Real Time Learning in Humanoids: A challenge for scalability of Online Algorithms

Sethu Vijayakumar and Stefan Schaal

Computer Science & Neuroscience and Kawato Dynamic Brain Project
University of Southern California, Los Angeles, CA 90089-2520, USA
{*sethu,sschaal*}*@usc.edu*

Abstract. While recent research in neural networks and statistical learning has focused mostly on learning from finite data sets without stringent constraints on computational efficiency, there is an increasing number of learning problems that require real-time performance from an essentially infinite stream of incrementally arriving data. This paper demonstrates how even high-dimensional learning problems of this kind can successfully be dealt with by techniques from nonparametric regression and locally weighted learning. As an example, we describe the application of one of the most advanced of such algorithms, Locally Weighted Projection Regression (LWPR), to the on-line learning of the inverse dynamics model of an actual seven degree-of-freedom anthropomorphic robot arm. LWPR's linear computational complexity in the number of input dimensions, its inherent mechanisms of local dimensionality reduction, and its sound learning rule based on incremental stochastic leave-one-out cross validation allows – to our knowledge for the first time – implementing inverse dynamics learning for such a complex robot with real-time performance. In our sample task, the robot acquires the local inverse dynamics model needed to trace a figure-8 in only 60 seconds of training.

1 Introduction

An increasing number of learning problems involves real-time modeling of complex high-dimensional processes. Typical examples include the on-line modeling of dynamic processes observed by visual surveillance, user modeling for advanced computer interfaces and game playing, and the learning of value functions, policies, and internal models for learning control. Among the characteristics of these learning problems are high dimensional input spaces with potentially redundant and irrelevant dimensions, nonstationary input and output distributions, essentially infinite training data sets with no representative validation sets, and the need for continual learning. Most of these learning tasks fall into the domain of regression problems.

Interestingly, this class of problems has so far not been conquered by the new developments in statistical learning. Bayesian inference [3] is usually computationally too expensive for real-time application as it requires representation of the complete joint probablity densities of the data. The framework of structural risk minimization[9], the most advanced in form of Support Vector Machines, excels in classification and finite batch learning problems, but has yet to show compelling performance in regression and incremental learning. In contrast, techniques from nonparametric regression, in particular the methods of locally weighted learning [2], have recently advanced to meet all the requirements of real-time learning in high-dimensional spaces. In this paper, we will describe how one of the most highly developed algorithms amongst them, Locally Weighted Projection Regression (LWPR), accomplishes learning of a highly nonlinear model for robot control– the inverse dynamics model of a seven degree-of-freedom (DOF) anthropomorphic robot. In the following sections, we will first explain the learning task, then spell out the LWPR algorithm, and finally illustrate learning results from real-time learning on the actual robot. To the best of our knowledge, this is the first time that real-time learning of such a complex model has been accomplished in robot control.

2 Inverse Dynamics Learning

A common strategy in robotic and biological motor control is to convert kinematic trajectory plans into motor commands by means of an inverse dynamics model. The inverse dynamics takes the desired positions, velocities, and accelerations of all DOFs of the robot and outputs the appropriate motor commands. For our robot, a seven DOF anthropomorphic robot arm, the inverse dynamics model receives 21 inputs and outputs 7 torque commands. If derived analytically using a rigid body dynamics assumption [1], the most compact recursive formulation of the inverse dynamics of our robot results in about 15 pages of compact C-code, filled with nested sine and cosine terms. For on-line learning, motor commands need to be generated from the model at 480Hz in our implementation.

Table 1. Pseudocode of PLS projection regression

1. Initialize: $\mathbf{X}_{res} = \mathbf{X}$, $\mathbf{y}_{res} = \mathbf{y}$
2. **For** $r = 1$ **to** R (# projections)
   (a) $\mathbf{u}_r = \mathbf{X}_{res}^T \mathbf{y}_{res}$; $\beta_r = \mathbf{s}_r^T \mathbf{y}_{res} / (\mathbf{s}_r^T \mathbf{s}_r)$ where $\mathbf{s}_r = \mathbf{X}_{res} \mathbf{u}_r$.
   (b) $\mathbf{y}_{res} = \mathbf{y}_{res} - \mathbf{s}_r \beta_r$; $\mathbf{X}_{res} = \mathbf{X}_{res} - \mathbf{s}_r \mathbf{p}_r{}^T$ where $\mathbf{p}_r = \mathbf{X}_{res}^T \mathbf{s}_r / (\mathbf{s}_r^T \mathbf{s}_r)$.

