ELSEVIER Neurocomputing 9 (1995) 243-269 NEUROCOMPUTING

Memory-based neural networks for robot learning

Christopher G. Atkeson *, Stefan Schaal

College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280, USA

Received 7 July 1994; accepted 22 February 1995

Abstract

This paper explores a memory-based approach to robot learning, using memory-based neural networks to learn models of the task to be performed. Steinbuch and Taylor presented neural network designs to explicitly store training data and do nearest neighbor lookup in the early 1960s. In this paper their nearest neighbor network is augmented with a local model network, which fits a local model to a set of nearest neighbors. This network design is equivalent to a statistical approach known as locally weighted regression, in which a local model is formed to answer each query, using a weighted regression in which nearby points (similar experiences) are weighted more than distant points (less relevant experiences). We illustrate this approach by describing how it has been used to enable a robot to learn a difficult juggling task.

Keywords: Memory-based; Robot learning; Locally weighted regression; Nearest neighbor; Local models

1. Introduction

An important problem in motor learning is approximating a continuous function from samples of the function's inputs and outputs. This paper explores a neural network architecture that simply remembers experiences (samples) and builds a local model to answer any particular query (an input for which the function's output is desired). Steinbuch [62,63] and Taylor [68,69] independently proposed neural network designs that explicitly remembered the training experiences and used a local representation to do nearest neighbor lookup. They pointed out that this approach could be used for control. They used a layer of hidden units to

* Corresponding author. Email: cga@cc.gatech.edu

compute an inner product of each stored vector with the input vector. A winner-take-all circuit then selected the hidden unit with the highest activation. This type of network can find nearest neighbors or best matches using a Euclidean distance metric [36]. In this paper their nearest neighbor lookup network (which we will refer to as the memory network) is augmented with a local model network, which fits a local model to a set of nearest neighbors.

The memory-based neural network design can represent smooth nonlinear functions, yet has simple training rules with a single global optimum for building a local model in response to a query. Our philosophy is to model complex continuous functions using simple local models. This approach avoids the difficult problem of finding an appropriate structure for a global model and allows complex nonlinear models to be identified (trained) quickly. A key idea is to form a training set for the local model network after a query to be answered is known. This approach allows us to include in the training set only relevant experiences (nearby samples), and to weight the experiences according to their relevance to the query. The local model network, which may be a simple network architecture such as a perceptron, forms a model of the portion of the function near the query point, much as a Taylor series models a function in a neighborhood of a point. This local model is then used to predict the output of the function, given the input. After answering the query, a new local model is trained to answer the next query. This approach minimizes interference between old and new data, and allows the range of generalization to depend on the density of the samples.

Currently we are using polynomials as the local models. Since the polynomial local models are linear in the unknown parameters, we can estimate these parameters using a linear regression. 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 this approach cross validation is no more computationally expensive than answering a query. This is quite different from a parametric neural network, where a new network must be trained for each cross validation training set. We extend this approach to give information about the reliability of the predictions and local linearizations generated by locally weighted regression. This allows the robot to monitor its own skill level and guide its exploratory behavior. The polynomial local models allow us to efficiently estimate local linear models for different points in the state space. These local linear models are used in several ways during learning.

We use several forms of indirect learning, where a model is learned and then control actions are chosen based on the model, rather than direct learning, where appropriate control actions are learned directly. Our starting point for modeling is that we do not know the structure or form of the system to be controlled. We assume we do know what constitutes a state of the system, and that we measure the complete state. Later papers will discuss how we could approach tasks in which complete measurements of the state are not available, or what constitutes a state is not even known.

The learned models are multidimensional functions that are approximated from sampled data (the previous experiences or attempts to perform the task). Goals for function approximation in robot learning go beyond being able to represent the training set and generalize appropriately. The learned models are used in a variety of ways to successfully execute the task. We would like the models to incorporate the latest information. The models will be continuously updated with a stream of new training data, so updating a model with new data should take a short period of time. There are also time constraints on how long it can take to use a model to make a prediction. Because we are interested in control methods that make use of local linearizations of the plant model, we want a representation that can quickly compute a local linear model of the represented transformation. We also need to be able to find first (and potentially second) derivatives of the learned function. We would like to minimize the negative interference from learning new knowledge on previously stored information. The ability to tell where in the input space the function is accurately approximated is very useful. This is typically based on the local density of samples, and an estimate of the local variance of the outputs. This ability is used in iterative use of the model to determine when to terminate search and collect more data.

2. Locally weighted regression

As the most generic approximator that satisfies many of these criteria, we are exploring a version of memory-based learning technique called locally weighted regression (LWR) [9,5]. A memory-based learning (MBL) system is trained by storing the training data in a memory. This allows MBL systems to achieve real-time learning. MBL avoids interference between new and old data by retaining and using all the data to answer each query. MBL approximates complex functions using simple local models, as does a Taylor series. Examples of types of local models include nearest neighbor, weighted average, and locally weighted regression (Fig. 1). Each of these local models combine points near to a query point to estimate the appropriate output. Nearest neighbor local models simply choose the closest point and use its output value. Weighted average local models sum the outputs of nearby points weighted by their distance to the query point. Locally weighted regression fit a surface to nearby points using a distance weighted regression.

The weights in the locally weighted regression depend on parameters used to calculate a distance metric and a weighting function, and stabilize the solution to the regression. We will refer to these parameters as 'fit parameters'. These parameters can be optimized automatically in a local fashion using cross validation.

Locally weighted regression uses a relatively complex regression procedure to form the local model, and is thus more expensive than nearest neighbor and weighted average memory-based learning procedures. For each query a new local model is formed. The rate at which local models can be formed and evaluated limits the rate at which queries can be answered. This section describes how locally weighted regression can be implemented in real time.

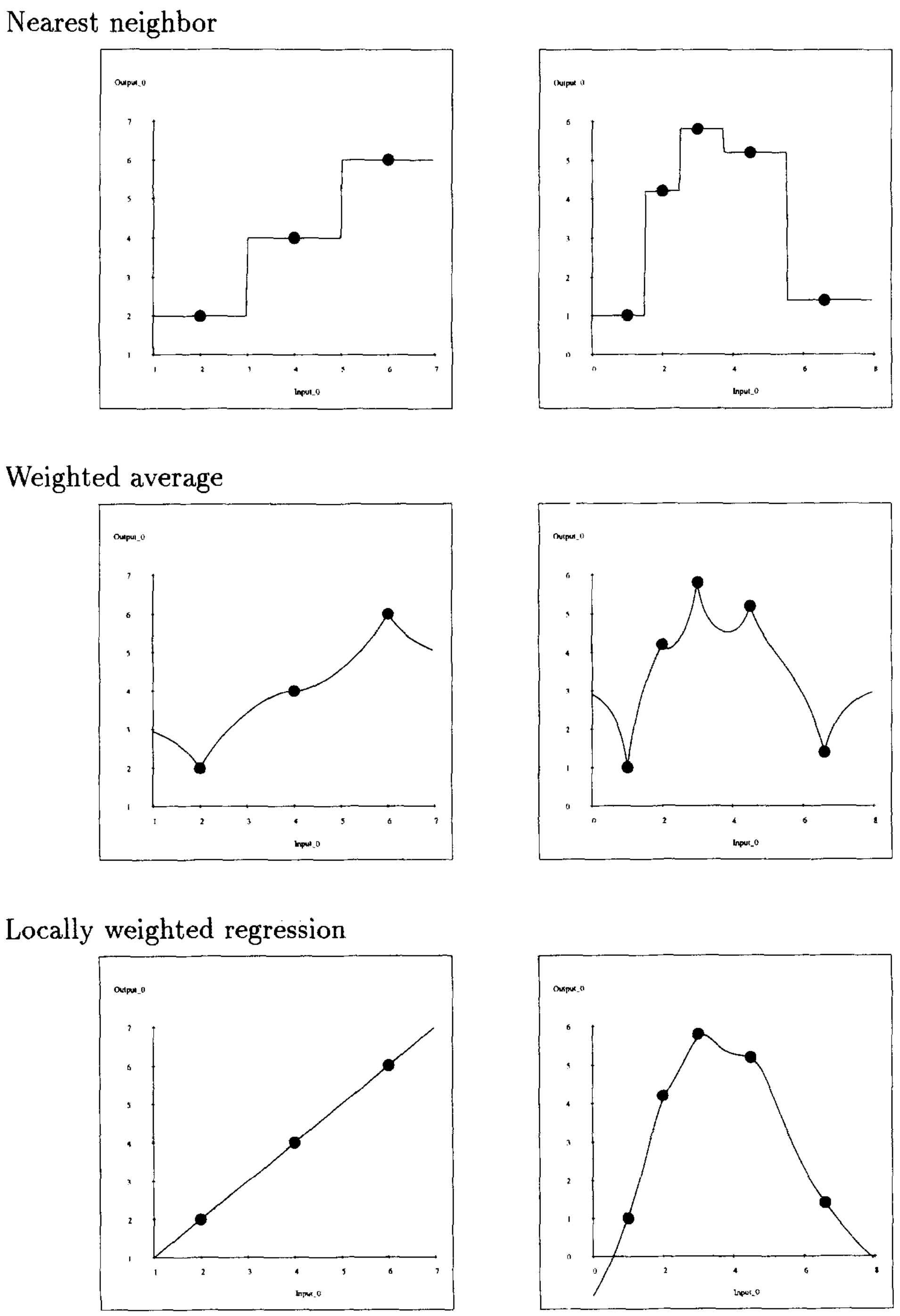


Fig. 1. Fits using different types of local models for three and five data points.

