Submitted to: *Applied Intelligence*

Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning

Stefan Schaal ‡⊕
sschaal@usc.edu
www-slab.usc.edu/sschaal

Christopher G. Atkeson *#
cga@cc.gatech.edu
www.cc.gatech.edu/fac/Chris.Atkeson

Sethu Vijayakumar ‡⊕
sethu@usc.edu
www-slab.usc.edu/sschaal /sethu

‡Computer Science and Neuroscience, HNB-103, University of Southern California, Los Angeles, CA 90089-2520
*College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280
⊕Kawato Dynamic Brain Project (ERATO/JST), 2-2 Hikaridai, Seika-cho, Soraku-gun, 619-02 Kyoto, Japan
#ATR Human Information Processing Laboratories, 2-2 Hikaridai, Seika-cho, Soraku-gun, 619-02 Kyoto, Japan

***Abstract**: Locally weighted learning (LWL) is a class of techniques from nonparametric statistics that provides useful representations and training algorithms for learning about complex phenomena during autonomous adaptive control of robotic systems. This paper introduces several LWL algorithms that have been tested successfully in real-time learning of complex robot tasks. We discuss two major classes of LWL, memory-based LWL and purely incremental LWL that does not need to remember any data explicitly. In contrast to the traditional belief that LWL methods cannot work well in high-dimensional spaces, we provide new algorithms that have been tested on up to 90 dimensional learning problems. The applicability of our LWL algorithms is demonstrated in various robot learning examples, including the learning of devil-sticking, pole-balancing by a humanoid robot arm, and inverse-dynamics learning for a seven and a 30 degree-of-freedom robot. In all these examples, the application of our statistical neural networks techniques allowed either faster or more accurate acquisition of motor control than classical control engineering.*

***Keywords**: Nonparametric Regression • Locally Weighted Learning • Motor Control • Internal Models • Incremental Learning*

1 Introduction

The necessity for self-improvement in control systems is becoming more apparent as fields such as robotics, factory automation, and autonomous vehicles become impeded by the complexity of inventing and programming satisfactory control laws. Learned models of complex tasks can aid the design of appropriate control laws for these tasks, which often involve decisions based on streams of information from sensors and actuators, and where data is relatively plentiful. Learning also seems to be the only viable research approach toward the generation of flexible autonomous robots that can perform multiple tasks (Schaal, 1999), with the hope of creating an autonomous humanoid robot at some point and to approach human abilities in sensorimotor control.

When approaching a learning problem, many alternative learning methods are available from the neural network, statistical, and machine learning literature. The current focus in learning research lies on increasingly more sophisticated algorithms for the offline analysis of finite data sets, without severe constraints on the computational complexity of the algorithms. Examples of such algorithms include the revival of Bayesian inference (Bishop, 1995, Williams and Rasmussen, 1996) and the new algorithms developed in the framework of structural risk minimization (Vapnik, 1982, Cortes and Vapnik, 1995). Mostly, these methods target problems in classification and diagnostics, although several extensions to regression problems exist (e.g., Vapnik, Golowich, and Smola, 1996).

In motor learning, however, special constraints need to be taken into account when approaching a learning task. Most learning problems in motor learning require regression networks, for instance, as in the learning of internal models, coordinate transformations, control policies, or evaluation functions in reinforcement learning. Data in motor learning is usually not limited to a finite data set—whenever the robot moves new data are generated and should be included in the learning network. Thus, computationally inexpensive training methods are important in this domain, and on-line learning would be preferred. Among the most significant additional problems of motor learning is that the distributions of the learning data may change continuously. Input distributions change due to the fact that a flexible movement system may work on different tasks at different times, thus creating different kinds of training data. Moreover, the input-output relationship of the data—the conditional distribution—may change when the learning system changes its physical properties or when learning involves nonstationary training data as in reinforcement learning. Such changing distributions easily lead to catastrophic interference in many neural network paradigms, i.e., the unlearning of useful information when training on new data (Schaal and Atkeson, 1998). As a last element, motor learning tasks of complex motor systems can be rather high dimensional in the number of input dimensions, thus increasing the need for efficient learning algorithms. The current trend in learning research is largely orthogonal to the problems of motor learning.

In this paper, we advocate locally weighted learning methods (LWL) for motor learning, a learning technique derived from nonparametric statistics (Atkeson and Schaal, 1995, Cleveland and Loader, 1995, Hastie and Tibshirani, 1994). LWL provides a principled approach to learning models of complex phenomena, dealing with large amounts of data, training quickly, and avoiding interference between multiple tasks during control of complex systems (Atkeson, Moore, and Schaal, 1997a, Atkeson, Moore, and Schaal, 1997b). LWL methods can even deal successfully with high dimensional input data that have redundant and irrelevant inputs while keeping the computational complexity of the algorithms linear in the number of inputs. LWL methods come in two different strategies. Memory-based LWL is a "lazy learning" method (Aha, 1997) that simply stores all training data in memory and uses efficient lookup and interpolation techniques when a prediction for a new input has to be generated. This kind of LWL is useful when data need to be interpreted in flexible ways, for instance, as for-

ward *or* inverse transformations. Memory-based LWL is therefore a "least commitment" approach and very data efficient. Non-memory-based LWL has essentially the same statistical properties as memory-based LWL, but it avoids storing data in memory by using recursive system identification techniques (Ljung and Söderström, 1986). In this way, non-memory-based LWL caches the information about training data in compact representations, at the cost that a flexible re-evaluation of data becomes impossible, but lookup times for new data become significantly faster.

In the following, we will describe four LWL algorithms that are the most suitable to robot learning problems. The goal of the next section is to provide clear pseudo-code explanations of these algorithms. Afterwards, we will illustrate the successful application of some of the methods to real-time robot learning, involving dexterous manipulation tasks such as devil sticking and pole balancing with an anthropomorphic robot arm, and classical problems like the learning of high-dimensional inverse dynamics models.

2 Locally Weighted Learning

In all our algorithms we assume that the data generating model for our regression problems has the standard form $y = f(\mathbf{x}) + \varepsilon$, where $\mathbf{x} \in \Re^n$ is a n-dimensional input vector, the noise term has mean zero, $E\{\varepsilon\} = 0$, and, for simplicity, the output is one-dimensional.

The key concept of our LWL methods is to approximate nonlinear functions by means of piecewise linear models (Cleveland, 1979), similar to a first-order Taylor series expansion. Locally linear models have been demonstrated to be an excellent statistical compromise among the possible local polynomials that can be fit to data (Hastie and Loader, 1993). The key problem in LWL is to determine the region of validity in which a local model can be trusted, and how to fit the local model in this region.