Updating the learning system can take place at a lower rate but should remain above 10Hz to capture suffcient data in fast movements.

Learning regression problems in a 21-dimensional input space is a daunting problem from the view of the bias-variance trade-off. In learning control, training data is generated by the learning system itself, and it is impossible to assess a priori what structural complexity that data is going to have. Fortunately, actual movement systems do not fill the data space in a completely random way. Instead, when viewed locally, data distributions are low dimensional, e.g., about 4-6 dimensional for the inverse dynamics [8] of our robot instead of the global 21 input dimensions. This property will be a key element in our approach to learning such models.

3 Locally Weighted Projection Regression

The core concept of our learning approach is to approximate nonlinear functions by means of piecewise linear models [2]. The learning system automatically determines the appropriate number of local models, the parameters of the hyperplane in each model, and also the region of validity, called receptive field (RF), of each of the model, usually formalized as a Gaussian kernel:

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

Given a query point $\mathbf{x}$, each linear model calculates a prediction y_k. The total output of the learning system is the weighted mean of all K linear models:

$$\hat{y} = \sum_{k=1}^{K} w_k y_k / \sum_{k=1}^{K} w_k,$$

also illustrated in Fig.1. Learning in the system involves determining the linear regression parameter β_k and the distance metric $\mathbf{D}_k$. The center $\mathbf{c}_k$ of the RF remains fixed. Local models are created as and when needed as described in Section 3.3.

3.1 Local Dimensionality Reduction

Despite its appealing simplicity, the "piecewise linear modeling" approach becomes numerically brittle and computationally too expensive in high dimensional problems. Given the empirical observation that high dimensional data is often locally low dimensional, it is possible to develop a very efficient approach to exploit this property. Instead of using ordinary linear regression to fit the local hyperplanes, we suggest to employ Partial Least Squares (PLS) [11, 4]. PLS recursively computes orthogonal projections of the input data and performs single variable regressions along these projections on the residuals of the previous iteration step. Table 1 illustrates PLS in pseudocode for a global linear model where the input data is in the rows of the matrix $\mathbf{X}$, and the corresponding one dimensional output data is in the vector $\mathbf{y}$. The key ingredient in PLS is to use the direction of maximal correlation between the residual error and the input data as the projection direction at every regression step. Additionally, PLS regresses the inputs of the previous step against the projected inputs $\mathbf{s}_r$ in order to ensure the orthogonality of all the projections $\mathbf{u}_r$ (Step 2b). Actually, this additional regression could be avoided by replacing $\mathbf{p}_r$ with $\mathbf{u}_r$, similar to techniques used in principal component analysis [5]. However, using this regression step leads to better performance of the algorithm. This effect is due to the fact that PLS chooses the most effective projections if the input data has a spherical distribution: with only one projection, PLS will find the direction of the gradient and achieve optimal regression results. The regression step in 2b modifies the input data $\mathbf{X}_{res}$ such that each resulting data vectors have coefficients of minimal magnitude and, hence, push the distribution of $\mathbf{X}_{res}$ to become more spherical.

An incremental locally weighted version of the PLS algorithm[10] is derived in Table 2. Here, $\lambda \in [0, 1]$ denotes a forgetting factor that determines how quickly older data will be forgotten in the various PLS parameters,

Table 2. Incremental locally weighted PLS for one RF

Initialization:
$\mathbf{x}_0^0 = \mathbf{0}$, $\mathbf{u}^0 = \mathbf{0}$, $\beta_0^0 = 0$, $W^0 = 0$
Given: A training point $(\mathbf{x}, y)$

$$w = exp(-\frac{1}{2}(\mathbf{x} - \mathbf{c})^T \mathbf{D}(\mathbf{x} - \mathbf{c}))$$

Update the means :

$$W^{n+1} = \lambda W^n + w$$
$$\mathbf{x}_0^{n+1} = \frac{\lambda W^n \mathbf{x}_0^n + w\mathbf{x}}{W^{n+1}}$$
$$\beta_0^{n+1} = \frac{\lambda W^n \beta_0^n + wy}{W^{n+1}}$$

