解 説

Online Statistical Robot Learning

Stefan Schaal* Sethu Vijayakumar* Aaron D'Souza*
*Computer Science and Neuroscience, HNB-103, 3641 Watt Way, University of Southern California, Los Angeles
*, * and * *

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 actua- tors, and where data is relatively plentiful. Learning also seems to be the only viable research approach towards the generation of flexible autonomous robots that can perform multiple tasks [23], 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 [8] and the new algorithms developed in the framework of structural risk minimization [27]. Mostly, these methods target problems in classification and diagnostics, although several extensions to regression problems exist. 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 [24]. 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 [6] [11] [16]. 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 [4] [5]. 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 in-

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

puts. LWL meth- ods come in two different strategies. Memory-based LWL is a "lazy learning" method 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 forward 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 [19]. 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 review four LWL algorithms that are the most suitable to online robot learning problems, with a focus on Locally Weighted Projection Regression, the most efficient learning method that we developed. We will illustrate the successful application of some of these methods to real-time robot learning, involving dexterous manipulation tasks such as 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 $y = f(x) + \varepsilon$, where $x \in \Re^n$ 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 [10], 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 [15]. 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:

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

where c_k is the center of the k^{th} linear model, and 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 [4] 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) [2] [3] [6]. 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 x_q, the following weighted regression analysis is performed:

Algorithm 1 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 $\mathbf{W}$ 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
$\mathbf{X} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_p)^T$ where $\tilde{\mathbf{x}}_i = \begin{bmatrix} (\mathbf{x}_i - \mathbf{x}_q)^T & 1 \end{bmatrix}^T$
$\mathbf{y} = (y_1, y_2, \ldots, y_p)^T$

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

β_{n+1} denotes the (n+1)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 [21]. 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 Algorithm 1 is the distance metric D, introduced in Equation (1). After a sufficient amount of data has been incorporated, D can be optimized by leave one-out cross validation. To avoid too many open parameters, D is usually assumed to be a global diagonal matrix $D = h \cdot diag([n_1, n_2, \ldots, n_n])$, 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 cross validation is thus performed only as a one-dimensional search over the parameter h [25]. 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 [3] [6] [20].

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 Algorithm 1 becomes numerically unstable. There is a computational efficient technique from the statistics literature, Partial Least Squares Regression (PLS) [14] [25] [29] [30], 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 Algorithm 2. The only steps in LWPLS that may look unusual at a first glance are the ones indicated by (*) and (**) in Algorithm 2. 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 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 [29]. 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:

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

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 D, which can be optimized as mentioned in the previous section. The computational complexity of LWPLS is $O(rnp)$, where r is the number of projections, n the number of input dimensions, and p the number of data points. 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.

Algorithm 2 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
$w_{ii} = \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_q)^T \mathbf{D} (\mathbf{x}_i - \mathbf{x}_q)\right)$
b) build matrix $\mathbf{X}$ and vector $\mathbf{y}$ such that
$\bar{\mathbf{x}} = \left(\sum_{i=1}^p w_{ii}\mathbf{x}_i\right) / \left(\sum_{i=1}^p w_{ii}\right)$
$\beta_0 = \left(\sum_{i=1}^p w_{ii} y_i\right) / \left(\sum_{i=1}^p w_{ii}\right)$
$\mathbf{X} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_p)^T$ where $\tilde{\mathbf{x}}_i = (\mathbf{x}_i - \bar{\mathbf{x}})$
$\mathbf{y} = (\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_p)^T$ where $\tilde{y}_i = (y_i - \beta_0)$
c) recursively compute locally linear model
Initialize: $\mathbf{Z}_0 = \mathbf{X}$, $\mathbf{res}_0 = \mathbf{y}$
for $i = 1 : r$
$\mathbf{u}_i = \mathbf{Z}_{i-1}^T \mathbf{W} \mathbf{res}_{i-1}$
$\mathbf{s}_i = \mathbf{Z}_{i-1} \mathbf{u}_i$
$\beta_i = \frac{\mathbf{s}_i^T \mathbf{W} \mathbf{res}_{i-1}}{\mathbf{s}_i^T \mathbf{W} \mathbf{s}_i}$
$\mathbf{p}_i = \frac{\mathbf{s}_i^T \mathbf{W} \mathbf{Z}_{i-1}}{\mathbf{s}_i^T \mathbf{W} \mathbf{s}_i}$ (*)
$\mathbf{res}_i = \mathbf{res}_{i-1} - \mathbf{s}_i \beta_i$
$\mathbf{Z}_i = \mathbf{Z}_{i-1} - \mathbf{s}_i \mathbf{p}_i$ (**)
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 $i = 1 : r$
$s_i = \mathbf{z}_{i-1}^T \mathbf{u}_i$
$\hat{y}_q \leftarrow \hat{y}_q + s_i \beta_i$
$\mathbf{z}_i = \mathbf{z}_{i-1} - s_i \mathbf{p}_i^T$