In all following algorithms, we compute the region of validity, called a *receptive field*, of each linear model from a Gaussian kernel:

$$w_k = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{c}_k)^T \mathbf{D}_k (\mathbf{x}-\mathbf{c}_k)\right) \tag{1}$$

where $\mathbf{c}_k$ is the center of the k^{th} linear model, and $\mathbf{D}_k$ corresponds to a positive semi-definite distance metric that determines the size and shape of region of validity of the linear model. Other kernel functions are possible (Atkeson et al., 1997a) but add only minor differences to the quality of function fitting.

2.1 Locally Weighted Regression

The most straightforward LWL algorithm with locally linear models is memory-based Locally Weighted Regression (LWR) (Atkeson, 1989, Atkeson, 1992, Atkeson and Schaal, 1995). Training of LWR is very fast: it just requires adding new training data to

the memory. Only when a prediction is needed for a query point $\mathbf{x}_q$, the following weighted regression analysis is performed:

The LWR Algorithm:

Given:

A query point $\mathbf{x}_q$ and p training points $\{\mathbf{x}_i, y_i\}$ in memory

Compute Prediction:

a) compute diagonal weight matrix $\mathbf{W}$

$$\text{where } w_{ii} = \exp\left(-\frac{1}{2}\left(\mathbf{x}_i - \mathbf{x}_q\right)^T \mathbf{D}\left(\mathbf{x}_i - \mathbf{x}_q\right)\right)$$

b) build matrix $\mathbf{X}$ and vector $\mathbf{y}$ such that (2)

$$\mathbf{X} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_p)^T \text{ where } \tilde{\mathbf{x}}_i = \left[\left(\mathbf{x}_i - \mathbf{x}_q\right)^T \; 1\right]^T$$

$$\mathbf{y} = \left(y_1, y_2, \ldots, y_p\right)^T$$

c) compute locally linear model

$$\beta = \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y}$$

d) the prediction for $\mathbf{x}_q$ is thus

$$\hat{y}_q = \beta_{n+1}$$

β_{n+1} denotes the (n+1)$^{\text{th}}$ element of the regression vector β. The computational complexity of LWR is proportional to $O(pn^2)$. Since normally most of the p training data points receive an approximately zero weight as they are too far away from the query point, the computational complexity of LWR can be reduced significantly, particularly when exploiting efficient data structures like *kd*-trees for keeping the data in memory (Moore, 1990). Thus, LWR can be applied efficiently in real-time for problems that are not too high dimensional in the number of inputs n and that do not accumulate too much data in one particular area of the input space.

The only open parameter in (2) is the distance metric $\mathbf{D}$, introduced in Equation (1). After there is a significant amount of data, $\mathbf{D}$ can be optimized by leave-one-out cross validation. To avoid too many open parameters, $\mathbf{D}$ is usually assumed to be a global diagonal matrix $\mathbf{D} = h \cdot diag([n_1, n_2, \ldots, n_n])$, where h is a scale parameter, and the n_i normalize the range of the input dimensions, e.g., by the variance of each input dimension $n_i = 1/\sigma_i^2$. Leave-one-out crossvalidation is thus performed only as a one-dimensional search over the parameter h:

Leave – One – Out Cross Validation:

Given:

a set H of reasonable values h_r

Algorithm:

For all $h_r \in H$:

$sse_r = 0$

For i = 1:p (3)

a) $\mathbf{x}_q = \mathbf{x}_i$

b) temporarily exclude $\{\mathbf{x}_i, y_i\}$ from training data

c) compute LWR prediction $\hat{y}_q$ with reduced data

d) $sse_r = sse_r + \left(y_i - \hat{y}_q\right)^2$

Choose the optimal h_r^* such that $h_r^* = \min_r \{sse_r\}$

Of course, at an increased computational cost, leave-one-out cross validation can also be performed treating all coefficients of the distance metric as open parameters, usually by using gradient descent methods (Atkeson, 1992; Atkeson and Schaal, 1995; Lowe, 1995).

2.2 Locally Weighted Partial Least Squares

Under two circumstances it is necessary to enhance the LWR algorithm above: if the number of input dimensions grows large, or if there are redundant input dimensions such that the matrix inversion in (2) becomes numerically unstable. There is a computational efficient technique from the statistics literature, Partial Least Squares Regression (PLS) (Wold, 1975, Frank and Friedman, 1993, Schaal, Vijayakumar, and Atkeson, 1998, Vijayakumar and Schaal, 2000), that is ideally suited to reduce the computational complexity of LWR and to avoid numerical problems. The essence of PLS is to fit linear models using a hierarchy of univariate regressions along selected projections in input space. The projections are chosen according to the correlation of input and output data, and the algorithm assures that subsequent projections are orthogonal in input space. It is straightforward to derive a locally weighted PLS algorithm (LWPLS), as shown in Equation (4). The only steps in LWPLS that may look unusual at a first glance are the ones indicated by (*) and (**) in Equation (4). At these steps, the input data is regressed against the current projection s (*), and subsequently, the input space is reduced (**), This procedure ensures that the next projection direction $\mathbf{u}_{i+1}$ is guaranteed to be orthogonal with respect to all previous projection directions.

The LWPLS Algorithm:

Given:

a query point $\mathbf{x}_q$ and p training points $\{\mathbf{x}_i, y_i\}$ in memory

Compute Prediction:

a) compute diagonal weight matrix $\mathbf{W}$

where $w_{ii} = \exp\left(-\frac{1}{2}\left(\mathbf{x}_i - \mathbf{x}_q\right)^T \mathbf{D}\left(\mathbf{x}_i - \mathbf{x}_q\right)\right)$

b) build the matrix $\mathbf{X}$ and vector $\mathbf{y}$ such that

$$\bar{\mathbf{x}} = \sum_{i=1}^{p} w_{ii}\mathbf{x}_i \Big/ \sum_{i=1}^{p} w_{ii};\ \beta_0 = \sum_{i=1}^{p} w_{ii} y_i \Big/ \sum_{i=1}^{p} w_{ii};$$

$$\mathbf{X} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_p)^T \text{ where } \tilde{\mathbf{x}}_i = \left(\mathbf{x}_i - \bar{\mathbf{x}}\right)$$

$$\mathbf{y} = \left(\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_p\right)^T \text{ where } \tilde{y}_i = \left(y_i - \beta_0\right)$$

c) recursively compute locally linear model

Initialize: $\mathbf{Z}_0 = \mathbf{X}$, $\mathbf{res}_0 = \mathbf{y}$

For i = 1:r (4)