Update the local model:
Initialize:
$\mathbf{z} = \mathbf{x} - \mathbf{x}_0^{n+1}$, $res_1 = y - \beta_0^{n+1}$
For $r = 1 : R$ (# projections)
1. $\mathbf{u}_r^{n+1} = \lambda \mathbf{u}_r^n + w\mathbf{z}\, res_r$
2. $s_r = \mathbf{z}^T \mathbf{u}_r^{n+1}$
3. $SS_r^{n+1} = \lambda SS_r^n + w\, s_r^2$
4. $SR_r^{n+1} = \lambda SR_r^n + w\, s_r\, res_r$
5. $\beta_r^{n+1} = SR_r^{n+1}/SS_r^{n+1}$
6. $res_{r+1} = res_r - s_r \beta_r^{n+1}$
7. $MSE_r^{n+1} = \lambda MSE_r^n + w\, res_{r+1}^2$
8. $SZ_r^{n+1} = \lambda SZ_r^n + w\mathbf{z}s_r$
9. $\mathbf{p}_r^{n+1} = SZ_r^{n+1}/SS_r^{n+1}$
10. $\mathbf{z} = \mathbf{z} - s\mathbf{p}_r^{n+1}$

Predicting with novel data ($\mathbf{x}_q$): Initialize: $y = \beta_0, \mathbf{z} = \mathbf{x}_q - \mathbf{x}_0$
Repeat for r=1:R
- $y = y + \beta_r s$ where $s = \mathbf{u}_r^T \mathbf{z}$
- $\mathbf{z} = \mathbf{z} - s\mathbf{p}_r^n$

similar to the recursive system identification techniques [12]. The variables SS_r, SR_r and SZ_r are memory terms that enable us to do the univariate regression in step (7) in a recursive least squares fashion, i.e., a fast Newton-like method.

Since PLS selects the univariate projections very efficiently, it is even possible to run locally weighted PLS with only *one* projection direction (denoted as LWPR-1). The optimal projection is in the direction of the local gradient of the function to be approximated. If the locally weighted input data forms a spherical distribution in a local model, the single PLS projection will suffice to find the optimal direction. Otherwise, the distance metric (and hence, weighting of the data) will need to be adjusted to make the local distribution more spherical. The learning rule of the distance metric can accomplish this adjustment, as explained below. It should be noted that Steps 8-10 in Table 2 become unnecessary for the uni-projection case.

3.2 Learning the Distance Metric

The distance metric $\mathbf{D}_k$ and hence, the locality of the receptive fields, can be learned for each local model individually by stochastic gradient descent in a leave-one-out cross validation cost function. Note that this update does *not* require competitive learning – only a completely local learning rule is needed, and leave-one-out cross validation can be performed *without* keeping data in memory [7]. The update rule can be written as :

$$\mathbf{M}^{n+1} = \mathbf{M}^n - \alpha \frac{\partial J}{\partial \mathbf{M}} \text{ where } \mathbf{D} = \mathbf{M}^T\mathbf{M} \text{ (for positive definiteness)} \tag{2}$$

and the cost function to be minimized is:

$$J = \frac{1}{\sum_{i=1}^M w_i} \sum_{i=1}^M \sum_{r=1}^R \frac{w_i res_{r+1,i}^2}{(1 - w_i \frac{s_{r,i}^2}{\mathbf{s}_r^T \mathbf{W} \mathbf{s}_r})^2} + \frac{\gamma}{N} \sum_{i,j=1}^N D_{ij}^2 = \sum_{r=1}^R (\sum_{i=1}^M J_{1,r}) + J_2. \tag{3}$$

where M denotes the number of training data, and N the number of input dimensions. A stochastic version of the gradient $\frac{\partial J}{\partial \mathbf{M}}$ can be derived from the cost function by keeping track of several "memory terms" as shown in Table 3.