2.1 An example

As an example of modeling a function using locally weighted regression, we will consider a problem from motor control and robotics, two-joint arm inverse dynamics (Fig. 2). We will predict torques at the shoulder, τ_1, and elbow, τ_2, on the basis of joint positions, θ_1 and θ_2, joint velocities, $\dot{\theta}_1$ and $\dot{\theta}_2$, and joint accelerations, $\ddot{\theta}_1$ and $\ddot{\theta}_2$. We use this example because we already know the idealized function, and will be able to assess how well the locally weighted regression procedure is doing and interpret the parameters used to improve the fit. In an actual application a structured model ([3], for example) would be used to fit the dynamics data, and the locally weighted regression would be used to fit the errors (residuals) of the structured model.

We have a query point $(\theta_1^*, \theta_2^*, \dot{\theta}_1^*, \dot{\theta}_2^*, \ddot{\theta}_1^*, \ddot{\theta}_2^*)$ for which we want to predict the shoulder and elbow torques. We will first show how an unweighted regression can be used to form a global model. Then we will show how a weighted regression can be used to form a local model appropriate to answer this particular query. For the purposes of this example we will assume a quadratic model is used in the regression. In this dynamics example there are 28 terms in the quadratic model:

$$\begin{array}{ccccccc}
1 & \theta_1, & \theta_2, & \dot{\theta}_1, & \dot{\theta}_2, & \ddot{\theta}_1, & \ddot{\theta}_2, \\
 & \theta_1 * \theta_1, & \theta_1 * \theta_2, & \theta_1 * \dot{\theta}_1, & \theta_1 * \dot{\theta}_2, & \theta_1 * \ddot{\theta}_1, & \theta_1 * \ddot{\theta}_2, \\
 & & \theta_2 * \theta_2, & \theta_2 * \dot{\theta}_1, & \theta_2 * \dot{\theta}_2, & \theta_2 * \ddot{\theta}_1, & \theta_2 * \ddot{\theta}_2, \\
 & & & \dot{\theta}_1 * \dot{\theta}_1, & \dot{\theta}_1 * \dot{\theta}_2, & \dot{\theta}_1 * \ddot{\theta}_1, & \dot{\theta}_1 * \ddot{\theta}_2, \\
 & & & & \dot{\theta}_2 * \dot{\theta}_2, & \dot{\theta}_2 * \ddot{\theta}_1, & \dot{\theta}_2 * \ddot{\theta}_2, \\
 & & & & & \ddot{\theta}_1 * \ddot{\theta}_1, & \ddot{\theta}_1 * \ddot{\theta}_2, \\
 & & & & & & \ddot{\theta}_2 * \ddot{\theta}_2
\end{array}$$

where 1 represents the constant term in the model.

Let us assume we have 1000 samples of the two joint arm dynamics function. To form a local model of the shoulder torque involves finding estimates of the 28 terms or parameters of the local quadratic model. The equation to be solved is

$$\mathbf{X}\beta = \mathbf{y} \tag{1}$$

where $\mathbf{X}$ is a 1000 by 28 data matrix, in which each row has the 28 terms of the quadratic model corresponding to a point (sample of the function), and each column corresponds to a particular term in the quadratic model. β is the vector of 28 estimated parameters of the quadratic model, and $\mathbf{y}$ is the vector of 1000 shoulder torques from the 1000 points included in the regression.

An unweighted regression finds the solution to the normal equations:

$$(\mathbf{X}^T\mathbf{X})\beta = \mathbf{X}^T\mathbf{y} \tag{2}$$

The estimated parameters are used, with the query point, to predict the shoulder torque for the query point. Another set of parameters are estimated for the elbow torque.

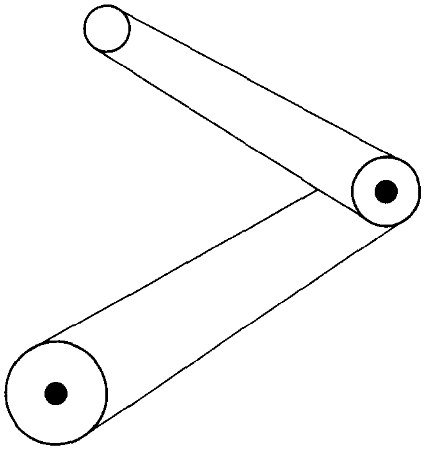


Fig. 2. Simulated planar two-joint arm.

However, we assume the global quadratic model is not the correct model structure for predicting the torques. These structural modeling errors imply that different sets of parameters are estimated by the regression, given different data sets. The data set can be tailored to the query point by emphasizing nearby points in the regression. The origin of the input data is first shifted by subtracting the query point from each data point. Then each data point is given a weight.

Unweighted regression gives distant points equal influence with nearby points on the ultimate answer to the query, for equally spaced data. To weight similar points more, locally weighted regression is used. First, a distance is calculated from each of the stored data points to the query point $\mathbf{q}$:

$$d_i^2 = \sum_j m_j^2 (\mathbf{X}_{ij} - \mathbf{q}_j)^2 \tag{3}$$

For the robot arm dynamics example, d_i^2 is calculated for each point in the following way:

$$\begin{aligned} d^2 = {} & m_1^2(\theta_1 - \theta_1^*)^2 + m_2^2(\theta_2 - \theta_2^*)^2 + m_3^2(\dot\theta_1 - \dot\theta_1^*)^2 \\ & + m_4^2(\dot\theta_2 - \dot\theta_2^*)^2 + m_5^2(\ddot\theta_1 - \ddot\theta_1^*)^2 + m_6^2(\ddot\theta_2 - \ddot\theta_2^*)^2 \end{aligned} \tag{4}$$

The superscript * indicates the query point, and the m_j are the components of the distance metric.

The weight for each stored data point is a function of that distance:

$$w_i = f(d_i^2) \tag{5}$$

Each row i of $\mathbf{X}$ and $\mathbf{y}$ is multiplied by the corresponding weight w_i. A simple weighting function just raises the distance to a negative power. The magnitude of the power determines how local the regression will be (the rate of dropoff of the weights with distance).

$$w_i = \frac{1}{d_i^p} \tag{6}$$

This type of weighting function goes to infinity as the query point approaches a stored data point. This forces the locally weighted regression to exactly match that stored point. If the data is noisy, exact interpolation is not desirable, and a weighting scheme with limited magnitude is desired. One such scheme, which we use in implementations on actual robots, is a Gaussian kernel:

$$w_i = \exp\left(\frac{-d_i^2}{2k^2}\right) \tag{7}$$

The parameter k scales the size of the kernel to determine how local the regression will be.

A potential problem is that the data points may be distributed in such a way as to make the regression matrix **X** nearly singular. Ridge regression [17] is used to prevent problems due to a singular data matrix. The following equation, with **X** and **y** already weighted, is solved for β:

$$(\mathbf{X}^T\mathbf{X} + \Lambda)\beta = \mathbf{X}^T\mathbf{y} \tag{8}$$

where Λ is a diagonal matrix with small positive diagonal elements λ_i^2. This is equivalent to adding i extra rows to **X**, each having a single non-zero element, λ_i, in the ith column. Adding additional rows can be viewed as adding 'fake' data, which, in the absence of sufficient real data, biases the parameter estimates to zero [17]. Another view of ridge regression parameters is that they are the Bayesian assumptions about the apriori distributions of the estimated parameters [56].

2.2. *Assessing the computational cost*

Lookup has three stages: forming weights, forming the regression matrix, and solving the normal equations. Let us examine how the cost of each of these stages grows with the size of the data set and dimensionality of the problem. We will assume a linear local model.

Forming and applying the weights involves scanning the entire data set, so it scales linearly with the number of data points in the database (n). For each of d input dimensions there are a constant number of operations, so the number of operations scales linearly with the number of input dimensions. Note that we can eliminate points whose distance is above a threshold, reducing the number of points considered in subsequent stages of the computation.

Each element of $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$ is the inner (dot) product of two columns of **X** or **y**. The architecture of digital signal processors is ideally suited for this computation, which consists of repeated multiplies and accumulates. The computation is linear in the number of rows n and quadratic in the number of columns $(d^2 + d * o)$, where d is the number of input dimensions and o is the number of output dimensions.

Solving the normal equations is done using a LDL^T (Cholesky) decomposition, which is cubic in the number of input dimensions, and independent of the number of data points. Other more sophisticated and more expensive decompositions, such as the singular value decomposition, do not need to be used since the ridge regression procedure guarantees that the normal equations will be well-conditioned.

The most straightforward parallel implementation of locally weighted regression would distribute the data points among several processors. Queries can be broadcast to the processors, and each processor can weight its data set and form its contribution to $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$. These contributions can be summed and the full normal equations solved on a single processor. The communication costs are linear in the number of processors and quadratic in the number of columns ($d^2 + d * o$), and independent of the total number of points.

We have implemented the local weighted regression procedure on a 33MHz Intel i860 microprocessor. The peak computation rate of this processor is 66 MFlops. We have achieved effective computation rates of 15 MFlops on a learning problem with 10 input dimensions and 5 output dimensions, using a linear local model. This leads to a lookup time of approximately 15 milliseconds on a database of 1000 points.

