REINFORCEMENT LEARNING WITH HIGH-DIMENSIONAL, CONTINUOUS ACTIONS

Leemon C. Baird III and A. Harry Klopf

Technical Report WL-TR-93-1147

Wright Laboratory

Wright-Patterson Air Force Base, OH 45433-7301

Address:	WL/AAAT, Bldg 635 2185 Avionics Circle Wright-Patterson Air Force Base, OH 45433-7301 U.S.A.
Telephone:	(513) 255-7644
Fax:	(513) 476-4302
E-mail:	bairdlc@wL.wpafb.af.mil

ABSTRACT

Many reinforcement learning systems, such as Q-learning (Watkins, 1989), or advantage updating (Baird, 1993), require that a function $f(\mathbf{x},\mathbf{u})$ be learned, and that the value of $\arg\max_{\mathbf{u}} f(\mathbf{x},\mathbf{u})$ be calculated quickly for any given $\mathbf{x}$. The function f could be learned by a function approximation system such as a multilayer perceptron, but the maximum of f for a given $\mathbf{x}$ cannot be found analytically and is difficult to approximate numerically for high-dimensional $\mathbf{u}$ vectors. A new method is proposed, *wire fitting*, in which a function approximation system is used to learn a set of functions called *control wires*, and the function f is found by fitting a surface to the control wires. Wire fitting has the following four properties: (1) any continuous f function can be represented to any desired accuracy given sufficient parameters; (2) the function $f(\mathbf{x},\mathbf{u})$ can be evaluated quickly; (3) $\arg\max_{\mathbf{u}} f(\mathbf{x},\mathbf{u})$ can be found exactly in constant time after evaluating $f(\mathbf{x},\mathbf{u})$; (4) wire fitting can incorporate any general function approximation system. These four properties are discussed and it is shown how wire fitting can be combined with a memory-based learning system and *Q*-learning to control an inverted-pendulum system.

Key words: reinforcement learning, wire fitting, maximization, *Q*-learning, advantage updating

TABLE OF CONTENTS

List of Figures iv

Acknowledgments v

1. Introduction 1

2. Maximization of a Function 2

3. Maximization of a Cross Section 4

4. Memory-Based Learning 9

5. Simulation Results 11

6. Conclusion 12

7. References 13

LIST OF FIGURES

Figure 1. Method for storing a function $f(u)$ such that the maximum can be found quickly 2

Figure 2. The wire fitting architecture 4

Figure 3. An example of a function $f(\mathbf{x},\mathbf{u})$ whose shape is determined by three wires 7

ACKNOWLEDGMENTS

This research was supported under Task 2312R1 by the Life and Environmental Sciences Directorate of the United States Air Force Office of Scientific Research. The author gratefully acknowledges the contributions of Mance Harmon, Jim Morgan, Gábor Bartha, Scott Weaver, and Tommi Jaakkola.

1. INTRODUCTION

A number of reinforcement learning systems have been proposed recently, such as the associative control process (ACP) network (Klopf, Morgan, and Weaver, 1993a, 1993b, Baird and Klopf, 1993a, 1993b), ADHDP (Werbos, 1989), Dyna (Sutton, 1990), other systems described in Barto and Bradtke (1991) based on *Q-learning* (Watkins, 1989, Watkins and Dayan, 1992), and systems based on advantage updating (Baird 1993). These systems learn to be optimal controllers of nonlinear plants, typically requiring that a function $f(\mathbf{x}, \mathbf{u})$ be learned. They also require that the value of $\arg\max_{\mathbf{u}} f(\mathbf{x}, \mathbf{u})$ be calculated repeatedly for various values of $\mathbf{x}$, both during learning, and when using the system as a controller. If the state $\mathbf{x}$ and action $\mathbf{u}$ are discretized, then the function can be represented as a finite lookup table. If the state $\mathbf{x}$ is a real-valued vector, then the function can be represented using standard function-representation techniques such as multilayer perceptrons, radial basis function networks, and memory-based learning and interpolation systems (Atkeson, 1990). However, if the action $\mathbf{u}$ is also a real-valued vector, then finding the maximum is extremely difficult with most function approximation systems. Although Tesauro's TD-Gammon program (Tesauro, 1990, 1992) demonstrates that some difficult problems can be solved using discrete values for $\mathbf{u}$, most practical problems require real-valued vectors. The optimization algorithm described in Baird (1992) can approximate the maximum for optimal control problems (or the saddle point for differential games), but there may be errors in the maximization during learning. Systems using the stochastic real-valued unit (Gullapalli, 1990, 1991) or the Analog Learning Element (Millington, 1991) can learn real-valued actions without maximizing learned functions, but they require the use of a particular exploration scheme. It is desirable for a system to be able to learn under any exploration scheme that tries all actions in all states sufficiently often. *Q*-learning and advantage updating, for example, have this property. Also, it would be useful if the power of any general function approximation system could be harnessed to learn the function, while still allowing the maximum of the function to be found quickly and exactly. A method is proposed, *wire fitting*, that has these desirable properties.