$$\mathbf{u}_i = \mathbf{Z}_{i-1}^T \mathbf{W}\,\mathbf{res}_{i-1}$$

$$\mathbf{s}_i = \mathbf{Z}_{i-1}\mathbf{u}_i$$

$$\beta_i = \frac{\mathbf{s}_i^T \mathbf{W}\,\mathbf{res}_{i-1}}{\mathbf{s}_i^T \mathbf{W}\mathbf{s}_i}$$

$$\mathbf{p}_i = \frac{\mathbf{s}_i^T \mathbf{W}\mathbf{Z}_{i-1}}{\mathbf{s}_i^T \mathbf{W}\mathbf{s}_i} \quad (*)$$

$$\mathbf{res}_i = \mathbf{res}_{i-1} - \mathbf{s}_i \beta_i$$

$$\mathbf{Z}_i = \mathbf{Z}_{i-1} - \mathbf{s}_i \mathbf{p}_i \quad (**)$$

d) the prediction for $\mathbf{x}_q$ is thus

Initialize: $\mathbf{z}_0 = \mathbf{x}_q - \bar{\mathbf{x}}, \hat{y}_q = \beta_0$

For i = 1:r

$$s_i = \mathbf{z}_{i-1}^T \mathbf{u}_i$$

$$\hat{y}_q \leftarrow \hat{y}_q + s_i \beta_i$$

$$\mathbf{z}_i = \mathbf{z}_{i-1} - s_i \mathbf{p}_i^T$$

There is a remarkable property of LWPLS: if the input data is locally statistically independent (i.e., has a diagonal covariance matrix) and is approximately locally linear, LWPLS will find an optimal linear approximation for the data with a *single* projection

(Vijayakumar and Schaal, 2000). This statement is true since LWPLS will immediately find the optimal projection direction, i.e., the gradient of the data. An important question is thus how many projections r should be chosen if the input data are not statistically independent. Typically, the squared error res_i^2 at iteration i should be significantly lower than that of the previous step. Thus, a simple heuristic to stop adding projections is to require that for every new projection, the squared error be reduced at least by a certain ratio:

$$\frac{res_i^2}{res_{i-1}^2} < \phi, \text{ where } \phi \in [0,1] \tag{5}$$

We usually use $\phi = 0.5$ for all our learning tasks. Thus, as in LWR, the only open parameter in LWPLS becomes the distance metric $\mathbf{D}$, which can be optimized according to the strategy in (3).

The computational complexity of LWPLS is $O(rnp)$. If one assumes that most of the data has a zero weight and that only a fixed number of projections are needed to achieve a good fit, the computational complexity tends towards linear in the number of input dimensions. This property constitutes a significant saving over the more than quadratic cost of LWR, particularly in high dimensional input spaces. Additionally, the correlation step to select the projection direction eliminates irrelevant and redundant input dimensions and results in excellent numerical robustness of LWPLS.

2.3 Receptive Field Weighted Regression

Two points of concern remain with LWR and LWPLS. If the learning system receives a large, possibly never ending stream of input data, as is typical in online robot learning, both memory requirements to store all data as well as the computational cost to evaluate algorithms (2) or (4) become too large. Under these circumstances, a non-memory-based version of LWL is desirable such that each new data point is incrementally incorporated in the learning system and lookup speed becomes accelerated.

A first approach to an incremental LWL algorithm was suggested in previous work (Schaal and Atkeson, 1998), using LWR as the starting point. The idea of the algorithm is straightforward: instead of postponing the computation of a local linear model until a prediction needs to be made, local models are built continuously in the entire support area of the input data at selected points in input space (see details below). The prediction for a query point is then formed as the weighted average of the predictions of all local models:

$$\hat{y}_q = \frac{\sum_{k=1}^{K} w_k \hat{y}_{q,k}}{\sum_{k=1}^{K} w_k} \tag{6}$$

The weights in (6) are computed according to the weighting kernel of each local model in Equation (7). Incremental updates of the parameters of the linear models can be accomplished with recursive least squares techniques (Ljung and Söderström, 1986). Thus,

the LWR algorithm from (2) becomes the Receptive Field Weighted Regression (RFWR) algorithm (Schaal and Atkeson, 1998) as given in the following equations (7).

The RFWR Algorithm:

Given:

A training point $(\mathbf{x}, y)$

Update all K local models:

$$w_k = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{c}_k)^T \mathbf{D}_k (\mathbf{x}-\mathbf{c}_k)\right) \tag{7}$$

$$\beta_k^{n+1} = \beta_k^n + w_k \mathbf{P}_k^{n+1} \tilde{\mathbf{x}} e_{cv,k} \quad \text{where } \tilde{\mathbf{x}} = \begin{bmatrix} (\mathbf{x}-\mathbf{c}_k)^T & 1 \end{bmatrix}$$

$$\text{and} \quad \mathbf{P}_k^{n+1} = \frac{1}{\lambda}\left(\mathbf{P}_k^n - \frac{\mathbf{P}_k^n \tilde{\mathbf{x}} \tilde{\mathbf{x}}^T \mathbf{P}_k^n}{\frac{\lambda}{w_k} + \tilde{\mathbf{x}}^T \mathbf{P}_k^n \tilde{\mathbf{x}}}\right) \quad \text{and} \quad e_{cv,k} = \left(y - \beta_k^{nT} \tilde{\mathbf{x}}\right)$$

Compute Prediction for query point $\mathbf{x}_q$:

$$\hat{y}_k = \beta_k^T \tilde{\mathbf{x}}_q$$

In the above equations, $\lambda \in [0,1]$ is a forgetting factor that determines how much old data in the regression parameters will be forgotten, similar as in recursive system identification techniques (Ljung and Söderström, 1986). The variables $\mathbf{P}_k$ cache the inverse of the covariance matrix of the input variables and needs to be initialized as $\mathbf{P}_k = \mathbf{I}\,r$, where $\mathbf{I}$ is the identity matrix and r is usually a large number, e.g., 10^{10}.

Thus, as in all the previous LWL algorithms, the only remaining open parameter is the distance metric $\mathbf{D}$. In contrast to the algorithm in (3) that only determined $\mathbf{D}$ as a global parameter to be used everywhere in input space, it is now possible to optimize the $\mathbf{D}_k$ for every local model individually. In (Schaal and Atkeson, 1998) we developed an incremental optimization of $\mathbf{D}$ by means of gradient descent in a stochastic leave-one-out crossvalidation criterion. The following equations summarize the update formulae. Since the gradient descent update is the same for all local models, the index k is omitted.