This memory-based approach can also be simulated using k-d tree data structures [28] on a standard serial computer and using parallel search on a massively parallel computer, the Connection Machine [33].

2.3. Implementing locally weighted regression in a neural network

The memory network of Steinbuch and Taylor can be used to find the nearest stored vectors to the current input vector. The memory network computes a measure of the distance between each stored vector and the input vector in parallel, and then a 'winner take all' network selects the nearest vector (nearest neighbor). Euclidean distance has been chosen as the distance metric, because the Euclidean distance is invariant under rotation of the coordinates used to represent the input vector.

The memory network consists of three layers of units: input units, hidden or memory units, and output units. The input units are fully connected to the hidden units. The squared Euclidean distance between the input vector ($\mathbf{i}$) and a weight vector ($\mathbf{w}_k$) for the connections of the input units to hidden unit k is given by:

$$d_k^2 = (\mathbf{i} - \mathbf{w}_k)^{\mathrm{T}}(\mathbf{i} - \mathbf{w}_k) = \mathbf{i}^{\mathrm{T}}\mathbf{i} - 2\mathbf{i}^{\mathrm{T}}\mathbf{w}_k + \mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k$$

Since the quantity $\mathbf{i}^{\mathrm{T}}\mathbf{i}$ is the same for all the hidden units, minimizing the distance between the input vector and the weight vector for each hidden unit is equivalent to maximizing:

$$\mathbf{i}^{\mathrm{T}}\mathbf{w}_k - 1/2\mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k$$

This quantity is the inner product of the input vector and the weight vector for hidden unit k, biased by half the squared length of the weight vector. Maximizing this quantity using a winner take all circuit allows the unit with the smallest distance to be selected.

Dynamics of the memory network neurons allow the memory network to output a sequence of nearest neighbors. These nearest neighbors form the selected training sequence for the local model network. Memory unit dynamics can also be used to allocate 'free' memory units to new experiences, and to forget old training points when the capacity of the memory network is fully utilized.

The local model network consists of only one layer of modifiable weights preceded by any number of layers with fixed connections. There may be arbitrary preprocessing of the inputs of the local model, but the local model is linear in the parameters used to form the fit. The local model network using the LMS training algorithm performs a linear regression of the transformed inputs against the desired outputs. Thus, the local model network can be used to fit a linear regression model to the selected training set. With multiplicative interactions between inputs the local model network can be used to fit a polynomial surface (such as a quadratic) to the selected training set. An alternative implementation of the local model network could use a single layer of 'sigma-pi' units [18].

This network design has simple training rules. In the memory network the weights are simply the values of the components of input and output vectors, and the bias for each memory unit is just half the squared length of the corresponding input weight vector. No search for weights is necessary, since the weights are directly given by the data to be stored. The local model network is linear in the weights, leading to a single optimum which can be found by linear regression or gradient descent. Thus, convergence to the global optimum is guaranteed when forming a local model to answer a particular query.

3. Related work

Memory-based representations have a long history. Approaches which represent previous experiences directly and use a similar experience or similar experiences to form a local model are often referred to as nearest neighbor or k-nearest neighbor approaches. Local models (often polynomials) have been used for many years to smooth time series [58,59,74,43] and interpolate and extrapolate from limited data. [6,54] survey the use of nearest neighbor interpolators to fit surfaces to arbitrarily spaced points. [19] surveys the use of nearest neighbor estimators in nonparametric regression. [39] refer to nearest neighbor approaches as 'moving least squares' and survey their use in fitting surfaces to data. [21,22] survey the use of nearest neighbor and local model approaches in modeling chaotic dynamic systems. [34,35] describe approaches to memory-based control of robots. [60] describes a memory-based neural network approach based on a probabilistic model that motivates using weighted averaging as the local model.

An early use of direct storage of experience was in pattern recognition. [24,25] suggested in the early 1950s that a new pattern could be classified by searching for similar patterns among a set of stored patterns, and using the categories of the similar patterns to classify the new pattern. Steinbuch and Taylor proposed a neural network implementation of the direct storage of experience and nearest-neighbor search process for pattern recognition [62,68] and pointed out that this approach could be used for control [63]. [61] proposed using directly stored experience to learn pronunciation, using a Connection Machine and parallel search to find relevant experience. They have also applied their approach to medical diagnosis [72] and protein structure prediction.

Nearest neighbor approaches have also been used in nonparametric regression and fitting surfaces to data. Often, a group of similar experiences, or nearest neighbors, is used to form a local model, and then that model is used to predict the desired value for a new point. Local models are formed for each new access to the memory. [73,53,14,12,57] proposed using a weighted average of a set of nearest neighbors. [30,7] analyze such weighted average schemes. [14,20,45] suggested using a weighted regression to fit a local polynomial model at each point a function evaluation was desired. All of the available data points were used. Each data point was weighted by a function of its distance to the desired point in the regression. [44,52,40,51,71,42,65,27] suggested fitting a polynomial surface to a set of nearest neighbors, also using distance weighted regression. Stone scaled the values in each dimension when the experiences where stored. The standard deviations of each dimension of previous experiences were used as the scaling factors, so that the range of values in each dimension were approximately equal. This affects the distance metric used to measure closeness of points. [9] proposed using robust regression procedures to eliminate outlying or erroneous points in the regression process. A program implementing a refined version of this approach (LOESS) is available by sending electronic mail containing the single line, *send dloess from a*, to the address netlib@research.att.com [31]. [11] analyzes the statistical properties of the LOESS algorithm and [10] shows examples of its use. [66,67,16,37,38,8,41, 23,49] provide analyses of nearest neighbor approaches. [26] compares the performance of nearest neighbor approaches with other methods for fitting surfaces to data.

4. Learning in simulation

The network has been used for motor learning of a simulated arm and a simulated running machine. The network performed surprisingly well in these simple evaluations. The simulated arm was able to follow a desired trajectory after only a few practice movements. Performance of the simulated running machine in following a series of desired velocities was also improved. We will report only on the arm trajectory learning here.

Fig. 2 shows the simulated 2-joint planar arm. The problem faced in this simulation is to learn the correct joint torques to drive the arm along the desired trajectory (the inverse dynamics problem). In addition to the feedforward control signal produced by the network described in this paper, a feedback controller was also used.

Fig. 3 shows several learning curves for this problem. The first point in each of the curves shows the performance generated by the feedback controller alone. The

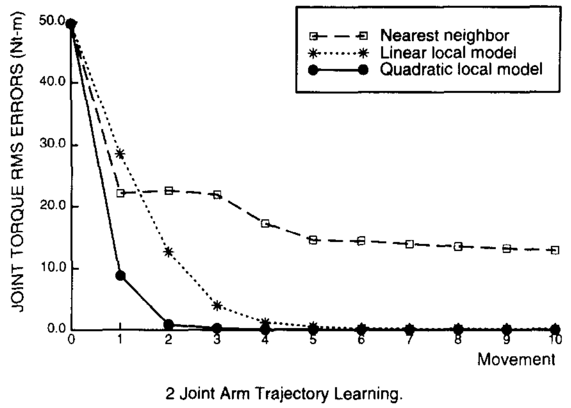


Fig. 3. Learning curves from 3 different network designs on the two joint arm trajectory learning problem.

error measure is the RMS torque error during the movement. The highest curve shows the performance of a nearest neighbor method without a local model. On each time step the nearest point was used to generate the torques for the feedforward command, which were then summed with the output from the feedback controller. The second curve shows the performance using a linear local model. The third curve shows the performance using a quadratic local model. Adding the local model network greatly speeds up learning. The network with the quadratic local model learned more quickly than the one with the local linear model.

5. Interference

To illustrate the differences between some proposed neural network representations and a memory-based representation, two neural network methods, CMAC [2,1] and sigmoidal feedforward neural networks, were compared to the approach explored in this paper. The parameters for the CMAC approach were taken from [46] who used the CMAC to model arm inverse dynamics. The architecture for the sigmoidal feedforward neural network was taken from [29, section 6] who also modeled arm inverse dynamics.

The ability of each of these methods to predict the torques of the simulated two joint arm at 1000 random points was compared. Fig. 4 plots the normalized RMS prediction error. The points were sampled uniformly using ranges comparable to those used in [46]. Initially, each method was trained on a training set of 1000 random samples of the two joint arm dynamics function, and then the predictions of the torques on a separate test set of 1000 random samples of the two joint arm dynamics function were assessed (points 1, 3, and 5). Each method was then trained on 10 attempts to make a particular desired movement. Each method successfully learned the desired movement. After this second round of training, performance on the random test set was again measured (points 2, 4, and 6).

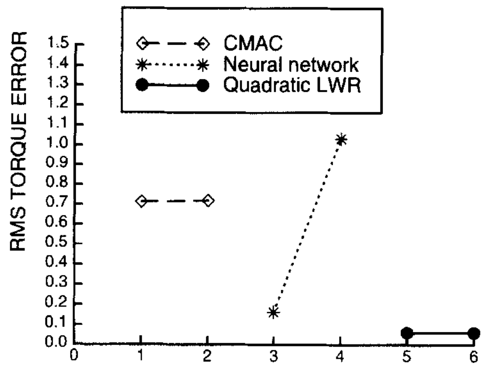


Fig. 4. Performance of various methods on two joint arm dynamics.