Table 3. Derivatives for distance metric update

$$\frac{\partial J}{\partial \mathbf{M}} \approx \sum_{r=1}^{R}(\sum_{i=1}^{M}\frac{\partial J_{1,r}}{\partial w})\frac{\partial w}{\partial \mathbf{M}} + \frac{w}{W^{n+1}}\frac{\partial J_2}{\partial \mathbf{M}} \text{ (stochastic update)}$$

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

$$\frac{\partial D_{ij}}{\partial M_{kl}} = M_{kj}\delta_{il} + M_{ik}\delta_{jl};\; \text{where } \delta_{ij}=1 \text{ if } i=j \text{ else } \delta_{ij}=0.$$

Compute the following for each projection direction r:

$$\sum_{i=1}^{M}\frac{\partial J_{1,r}}{\partial w} = \frac{e_{cv,r}^2}{W^{n+1}} - 2\frac{(P_r^{n+1}s_r e_r)}{W^{n+1}}H_r^n - 2\frac{(P_r^{n+1}s_r)^2}{W^{n+1}}R_r^n - \frac{E_r^{n+1}}{(W^{n+1})^2}$$
$$+[T_r^{n+1} - 2R_r^{n+1}P_r^{n+1}C^{n+1}]\frac{(\mathbf{I}-\mathbf{u}_r\mathbf{u}_r^T)\mathbf{z}\; res_r}{W^{n+1}\sqrt{\mathbf{u}_r^T\mathbf{u}_r}}$$

$$C^{n+1} = \lambda C^n + w s_r res_r,\;\; e_r = res_{r+1},\;\; e_{cv,r} = \frac{e_r}{1-wP_r^{n+1}s_r^2},\;\; P_r^{n+1} = \frac{1}{SS_r^{n+1}}$$

$$H_r^{n+1} = \lambda H_r^n + \frac{w\; e_{cv,r}s_r}{(1-w\; h_r)};\;\; R_r^{n+1} = \lambda R_r^n + \frac{w^2 s_r^2 e_{cv,r}^2}{(1-w\; h_r)} \text{ where } h_r = P_r^{n+1}s_r^2$$

$$E_r^{n+1} = \lambda E_r^n + w e_{cv,r}^2;\;\; T_r^{n+1} = \lambda T_r^n + \frac{w(2we_{cv,r}^2 s_r P_r^{n+1} - e_{cv,r}\beta_r^{n+1})res_r}{(1-w\; h_r)}$$

3.3 The Complete LWPR Algorithm

All the ingredients above can be combined in an incremental learning scheme that automatically allocates new locally linear models as needed. The final learning network is illustrated in Fig. 1 and an outline of the algorithm is shown below.

LWPR outline

- Initialize the LWPR with no receptive field (RF);
- **For** every new training sample (x,y):
  - **For** k=1 to K:
    - calculate the activation from eq.(1)
    - update projections & regression (Table 2) and Distance Metric (Table 3)
  - **If** no RF was activated by more than w_{gen};
    - create a new RF with $r=2$, $\mathbf{c}=\mathbf{x}$, $\mathbf{D}=\mathbf{D}_{def}$

In this pseudo-code, w_{gen} is a threshold that determines when to create a new receptive field, and $\mathbf{D}_{def}$ is the initial (usually diagonal) distance metric in eq.(1). The initial number of projections is set to $r=2$. The algorithm has a simple mechanism of determining whether r should be increased by recursively keeping track of the mean-squared error (MSE) as a function of the number of projections included in a local model, i.e., Step 7 in the incremental PLS pseudocode (Table 2). If the MSE at the next projection does not decrease more than a certain percentage of the previous MSE, i.e., $\frac{MSE_{i+1}}{MSE_i} > \phi$, where $\phi \in [0,1]$, the algorithm will stop adding new projections locally. For a diagonal distance metric $\mathbf{D}$ and under the assumption that the number of projections R remains small, the computational complexity of the update of all parameters of LWPR is linear in the number of input dimensions. The LWPR-1 variant on the other hand uses just one projection direction.

4 Real-Time Learning of Inverse Dynamics

4.1 Performance Comparison on a Static Data Set

Before demonstrating the applicability of LWPR in real-time, a comparison with alternative learning methods will serve to demonstrate the complexity of the learning task. We collected 50,000 data points from various movement