RFWR Gradient descent update of D:

Cost Function for Gradient Descent:

$$J = \frac{1}{\sum_{i=1}^{p} w_i} \sum_{i=1}^{p} \frac{w_i \left\| \mathbf{y}_i - \hat{\mathbf{y}}_i \right\|^2}{\left(1 - w_i \tilde{\mathbf{x}}_i^T \mathbf{P} \tilde{\mathbf{x}}_i\right)^2} + \gamma \sum_{i,j=1}^{n} D_{ij}^2$$

Recursive Update Algorithm:

$$\mathbf{M}^{n+1} = \mathbf{M}^n - \alpha \frac{\partial J}{\partial \mathbf{M}} \quad \text{with} \quad \mathbf{D} = \mathbf{M}^T\mathbf{M}, \text{where } \mathbf{M} \text{ is upper triangular}$$

$$W^{n+1} = \lambda W^n + w$$

$$E^{n+1} = \lambda E^n + w\, e_{cv}^2$$

$$\mathbf{H}^{n+1} = \lambda \mathbf{H}^n + \frac{w\,\tilde{\mathbf{x}}\, e_{cv}}{1-h}, \quad \text{where} \quad h = w\,\tilde{\mathbf{x}}^T \mathbf{P}^{n+1}\,\tilde{\mathbf{x}}$$

$$\mathbf{R}^{n+1} = \lambda \mathbf{R}^n + \frac{w^2\, e_{cv}^2 \tilde{\mathbf{x}}\,\tilde{\mathbf{x}}^T}{1-h}$$

$$\frac{\partial J}{\partial \mathbf{M}} \approx \frac{\partial w}{\partial \mathbf{M}} \sum_{i=1}^{p} \frac{\partial J_{1,i}}{\partial w} + \frac{w}{W^{n+1}} \frac{\partial J_2}{\partial \mathbf{M}} \tag{8}$$

where:

$$\frac{\partial w}{\partial M_{rl}} = -\frac{1}{2} w (\mathbf{x}-\mathbf{c})^T \frac{\partial \mathbf{D}}{\partial M_{rl}} (\mathbf{x}-\mathbf{c}), \quad \frac{\partial J_2}{\partial M_{rl}} = 2\gamma \sum_{i,j=1}^{n} D_{ij} \frac{\partial D_{ij}}{\partial M_{rl}}$$

$$\frac{\partial D_{ij}}{\partial M_{rl}} = \delta_{lj} M_{ri} + \delta_{il} M_{rj} \quad (\delta \text{ is the Kronecker operator})$$

$$\sum_{i=1}^{p} \frac{\partial J_{1,i}}{\partial w} \approx -\frac{E^{n+1}}{\left(W^{n+1}\right)^2} +$$

$$\frac{1}{W^{n+1}} \left(e_{cv}^2 - \left(2\,\mathbf{P}^{n+1}\,\tilde{\mathbf{x}} \left(y - \tilde{\mathbf{x}}^T \beta^{n+1} \right) \right) \otimes \mathbf{H}^n - \left(2\,\mathbf{P}^{n+1}\,\tilde{\mathbf{x}}\,\tilde{\mathbf{x}}^T \mathbf{P}^{n+1} \right) \otimes \mathbf{R}^n \right)$$

where the operator $\otimes$ denotes an element-wise multiplication of two homomorphic matrices or vectors with a subsequent summation of all coefficients, $\mathbf{Q} \otimes \mathbf{V} = \Sigma Q_{ij} V_{ij}$. All recursive variables are initialized with zeros, except for the initial distance metric that is set to a manually chosen default value $\mathbf{D}=\mathbf{D}_{def}$.

The above learning rules can be embedded in an incremental learning system that automatically allocates new locally linear models as needed (Schaal and Atkeson, 1998):

Initialize RFWR with no receptive field (RF);
For every new training sample (**x**,*y*):
  For *k*=1 to *K*:
    calculate the activation from (1)
    update according to (7) and (8)
  end;
  If no linear model was activated by more than w_{gen};
    create a new RF with **c**=**x**, **D**=$\mathbf{D}_{def}$
  end;
end;

In this pseudo-code algorithm, w_{gen} is a threshold that determines when to create a new receptive field, e.g., w_{gen}=0.1, and $\mathbf{D}_{def}$ is the initial (usually diagonal) distance metric in Equation (1). The decomposition of $\mathbf{D}$ into $\mathbf{M}^T\mathbf{M}$ can be achieved with a Cholesky decomposition (Press, Flannery, Teukolsky, and Vetterling, 1989).

2.4 Locally Weighted Projection Regression

While RFWR is a powerful algorithm for incremental function approximation, it becomes computationally too expensive in high dimensional spaces. For such cases, LWPLS can be reformulated as an incremental algorithm, too, called Locally Weighted Projection Regression (LWPR). The update equations for one local model become:

Locally Weighted Projection Regression

Given: A training point $(\mathbf{x}, y)$

Update the means of inputs and output:

$$\mathbf{x}_0^{n+1} = \frac{\lambda W^n \mathbf{x}_0^n + w\,\mathbf{x}}{W^{n+1}}$$

$$\beta_0^{n+1} = \frac{\lambda W^n \beta_0^{n+1} + w\,y}{W^{n+1}}$$

where $W^{n+1} = \lambda W^n + w$

Update the local model:

Initialize: $\mathbf{z}_0 = \mathbf{x} - \mathbf{x}_0^{n+1}$, $res_0 = y - \beta_0^{n+1}$

For $i = 1{:}\,r$,

a) $\mathbf{u}_i^{n+1} = \lambda\,\mathbf{u}_i^n + w\,\mathbf{z}_{i-1}\,res_{i-1}$ (9)

b) $s_i = \mathbf{z}_{i-1}^T \mathbf{u}_i^{n+1}$

c) $SS_i^{n+1} = \lambda\,SS_i^n + w\,s_i^2$

d) $SR_i^{n+1} = \lambda\,SR_i^n + w\,s_i^2\,res_{i-1}$

e) $SZ_i^{n+1} = \lambda\,SZ_i^n + w\,\mathbf{z}_{i-1}\,s_i$

f) $\beta_i^{n+1} = SR_i^{n+1} / SS_i^{n+1}$

g) $\mathbf{p}_i^{n+1} = SZ_i^{n+1} / SS_i^{n+1}$

h) $\mathbf{z}_i = \mathbf{z}_{i-1} - s_i \mathbf{p}_i^{n+1}$

i) $res_i = res_{i-1} - s_i \beta_i^{n+1}$

j) $SSE_i^{n+1} = \lambda\,SSE_i^n + w\,res_i^2$

In the above equations, the variables *SS*, *SR*, and *SZ* are memory terms that enable us to achieve the univariate regression in step f) in a recursive least squares fashion, i.e., a fast Newton-like method as in RFWR (cf. Equation (7)). The other steps are incremental

counterparts of the LWPLS algorithm above. Step j) computes the sum of squared errors that is used to determine when to stop adding projections according to (5). Predictions for a query point are formed exactly as in Eqn (4)-d.