2. MAXIMIZATION OF A FUNCTION

First consider the simpler problem of learning a function $f(u)$ such that it is possible to quickly find the maximum of the function. Figure 1 shows one approach to solving this problem.

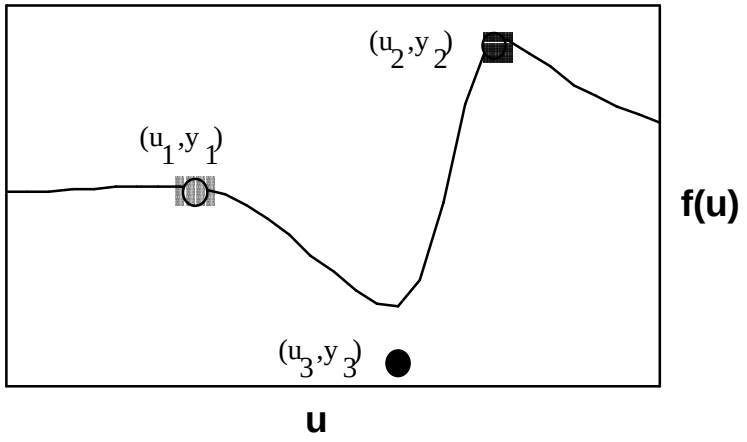

Figure 1. Method for storing a function $f(u)$ such that the maximum can be found quickly.

The shape of the function is determined by three control points (circles). Six parameters u_1,u_2,u_3,y_1,y_2,y_3 are initialized to arbitrary values. As training samples are observed, the six parameters are adjusted so that $f(u)$ (dotted line) is a good fit to the training data. The value of $f(u)$ at point u is defined as a weighted average of the three y_i values, weighted by distance between u and u_i, and also by the distance between y_i and y_{max}. This ensures that the maximum of $f(u)$ always occurs at one of the control points, (u_i,y_i).

The shape of the function $f(u)$ is controlled by six parameters which specify the location of three control points. The function $f(u)$ is defined as:

$$f(u) = \frac{\sum_i y_i \left[|u - u_i| + \max_k y_k - y_i \right]^{-1}}{\sum_i \left[|u - u_i| + \max_k y_k - y_i \right]^{-1}} \qquad (1)$$

The function is defined by a weighted-nearest-neighbor interpolation of the three control points. If equation (1) is undefined for a given value of u, then $f(u)$ is defined to be $\max_i y_i$ for that value of u. The function may not go through every control point, but it is guaranteed to go through the highest point. Also, the function is guaranteed never to go above the highest point or below the lowest point. Therefore, the maximum of the function is guaranteed to be located at the u_i, which has the same subscript as the maximum y_i value.

This function approximation system resembles a memory based learning system, but is different. In a memory based learning system, a set of training data is stored and interpolated to give the function $f(u)$. In the system described here, the control points are initialized to arbitrary values. Then, as training data is observed, the control points shift until $f(u)$ approximates the training data. For example, if all of the training data lies on the curve shown in Figure 1, then a gradient-descent learning algorithm will learn to place the three control points as shown in Figure 1. The control point (u_3,y_3), therefore, learns to be much lower than any of the training data. Equation (1) might not be a good algorithm for interpolating raw training data, but it may be useful for learning if the control points shift during learning. The maximum of the curve f can be found in even less time than it takes to evaluate $f(u)$ for an arbitrary u, because the maximum can be found without using equation (1).

