensional manifold or a thin slice, even if the number of state variables and control inputs is several times larger.

  In any event we expect the performance of locally weighted regression to be as good as any other method as the dimensionality of the problem increases, as locally weighted learning can become global if necessary to emulate global models, and can become global or local in particular directions to emulate projection pursuit models (e.g., the distance function can be set to choose a projection direction, for example, but for multiple projection directions multiple distance functions must be used in additive locally weighted fits) (Friedman and Stuetzle, 1981). We expect locally weighted learning to degrade gracefully as the problem dimensionality increases.

- **Lazy learning depends on having good representations already selected.** Good representational choices (i.e., choices of the elements of the state and control vectors, etc.) can dramatically speed up learning or make learning possible at all. Feature selection and scaling algorithms are a crude form of choosing new representations (Atkeson et al., 1995). However, we have not solved the representation problem, and locally weighted learning and all other machine learning approaches depend on prior representational decisions.

5 Conclusions

This paper has explored methods for using lazy learning to learn task models for control, emphasizing how forward and inverse learned models can be used. The implementations all used lazy models. The last section discussed in more detail the pros and cons of lazy learning as the specific choice of model learner.

There is little doubt that these advances can be converted into general purpose software packages for the benefit of robotics and process control. But it should also be understood that we are still a considerable way from full autonomy. A human programmer has to decide what the state and action variables are for a problem, how the task should be specified, and what class of control task it is. The engineering of real-time systems, sensors and actuators is still required. A human must take responsibility for safety and supervision of the system. Thus, at this stage, if we are given a problem, the relative effectiveness of learning control, measured as the proportion of human effort eliminated, is heavily dependent on problem-specific issues.

Appendix A: Simple Linear Quadratic Regulator derivation

This appendix provides a simplified, self-contained introduction to LQR control for readers who wish to understand the ideas behind Equations 17 and 18. Assume a scalar state and action, and assume that desired state and action are zero ($x_d = u_d = 0$). Assume linear dynamics:

$$x_{k+1} = ax_k + bu_k \tag{25}$$

where a and b are constants. Define $V_k^*(x)$ to be the minimum possible sum of future costs, starting from state x, assuming we are at time-step k. Assume the system stops at time $k = N$, and the stopping cost is qx_N^2. For all other steps (i.e. $k < N$) the cost is $qx_k^2 + ru_k^2$.

$$V_k^*(x) = \sum_{j=k}^{N-1} \left(qx_j^2 + ru_j^2\right) + qx_N^2 \qquad \text{assuming } u_k, u_{k+1}, \ldots, u_{N-1} \text{ chosen optimally.} \tag{26}$$

$V_k^*(x)$ can be defined inductively:

$$\begin{aligned} V_N^*(x) &= qx_N^2 & (27)\\ V_k^*(x) &= \arg\min_{u_k} \left(qx_k^2 + ru_k^2 + V_{k+1}^*(x_{k+1})\right) & (28) \end{aligned}$$

by the principal of optimality, which says that your best bet for minimal costs is to minimize over your first step for the cost of that step plus the minimum possible costs of future steps. We will now prove by induction that $V_k^*(x)$ is a quadratic in x, with the quadratic coefficient dependent on k: $V_k^*(x) = p_k x^2$ for some $p_0, p_1, \ldots, p_N$.

- **Base case:** $p_N = q$ from Equation 27.
- **Inductive step:** Assume $V_{k+1}^*(x) = p_{k+1}x^2$; we'll prove $V_k^*(x) = p_k x^2$ for some p_k.

From here on, all that remains is algebra. We begin with Equation 28, in which we replace x_{k+1} with $ax_k + bu_k$ from Equation 25:

$$V_k^*(x) = \arg\min_{u_k} \left(qx_k^2 + ru_k^2 + V_{k+1}^*(ax_k + bu_k)\right) \tag{29}$$

Then we use the inductive assumption $V^*_{k+1}(x) = p_{k+1}x^2$

$$V^*_k(x) = \arg\min_{u_k} \left(qx_k^2 + ru_k^2 + p_{k+1}(ax_k + bu_k)^2 \right) \tag{30}$$