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 is typical in online robot learning, both memory requirements to store all data as well as the computational cost to evaluate

Algorithms 1 or 2 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 [24], using LWR as the starting point. The idea of the algorithm is straight forward: 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:

$$\hat{y}_q = \frac{\sum_{k=1}^{K} w_k \hat{y}_{q,k}}{\sum_{k=1}^{K} w_k} \qquad (3)$$

The weights in (3) 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 [19]. Thus, the LWR algorithm from Algorithm 1 becomes the Receptive Field Weighted Regression (RFWR) algorithm, as described in detail in previous work [24]. 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 is shown in Algorithm 3. 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. $\lambda \in [0, 1]$ is a forgetting factor that determines how much old data in the regression parameters will be forgotten, similar as in recursive system identification techniques [19]. 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 (2). Predictions for a query point are formed exactly as in Algorithm 2 (d). An incremental update of the distance metric D can be derived based on stochastic one-leave-out cross validation as detailed in [28]. The resulting learning rules can be embedded in an incremental learning system that automatically allocates new locally linear models as needed [24] (c.f. Algorithm 4).

Algorithm 3 Locally Weighted Projection Regression

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

Update the means of inputs and output:

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

$\beta_0^{n+1} = \frac{\lambda W^n \beta_0^{n+1} + wy}{W^{n+1}}$

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

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

c) $SS_i^{n+1} = \lambda SS_i^n + ws_i^2$

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

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

Algorithm 4 The complete LWPR pseudocode

Initialize RFWR with no receptive field (RF);
for every new training sample (x, y)
  for $k = 1 : K$
    calculate the activation from (1)
    update according to (7) and (8)
  IF no linear model was activated by more than *wgen*:
    create a new RF with $c = x$, $D = D_def$, $r = 2$

In this pseudocode algorithm, w_{gen} is a threshold that determines when to create a new receptive field, e.g., $w_{gen} = 0.1$, and D_{def} is the initial (usually diagonal) distance met- ric in Equation (1). The decomposition of D into $M^T M$ can be achieved with a Cholesky decomposition [22].

2. 4 Computational Complexity of LWPR

For a diagonal distance metric D_k 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. This property, besides the very fast learning abilities of LWPR, is among the most outstanding characteristics of the learning algorithm, unparalleled by any previous work to the best of our knowledge. 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. Robot Learning Examples

We applied RFWR and LWPR to various robot-learning problems. Common to all these examples is the fact that the learning algorithms acquired an internal model of a part of the control task, and that this model was used subsequently in designing a controller. Such model-based control and learning has also become increasingly more an accepted hypothesis of how biological movements acquire and perform skilled movement [18].

3. 1 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 1). The low-level robot controller ran in a compute-torque mode [12] 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 tech- niques from control theory [1]. 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 [7]. 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 2 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.

