Schaal, S, Atkeson, CG, Vijayakumar, S (2000). Real-time robot learning with locally weighted statistical learning, International Conference on Robotics and Automation (ICRA2000). San Francisco, April 2000.

Real-Time Robot Learning With Locally Weighted Statistical Learning

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

Christopher G. Atkeson *⊕
cga@cc.gatech.edu
http://www.cc.gatech.edu/fac/Chris.Atkeson

Sethu Vijayakumar #
sethu@brain.riken.go.jp
http://www.islab.brain.riken.go.jp/~sethu

‡Computer Science and Neuroscience, HNB-103, Univ. 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
#⊕Laboratory for Information Synthesis, Riken Brain Science Research Institute, Wako, Saitama, Japan

***Abstract**: Locally weighted learning (LWL) is a class of statistical learning techniques 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 beliefs that LWL methods cannot work well in high-dimensional spaces, we provide new algorithms that have been tested in up to 50 dimensional learning problems. The applicability of our LWL algorithms is demonstrated in various robot learning examples, including the learning of devil-sticking, pole-balancing of a humanoid robot arm, and inverse-dynamics learning for a seven degree-of-freedom robot.*

1 Introduction

The necessity for self-improvement in control systems is becoming more apparent as fields such as robotics, factory automation, and autonomous vehicles become impeded by the complexity of inventing and programming satisfactory control laws. Learned models of complex tasks can aid the design of appropriate control laws for these tasks, which often involve decisions based on streams of information from sensors and actuators, where data is relatively plentiful. Learning also seems to be the only viable research approach toward the generation of flexible autonomous robots that can perform multiple tasks ([1]), with the hope of creating an autonomous humanoid robot at some point.

When approaching a learning problem, there are many alternative learning methods that can be chosen, either from the neural network, the statistical, or the machine learning literature. The current focus in learning research lies on increasingly more sophisticated algorithms for the off-line 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 ([2], [3]) and the new algorithms developed in the framework of structural risk minimization ([4], [5]). Mostly, these methods target problems in classification and diagnostics, although several extensions to regression problems exist (e.g., [6]).

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 is 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 on different days, 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 ([7]). As a last element, motor learning tasks of complex motor systems can be rather high dimensional in the number of input dimensions, thus amplifying 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 ([8], [9], [10]). LWL provides an approach to learning models of complex phenomena, dealing with large amounts of data, training quickly, and avoiding interference between multiple tasks during control of complex systems ([11], [12]). LWL meth-

ods 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 ([13]) 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 needs to be interpreted in flexible ways, for instance, as forward *or* inverse transformation. 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 ([14]). 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 three 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 implementations of real-time robot learning, involving dexterous manipulation tasks like 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 the output is one-dimensional. The key concept of our LWL methods is to approximate nonlinear functions by means of piecewise linear models ([15]), 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 ([16]). 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:

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

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 ([11]) 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) ([17]). 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 $\mathbf{x}_q$

p training points $\{\mathbf{x}_i, y_i\}$ in memory

Compute Prediction**:**

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

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

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

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

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

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 structure like *kd*-trees for keeping the data in memory ([18]). 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 distance metric $\mathbf{D}$, introduced in Equation (1). After the data in memory increased by a significant amount, $\mathbf{D}$ should be optimized by leave-one-out cross validation. To avoid too many open parameters, $\mathbf{D}$ is usually assumed to be a 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

For all $h_r \in H$:

$sse_r = 0$

For i = 1:p

a) $\mathbf{x}_q = \mathbf{x}_i$ (3)

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 $h_r^* = \min_r \{sse_r\}$

2.2 Locally Weighted Partial Least Squares

Under two circumstances the LWR algorithm above needs to be enhanced: 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) ([19], [20]), 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 through 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 Equation (5). The only steps in LWPLS that may look unusual at the first glance are the ones indicated by (*) and (**) in Equation (5). 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.