Next we simplify with three new variables, α, β, γ:

$$\begin{aligned} V^*_k(x) &= \arg\min_{u_k} \left(\alpha x_k^2 + 2\beta x_k u_k + \gamma u_k^2 \right) \qquad \text{where} && (31) \\ \alpha &= q + p_{k+1}a^2 && (32) \\ \beta &= p_{k+1}ab && (33) \\ \gamma &= r + p_{k+1}b^2 && (34) \end{aligned}$$

To minimize Equation 31 with respect to u we differentiate and set to zero the bracketed expression giving:

$$2\beta x + 2\gamma u^*_k = 0 \tag{35}$$

where u^*_k is the optimal action. Thus

$$u^*_k = -(\beta/\gamma)x_k \tag{36}$$

Since u^*_k minimizes Equation 31 we have

$$V^*_k(x) = \alpha x^2 + 2\beta x u^*_k + \gamma(u^*_k)^2 \tag{37}$$

So from Equation 36

$$V^*_k(x) = \alpha x^2 + 2\beta x(-\beta/\gamma)x + \gamma(-\beta/\gamma)^2 x^2 = \left(\alpha - 2\beta^2/\gamma + \beta^2/\gamma\right) x^2 = \left(\alpha - \beta^2/\gamma\right) x^2 \tag{38}$$

so that we have shown $V^*_k(x) = p_k x^2$ where

$$p_k = \left(\alpha - \beta^2/\gamma\right) \tag{39}$$

Inserting back the substitutions of Equations 32, 33, 34 into Equations 36 and 39:

$$\begin{aligned} u^*_k &= \left(\frac{-p_{k+1}ab}{r + p_{k+1}b^2} \right) x_k && (40) \\ V^*_k(x) &= p_k x^2 \text{where } p_k = q + a^2 p_{k+1}\left(1 - \tfrac{p_{k+1}b^2}{r + p_{k+1}b^2}\right) && (41) \end{aligned}$$

Assuming that there are $N - k$ steps remaining, to compute the cost-to-go from state x we set $p := q$ and then iterate the assignment $p := q + a^2 p(1 - \frac{pb^2}{r+pb^2})$ a total of $N - k$ times. As $N - k$ becomes large p converges to a constant value (not proven here). This gives the cost-to-go value function of px^2, assuming that the system will run forever.

6 Acknowledgments

Support for C. Atkeson and S. Schaal was provided by the ATR Human Information Processing Research Laboratories. Support for C. Atkeson was provided under Air Force Office of Scientific Research grant F49-6209410362, and by a National Science Foundation Presidential Young Investigator Award. Support for S. Schaal was provided by the German Scholarship Foundation and the Alexander von Humboldt Foundation. Support for A. Moore was provided by the U. K. Science and Engineering Research Council, NSF Research Initiation Award # IRI-9409912, and a Research Gift from the 3M Corporation.

References

Aha, D. W. and Salzberg, S. L. (1993). Learning to catch: Applying nearest neighbor algorithms to dynamic control tasks. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 363–368, Ft. Lauderdale, FL.

Albus, J. S. (1981). *Brains, Behaviour and Robotics.* BYTE Books, McGraw-Hill.

Atkeson, C. G. (1990). Using local models to control movement. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 316–323. Morgan Kaufmann, San Mateo, CA.

Atkeson, C. G. (1994). Using local trajectory optimizers to speed up global optimization in dynamic programming. In Hanson, S. J., Cowan, J. D., and Giles, C. L., 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. (1995). Locally weighted learning. Submitted to the *Artificial Intelligence Review* special issue on Lazy Learning.

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.

Barto, A. G., Sutton, R. S., and Watkins, C. J. C. H. (1990). Learning and Sequential Decision Making. In Gabriel, M. and Moore, J. W., editors, *Learning and Computational Neuroscience*, pages 539–602. MIT Press, Cambridge, MA.

Bellman, R. E. (1957). *Dynamic Programming.* Princeton University Press, Princeton, NJ.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation.* Prentice Hall.

Cannon, R. H. (1967). *Dynamics of Physical Systems.* McGraw-Hill.

Cohn, D. A., Ghahramani, Z., and Jordan, M. I. (1995). Active learning with statistical models. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7.* MIT Press.

Connell, M. E. and Utgoff, P. E. (1987). Learning to control a dynamic physical system. In *Sixth National Conference on Artificial Intelligence*, pages 456–460, Seattle, WA. Morgan Kaufmann, San Mateo, CA.

Conte, S. D. and De Boor, C. (1980). *Elementary Numerical Analysis.* McGraw Hill.