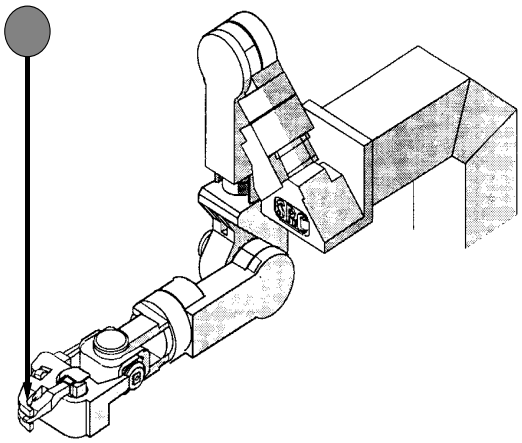


Fig1 Sarcos Dexterous Robot Arm

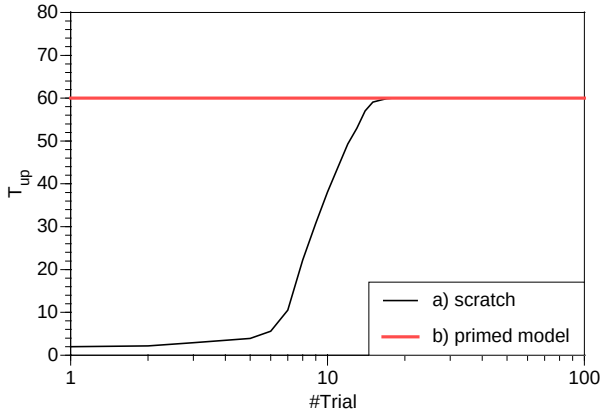


Fig2 Smoothed average of 10 learning curves of the robot for pole balancing. The trials were aborted after successful balancing of 60 seconds. We also tested long term performance of the learning system by running pole balancing for over an hour – the pole was never dropped.

3. 2 Inverse Dynamics Learning

As all our anthropomorphic robots are rather lightweight, 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 1) from a data set consisting of 45,000 data points, collected at 100Hz from the actual robot performing

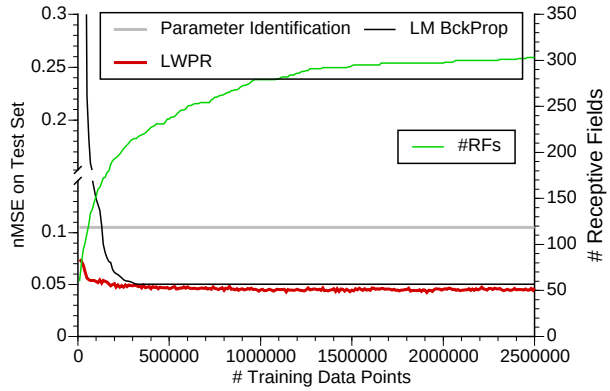


Fig3 Learning curve for learning the inverse dynamics model of the robot from a 50 dimensional data set that included 29 irrelevant dimensions.

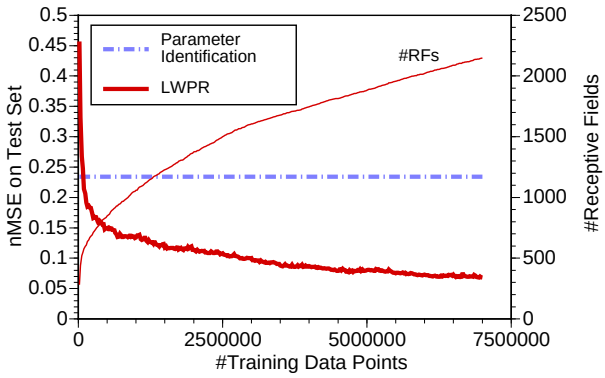


Fig4 Inverse dynamics learning for the right shoulder motor of the humanoid robot

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(0,0.052) 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 3 shows the learning results in comparison to parametric estimation of the inverse dynamics based on rigid body dynamics [1], 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 D 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.

We also applied LWPR to an even more complex robot, a 30 DOFs humanoid robot (For the robot architecture, refer to the paper by Kawato and Shibata in the current issue). 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 4, are similar to Figure 3. 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.

3. 3 Inverse Kinematics Learning

The major obstacle in learning inverse kinematics lies in the problem that the inverse kinematics of a redundant kinematic chain has infinitely many solutions. Thus, the learning algorithm has to acquire a particular inverse, and moreover, has to make sure that the inverse is actually a valid solution [17]. As noted by Bullock et al. [9], it is possible to transform the usually ill-defined problem of inverse kinematics learning into a well-defined problem by spatially localizing the learning, i.e. by learning a mapping from end effector velocity $\dot{x}$ and joint position θ to joint velocity $\dot{\theta}$ $((\dot{x}, q) \rightarrow \dot{q})$ while employing a spatially localized learning algorithm. LWPR has all the prerequisites to learn inverse kinematics in this suggested way. The inputs to the learning system are $z = (\dot{x}, q)$, and the outputs are $y = \dot{q}$. $\dot{x}$ can be in Cartesian coordinates if a calibrated