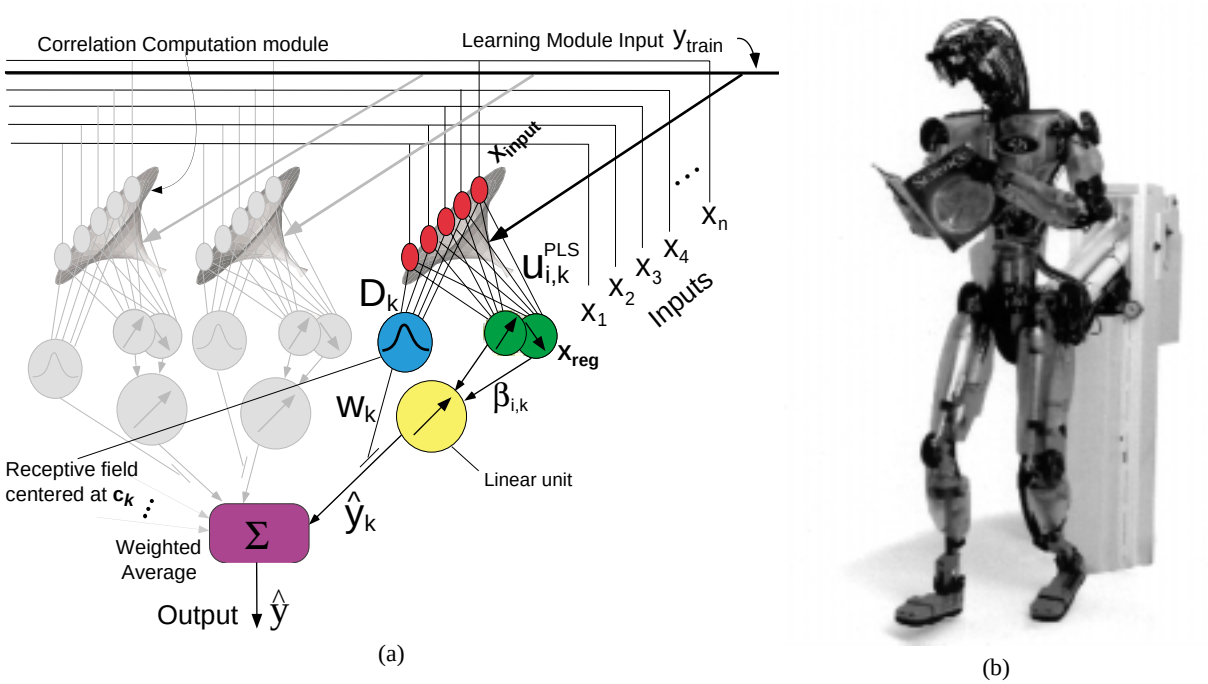


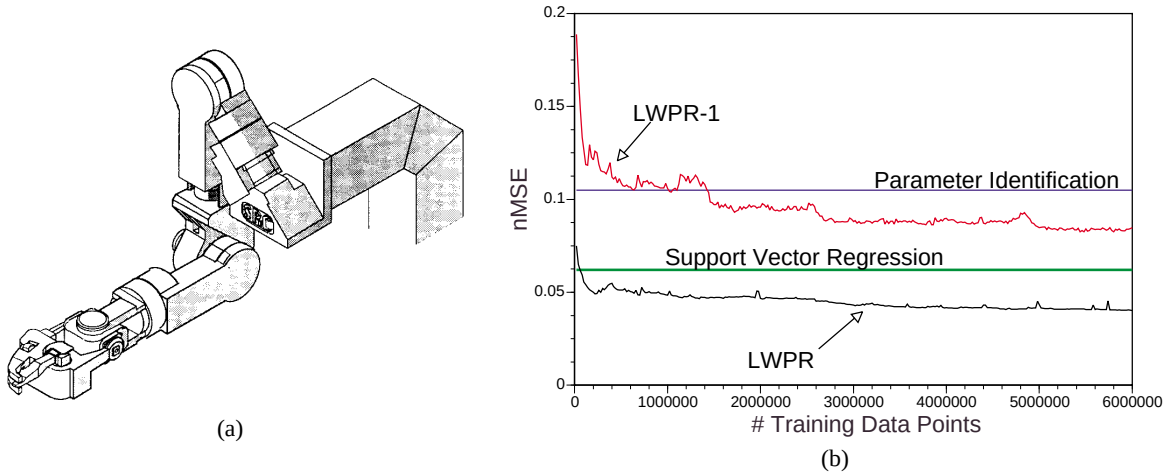
Fig. 1. (a) Information processing unit of LWPR (b) Humanoid Robot used for learning in our laboratory

patterns from a 7 DOF anthropomorphic robot (Fig.2a) at 50Hz sampling frequency. 10 percent of this data was excluded as a test set. The training data was approximated by 4 different methods: i) parameter estimation based on an analytical rigid body dynamics model [1], ii) Support Vector Regression[6], iii) LWPR-1, and iv) full LWPR. It should be noted that neither i) nor ii) are incremental methods. Using a parametric model as suggested in i) and just approximating its open parameters from data results in a global model of the inverse dynamics and is theroretically the most powerful method. However, given that our robot is actuated hydraulically and rather lightweight and compliant, we know that the rigid body dynamics assumption is not fully justified. In all our evaluations, the inverse dynamics model of each DOF was learned separately, i.e., all models had a univariate output and 21 inputs. LWPR employed a diagonal distance metric.