In analogy with RFWR, an incremental update of the distance metric **D** can be derived based on stochastic one-leave-out cross validation. Due to the hierarchical fitting procedure of PLS, the one-leave-out crossvalidation cost function can only be formulated in approximation by treating all projections as independent:

LWPR Gradient descent update of D:

Cost Function for Gradient Descent:

$$J = \frac{1}{\sum_{i=1}^{p} w_i} \sum_{i=1}^{p} \sum_{k=1}^{r} \frac{w_i res_{k,i}^2}{\left(1 - w_i \frac{s_{k,i}^2}{\mathbf{s}_k^T \mathbf{W} \mathbf{s}_k}\right)^2} + \gamma \sum_{i,j=1}^{n} D_{ij}^2$$

Recursive Update Algorithm:

$$\mathbf{M}^{n+1} = \mathbf{M}^n - \alpha \frac{\partial J}{\partial \mathbf{M}} \quad \text{with } \mathbf{D} = \mathbf{M}^T \mathbf{M}, \text{where } \mathbf{M} \text{ is upper triangular}$$

$$W^{n+1} = \lambda W^n + w$$

$$E_k^{n+1} = \lambda E_k^n + w\, res_{k-1}^2$$

$$\mathbf{H}_k^{n+1} = \lambda \mathbf{H}_k^n + \frac{w\, s_k\, res_{k-1}}{1 - h_k}, \quad \text{where} \quad h_k = w \frac{s_k^2}{SS_k^{n+1}}$$

$$\mathbf{R}_k^{n+1} = \lambda \mathbf{R}_k^n + \frac{w^2\, res_{k-1}^2 s_k^2}{1 - h_k} \tag{10}$$

$$\frac{\partial J}{\partial \mathbf{M}} \approx \frac{\partial w}{\partial \mathbf{M}} \sum_{i=1}^{p} \sum_{k=1}^{r} \frac{\partial J_{1,r,i}}{\partial w} + \frac{w}{n\, W^{n+1}} \frac{\partial J_2}{\partial \mathbf{M}}$$

where:

$$\frac{\partial w}{\partial M_{rl}} = -\frac{1}{2} w (\mathbf{x} - \mathbf{c})^T \frac{\partial \mathbf{D}}{\partial M_{rl}} (\mathbf{x} - \mathbf{c}), \quad \frac{\partial J_2}{\partial M_{rl}} = 2\gamma \sum_{i,j=1}^{n} D_{ij} \frac{\partial D_{ij}}{\partial M_{rl}}$$

$$\frac{\partial D_{ij}}{\partial M_{rl}} = \delta_{lj} M_{ri} + \delta_{il} M_{rj} \quad (\delta \text{ is the Kronecker operator})$$

$$\sum_{i=1}^{p} \sum_{k=1}^{r} \frac{\partial J_{1,r,i}}{\partial w} \approx \sum_{k=1}^{r} \left[-\frac{E_k^{n+1}}{\left(W^{n+1}\right)^2} + \frac{1}{W^{n+1}} \left(res_{k-1}^2 - 2 \frac{res_k s_k}{SS_k^{n+1}} \mathbf{H}_k^n - 2 \left(\frac{s_k}{SS_k^{n+1}} \right)^2 \mathbf{R}_k^n \right) \right]$$

Thus, the ensuing learning algorithm is the same as for RFWR by using the update equations (9) and (10) instead of (7) and (8), and by initializing a new receptive field with r=2, i.e., two initial projections in order to allow deciding whether to add new projections according to Equation (5). For a diagonal distance metric $\mathbf{D}_k$ and under the

assumption that r remains small, the computational complexity of the update of all parameters of LWPR is linear in the number of input dimensions. Thus, LWPR constitutes the computationally most efficient algorithm in the series of Locally Weighted Learning algorithms, and, as will be demonstrated below, even works successfully in very high-dimensional input spaces.

3 Empirical Evaluations

The following examples illustrate how our LWL techniques have been applied to various robot learning problems. In common to all these application is that the learning algorithm acquired an internal model of a dynamic system, and that this model was used subsequently in designing a controller for a robot task. Such model-based control and learning has also become increasingly more an accepted hypothesis of how biological movements acquire and perform skilled movement (Kawato, 1999; Wolpert, Miall, and Kawato, 1998).

3.1 Learning Devil-Sticking

Devil sticking is a juggling task where a center stick is batted back and forth between two handsticks (Figure 1a). Figure 1b shows a sketch of our devil-sticking robot. The robot uses its top two joints to perform planar devil-sticking; more details can be found in (Schaal and Atkeson, 1994). The task of the robot is to learn a continuous left-right-left-etc. juggling pattern. For learning, the task is modeled as a discrete function that maps impact states on one hand to impact states on the other hand. A state is given as a five dimensional vector $\mathbf{x} = (p, \theta, \dot{x}, \dot{y}, \dot{\theta})^T$, comprising impact position, angle, and velocities of the center of the devil stick and angular velocity (Figure 1b), respectively. The task command $\mathbf{u} = (x_h, y_h, \dot{\theta}_t, v_x, v_y)^T$ is given by a catch position (x_h, y_h), an angular trigger velocity $(\dot{\theta}_t)$when to start the throw, and the two dimensional throw direction (v_x, v_y). In order to compute appropriate Linear Quadratic Regulator (LQR) controllers for this task (Dyer and McReynolds, 1970), the robot learns the nonlinear mapping between the current state and command, and the next state, i.e., a 10 dimensional input to five dimensional output function. This task is ideally suited for LWR as it is not too high dimensional and new training data are only generated at about 1-2Hz, i.e., whenever the center stick hits one of the handsticks. Moreover, memory-based learning also supports efficient search of the state-action space for statistically good new commands (Schaal and Atkeson, 1994). As a result, successful (i.e., more than 1000 consecutive hits) devil sticking could be achieved in about 40-80 trials, corresponding to about 300-800 training points in memory (Figure 2). This is a remarkable learning rate given that humans need about one week of 1 hour practicing a day before they learn to juggle the devilstick. The final performance of the robot was also quite superior to our early attempts to implement the task based on classical system identification and controller design, which only accomplished up to 100 consecutive hits.