The data indicate that the locally weighted regression approach (filled in circles) and the sigmoidal feedforward network approach (asterisks) both generalize well on this problem (points 3 and 5 have low error). The CMAC (diamonds) did not generalize well on this problem (point 1 has a large error), although it represented the original training set with a normalized RMS error of 0.000001. A variety of CMAC resolutions were explored, ranging from a basic CMAC cell size covering the entire range of data to a cell size covering a fifth of the data range in each dimension. A cell size covering one half the data ranges in each dimension generalized best (the data shown here).

After training on a different training set (the attempts to make a particular desired movement), the sigmoidal feedforward neural network lost its memory of the full dynamics (point 4), and represented only the dynamics of the particular movements being learned in the second training set. This interference between new and previously learned data was not prevented by increasing the number of hidden units in the single layer network from 10 up to 100. The other methods explored did not show this interference effect (points 2 and 6).

6. Tuning fit parameters globally

For the example problem of two joint arm inverse dynamics, we have introduced 34 free parameters into the local regression process: the weighting function dropoff parameter p, the 6 elements of the distance metric m_i, and the 27 variable diagonal elements of Λ (the ridge regression parameters λ_i). The element of Λ corresponding to the constant term, λ_1, is held fixed.

A cross validation approach is used to choose values for these fit parameters. For each point a query is done to estimate the output at that point, after removing the point from the database. The difference between the estimate and the actual value for that point is the cross validation error for that point. The sum of the squared cross validation errors is minimized using a nonlinear parameter estimation procedure (MINPACK [48] or NL2SOL [15], for example). Because the local model is linear in the unknown parameters we can analytically take the derivative of the cross validation error with respect to the parameters to be estimated, which greatly speeds up the search process. In the memory-based approach computing the cross validation error for a single point is no more computationally expensive than answering a query. This is quite different from a parametric neural network, where a new network must be trained for each cross validation training set with a particular point removed.

The cross validation to optimize the fit parameters may be done globally, using all the experiences in the memory to produce one set of fit parameters. Different fit parameters can be used for different outputs. The cross validation may also be done locally, either with each query, or separately for different regions of the input space, producing different sets of fit parameters specialized for particular queries, as discussed in the next section.

We can use the optimized distance metric to find which input variables are irrelevant to the function being represented. In the horizontal two-joint arm inverse dynamics problem, m_1, the weight on the distances in the θ_1 direction typically drops to zero, indicating that the input variable θ_1 is irrelevant to predicting τ_1 and τ_2. This is actually the case, as θ_1 does not appear in the true dynamics equations for an arm operating in a horizontal plane.

We can also interpret the ridge regression parameters, λ_i. Since the arm dynamics are linear in acceleration, the terms in the local model that are quadratic in acceleration ($\ddot{\theta}_1^2$, $\ddot{\theta}_1 * \ddot{\theta}_2$, $\ddot{\theta}_2^2$) are not relevant to predicting torques. Similarly the products of velocity and acceleration ($\dot{\theta}_1 * \ddot{\theta}_1$, $\dot{\theta}_1 * \ddot{\theta}_2$, $\dot{\theta}_2 * \ddot{\theta}_1$, $\dot{\theta}_2 * \ddot{\theta}_2$) are also not relevant to the dynamics. The ridge regression parameter for each of these terms becomes very large in the parameter optimization. The effect of this is to force the estimated parameter β_i for these terms to be zero and the terms to have no effect on the regression.

We have also explored stepwise regression procedures to determine which terms of the local model are useful [17] with similar results.

7. Tuning fit parameters locally

In the process of implementing various robot learning algorithms it has become clear to us that the fit parameters should depend on the location of the query point. In this section we describe new procedures that estimate local values of the fit parameters optimized for the site of the current query point. We want to demonstrate the differences between local and global fitting in an example where we only focus on the kernel width k of a Gaussian weighting function (Eq. 7). In Fig. 5(a), a noisy data set of the function $y = x - \sin^3(2\pi x^3)\cos(2\pi x^3)\exp(x^4)$ was fitted by locally weighted regression with a globally optimized constant k. In the left half of the plot, the regression starts to fit noise because k had to be rather small to fit the high frequency regions on the right half of the plot. The prediction intervals, which are shown by the shaded areas in Fig. 5 and which will be introduced below, demonstrate high uncertainty in several places. To avoid such undesirable behavior, a *local* optimization criterion is needed. Standard linear regression analysis provides a series of well-defined statistical tools to assess the quality of fits, such as coefficients of determination, t-tests, F-test, the PRESS-statistic, Mallow's Cp-test, confidence intervals, prediction intervals, and many more [50]. These tools can be adapted to locally weighted regression. We do not want to discuss all possible available statistics here but rather focus on two that have proved to be quite helpful.

Cross validation has a relative in linear regression analysis called the PRESS residual error. The PRESS statistic performs leave-one-out cross validation, however, without the need to recalculate the regression parameters for every excluded point. This is computationally very efficient. The PRESS residual can be expressed as a mean squared cross validation error MSE_{cross}:

$$MSE_{cross} = \frac{1}{n}\sum_i \left(\frac{w_i(y_i - \mathbf{x}_i^T\beta)}{1 - w_i^2\mathbf{x}_i^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_i} \right)^2 \tag{9}$$

where

$$n = \sum_i w_i^2 \tag{10}$$

where the matrix **X** is already weighted. In Fig. 5(b), the same data as in Fig. 5(a) was fitted by adjusting k to minimize MSE_{cross} at each query point. The outcome is much smoother than that of global cross validation, and also the prediction intervals (a measure of uncertainty described below and indicated by the shaded regions in Fig. 5) are narrower. It should be noted that the extrapolation properties on both sides of the graph are quite appropriate (compared to the known underlying function), in comparison to Fig. 5(a) and Fig. 5(c).

Prediction intervals I_i^+ and I_i^- are expected bounds of the prediction error at a query point $\mathbf{x}_i$ [50]:

$$I_i^{\pm} = \mathbf{x}_i^T\beta \pm t_{\alpha/2,n-dof}\, s\sqrt{1 + \mathbf{x}_i^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_i} \tag{11}$$

where

$$dof = \sum_i w_i^4\mathbf{x}_i^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_i \tag{12}$$

s denotes the estimate of the local variance, *dof* the local degrees of freedom of the regression [55], while the other notation is the same as above. Besides using the intervals to assess the confidence in the fit at a certain point, they provide another optimization measure. Fig. 5(c) demonstrates the result when applying this statistic for optimization of k at each query point. Again, the fitted curve is significantly smoother than the global cross validation fit. A rather interesting and also typical effect happens at the right side of the plot. When starting to extrapolate, the prediction intervals suddenly favor a global regression instead of the local regression, i.e. the k was chosen to be rather large. It turns out that in local fit parameter optimization one always finds a competition between local and global

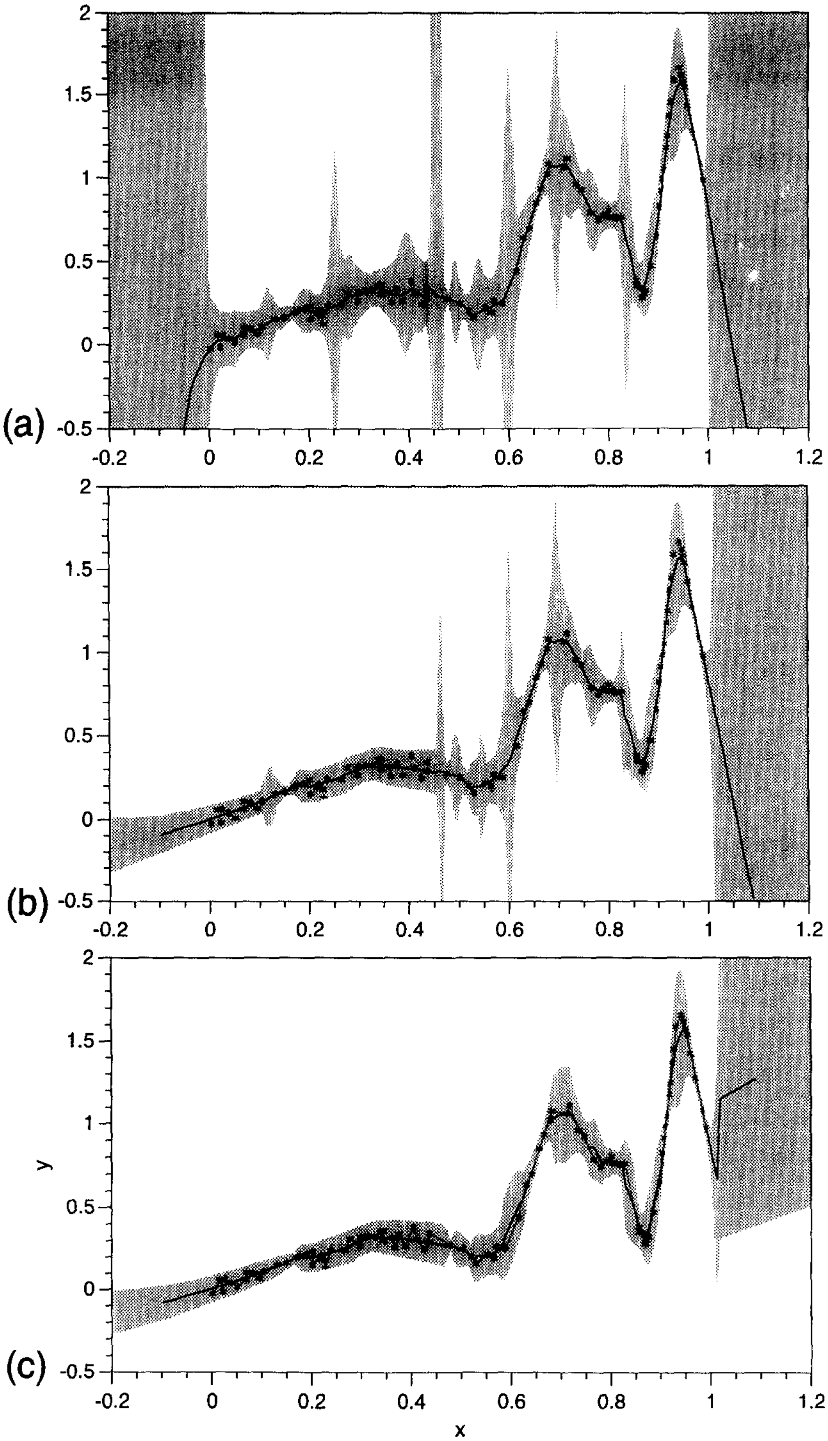


Fig. 5. Optimizing the LWR fit using different measures: (a) global cross validation, (b) local cross validation, (c) local prediction intervals.

regression. But sudden jumps from one mode into the other typically take place only when the prediction intervals are so large that the data is not reliable anyway.