Deng, K. and Moore, A. W. (1995). Multiresolution Instance-based Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence, Montreal, 1995.* Morgan Kaufmann.

Farmer, J. D. and Sidorowich, J. J. (1987). Predicting chaotic time series. *Physical Review Letters*, 59(8):845–848.

Farmer, J. D. and Sidorowich, J. J. (1988a). Exploiting chaos to predict the future and reduce noise. In Lee, Y. C., editor, *Evolution, Learning, and Cognition*, pages 277–??? World Scientific Press, NJ. also available as Technical Report LA-UR-88-901, Los Alamos National Laboratory, Los Alamos, New Mexico.

Farmer, J. D. and Sidorowich, J. J. (1988b). Predicting chaotic dynamics. In Kelso, J. A. S., Mandell, A. J., and Schlesinger, M. F., editors, *Dynamic Patterns in Complex Systems*, pages 265–292. World Scientific, NJ.

Friedman, J. H. and Stuetzle, W. (1981). Projection Pursuit Regression. *Journal of the American Statistical Association*, 76(376):817–823.

Gorinevsky, D. and Connolly, T. H. (1994). Comparison of some neural network and scattered data approximations: The inverse manipulator kinematics example. *Neural Computation*, 6:521–542.

Grosse, E. (1989). LOESS: Multivariate Smoothing by Moving Least Squares. In C. K. Chul, L. L. S. and Ward, J. D., editors, *Approximation Theory VI*. Academic Press.

Hastie, T. and Loader, C. (1993). Local regression: Automatic kernel carpentry. *Statistical Science*, 8(2):120–143.

Huang, P. S. (1996). *Planning For Dynamic Motions Using A Search Tree*. MS thesis, University of Toronto, Graduate Department of Computer Science. http://www.dgp.utoronto.ca/people/psh/home.html.

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

Jordan, M. I. and Rumelhart, D. E. (1992). Forward Models: Supervised Learning with a Distal Teacher. *Cognitive Science*, 16:307–354.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT Press, Cambridge, MA.

Kuperstein, M. (1988). Neural Model of Adaptive Hand-Eye Coordination for Single Postures. *Science*, 239:1308–3111.

MacKay, D. J. C. (1992). Bayesian Model Comparison and Backprop Nets. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 839–846. Morgan Kaufmann, San Mateo, CA.

Mahadevan, S. (1992). Enhancing Transfer in Reinforcement Learning by Building Stochastic Models of Robot Actions. In *Machine Learning: Proceedings of the Ninth International Workshop*. Morgan Kaufmann.

Maron, O. and Moore, A. (1994). Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. In *Advances in Neural Information Processing Systems 6*, pages 59–66. Morgan Kaufmann, San Mateo, CA.

McCallum, R. A. (1995). Instance-based utile distinctions for reinforcement learning with hidden state. In Prieditis and Russell (1995), pages 387–395.

Mel, B. W. (1989). MURPHY: A Connectionist Approach to Vision-Based Robot Motion Planning. Technical Report CCSR-89-17A, University of Illinois at Urbana-Champaign.

Miller, W. T. (1989). Real-Time Application of Neural Networks for Sensor-Based Control of Robots with Vision. *IEEE Transactions on Systems, Man and Cybernetics*, 19(4):825–831.

Moore, A. W. (1990). Acquisition of Dynamic Control Knowledge for a Robotic Manipulator. In *Proceedings of the 7th International Conference on Machine Learning*. Morgan Kaufmann.

Moore, A. W. (1991a). Knowledge of Knowledge and Intelligent Experimentation for Learning Control. In *Proceedings of the 1991 Seattle International Joint Conference on Neural Networks*.

Moore, A. W. (1991b). Variable Resolution Dynamic Programming: Efficiently Learning Action Maps in Multivariate Real-valued State-spaces. In Birnbaum, L. and Collins, G., editors, *Machine Learning: Proceedings of the Eighth International Workshop*. Morgan Kaufmann.

Moore, A. W. (1992). Fast, Robust Adaptive Control by Learning only Forward Models. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 571–578. Morgan Kaufmann, San Mateo, CA.

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., Hill, D. J., and Johnson, M. P. (1992). An Empirical Investigation of Brute Force to Choose Features, Smoothers and Function Approximators. In Hanson, S., Judd, S., and Petsche, T., editors, *Computational Learning Theory and Natural Learning Systems, Volume 3*. MIT Press.