There may be uses for a system that can learn $f(u)$ and find the maximum. It is more useful, however, to have a system that can learn $f(x,u)$ and can find the u that maximizes the function for any given x. This can be done using the same method shown above, but with the parameters u_i and y_i replaced with functions $u_i(x)$ and $y_i(x)$. In this case, the control points become control wires in a higher-dimensional space, and the function is a surface fitted to those wires.

3. MAXIMIZATION OF A CROSS SECTION

Wire fitting is a function approximation method designed to facilitate finding the maximum of the function $f(\mathbf{x}, \mathbf{u})$ for any given $\mathbf{x}$. When using wire fitting, the function $f(\mathbf{x}, \mathbf{u})$ is evaluated for a given $\mathbf{x}$ and $\mathbf{u}$ as shown in Figure 2.

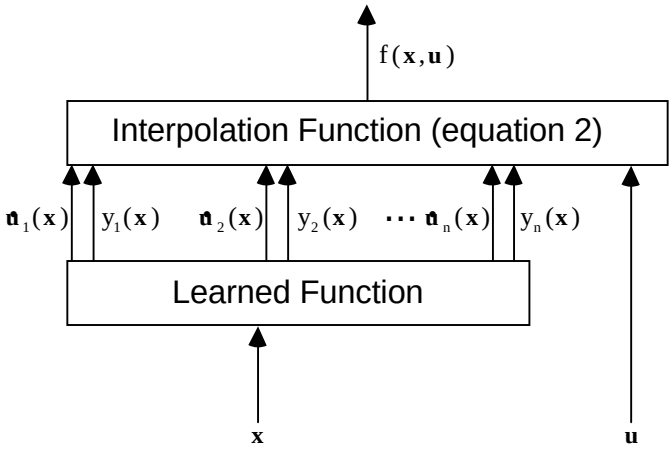

Figure 2. The wire fitting architecture.

> A function approximation system learns the function in the lower block. Given the state $\mathbf{x}$, this generates a set of control points. The interpolating function then fits a function to the set of control points and calculates $f(\mathbf{x},\mathbf{u})$, in the same manner as in Figure 1.

Any general function approximation system can be used to learn the function marked "learned function" in Figure 2. This function generates a set of control points based upon the value of $\mathbf{x}$. A function is then fitted to the set of control points, and the value of f is then calculated from $\mathbf{u}$ in the manner illustrated in the previous section. Since there is a set of control points for every possible $\mathbf{x}$, the control points are actually control curves, or wires, in a higher-dimensional space. Thus, the function is actually being fitted to a set of wires rather than to a set of points. The action $\mathbf{u}$ and the functions $\hat{\mathbf{u}}_i$ are all vectors with the same number of elements. The state $\mathbf{x}$ is also a vector, possibly with a different number of elements. The function f and the functions y_i are all scalars, and f is a weighted average of the set of y_i. In a reinforcement learning system, the function $f(\mathbf{x}, \mathbf{u})$ typically represents the utility of performing action $\mathbf{u}$ in state $\mathbf{x}$, so the $\mathbf{u}$ that maximizes $f(\mathbf{x},\mathbf{u})$ is the optimal action to perform in state $\mathbf{x}$. The lower box in Figure 2 can be any function approximation system, such as a multilayer perceptron trained by backpropagation. Its only input is the state $\mathbf{x}$. Its output is a set of vector pairs $(\hat{\mathbf{u}}_i, y_i)$, which control the shape of the function in state $\mathbf{x}$. Equation (2) is a continuous, smooth function of its inputs, so it is possible to backpropagate errors in f back through equation (2) to update weights in the learning system:

$$f(\mathbf{x},\mathbf{u}) = \lim_{\varepsilon \to 0^+} \frac{\sum_i y_i(\mathbf{x}) \left[\left\| \mathbf{u} - \hat{\mathbf{u}}_i(\mathbf{x}) \right\|^2 + c_i (y_{max}(\mathbf{x}) - y_i(\mathbf{x})) + \varepsilon \right]^{-1}}{\sum_i \left[\left\| \mathbf{u} - \hat{\mathbf{u}}_i(\mathbf{x}) \right\|^2 + c_i (y_{max}(\mathbf{x}) - y_i(\mathbf{x})) + \varepsilon \right]^{-1}} \quad (2)$$