8. Assessing the quality of the local model

Both the local cross validation error MSE_{cross} and the prediction interval I_i may serve to assess the quality of the local fit:

$$Q_{fit} = \frac{\sqrt{MSE_{cross}}}{c} \tag{13}$$

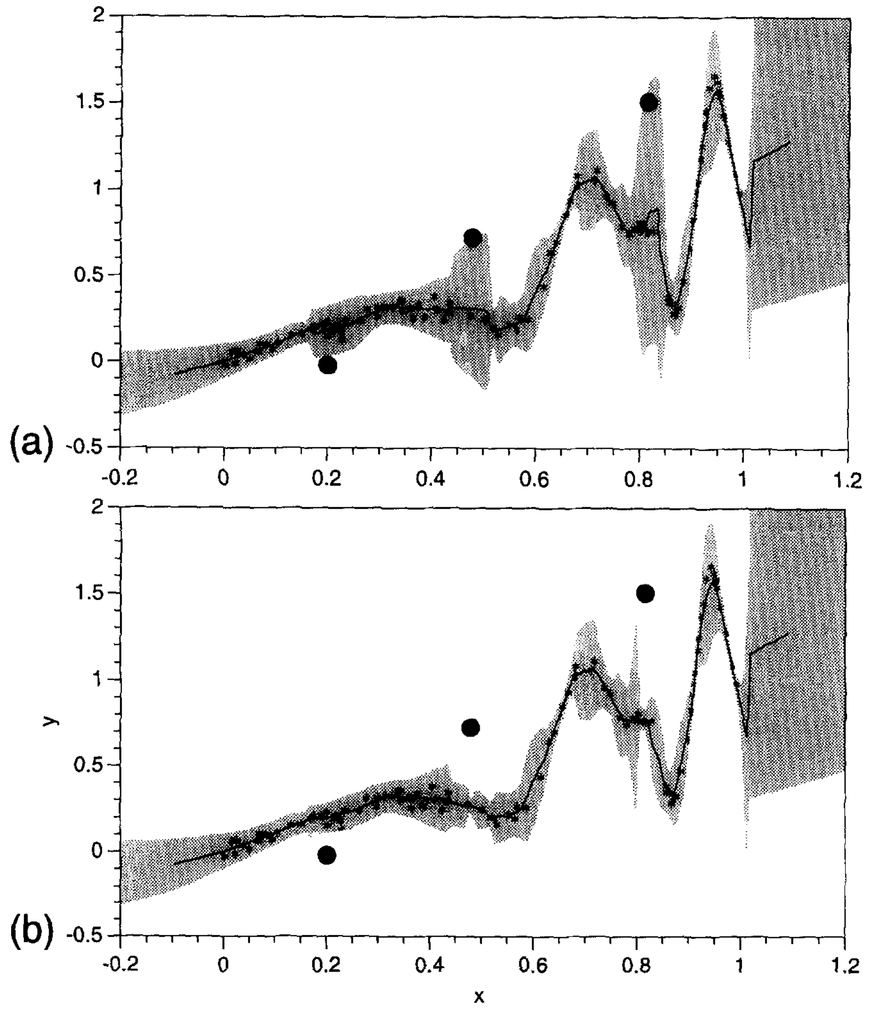


Fig. 6. Influence of outliers on LWR (the black dots are the outliers): (a) no outlier removal, (b) with outlier removal.

or

$$Q_{fit} = \frac{I_i^+ - I_i^-}{c} \tag{14}$$

The factor c makes Q_{fit} dimensionless and normalizes it with respect to some user defined quantity. In our applications, we usually preferred Q_{fit} based on the prediction intervals, which is the more conservative assessment.

9. Dealing with outliers

Linear regression is not robust with respect to outliers. This also holds for locally weighted regression, although the influence of outliers will not be noticed unless the outliers lie close enough to a query point. In Fig. 6(a) we added three outliers to the test data of Fig. 5 to demonstrate this effect; the plots in Fig. 6 should be compared to Fig. 5(c). [47] applied the median absolute deviation procedure from robust statistics [32] to globally remove outliers in LWR. Again, we would like to localize our criterion for outlier removal. The PRESS statistic can be modified to serve as an outlier detector in LWR. For this, we need the standardized individual PRESS residual (also called Studentized residual):

$$e_{PRESS} = \frac{w_i\left(y_i - \mathbf{x}_i^T\beta\right)}{s\sqrt{1 - w_i^2\mathbf{x}_i^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_i}} \tag{15}$$

This measure has zero mean and unit variance and assumes a locally normal distribution of the error, a mild assumption. If, for a given data point it deviates from zero more than a certain threshold, the point can be called an outlier. A conservative threshold would be 1.96, discarding all points lying outside the 95% area of the normal distribution. In our applications, we used 2.57, cutting off all data outside the 99% area of the normal distribution. As can be seen in Fig. 6(b), the effects of the outliers are reduced.

10. A testbed for learning algorithms: Robot juggling

We have constructed a system for experiments in real-time motor learning [70]. The task is a juggling task known as ‘devil sticking’. A center stick is batted back and forth between two handsticks (Fig. 7(a)). Fig. 7(b) 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 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

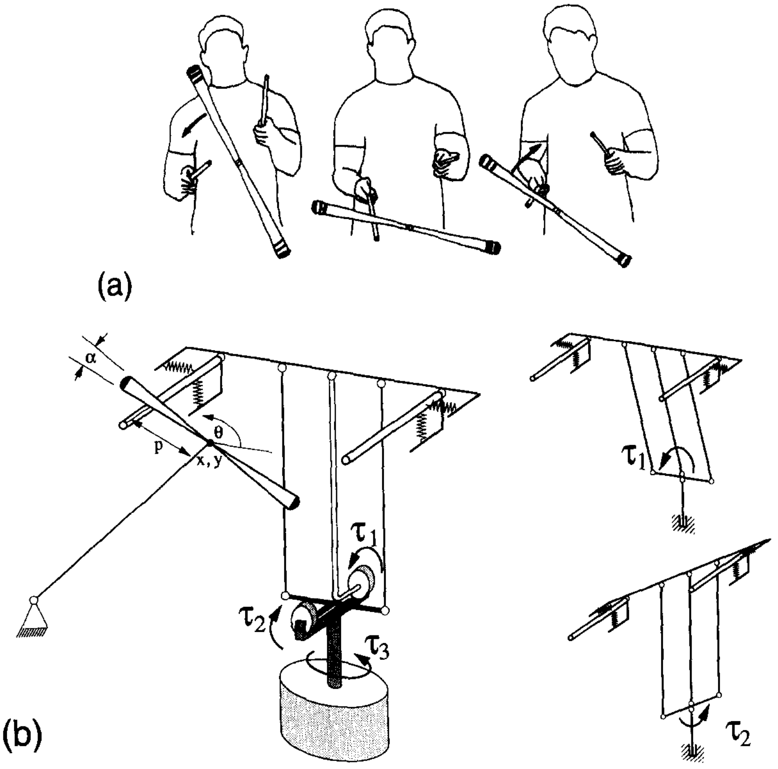


Fig. 7. (a) An illustration of devil sticking, (b) a devil sticking robot.

stick. Ultimately we want to be able to perform unconstrained three dimensional devil sticking using vision to provide sensing of the center stick state.

The task state is the predicted location of where the center stick would hit the hand stick if the hand stick was held in a nominal position. Standard ballistics equations for the flight of the center stick are used to map flight trajectory measurements $(x(t), y(t), \theta(t))$ into a task state:

$$\mathbf{x} = \left(p, \theta, \dot{x}, \dot{y}, \dot{\theta}\right) \tag{16}$$

p is the distance from the middle of the center stick that the hand stick at the nominal position contacts the center stick, θ is the angle of the center stick at nominal contact, and $\dot{x}$, $\dot{y}$, and $\dot{\theta}$, are the velocities and angular velocity of the center stick at nominal contact (Fig. 7).