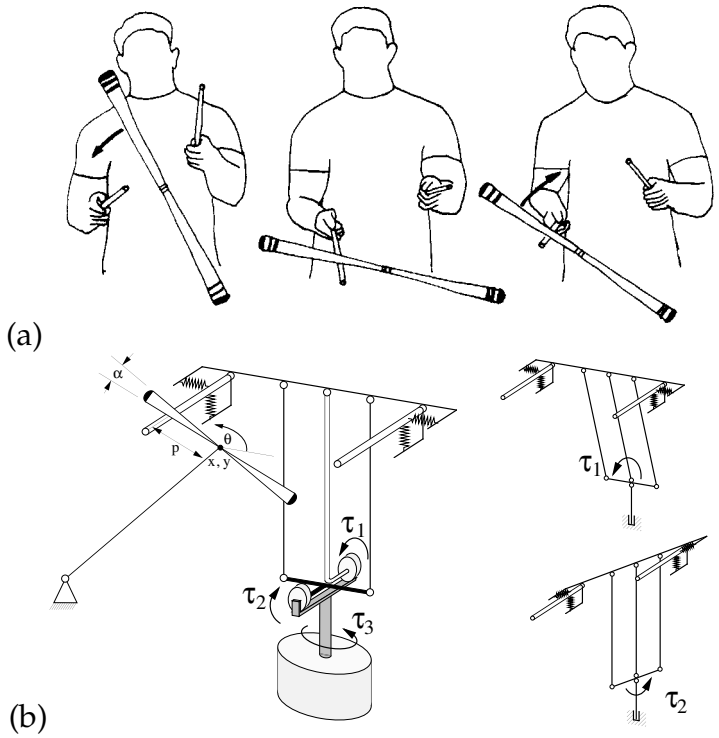

Figure 1: (a) an illustration of devil sticking, (b) sketch of our devil sticking robot: the flow of force from each motor into the robot is indicated by different shadings of the robot links, and a position change due to an application of motor 1 or motor 2, respectively, is indicated in the small sketches

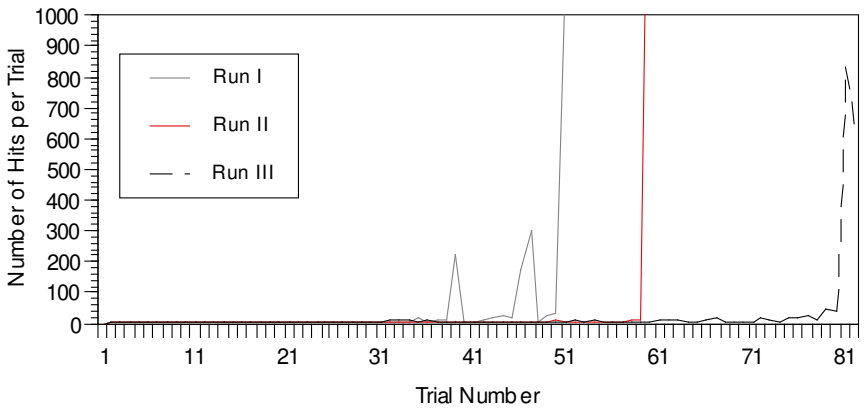

Figure 2: Learning curves of devil sticking for three learning runs.

3.2 Learning Pole Balancing

We implemented learning of the task of balancing a pole on a fingertip with a 7-degree-of-freedom anthropomorphic robot arm (Figure 3a). The low-level robot controller ran in a compute-torque mode (Craig, 1986) at 480Hz out of 8 parallel processors located in a VME bus, running the real-time operating system vxWorks. The inverse dynamics

model of the robot had been estimated using established parameter estimation techniques from control theory (An, Atkeson, and Hollerbach, 1988). The goal of learning was to generate appropriate task level commands, i.e., Cartesian accelerations of the fingertip, to keep the pole upright. Task level commands were converted to actuator space by means of the extended Jacobian method (Baillieul and Martin, 1990). As input, the robot received data from its color-tracking-based stereo vision system with more than 60ms processing delays. Learning was implemented on-line using RFWR. The task of RFWR was to acquire a discrete time forward dynamics model of the pole that was both used to compute an LQR controller and to realize a Kalman predictor to eliminate the delays in visual input. The forward model had 12 input dimensions (3 positions of the lower pole end, 2 angular positions, the corresponding 5 velocities, and 2 horizontal accelerations of the fingertip) that are mapped to 10 outputs, i.e., the next state of the pole. The robot only received training data when it actually moved.

Figure 3b shows the results of learning. It took about 10-20 trials before learning succeeded in reliable performance longer than one minute. We also explored learning from demonstration, where a human demonstrated how to balance a pole for 30 seconds while the robot was learning the forward model by just "watching". From the demonstration, the robot could extract the forward dynamics model of the pole-balancing task and synthesize the LQR controller. Now learning was reliably accomplished in one *single* trial, using a large variety of physically different poles and using demonstrations from arbitrary people in the laboratory.

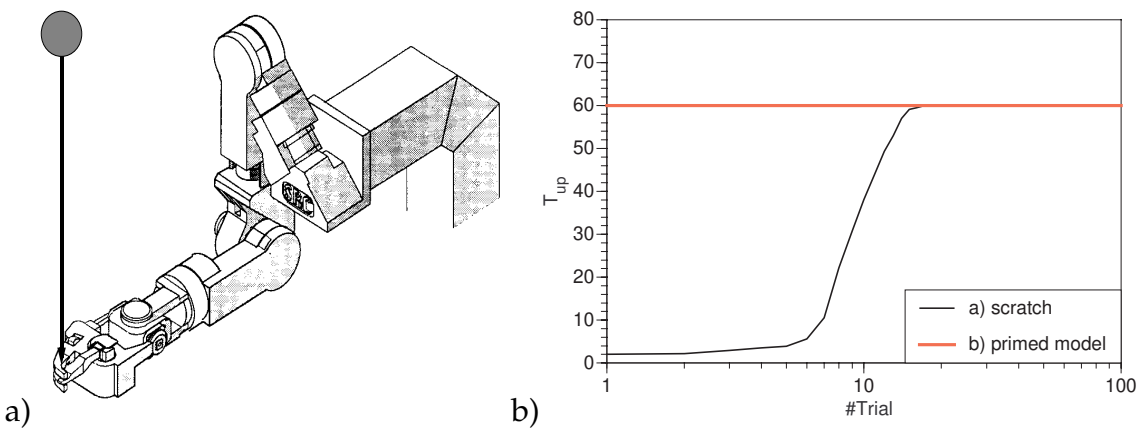

Figure 3: a) Sarcos Dexterous Robot Arm; b) Smoothed average of 10 learning curves of the robot for pole balancing. The trials were aborted after successful balancing of 60 seconds. We also tested long term performance of the learning system by running pole balancing for over an hour—the pole was never dropped.