For a given state, the set of vector pairs ($\hat{\mathbf{u}}_i$, y_i) are interpolated to give $f(\mathbf{x},\mathbf{u})$. The value y_{max} is simply the maximum of the y_i values. Equation (2) defines $f(\mathbf{x},\mathbf{u})$ for a particular $\mathbf{u}$ to be a weighted average of y_i values. If $\mathbf{u}$ is near a particular $\hat{\mathbf{u}}_i$, then the corresponding y_i is given more weight. The nonnegative constant parameters c_i determine the amount of smoothing. If all c_i =0, then the interpolation "honors the data", and $f = y_i$ when $\mathbf{u} = \hat{\mathbf{u}}_i$. If the c_i values are positive, the interpolated function is smoother, and f may not be exactly equal to y_i, even when $\mathbf{u} = \hat{\mathbf{u}}_i$.. The constants c_i can be chosen a priori, or they can be learned. As will be shown below, if the learned function is trained with a memory-based learning method, then the values for c_i can be chosen arbitrarily, with no effect on learning or performance. The limit in equation (2) is merely for mathematical completeness. It ensures that the function is defined when $\mathbf{u} = \hat{\mathbf{u}}_i$. The equation can be written without the limit and ε, if it is stated that $f(\mathbf{x},\mathbf{u}) = y_i$ whenever the coefficient of y_i in the summation would be undefined.

The control points ($\hat{\mathbf{u}}_i$, y_i) serve to shape the function in a given state. Each control point plays a role analogous to a knot for a spline or a data point for an interpolation function. It is also analogous to the parameters associated with one radial basis function in a radial basis function network. In each case, the parameters have a local effect on the shape of the function. However, Equation 2 has one property that distinguishes it from other interpolation algorithms. No matter what values the vector pairs have, it is always the case that:

$$\max_{\mathbf{u}} f(\mathbf{x},\mathbf{u}) = \max_i y_i(\mathbf{x}) \equiv y_{max}(\mathbf{x}) \quad (3)$$

This is easily proved. First, consider a value of $\mathbf{u}$ not equal to any $\hat{\mathbf{u}}_i$. In this case, the expression in Equation 1 is defined for ε=0. f is then a weighted average of the y_i, with each weight between zero and one and the sum of the weights equal to one. A weighted average of several numbers cannot exceed the largest number, so f is less than or equal to the maximum y_i, which is y_{max} . Next, consider the case in which $\mathbf{u}$ is equal to $\hat{\mathbf{u}}_{max}$, where $\hat{\mathbf{u}}_{max}$, is the $\hat{\mathbf{u}}_i$ with the same subscript as y_{max} . In this case, as ε goes to zero, the sum in the numerator comes to be dominated by the term containing $\hat{\mathbf{u}}_{max}$ and y_{max} , so in the limit $f = y_{max}$. Lastly, consider the case in which $\mathbf{u} = \mathbf{u}_i \_ \hat{\mathbf{u}}_{max}$. By a similar argument, if c_i=0, then $f = y_i \_ y_{max}$. If $c_i \_ 0$, then f is simply a

weighted sum of y_i, so $f \_ y_{max}$. Thus, when $\mathbf{u} = \hat{\mathbf{u}}_{max}$, $f = y_{max}$, and for every $\mathbf{u} \_ \hat{\mathbf{u}}_{max}$, $f \_ y_{max}$. Therefore, Equation 3 is true.

Given this method for representing a function *f*, it is possible to implement a reinforcement learning system that learns from any sequence of actions. Any function approximation system can be used as the lower box in Figure 2. The system in Figure 2 can be used to quickly calculate the *f* value for a given state-action pair, $f(\mathbf{x}, \mathbf{u})$, or the optimal action in a state, $\hat{\mathbf{u}}_{max}(\mathbf{x})$, or the maximum *f* value for a state, $y_{max}(\mathbf{x})$. If action **u** is performed in state **x**, then $f(\mathbf{x}, \mathbf{u})$ can be calculated immediately. On the next time step (or several time steps later for multistep learning), an improved estimate can be calculated for $f(\mathbf{x}, \mathbf{u})$ by the reinforcement learning algorithm, using the value of the new states and the reinforcement received. This can be used to calculate an error in $f(\mathbf{x}, \mathbf{u})$. If the learning system is gradient-based, then the error can be propagated back through Equation 2 and through the learned function, so that $f(\mathbf{x}, \mathbf{u})$ moves toward the improved estimate for $f(\mathbf{x}, \mathbf{u})$. Thus, this method for representing *f* is flexible, and can be incorporated in a variety of reinforcement learning systems.