The task command is given by a displacement of the hand stick from the nominal position (x_h, y_h), a center stick angular velocity threshold $(\dot{\theta}_t)$ to trigger the start of a throwing motion, and a throw velocity vector (v_x, v_y).

$$\mathbf{u} = \left(x_h, y_h, \dot{\theta}_t, v_x, v_y\right) \tag{17}$$

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 after the hit. Thus, a forward or an inverse model would have 10 input dimensions and 5 output dimensions.

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

$$\mathbf{u}_k = \hat{f}^{-1}(\mathbf{x}_k, \mathbf{x}_{k+1}) \tag{18}$$

Before each hit the system looked up a command with the predicted nominal impact state and the desired result state:

$$\mathbf{u}_k = \hat{f}^{-1}(\mathbf{x}_k, \mathbf{x}_d) \tag{19}$$

Inverse model learning 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, which, on average, was about 400 milliseconds. 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 default 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 memory-based system (approximately 100 hits). Learning with small amounts of initial data was not possible.

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 memory-based forward model with the same data:

$$\hat{\mathbf{x}}_{k+1} = \hat{f}(\mathbf{x}_k, \hat{\mathbf{u}}_k) \tag{20}$$

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 β. 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.

$$\frac{\partial \hat{f}}{\partial \mathbf{u}} \Delta \hat{\mathbf{u}}_k = \hat{\mathbf{x}}_{k+1} - \mathbf{x}_d \tag{21}$$

The pseudo-inverse of the matrix $\partial \hat{f} / \partial \mathbf{u}$ is used to solve the above equation for $\Delta \hat{\mathbf{u}}_k$, to handle situations in which the matrix is singular or there are a different number of commands and states (which does not apply for devil sticking). This process of command refinement can be repeated until the forward model no longer produces accurate predictions of the outcome. This 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. The outcome, however, 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 unreasonably large commands to correct for deviations from the setpoint. The workspace bounds and command bounds of our devil sticking robot limit the size of the commands. In addition, deadbeat control in the presence of errors in the model seems to lead to large inappropriate commands. Second, the ten dimensional input space is large, and 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. Second, the control must have as the primary goal increasing the data density in the current region of the state-action space, and as a secondary goal to arrive at the desired goal state. Both requirements are fulfilled by a simple exploration algorithm we have developed, the shifting setpoint algorithm (SSA). Applied to devil sticking, the SSA proceeds as follows:

(1) Regardless of the poor juggling quality of the robot (i.e. at most two or three hits per trial), the SSA makes the robot repeat these initial actions with small random perturbations until a cloud of data was collected somewhere in state-action space of each hand. An abstract illustration for this is given in Figs. 8(a–b).
(2) Each point in the data cloud of each hand is 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 becomes the setpoint of the hand and an linear quadratic (LQ) controller is calculated from its local linear model [4]. By means of these controllers, the amount of data around the setpoints can quickly be increased until the quality of the local models exceeds a chosen statistical threshold.
(3) At this point, the setpoints are gradually shifted towards the goal setpoints until the data support of the local models falls below a statistical value. After shifting, the smoothing kernel is optimized by minimizing the local cross validation error.
(4) The SSA continues by collecting data in the new regions of the workspace until the setpoints can be shifted again (Fig. 8(c)). The procedure terminates by reaching the goal, leaving a (hyper-) ridge of data in space (Fig. 8(d)).

The linear quadratic controllers play a crucial role for devil sticking. It is difficult to build good local linear models in the high dimensional forward models, particularly at the beginning of learning. Linear quadratic control is robust even if

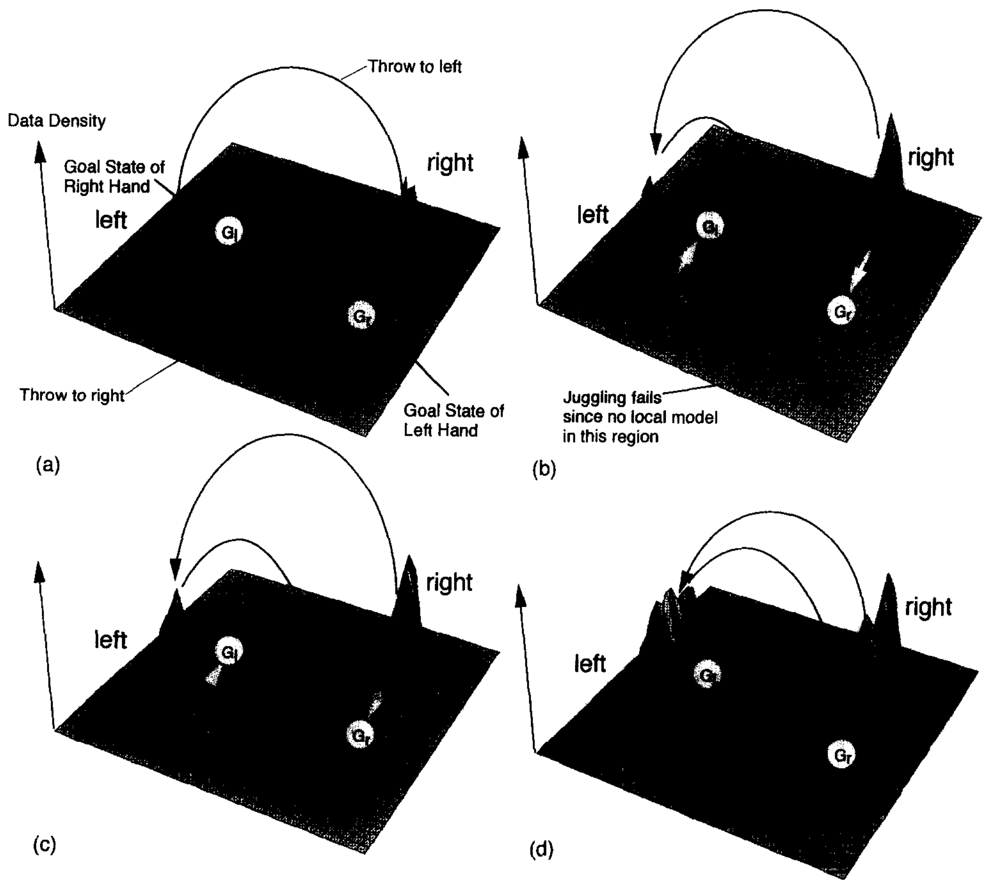


Fig. 8. Abstract illustration how the SSA algorithm collects data in space: (a) sparse data after the first few hits; (b) high local data density due to local control in this region; (c) increased data density on the way to the goals due to shifting the setpoints; (d) ridge of data density after the goal was reached.

the underlying linear models are imprecise. We tested the SSA in a noise corrupted simulation and on the real robot. Learning curves are given in Fig. 9(a) and Fig. 9(b).

The learning curves are typical for the given problem. It takes roughly 40 trials before the setpoint of each hand has moved close enough to the other hand's setpoint. For the simulation a break-through occurs and the robot rarely loses the devilstick after that. The real robot takes more trials to achieve longer juggling runs, and its performance is less consistent. The devil sticking robot is a very fast robot, but its positioning accuracy achieves at most ± 1 cm. Additionally, the direct drive motors do not always deliver the torque as commanded. The simulation does not model such disturbances. It only copes with various levels of Gaussian noise, which is rather well-behaved in comparison to what the real robot experiences.

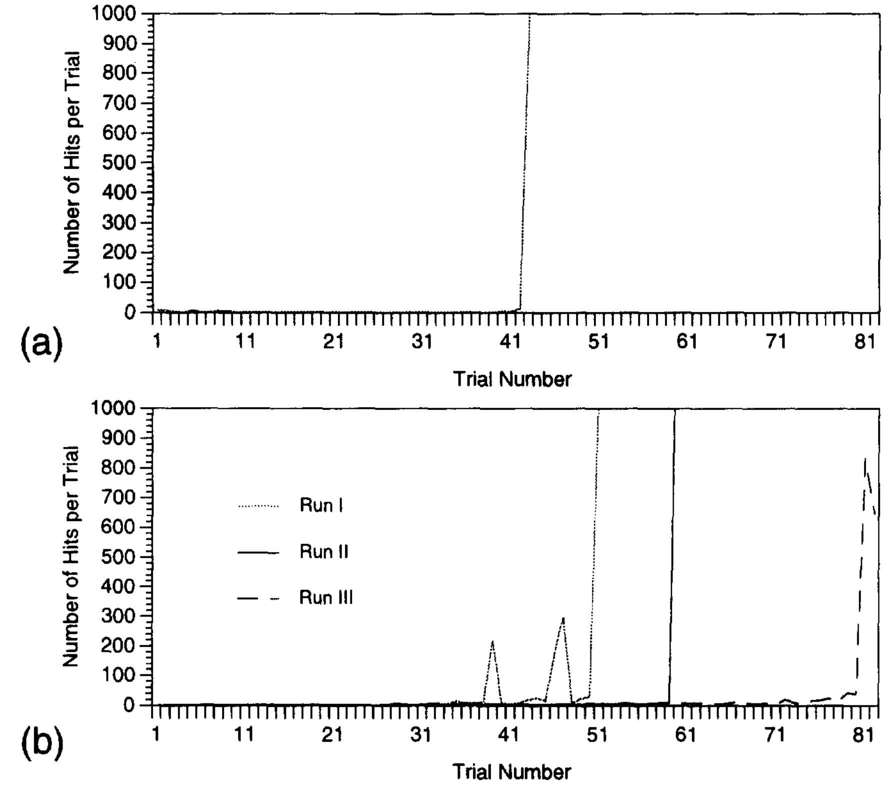


Fig. 9. Learning curves of devil sticking using the SSA algorithm: (a) simulation results, (b) real robot results.