3.3 Inverse Dynamics Learning

As all our anthropomorphic robots are rather light-weight, compliant, and driven by direct-drive hydraulic motors, using methods from rigid body dynamics to estimate their

inverse models resulted in rather inaccurate performance due to unknown nonlinearities in these systems. Therefore, the goal of the following learning experiments was to approximate the inverse dynamics of a 7-degree-of-freedom anthropomorphic robot arm (Figure 3a) from a data set consisting of 45,000 data points, collected at 100Hz from the actual robot performing various rhythmic and discrete movement tasks (this corresponds to 7.5 minutes of data collection). The data consisted of 21 input dimensions: 7 joint positions, velocities, and accelerations. The goal of learning was to approximate the appropriate torque command of the shoulder motor in response to the input vector. To increase the difficulty of learning, we added 29 irrelevant dimensions to the inputs with $N(0,0.05^2)$ Gaussian noise. 5,000 data points were excluded from the training data as a test set.

The high dimensional input space of this learning problem requires an application of LWPR. Figure 4 shows the learning results in comparison to parametric estimation of the inverse dynamics based on rigid body dynamics (An et al., 1988), and in comparison to a sigmoidal feedforward neural network with 30 hidden units trained with Levenberg-Marquardt optimization. From the very beginning, LWPR outperformed the parametric model. Within about 500,000 training points (about 30 minutes training on a 500Mhz DEC Alpha Computer), LWPR converged to the excellent result of $nMSE$=0.042. It employed an average of only 3.8 projections per local model despite the fact that the input dimensionality was 50. During learning, the number of local models increased by a factor of 6 from about 50 initial models to about 310 models. This increase is due to the adjustment of the distance metric $\mathbf{D}$ in Equation (10) that was initialized to form a rather large kernel. Since this large kernel oversmoothes the data, LWPR reduced the kernel size, and in response more kernels needed to be allocated. The sigmoidal neural network achieved the same learning results as LWPR, however, it required one night of training and could only operate in batch mode.

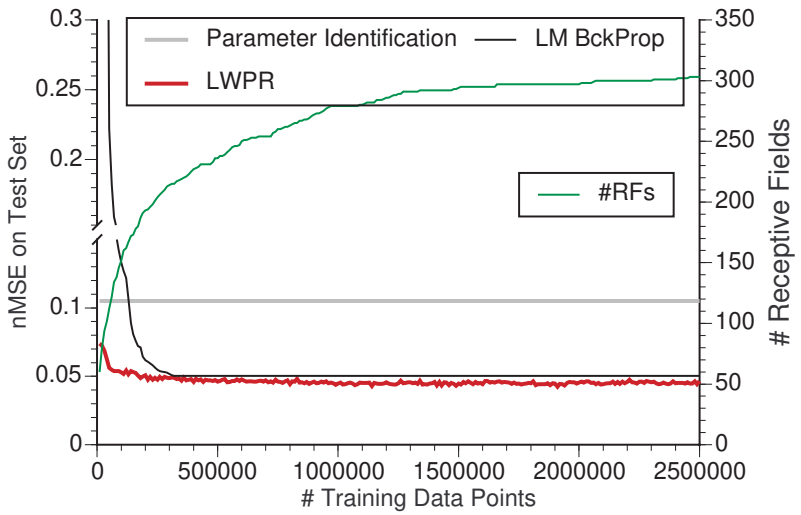

Figure 4: Learning curve for learning the inverse dynamics model of the robot from a 50 dimensional data set that included 29 irrelevant dimensions.

We also applied LWPR to an even more complex robot, a 30 DOFs humanoid robot as shown in Figure 5a. Again, we learned the inverse dynamics model for the shoulder motor, however, this time involving 90 input dimensions (i.e., 30 positions, 30 velocities, and 30 accelerations of all DOFs). The learning results, shown in Figure 5b, are similar to Figure 4. Very quickly, LWPR outperformed the inverse dynamics model estimated from rigid body dynamics and settled at a result that was more than three times more accurate. The huge learning space required more than 2000 local models, using about 2.5 local projections on average. In our real-time implementation of LWPR on this robot, the learned models achieve by far better tracking performance than the parameter estimation techniques.

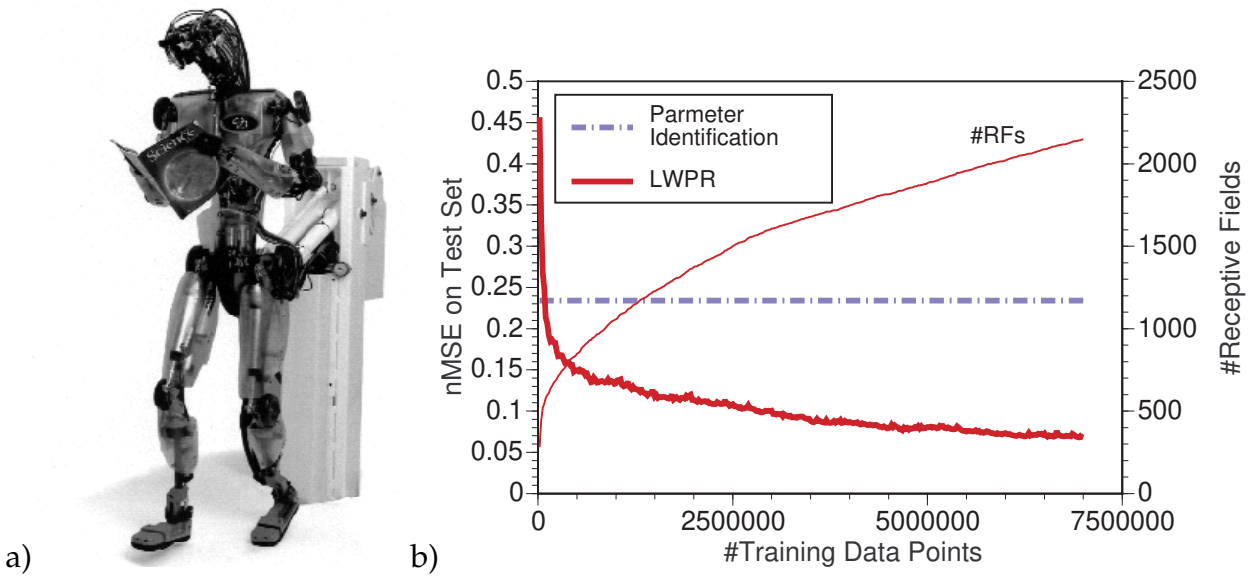

Figure 5: a) Humanoid robot in our laboratory; b) inverse dynamics learning for the right shoulder motor of the humanoid.