This method for representing the function $f(\mathbf{x}, \mathbf{u})$ can be represented graphically, as shown in Figure 3.

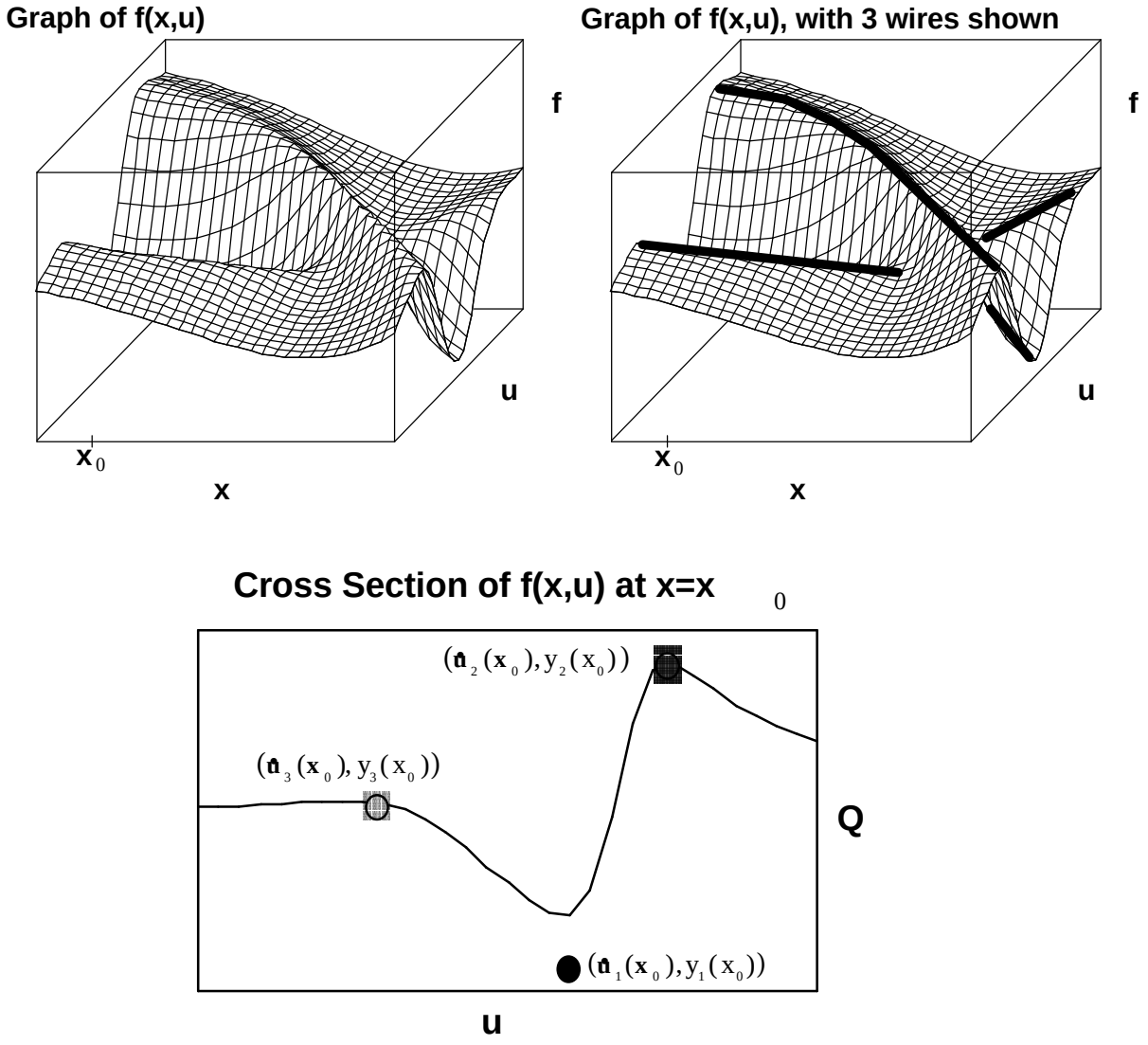

Figure 3. An example of a function $f(\mathbf{x},\mathbf{u})$ whose shape is determined by three wires. In any given state, such as $\mathbf{x}_0$, the wires intersect the plane of that state at three points. These three points are the control points that determine the shape of the function for that value of $\mathbf{x}$. The shape of the function in that plane is determined by the location of the three wires, and the function is guaranteed to pass through the point $(\hat{\mathbf{u}}_{\max}(\mathbf{x}_0),\ y_{\max}(\mathbf{x}_0))$, which in this example is the point $(\hat{\mathbf{u}}_2(\mathbf{x}_0),\ y_2(\mathbf{x}_0))$.

The upper graphs in Figure 2 show an example of a function $f(\mathbf{x},\mathbf{u})$, where $\mathbf{x}$ and $\mathbf{u}$ are scalars. The graph on the left is the f function itself, while the one on the right shows three control wires superimposed on the picture. The lower graph is a cross section of the function, taken at state $\mathbf{x}_0$. The set of all points of the form $(\mathbf{x},\ \hat{\mathbf{u}}_i(\mathbf{x}),\ y_i(\mathbf{x}))$ forms the ith wire in 3-D space. The shape of the surface is then determined by the shape and location of the control wires. The shape of the function in this example is determined by three wires: A high, curved wire

(dark gray), a medium, curved wire (light gray), and a low, straight wire (black). Although the surface does not touch the wires at every point, it is drawn toward them, and so consists of two intersecting ridges with a valley between them. Where the ridges intersect, the surface rises to the highest wire. In this example each wire has a constant height but, in general, a wire could have a varying height. The lower picture in Figure 2 shows a cross section of the graph on the right for a particular state, $\mathbf{x}_0$. Each wire intersects the plane of $\mathbf{x}_0$ at a point, so the three wires define three control points. The learning system learns the location of each control point in each state. The surface is defined by Equation 2, which ensures that the highest point on the surface will lie on one of the control points within the cross section at any given state.