Nevertheless, the real robot learned the task in 50–100 trials and was then able to accomplish runs of more the 2000 consecutive hits. On average, humans need roughly a week of one hour practice a day before they learn to juggle the devilstick. With respect to this, the robot learned rather quickly. Future work will attempt to improve the learning rate and robustness; the results shown stem from very recent work.

11. Discussion

Memory-based neural networks are useful for motor learning. Fast training is achieved by modularizing the network architecture: the memory network does not need to search for weights in order to store the samples, and local models can be linear in the unknown parameters, leading to a single optimum which can be found by linear regression or gradient descent. The combination of storing all the data and only using a certain number of nearby samples to form a local model minimizes interference between old and new data, and allows the range of generalization to depend on the density of the samples.

Table 1
Comparison of parametric and memory-based approaches

	Training	Lookup	Tuning
Nonlinear parametric model	Nonlinear parameter estimation	Cheap	Nonlinear parameter estimation
Memory-based model	Cheap	Linear parameter estimation	Nonlinear parameter estimation

It is useful to compare memory-based function approximation and other nonlinear parametric modeling approaches (Table 1). Training a memory-based model is computationally inexpensive, as the data is simply stored in a memory. Training a nonlinear parametric model typically requires an iterative search for the appropriate parameters. Examples of iterative search are the various gradient descent techniques used to train neural network models. Lookup or evaluating a memory-based model is computationally expensive, as described in this paper. Lookup in a nonlinear parametric model is often relatively inexpensive. If there is a situation in which a fixed set of training data is available, and there will be many queries to the model after the training data is processed, then it makes sense to use a nonlinear parametric model. However, if there is a continuous stream of new training data intermixed with queries, as there typically is in many motor learning problems, it may be less expensive to train and query a memory-based model then it is to train and query a nonlinear parametric model.

A potential disadvantage of the memory-based approach is the limited capacity of the memory network. In this version of the proposed neural network architecture, every experience is stored. Eventually all the memory units will be used up. We have not yet needed to address this issue in our experiments. However, we plan to explore how memory use can be minimized based on several approaches. One approach is to only store ‘surprises’. The system would try to predict the outputs of a data point before trying to store it. If the prediction is good, it is not necessary to store the point. Another approach is to forget data points. Points can be forgotten or removed from the database based on age, proximity to queries, or other criteria. It is an empirical question as to how large a memory capacity is necessary for this network design to be useful. Because memory-based learning retains the original training data, forgetting can be explicitly controlled.

The cross validation approach to optimizing the fit parameters reduces the number of arbitrary choices that need to be made before the training data is collected. However, like other modeling approaches, the choice of representation of the data (number and selection of dimensions to be measured, etc.) play a large role in determining the success of the approach.

In this learning paradigm the feedback controller serves as the teacher, or source of new data for the network. If the feedback controller is of poor quality, the nearest neighbor function approximation method tends to get ‘stuck’ with a non-zero error level. The use of a local model seems to eliminate this stuck state, and reduce the dependence on the quality of the feedback controller.

Much work remains ahead in developing new learning paradigms. We need to develop learning systems that maintain multiple levels of models, allowing generalization via abstract models of the task. We need paradigms that are capable of finding new strategies for a task, and learning and generalizing across multiple tasks. We look forward to paradigms that perform qualitative physical reasoning and guide learning using this information. Finally, careful control of exploration is needed for improvements in learning efficiency.

Acknowledgments

B. Widrow and J.D. Cowan made the authors aware of the work of Steinbuch and Taylor [64,13].

Support was provided under Air Force Office of Scientific Research grant F49-6209410362, and by the ATR Human Information Processing Research Laboratories. Support for Stefan Schaal was provided by the German Scholarship Foundation and the Alexander von Humboldt Foundation. Support for Christopher Atkeson was provided by a National Science Foundation Presidential Young Investigator Award. We thank Gideon Stein for implementing the first version of LWR on the i860 microprocessor, and Gerrie van Zyl for building the devil stick robot and implementing the first version of devil stick learning [70].

References

[1] J.S. Albus, Data storage in the cerebellar model articulation controller (CMAC), *ASME J. Dynamic Systems, Measurement, and Control* 97 (Sep. 1975) 228–233.

[2] J.S. Albus, A new approach to manipulator control: The cerebellar model articulation controller (CMAC), *ASME J. Dynamic Systems, Measurement, and Control* 97 (Sep. 1975) 220–227.

[3] C.H. An, C.G. Atkeson and J.M. Hollerbach, *Model-Based Control of a Robot Manipulator* (MIT Press, Cambridge, MA, 1988).

[4] B.D.O. Anderson and J.B. Moore, *Optimal Control: Linear Quadratic Methods* (Prentice Hall, Englewood Cliffs, NJ, 1990).

[5] C.G. Atkeson, Memory-based approaches to approximating continuous functions, in: M. Casdagli and S. Eubank, eds, *Non-linear Modeling and Forecasting*, Proc. Vol. XII in the Santa Fe Institute Studies in the Sciences of Complexity, pp. 503–521 (Addison Wesley, New York, NY, 1992), Proc. Workshop on Nonlinear Modeling and Forecasting (Sep. 17–21, 1990) Santa Fe, New Mexico.

[6] R.E. Barnhill, Representation and approximation of surfaces, in: J.R. Rice, ed., *Mathematical Software III* (Academic Press, New York, NY, 1977) 69–120.

[7] R.E. Barnhill, R.P. Dube and F.F. Little, Properties of Shepard's surfaces, *Rocky Mountain J. Math.* 13(2) (1983) 365–382.

[8] P.E. Cheng, Strong consistency of nearest neighbor regression function estimators, *J. Multivariate Analysis* 15 (1984) 63–72.

[9] W.S. Cleveland, Robust locally weighted regression and smoothing scatterplots, *J. Amer. Statistical Assoc.* 74 (1979) 829–836.

[10] W.S. Cleveland and S.J. Devlin, Locally weighted regression: An approach to regression analysis by local fitting, *J. Amer. Statistical Assoc.* 83 (1988) 87–114.

[11] W.S. Cleveland, S.J. Devlin and E. Grosse, Regression by local fitting: Methods, properties, and computational algorithms, *J. Econometrics* 37 (1988) 87–114.

[12] T.M. Cover, Estimation by the nearest neighbor rule, *IEEE Trans. Informat. Theory* IT-14 (1968) 50–55.

[13] J.D. Cowan and D.H. Sharp, Neural nets, *Quarterly Rev. Biophysics* 21(3) (1988) 365–427.

[14] I.K. Crain and B.K. Bhattacharyya, Treatment of nonequispaced two dimensional data with a digital computer, *Geoexploration* 5 (1967) 173–194.

[15] J.E. Dennis, D.M. Gay and R.E. Welsch, An adaptive nonlinear least-squares algorithm, *ACM Trans. Math. Software* 7(3) (1981) 173–194.

[16] L. Devroye, On the almost everywhere convergence of nonparametric regression function estimates, *Annals Statistics* 9(6) (1981) 1310–1319.

[17] N.R. Draper and H. Smith, *Applied Regression Analysis* (Wiley, New York, NY, second ed., 1981).

[18] R. Durbin and D.E. Rumelhart, Product units: a computationally powerfull and biologically plausible extension to backpropagation networks, *Neural Computat.* 1 (1989) 133.

[19] R.L. Eubank, *Spline Smoothing and Nonparametric Regression* (Marcel Dekker, New York, NY, 1988).

[20] K.J. Falconer, A general purpose algorithm for contouring over scattered data points, Technical Report NAC 6, National Physical Laboratory, Teddington, Middlesex, UK, TW11 0LW, 1971.

[21] J.D. Farmer and J.J. Sidorowich, Exploiting chaos to predict the future and reduce noise, Technical Report LA-UR-88-901, Los Alamos National Laboratory, Los Alamos, New Mexico, 1988.

[22] J.D. Farmer and J.J. Sidorowich, Predicting chaotic dynamics, in: J.A.S. Kelso, A.J. Mandell and M.F. Schlesinger, eds, *Dynamic Patterns in Complex Systems* (World Scientific, NJ, 1988) 265–292.

[23] R. Farwig, Multivariate interpolation of scattered data by moving least squares methods, in: J.C. Mason and M.G. Cox, eds, *Algorithms for Approximation* (Clarendon Press, Oxford, 1987) 193–211.

[24] E. Fix and J.L. Hodges, Discriminatory analysis, nonparametric discrimination: Consistency properties, in: B.V. Dasarathy, ed., *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques* (IEEE Computer Society Press, Los Alamitos, CA, 1991) 32–39. Originally published as Project 21-49-004, Report No. 4. USAF School of Aviation Medicine Randolph Field, Texas. Contract AF-41-(128)-31, Feb. 1951.

[25] E. Fix and J.L. Hodges, Discriminatory analysis: Small sample performance, in: B.V. Dasarathy, ed., *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques* (IEEE Computer Society Press, Los Alamitos, CA, 1991) 40–56. Originally published as Project 21-49-004, Rep. 11 USAF School of Aviation Medicine Randolph Field, Texas. Contract AF-41-(128)-31, Aug. 1952.