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. This is true since LWPLS will chose the optimal projection direction, the gradient of the data. This opens the question of how many projections r should be chosen if the input data are not statistically independent. Typically, the squared error $\mathbf{res}_i^T \mathbf{res}_i$ 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 at every new projection, the squared error reduces by a certain ratio:

$$\frac{\mathbf{res}_i^T \mathbf{res}_i}{\mathbf{res}_{i-1}^T \mathbf{res}_{i-1}} < \phi, \text{ where } \phi \in [0,1] \quad (4)$$

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 LWPLS Algorithm**:**

Given: a query point $\mathbf{x}_q$

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

For i = 1:r (5)

$\mathbf{u}_i = \mathbf{Z}^T \mathbf{W} \mathbf{res}$

$\mathbf{s} = \mathbf{Z}\mathbf{u}_i$

$\beta_i = \dfrac{\mathbf{s}^T \mathbf{W} \mathbf{res}}{\mathbf{s}^T \mathbf{W} \mathbf{s}}$

$\mathbf{p}_i = \dfrac{\mathbf{s}^T \mathbf{W} \mathbf{Z}}{\mathbf{s}^T \mathbf{W} \mathbf{s}}$ (*)

$\mathbf{res} \leftarrow \mathbf{res} - \mathbf{s}\beta_i$

$\mathbf{Z} \leftarrow \mathbf{Z} - \mathbf{s}\mathbf{p}_i$ (**)

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

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

For i = 1:r

$s = \mathbf{z}^T \mathbf{u}_i$

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

$\mathbf{z} \leftarrow \mathbf{z} - s\mathbf{p}_i^T$

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 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 Locally Weighted Projection 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 typical in online robot learning, both memory requirements to store all data as well as the computational cost to evaluate algorithms (2) or (5) become to

large. Under these circumstances, a non-memory-based version of LWL is desirable such that each incoming 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 ([7]), 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 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 (1). Incremental updates of the parameters of the linear models can be accomplished with recursive least squares techniques ([14]). In the following, we give the incremental update rule for a new learning system based on LWPLS, called Locally Weighted Projection Regression (LWPR). Note that we omit the index k unless it is necessary to distinguish explicitly between different linear models.

***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: $\mathbf{z} = \mathbf{x}, res_1 = y - \beta_0^{n+1}$

For i = 1:r,

$$\begin{aligned}
&\text{a)} \quad \mathbf{u}_i^{n+1} = \lambda\,\mathbf{u}_i^n + w\,\mathbf{z}\,res_i \\
&\text{b)} \quad s = \mathbf{z}^T \mathbf{u}_i^{n+1} \\
&\text{c)} \quad SS_i^{n+1} = \lambda\,SS_i^n + w\,s^2 \\
&\text{d)} \quad SR_i^{n+1} = \lambda\,SR_i^n + w\,s\,res_i \\
&\text{e)} \quad SZ_i^{n+1} = \lambda\,SZ_i^n + w\,\mathbf{z}\,s \\
&\text{f)} \quad \beta_i^{n+1} = SR_i^{n+1} / SS_i^{n+1} \\
&\text{g)} \quad \mathbf{p}_i^{n+1} = SZ_i^{n+1} / SS_i^{n+1} \\
&\text{h)} \quad \mathbf{z} \leftarrow \mathbf{z} - s\mathbf{p}_i^{n+1} \\
&\text{i)} \quad res_{i+1} \leftarrow res - s\beta_i^{n+1} \\
&\text{j)} \quad SSE_i^{n+1} = \lambda\,SSE_i^{n} + w\,res_{i+1}^2
\end{aligned} \tag{7}$$

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 ([14]). 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. 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 (4). Predictions for a query point are formed exactly as in Eqn (5)-d.

Thus, as in all the other LWL algorithms before, the only remaining open parameter is the distance metric **D**. In contrast to the algorithm in (3) that only determined **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 ([7]) we developed an incremental optimization of **D** by means of gradient descent based on a stochastic leave-one-out crossvalidation criterion. This derivation can be adapted for LWPR, and due to space constraints, we only give the new cost function that is needs to be minimized as in :analogy to ([7]).

***Gradient descent update of* D:**

$$\begin{aligned}
&\mathbf{D} = \mathbf{M}^T\mathbf{M}, \text{where } \mathbf{M} \text{ is upper triangular} \\
&\mathbf{M}^{n+1} = \mathbf{M}^n - \alpha \frac{\partial J}{\partial \mathbf{M}} \\
&J = \frac{1}{W}\sum_{i=1}^{p}\sum_{k=1}^{r} \frac{w_i res_{k+1,i}^2}{\left(1 - w_i \frac{s_{k,i}^2}{\mathbf{s}_k^T \mathbf{W} \mathbf{s}_k}\right)^2} + \gamma \sum_{i,j=1}^{n} D_{ij}^2
\end{aligned} \tag{8}$$

The above learning rules can be embedded in an incremental learning system that automatically allocates new locally linear models as needed ([7]):

Initialize the LWPR with no receptive field (RF);
For every new training sample $(\mathbf{x}, y)$:
  For k=1 to #RF:
    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 r=2, **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, and $\mathbf{D}_{\text{def}}$ is the initial (usually diagonal) distance metric in (1). The initial number of projections is set to r=2 and grows if the criterion (4) is fulfilled. For a diagonal distance metric $\mathbf{D}_{\text{k}}$ 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.