4. MEMORY-BASED LEARNING

The method presented here for representing a function can be used with a variety of function representation systems. It is clear how it could be used with a gradient-based function approximation system. The error in f can be propagated back through equation (2) (which is differentiable), to change the weights in the learning system. This causes the control wires to shift until the surface has the appropriate shape to minimize the mean squared error in f. It may be less clear how it could be used with a memory-based function approximation system, so we elaborate upon that alternative in this section.

For a memory-based function approximation system, the stored information will comprise a set of triplets $(\mathbf{x}_t, \mathbf{u}_t, E_t)$. If action u_t is performed in state x_t at time t, the system will output $f(\mathbf{x}_t,\mathbf{u}_t)$. The reinforcement learning algorithm then calculates an estimate E_t of what $f(\mathbf{x}_t,\mathbf{u}_t)$ should have been, based on the results of performing action $\mathbf{u}_t$ in state $\mathbf{x}_t$. Once this estimate has been calculated, the triplet $(\mathbf{x}_t, \mathbf{u}_t, E_t)$ can be stored. The functions $\hat{\mathbf{u}}_i(\mathbf{x})$ and $y_i(\mathbf{x})$ can be calculated from the set of stored memories. If old memories are eventually lost, perhaps because of a finite-sized memory set, then the $\hat{\mathbf{u}}_i(\mathbf{x})$ and $y_i(\mathbf{x})$ functions would be expected to improve with experience, yielding memory-based learning.

Memory-based learning has an advantage relative to gradient learning systems when used with wire fitting. It is possible to calculate and store each triplet without calculating $f(\mathbf{x}, \mathbf{u})$. In a gradient learning system, the output of the system must be calculated so that an error can be found to drive learning. In a memory-based system, examples of inputs and desired outputs are simply stored, and the actual outputs $f(\mathbf{x}, \mathbf{u})$ need not be calculated. Thus, for the particular case of a memory-based learning system, Equation 2 need never be evaluated. This not only saves calculation time, but also simplifies the system because the constants c_i do not have to be chosen or learned.