Fig.2b illustrates the function approximation results for the shoulder motor command graphed over the number of training iterations (one iteration corresponds to the update from one data point). Surprisingly, rigid body parameter estimation achieved the worst results. LWPR-1 outperformed parameter estimation, but fell behind SVM regression. Full LWPR performed the best. The results for all other DOFs were analogous. For the final result, LWPR employed 260 local models, using an average of 3.2 local projections. LWPR-1 did not perform better because we used a diagonal distance metric. The abilities of a diagonal distance metric to "carve out" a locally spherical distribution are too limited to accomplish better results – a full distance metric can remedy this problem, but would make the learning updates quadratic in the number of inputs. These results demonstrate that LWPR is a competitive function approximation technque.

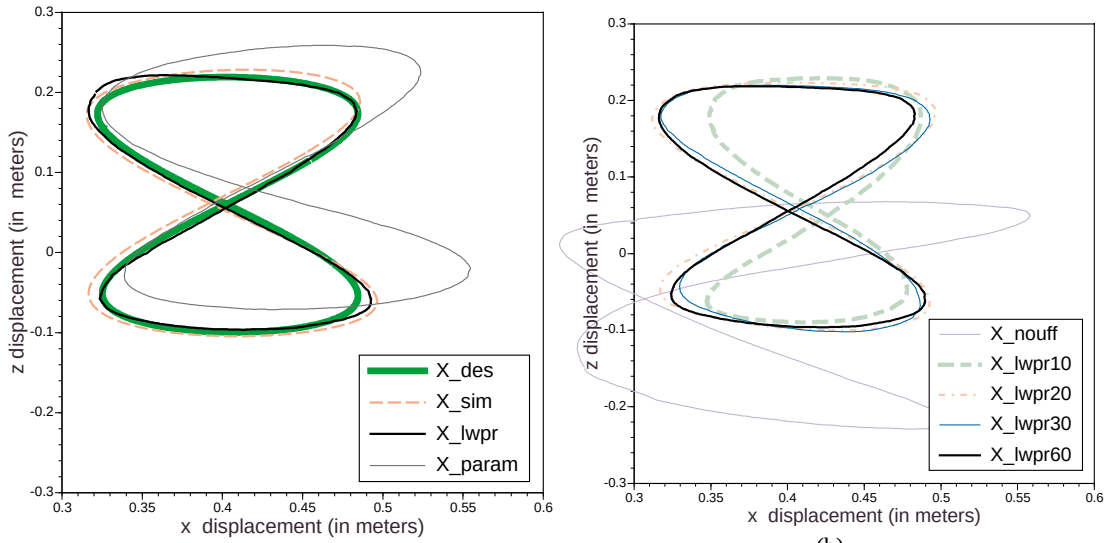
4.2 On-line Learning

We implemented full LWPR on our robotic setup. Out of the four parallel processors of the system, one 366Mhz PowerPC processor was completely devoted to lookup and learning with LWPR. Each DOF had its own LWPR learning system, resulting in 7 parallel learning modules. In order to accelerate lookup and training times, we added a special data structure to LWPR. Each local model maintained a list of all other local models that overlapped sufficiently with it. Sufficient overlap between two local model i and j can be determined from the centers and distance metrics. The point $\mathbf{x}$ in input space that is the closest to both centers in the sense of a Mahalanobis distance is $\mathbf{x} = (\mathbf{D}_i + \mathbf{D}_j)^{-1}(\mathbf{D}_i\mathbf{c}_i + \mathbf{D}_j\mathbf{c}_j)$. Inserting this point into eq.(1) of one of the local models gives the activation w due to this point. The two local models are listed as sufficiently overlapping if $w >= w_{gen}$ (cf. LWPR outline). For diagonal distance metrics, the overlap computation is linear in the number of inputs. Whenever a new data point is added to LWPR, one neighborhood relation is checked for the maximally activated RF. An appropriate