3 Empirical Evaluations

3.1 Learning of Devil Sticking

Devil sticking is a juggling task where a center stick is batted back and forth between two handsticks (Figure 1a).

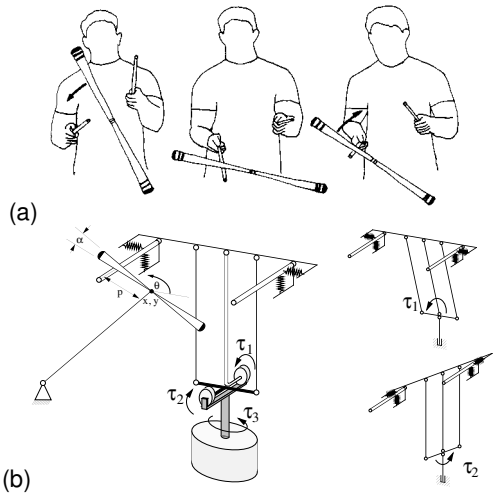

Figure 1: (a) an illustration of devil sticking, (b) sketch of our devil sticking robot: the flow of force from each motor into the robot is indicated by different shadings of the robot links, and a position change due to an application of motor 1 or motor 2, respectively, is indicated in the small sketches

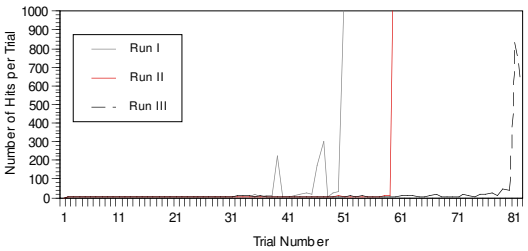

Figure 2: Learning curves of devil sticking for three runs.

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 [21]). The task of the robot is to learn a continuous left-right-left-etc. juggling pattern. For the purpose of 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 5 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 two 2 dimensional throw direction (v_x, v_y). In order to compute appropriate LQR controllers for this task, the robot learns the nonlinear mapping between current state, command, and 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. Moreover, the memory-based learning also allows to efficiently search the state-actions space for statistically good new commands ([21]). As a result, successful devil sticking can be achieved in about 40-80 trials, corresponding to about 300-800 training points in memory (Figure 2). This is a remarkable learning speed given that humans need about one week of 1 hour practicing a day before they learn to juggle the devilstick.

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 4a). The low level robot controller ran in a compute-torque mode at 480Hz out of 8 parallel processors located in a VME bus, running the real-time operating system vxWorks. 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 ([22]). 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 Receptive Field Weighted Regression (RFWR) ([7]) which is essentially the non-memory based counterpart of LWR. RFWR employs the same incremental learning strategies as LWPR. 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 has 12 input dimensions (3 positions of the lower pole end, 2 angular positions, and the corresponding 5 velocities, 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 3 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". 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.

3.3 Inverse Dynamics Learning

The goal of this learning task is to approximate the inverse dynamics model of a 7-degree-of-freedom anthropomorphic robot arm (Figure 4a) 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 robot 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.

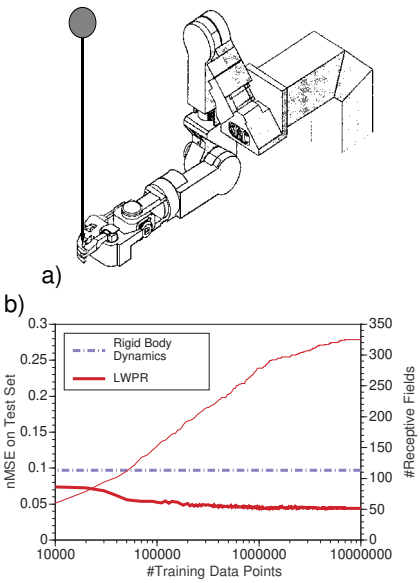

Figure 4: a) Sarcos Dexterous Robot Arm; b) Learning curve for learning the inverse dynamics model of the robot from a 50 dimensional data set that included 29 irrelevant dimensions.