An important question for a memory-based system is that of how the functions $\hat{\mathbf{u}}_i(\mathbf{x})$ and $y_i(\mathbf{x})$ can be calculated from the set of stored data. In Figure 3, this would correspond to the question of how several wires can be created that will generate a surface that is a reasonable approximation to a set of data points scattered throughout the cube. If there are n functions $\hat{\mathbf{u}}_i(\mathbf{x})$ and $y_i(\mathbf{x})$, then every state will intersect n of the wires. One possible solution is presented next.

For a given state $\mathbf{x}$, the functions $\hat{\mathbf{u}}_i(\mathbf{x})$ and $y_i(\mathbf{x})$ are defined by Equations 4 through 10. If there are n wires, then there will be a wire associated with each of the n data points nearest to state $\mathbf{x}$ (Euclidean distance). The ith wire will not necessarily go through the ith data point, but $\hat{\mathbf{u}}_i(\mathbf{x})$ will typically be fairly close to the $\mathbf{u}$ component

of the associated data point. In the equations that follow, t is an index that ranges over all stored data points. The index i ranges over those data points that are associated with wires. States and actions are vectors. The subscript k represents the kth element of an action vector, and the subscript L represents the Lth element of a state vector:

(4)

$$\bar{y}_i = \frac{\sum_j y_j d_{ij}}{\sum_j d_{ij}} \quad (5) \qquad \bar{\mathbf{u}}_{ik} = \frac{\sum_j \mathbf{u}_{jk} d_{ij}}{\sum_j d_{ij}} \quad (6)$$

$$\overline{yu}_{ik} = \frac{\sum_j y_j u_{jk} d_{ij}}{\sum_j d_{ij}} \quad (7) \qquad m_{ik} = \frac{\overline{y\mathbf{u}}_{ik} - \bar{y}_i \bar{\mathbf{u}}_{ik}}{\bar{\mathbf{u}}_{ik} - (\overline{y\mathbf{u}}_{ik})^2} \quad (8)$$

$$\mathbf{u}_{ik}(\mathbf{x}) = \bar{\mathbf{u}}_{ik} + \alpha\, m_{ik} \quad (9) \qquad y_i(\mathbf{x}) = \bar{y}_i + \alpha \sum_k m_{ik}^2 \quad (10)$$

Each of the n data points ($\mathbf{x}_t$, $\mathbf{u}_t$, E_t) is projected into the plane of the current state $\mathbf{x}$, to give a projected point ($\mathbf{x}$, $\mathbf{u}_t$, E_t). These points are locations where estimates of the value of f should be most reliable. All of the data points (not just the n closest) have an effect on the wire associated with each projected point. The effect of the tth data point on the ith wire is inversely proportional to its distance from the projection, and is given by Equation 4. Equations 5 through 8 perform weighted linear regression. This gives an estimate of the direction one should move from the projected point to maximize the function $f(\mathbf{x}, \mathbf{u})$. Equations 9 and 10 place the location of the ith wire ($\mathbf{u}_i(\mathbf{x})$, $y_i(\mathbf{x})$) near the projected point, slightly uphill in the direction found by weighted linear regression. Thus, each wire comprises a local estimate of an action that would maximize f, and an estimate of the f value for that action. The linear regression is done separately for each dimension. For high-dimensional action vectors, this is less computationally intensive than doing multidimensional linear regression. The results are the same when the stored values ($\mathbf{x}_t$, $\mathbf{u}_t$) are evenly distributed (have zero covariance). If the state-action space is explored unevenly, then the stored values may not be evenly distributed, and it may be necessary to perform an affine transformation on the data to give zero covariance.

5. SIMULATION RESULTS

The wire fitting approach was tested by incorporating it into a reinforcement learning system used to control an inverted pendulum hinged to a cart moving on an infinite track. *Q*-learning was used for the reinforcement learning algorithm, and a memory-based learning system was used as the function approximation system. The equations for the cart-pole system are:

$$(m_c + m_p)\ddot{x} + m_p l\ddot{\theta}\cos\theta - m_p l\dot{\theta}^2 \sin\theta = f - \mu_c \operatorname{sgn}\dot{x} \tag{11}$$

$$\tfrac{4}{3} m_p l^2 \ddot{\theta} + m_p l\ddot{x}\cos\theta - m_p g l \sin\theta = -\mu_p \dot{\theta} \tag{12}$$

where:

x	=		position of the cart (m)
ϕ	=		pole angle (rad)
g	=	9.8 m/s2	acceleration due to gravity
m_c	=	1.0 kg	mass of the cart
m_p	=	0.1 kg	mass of the pole
l	=	0.5 m	pole half-length
μ_c	=	0.0005 N	friction between cart and track
μ_p	=	0.000002 N_m_s	friction between pole and cart
$\|f\|$	_	10.0 N	force applied to cart