(a) (b)

Fig. 2. (a) SARCOS dexterous arm (b) Comparison of nMSE traces for different learning schemes



(a) (b)

Fig. 3. (a) Robot end effector motion traces under different control schemes (b) Progress of online learning with LWPR control

counter for each local model ensures that overlap with all other local models is checked exhaustively. Given this "nearest neighbor" data structure and the fact that a movement system generates temporally highly correlated data, lookup and learning can be confined to only few RFs. For every lookup (update), the identification number of the maximally activated RF is returned. The next lookup (update) will only consider the neighbors of this RF. It can be proved that this method performs as good as an exhaustive lookup (update) strategy that excludes RFs that are activated below a certain threshold w_{cutoff}.

The LWPR models were trained on-line while the robot performed a pseudo randomly drifting figure-8 pattern in front of its body. Lookup proceeded at 480Hz, while updating the learning model was achieved at about 70Hz. At certain intervals, learning was stopped and the robot attempted to draw a planar figure-8 at 2Hz frequency for the entire pattern. The quality of these drawing patterns is illustrated in Fig.3a,b. In Fig.3a, X_{des} denotes the desired figure-8 pattern, X_{sim} illustrates the figure-8 performed by our robot simulator that uses a perfect inverse dynamics model (but not necessarily a perfect tracking and numerical integration algorithm), X_{param} is the performance of the estimated rigid body dynamics model, and X_{lwpr} shows the results of LWPR. While the rigid body model has the worst performance, LWPR obtained the best results, even better than the simulator. Fig.3b illustrates the speed of LWPR learning. The X_{nouff} trace demonstrates the figure-8 patterns performed without any inverse dynamics model, just using a low gain PD controller. The other traces show how rapidly LWPR learned the figure-8 pattern during training: they denote performance after 10, 20, 30, and 60 *seconds* of training. After 60 seconds, the figure-8 is hardly distinguishable from the desired trace.

5 Conclusions

This paper illustrated an application of Locally Weighted Projection Regression to a complex robot learning task. The *O(n)* update complexity of LWPR, together with its statistically sound dimensionality reduction and learning rules allowed a reliable and successful real-time implementation of the algorithm on an actual anthropomorphic robot. From the viewpoint of learning control, these results demark the first time that complex internal models can be learned autonomously in real-time on sophisticated robotic devices. In an even more challenging application, we will implement LWPR for inverse dynamics learning on a 30 DOF full-body humanoid robot (Fig.1b) in the near future.

References

1. An,C.H., Atkeson,C. & Hollerbach, J. (1988) *Model Based Control of a Robot Manipulator* MIT Press.
2. Atkeson, C., Moore, A. & Schaal, S. (1997). Locally weighted learning. *Artif. Intel. Rev., 11*, 76–113.
3. Bishop, C. (1995) *Neural Networks for Pattern Recognition*. Oxford Press.
4. Frank, I. E. & Friedman, J. H. (1993). A stat. view of some chemo. reg. tools. *Technometrics*, *35*, 109–135.
5. Sanger, T. D. (1989). Optimal unsupervised learning in a single layer liner feedforward neural network. *Neural Networks*, *2*, 459–473.
6. Saunders,C., Stitson, M.O., Weston,J., Bottou,L., Schoelkopf,B., Smola,A. (1998) Support Vector Machine - Reference Manual. *TR CSD-TR-98-03*, Dept. of Computer Science, Royal Holloway, Univ. of London.
7. Schaal, S. & Atkeson, C. G. (1998). Const. inc. learning from only local info. *Neural Comp, 10*, pp.2047–2084.
8. Schaal, S., Vijayakumar, S. & Atkeson, C. G. (1998). Local dimensionality reduction. *NIPS 10*, pp.633–639.
9. Vapnik, V. (1995) *The Nature of Statistical Learning Theory*. Springer, New York.
10. Vijayakumar, S. & Schaal,S. (2000). Locally Weighted Projection Regression : An *O(n)* algorithm for incremental real time learning in high dimensional space. Proc. ICML 2000, pp.1079-1086.
11. Wold, H. (1975). Soft modeling by latent variables: the nonlinear iterative partial least squares approach. *Perspectives in Probability and Statistics*.
12. Ljung, L. & Soderstrom, T. (1986) *Theory and practice of recursive identification*. Cambridge MIT Press.