The high dimensional input space of this learning problem requires an application of LWPR. Figure 4b shows the learning results in comparison to a parametric estimation of the inverse dynamics based on rigid body dynamics ([23]). From the very beginning, LWPR outperformed the parametric model. Within about 500,000 training points, 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 325 models. This increase is due to the adjustment of the distance metric $\mathbf{D}$ in Equation (8) 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.

4 Conclusions

This paper presented Locally Weighted Learning algorithms for real-time robot learning. The algorithms are easy to implement, use sound statistical techniques at their core, converge fast 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 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. 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.

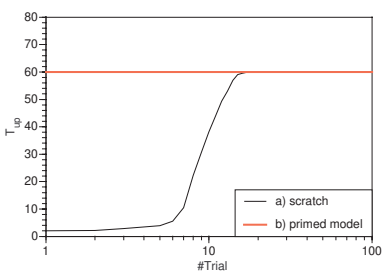

Figure 3: Smoothed average of 10 learning curves of the robot for pole balancing. The trials were aborted after successful balancing of 60 seconds. We also tested long term performance of the learning system by running pole balancing for over an hour—the pole was never dropped.

Acknowledgments

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

References

[1] S. Schaal, "Is imitation learning the route to humanoid robots?," *Trends in Cognitive Sciences*, vol. 3, pp. 233-242, 1999.

[2] C. M. Bishop, *Neural networks for pattern recognition*. New York: Oxford University Press, 1995.

[3] C. K. I. Williams and C. E. Rasmussen, "Gaussian processes for regression," in *Advances in Neural Information Processing Systems 8*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. Cambridge, MA: MIT Press, 1996, pp. 514-520.

[4] V. N. Vapnik, *Estimation of dependences based on empirical data*. Berlin: Springer, 1982.

[5] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273-297, 1995.

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

[7] S. Schaal and C. G. Atkeson, "Constructive incremental learning from only local information," *Neural Computation*, vol. 10, pp. 2047-2084, 1998.

[8] C. G. Atkeson and S. Schaal, "Memory-based neural networks for robot learning," *Neurocomputing*, vol. 9, pp. 1-27, 1995.

[9] W. S. Cleveland and C. Loader, "Smoothing by local regression: Principles and methods," : AT&T Bell Laboratories Murray Hill NY, 1995.

[10] T. J. Hastie and R. J. Tibshirani, "Nonparametric regression and classification: Part I: Nonparametric regression," in *From Statistics to Neural Networks: Theory and Pattern Recognition Applications. ASI Proceedings, subseries F, Computer and Systems Sciences*, V. Cherkassky, J. H. Friedman, and H. Wechsler, Eds.: Springer, 1994, pp. 120-143.

[11] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, pp. 11-73, 1997.

[12] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Artificial Intelligence Review*, vol. 11, pp. 75-113, 1997.

[13] D. Aha, "Lazy Learning," *Artificial Intelligence Review*, pp. 325-337, 1997.

[14] L. Ljung and T. Söderström, *Theory and practice of recursive identification*: Cambridge MIT Press, 1986.

[15] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *Journal of the American Statistical Association*, vol. 74, pp. 829-836, 1979.

[16] T. Hastie and C. Loader, "Local regression: Automatic kernel carpentry," *Statistical Science*, vol. 8, pp. 120-143, 1993.

[17] C. G. Atkeson, "Memory-based approaches to approximating continuous functions," in *Nonlinear Modeling and Forecasting*, M. Casdagli and S. Eubank, Eds. Redwood City, CA: Addison Wesley, 1992, pp. 503-521.

[18] A. W. Moore, "Efficient memory-based learning for robot control," : Computer Laboratory University of Cambridge October 1990, 1990.

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

[20] I. E. Frank and J. H. Friedman, "A statistical view of some chemometric regression tools," *Technometrics*, vol. 35, pp. 109-135, 1993.

[21] S. Schaal and C. G. Atkeson, "Robot juggling: An implementation of memory-based learning," *Control Systems Magazine*, vol. 14, pp. 57-71, 1994.

[22] J. Baillieul and D. P. Martin, "Resolution of kinematic redundancy," in *Proceedings of Symposia in Applied Mathematics*, vol. 41: American Mathematical Society, 1990, pp. 49-89.

[23] C. H. An, C. G. Atkeson, and J. M. Hollerbach, *Model-based control of a robot manipulator*. Cambridge, MA: MIT Press, 1988.