The cart-pole system was simulated by Euler integration at 50 Hz. Reinforcement was proportional to the pole angle squared, with an additional negative reinforcement when the pole exceeded 12 degrees from vertical. The learning system was allowed to learn for only 60 seconds of simulated time, during which a random action in the range [-10,10] newtons was chosen with uniform probability on each time step. This training data contained information on only a small portion of the state space, so the learning system was forced to generalize. The learning system was able to balance the pole indefinitely after 60 seconds of training time, after which learning was disabled. When the learning system was applied to a finite-track, cart-pole problem, it was not able to learn to control the cart and pole consistently. This appears to be due to the fact that a time step was only 0.02 second. Baird (1993) explains why *Q*-learning cannot learn in continuous time (or discrete time with small time steps), and proposes a new algorithm, advantage updating, which does not have this limitation. Advantage updating could be combined with wire fitting and a function approximation system; this remains an area for future research.

6. CONCLUSION

We have proposed *wire fitting*, a new method for representing functions using any general function approximation system. This method solves the maximization problem arising in reinforcement learning systems and offers several other advantages. We have presented an example of a memory-based system that may be used with the method to represent Q functions, and have shown how the method, combined with the memory-based system, can be used for reinforcement learning on a cart-pole control problem.

7. REFERENCES

Atkeson, C. G. (1990). Memory-based approaches to approximating continuous functions. *Proceedings of the Workshop on Nonlinear Modeling and Forecasting*, Santa Fe Institute NM.

Baird, L. C. (1992). Function minimization for dynamic programming using connectionist networks. *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics* (pp. 19-24). Chicago IL.

Baird, L. C. (1993). *Advantage Updating*. (Technical Report WL-TR-93-1146). Wright-Patterson Air Force Base Ohio: Wright Laboratory. (available from the Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145).

Baird, L. C., & Klopf, A. H. (1993a). Extensions of the associative control process (ACP) network: Hierarchies and provable optimality. *Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (pp. 163-171), Honolulu, Hawaii.

Baird, L. C., & Klopf, A. H. (1993b). A hierarchical network of provably optimal learning control systems: Extensions of the associative control process (ACP) network. *Adaptive Behavior*, **1**(3), 321-352.

Barto, A. G., & Bradtke, S. J. (1991). *Real-time learning and control using asynchronous dynamic programming* (Tech. Rep. 91-57). Department of Computer Science, University of Massachusetts, Amherst MA.

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

Gullapalli, V. (1991). Associative reinforcement learning of real-valued functions. *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics* (pp. 1453-1458), Charlottesville VA.

Klopf, A. H., Morgan, J. S., & Weaver, S. E. (1993a). Modeling nervous system function with a hierarchical network of control systems that learn. *Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (pp. 254-261), Honolulu, Hawaii.

Klopf, A. H., Morgan, J. S., & Weaver, S. E. (1993b). A hierarchical network of control systems that learn: Modeling nervous system function during classical and instrumental conditioning. *Adaptive Behavior*, **1**(3), 321-352.

Millington, P. (1991). *Associative reinforcement learning for optimal control*. Master's thesis, Massachusetts Institute of Technology, Cambridge MA.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*.

Tesauro, G. (1990). Neurogammon: A neural-network backgammon program. *Proceedings of the International Joint conference on Neural Networks* **3** (pp. 33-40), San Diego CA.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, **8**(3/4), 257-277.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Doctoral thesis, Cambridge University, Cambridge, England.

Watkins, C. J. C. H., & Dayan, P. (1992). Technical note: *Q-learning*. *Machine Learning*, **8**(3/4), 279-292.

Werbos, P. J. (1989). Neural networks for control and system identification. *Proceedings of the 28th Conference on Decision and Control* (pp. 260-265), Tampa FL.