Moore, A. W. and Lee, M. S. (1994). Efficient Algorithms for Minimizing Cross Validation Error. In Cohen, W. W. and Hirsh, H., editors, *Proceedings of the 11th International Conference on Machine Learning*. Morgan Kaufmann.

Moore, A. W. and Schneider, J. (1995). Memory-Based Stochastic Optimization. In *Proceedings of Neural Information Processing Systems Conference*.

Omohundro, S. M. (1987). Efficient Algorithms with Neural Network Behaviour. *Journal of Complex Systems*, 1(2):273–347.

Omohundro, S. M. (1991). Bumptrees for Efficient Function, Constraint, and Classification Learning. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 693–699. Morgan Kaufmann, San Mateo, CA.

Ortega, J. M. and Rheinboldt, W. C. (1970). *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press.

Peng, J. (1995). Efficient memory-based dynamic programming. In Prieditis and Russell (1995), pages 438–446.

Peng, J. and Williams, R. J. (1993). Efficient Learning and Planning Within the Dyna Framework. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. MIT Press.

Pomerleau, D. (1994). Reliability estimation for neural network based autonomous driving. *Robotics and Autonomous Systems*, 12.

Preparata, F. P. and Shamos, M. (1985). *Computational Geometry*. Springer-Verlag.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1988). *Numerical Recipes in C*. Cambridge University Press, New York, NY.

Prieditis, A. and Russell, S., editors (1995). *Twelfth International Conference on Machine Learning*, Tahoe City, CA. Morgan Kaufmann, San Mateo, CA.

Quinlan, J. R. (1993). Combining Instance-Based and Model-Based Learning. In *Machine Learning: Proceedings of the Tenth International Conference*.

Saitta, L., editor (1996). *Thirteenth International Conference on Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Schaal, S. and Atkeson, C. (1994a). Robot Juggling: An Implementation of Memory-based Learning. *Control Systems Magazine*, 14(1):57–71.

Schaal, S. and Atkeson, C. G. (1994b). Assessing the Quality of Local Linear Models. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 160–167. Morgan Kaufmann.

Stanfill, C. and Waltz, D. (1986). Towards Memory-Based Reasoning. *Communications of the ACM*, 29(12):1213–1228.

Stengel, R. F. (1986). *Stochastic Optimal Control*. John Wiley and Sons.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44.

Sutton, R. S. (1990). Integrated Architecture for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Proceedings of the 7th International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann.

Tadepalli, P. and Ok, D. (1996). Scaling up average reward reinforcement learning by approximating the domain models and the value function. In Saitta (1996). http://www.cs.orst.edu:80/∼tadepall/research/publications.html.

Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems (NIPS) 8*. http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/thrun/publications.html.

Thrun, S. and O'Sullivan, J. (1996). Discovering structure in multiple learning tasks: The TC algorithm. In Saitta (1996). http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/thrun/publications.html.

van der Smagt, P., Groen, F., and van het Groenewoud, F. (1994). The locally linear nested network for robot manipulation. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 2787–2792. ftp://ftp.fwi.uva.nl/pub/computer-systems/aut-sys/reports/SmaGroGro94b.ps.gz.

Watkins, C. J. C. H. (1989). Learning from Delayed Rewards. PhD. Thesis, King's College, University of Cambridge.

Zografski, Z. (1989). *Neuromorphic, Algorithmic, and Logical Models for the Automatic Synthesis of Robot Action*. PhD dissertation, University of Ljubljana, Ljubljana, Slovenia, Yugoslavia.

Zografski, Z. (1991). New methods of machine learning for the construction of integrated neuromorphic and associative-memory knowledge bases. In Zajc, B. and Solina, F., editors, *Proceedings, 6th Mediterranean Electrotechnical Conference*, volume II, pages 1150–1153, Ljubljana, Slovenia, Yugoslavia. IEEE catalog number 91CH2964-5.

Zografski, Z. (1992). Geometric and neuromorphic learning for nonlinear modeling, control and forecasting. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 158–163, Glasgow, Scotland. IEEE catalog number 92CH3110-4.

Zografski, Z. and Durrani, T. (1995). Comparing predictions from neural networks and memory-based learning. In *Proceedings, ICANN '95/NEURONIMES '95: International Conference on Artificial Neural Networks*, pages 221–226, Paris, France.