4 Conclusions

This paper presented Locally Weighted Learning algorithms for real-time robot learning. The algorithms are easy to implement, use sound statistical learning techniques, converge quickly to accurate learning results, and can be implemented in a purely incremental fashion. We demonstrated that the latest version of our algorithms is capable of dealing with high dimensional input spaces that even have redundant and irrelevant input dimensions while the computational complexity of an incremental update remained linear in the number of inputs. In several examples, we demonstrated how LWL algorithms were applied successfully to complex learning problems with actual robots. From the view point of function approximation, LWL algorithms are competitive methods of supervised learning of regression problem and achieve results that are comparable with state-of-the-art learning techniques. However, what makes the presented algo-

rithms special is their learning speed, numerical robustness in high dimensional spaces, and ability to learn incrementally. To the best of our knowledge, there is currently no comparable learning framework that combines all the required properties for real-time motor learning as well as Locally Weighted Learning. Our learning results demonstrate that autonomous learning can be applied successfully to rather complex robotic systems, and that learning can achieve performance that outperforms traditional techniques from control theory.

5 Acknowledgments

This work was made possible by award 9710312 and 9711770 of the National Science Foundation, the ERATO Kawato Dynamic Brain Project funded by the Japanese Science and Technology Agency, and the ATR Human Information Processing Research Laboratories.

6 References

Aha, D (1997). Lazy Learning. Artificial Intelligence Review:325-337.

An, CH, Atkeson, CG, Hollerbach, JM (1988). Model-based control of a robot manipulator. Cambridge, MA: MIT Press.

Atkeson, CG (1989). Using local models to control movement. In D. Touretzky (Ed.), Advances in Neural Information Processing Systems 1. San Mateo, CA: Morgan Kaufmann, pp 157-83.

Atkeson, CG (1992). Memory-based approaches to approximating continuous functions. In M. Casdagli, & S. Eubank (Eds.), Nonlinear Modeling and Forecasting. Redwood City, CA: Addison Wesley, pp 503-521.

Atkeson, CG, Moore, AW, Schaal, S (1997a). Locally weighted learning. Artificial Intelligence Review 11:11-73.

Atkeson, CG, Moore, AW, Schaal, S (1997b). Locally weighted learning for control. Artificial Intelligence Review 11:75-113.

Atkeson, CG, Schaal, S (1995). Memory-based neural networks for robot learning. Neurocomputing 9:1-27.

Baillieul, J, Martin, DP (1990). Resolution of kinematic redundancy, Proceedings of Symposia in Applied Mathematics: American Mathematical Society, pp 49-89.

Bishop, CM (1995). Neural networks for pattern recognition. New York: Oxford University Press.

Cleveland, WS (1979). Robust locally weighted regression and smoothing scatterplots. Journal of the American Statistical Association 74:829-836.

Cleveland, WS, Loader, C (1995). Smoothing by local regression: Principles and methods: AT&T Bell Laboratories Murray Hill NY.

Cortes, C, Vapnik, V (1995). Support vector networks. Machine Learning 20:273-297.

Craig, JJ (1986). Introduction to robotics. Reading, MA: Addison-Wesley.

Dyer, P, McReynolds, SR (1970). The computation and theory of optimal control. New York: Academic Press.

Frank, IE, Friedman, JH (1993). A statistical view of some chemometric regression tools. Technometrics 35:109-135.

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

Hastie, TJ, Tibshirani, RJ (1994). Nonparametric regression and classification: Part I: Nonparametric regression. In V. Cherkassky, J. H. Friedman, & H. Wechsler (Eds.), From Statistics to Neural Networks: Theory and Pattern Recognition Applications. ASI Proceedings, subseries F, Computer and Systems Sciences: Springer, pp 120-143.

Kawato, M (1999). Internal models for motor control and trajectory planning. Curr Opin Neurobiol 9:718-727.

Ljung, L, Söderström, T (1986). Theory and practice of recursive identification: Cambridge MIT Press.

Lowe, DG (1995). Similarity metric learning for a variable-kernel classifier. Neural Comput 7:72-85.

Moore, AW (1990). Efficient memory-based learning for robot control: Computer Laboratory University of Cambridge October 1990.

Press, WP, Flannery, BP, Teukolsky, SA, Vetterling, WT (1989). Numerical recipes in C – The art of scientific computing. Cambridge, MA: Press Syndiacate University of Cambridge.

Schaal, S (1999). Is imitation learning the route to humanoid robots? Trends in Cognitive Sciences 3:233-242.

Schaal, S, Atkeson, CG (1994). Robot juggling: An implementation of memory-based learning. Control Systems Magazine 14:57-71.

Schaal, S, Atkeson, CG (1998). Constructive incremental learning from only local information. Neural Comput 10:2047-2084.

Schaal, S, Vijayakumar, S, Atkeson, CG (1998). Local dimensionality reduction. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), Advances in Neural Information Processing Systems 10. Cambridge, MA: MIT Press, pp 633-639.

Vapnik, V, Golowich, S, Smola, A (1996). Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. I. Jordan, & T. Petsche (Eds.), Advances in Neural Information Processing Systems 9. Cambridge, MA: MIT Press, pp 281-287.

Vapnik, VN (1982). Estimation of dependences based on empirical data. Berlin: Springer.

Vijayakumar, S, Schaal, S (2000). Locally weighted projection regression, Machine Learning: Proceedings of the Seventeenth International Conference (ICML 2000). Stanford, CA.

Williams, CKI, Rasmussen, CE (1996). Gaussian processes for regression. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), Advances in Neural Information Processing Systems 8. Cambridge, MA: MIT Press, pp 514-520.

Wold, H (1975). Soft modeling by latent variables: the nonlinear iterative partial least squares approach. In J. Gani (Ed.), Perspectives in Probability and Statistics, Papers in Honour of M. S. Bartlett. London: Academic Press, pp 520-540.

Wolpert, DM, Miall, RC, Kawato, M (1998). Internal models in the cerebellum. Trends in Cognitive Sciences 2:338-347.