[26] R. Franke, Scattered data interpolation: Tests of some methods, *Math. Computat.* 38(157) (1982) 181–200.

[27] R. Franke and G. Nielson, Smooth interpolation of large sets of scattered data, *Int. J. Numerical Methods Eng.* 15 (1980) 1691–1704.

[28] J.H. Friedman, J.L. Bentley and R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, *ACM Trans. Math. Software* 3(3) (Sep. 1977) 209–226.

[29] K.Y. Goldberg and B. Pearlmutter, Using a neural network to learn the dynamics of the CMU Direct-Drive Arm II, Technical Report CMU-CS-88-160, Carnegie-Mellon University, Pittsburgh, PA, August 1988.

[30] W.J. Gordon and J.A. Wixom, Shepard's method of metric interpolation to bivariate and multivariate interpolation, *Math. Computat.* 32(141) (1978) 253–264.

[31] E. Grosse, LOESS: Multivariate smoothing by moving least squares, in: C.K. Chui, L.L. Schumaker and J.D. Ward, eds, *Approximation Theory VI* (Academic Press, Boston, MA, 1989) 1–4.

[32] F.R. Hampel, P. Rousseeuw, E. Ronchetti and W. Stahel, *Robust Statistics: The Approach Based On Influence Functions* (Wiley International, New York, NY, 1986).

[33] D. Hillis, *The Connection Machine* (MIT Press, Cambridge, MA, 1985).

[34] S. Kawamura and M. Nakagawa, Memory-based control for recognition of motion environment and planning of effective locomotion, in: *IEEE Int. Workshop on Intelligent Robots and Systems, IROS '90* (1990) 303–308.

[35] S. Kawamura, H. Noborio and M. Nakagawa, Hierarchical data structure in memory-based control of robots, in: Okyay Kaynak, ed., *IEEE Int. Workshop on Intelligent Motion Control*, Bogazici University, Istanbul (Aug. 20–22, 1990) 109–114 IEEE Cat. No: 90TH0272-5.

[36] H. Kazmierczak and K. Steinbuch, Adaptive systems in pattern recognition, *IEEE Trans. Electronic Comput.* EC-12 (Dec. 1963) 822–835.

[37] P. Lancaster, Moving weighted least-squares methods, in: B.N. Sahney, ed., *Polynomial and Spline Approximation* (D. Reidel, Boston, MA, 1979) 103–120.

[38] P. Lancaster and K. Šalkauskas, Surfaces generated by moving least squares methods, *Math. Computat.* 37(155) (1981) 141–158.

[39] P. Lancaster and K. Šalkauskas, *Curve and Surface Fitting* (Academic Press, New York, NY, 1986).

[40] M.P.C. Legg and R.P. Brent, Automatic contouring, in: *4th Australian Computer Conf.* (1969) 467–468.

[41] K.C. Li, Consistency for cross-validated nearest neighbor estimates in nonparametric regression, *Annals Statistics*, 12 (1984) 230–240.

[42] G.D. Lodwick and J. Whittle, A technique for automatic contouring field survey data, *Australian Comput. J.* 2 (1970) 104–109.

[43] F.R. Macauley, *The Smoothing of Time Series* (National Bureau of Economic Research, New York, NY, 1931).

[44] D.B. McIntyre, D.D. Pollard and R. Smith, Computer programs for automatic contouring, Technical Report Kansas Geological Survey Computer Contributions 23, University of Kansas, Lawrence, KA, 1968.

[45] D.H. McLain, Drawing contours from arbitrary data points, *Comput. J.* 17(4) (1974) 318–324.

[46] W.T. Miller, F.H. Glanz and L.G. Kraft, Application of a general learning algorithm to the control of robotic manipulators, *Int. J. Robotics Res.* 6 (1987) 84–98.

[47] A.W. Moore and C.G. Atkeson, An investigation of memory-based function approximators for learning control, in preparation.

[48] J.J. More, B.S. Garbow and K.E. Hillstrom, User guide for minpack-1 Technical Report ANL-80-74, Argonne National Laboratory, Argonne, IL, 1980.

[49] H.G. Müller, Weighted local regression and kernel methods for nonparametric curve fitting, *J. Amer. Statistical Assoc.* 82 (1987) 231–238.

[50] R.H. Myers, *Classical and Modern Regression with Applications* (PWS-KENT, Boston, MA, 1990).

[51] J.A.B. Palmer, Automatic mapping, in: *4th Australian Comput. Conf.* (1969) 463–466.

[52] C.R. Pelto, T.A. Elkins and H.A. Boyd, Automatic contouring of irregularly spaced data, *Geophysics*, 33 (1968) 424–430.

[53] R.M. Royall, A class of nonparametric estimators of a smooth regression function PhD dissertation, Stanford University, Department of Statistics, 1966. also published as Tech. Report No. 14, Public Health Service Grant USPHS-5T1 GM 25-09.

[54] M.A. Sabin, Contouring-a review of methods for scattered data, in: K.W. Brodlie, ed., *Mathematical Methods in Computer Graphics and Design* (Academic Press, New York, NY, 1980) 63–86.

[55] S. Schaal and C.G. Atkeson, Assessing the quality of learned local models, in: J.D. Cowan, G. Tesauro and J. Alspector, eds, *Proc. 1993 Neural Information Processing Systems Conf.*, no: 6 in Advances in Neural Information Processing Systems (San Francisco, CA, 1994, Morgan Kaufman) 160–167.

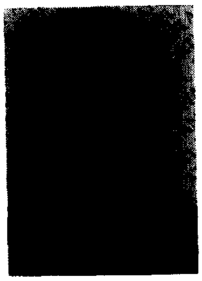
[56] G.A.F. Seber, *Linear Regression Analysis* (Wiley, New York, NY, 1977).

[57] D. Shepard, A two-dimensional function for irregularly spaced data, in: *23rd ACM Nat. Conf.* (1968) 517–524.

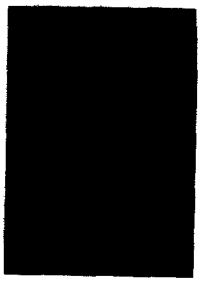
[58] W.F. Sheppard, Reduction of errors by means of negligible differences, in: E.W. Hobson and A.E.H. Love, eds, *Fifth Int. Congress of Mathematicians*, vol. II (Cambridge University Press, 1912) 348–384.

[59] C.W.M. Sherriff, On a class of graduation formulae, *Proc. Royal Soc. Edinburgh* XL (1920) 112–128.

[60] D.E. Specht, A general regression neural network, *IEEE Trans. Neural Networks* 2(6) (1991) 568–576.
[61] C. Stanfill and D. Waltz, Toward memory-based reasoning, *Commun. ACM* 29(12) (Dec. 1986) 1213–1228.
[62] K. Steinbuch, Die lernmatrix, *Kybernetik* 1 (1961) 36–45.
[63] K. Steinbuch and U.A.W. Piske, Learning matrices and their applications, *IEEE Trans. Electronic Comput.* EC-12 (Dec. 1963) 846–862.
[64] K. Steinbuch and B. Widrow, A critical comparison of two kinds of adaptive classification networks, *IEEE Trans. Electronic Comput.* EC-14 (Oct. 1965) 737–740.
[65] C.J. Stone, Nearest neighbor estimators of a nonlinear regression function, in: *Computer Science and Statistics: 8th Annual Symp. on the Interface* (1975) 413–418.
[66] C.J. Stone, Consistent nonparametric regression, *Annals Statistics* 5 (1977) 595–645.
[67] C.J. Stone, Optimal global rates of convergence for nonparametric regression, *Annals Statistics* 10(4) (1982) 1040–1053.
[68] W.K. Taylor, Pattern recognition by means of automatic analogue apparatus, *Proc. Instit. Electrical Engineers* 106B (1959) 198–209.
[69] W.K. Taylor, A parallel analogue reading machine, *Control* 3 (1960) 95–99.
[70] G. van Zyl, Design and control of a robotic platform for machine learning, MS dissertation, Massachusetts Institute of Technology, Mechanical Engineering Department, 1991.
[71] R.F. Walters, Contouring by machine: A user's guide, *Amer. Assoc. Petroleum Geologists Bull.* 53(11) (1969) 2324–2340.
[72] D.L. Waltz, Applications of the connection machine, *Computer* 20(1) (Jan. 1987) 85–97.
[73] G.S. Watson, Smooth regression analysis, *Sankhyā: Indian J. Statistics, Ser. A* 26 (1964) 359–372.
[74] E. Whittaker and G. Robinson, *The Calculus of Observations* (Blackie & Son, London, 1924).



Christopher G. Atkeson received the MS degree in Applied Mathematics (Computer Science) from Harvard University and the PhD degree in Brain and Cognitive Science from M.I.T. He joined the M.I.T. faculty in 1986, and moved to the Georgia Institute of Technology College of Computing in 1994. His research focuses on machine learning, and uses robotics as a domain in which to explore the behavior of learning algorithms. Chris Atkeson is a recipient of a National Science Foundation Presidential Young Investigator Award.



Stefan Schaal received the MS and PhD degrees in Mechanical Engineering from the Technical University of Munich. Since 1991 he has been a Postdoctoral Fellow at the Department of Brain and Cognitive Sciences at M.I.T. His research interests lie in the fields of biological and machine motor control and learning, nonlinear dynamics, and complex systems.