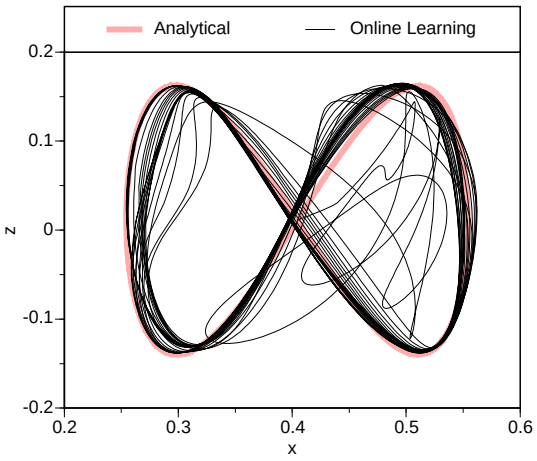


Fig5 Trajectory followed in the first 3 minutes when learning the inverse kinematics from scratch while attempting to perform figure-eight task

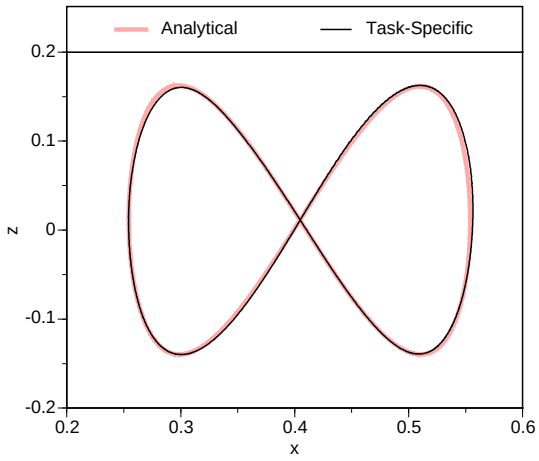


Fig6 Performance of the system after convergence of learning (about 10 minutes of training)

3D tracking system for the endeffector exists, but it could also be in uncalibrated image coordinates of two or more cameras – since LWPR can handle redundant inputs there is not restriction on the dimensionality of $\dot{x}$. For our humanoid robot, the dimensionality of the input z to LWPR is 29 (26 degrees-of-freedom neglecting the 4 degrees-of-freedom for the eyes, plus 3 Cartesian inputs), while the dimensionality of the output y is 26. By moving the robot while reading values for z and y from the sensors, training data is generated that can be added incrementally to the learning system – this process is often termed self-supervised learning.

An explicit optimization criterion can be incorporated in the learning process to bias the learning towards a particular solution of the inverse kinematics [13].

We implemented the inverse dynamics learning on our humanoid robot. The task goal was to track a figure-eight trajectory in Cartesian space created by simulated visual input to the robot. Figure 5 shows the progression of the systems performance from the beginning of the task to about 3 minutes into the learning. One can see that the system initially starts out making slow inaccurate movements. As it collects data, however, it rapidly converges towards the desired trajectory. Within a few more minutes of training on the task, the performance approached that seen in Figure 6, i.e., a tracking accuracy that is hardly distinguishable from an analytical solution to the inverse kinematics that we obtained in a previous work [26].

4. Conclusions

This paper presented Locally Weighted Learning algorithms for on-line 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 algorithms 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.

Reference

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

[2] C. G. Atkeson. Using local models to control movement. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 157–83. Morgan Kaufmann, 1989.

[3] C. G. Atkeson. Memory-based approaches to approximating continuous functions. In M. Casdagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting*, pages 503–521. Addison Wesley, Redwood City, CA, 1992.

[4] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[5] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75–113, 1997.

[6] C. G. Atkeson and S. Schaal. Memory-based neural networks for robot learning. *Neurocomputing*, 9:1–27, 1995.

[7] J. Baillieul and D. P. Martin. Resolution of kinematic redundancy. In *Proceedings of Symposia in Applied Mathematics*, volume 41, pages 49–89. American Mathematical Society, 1990.

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

[9] D. Bullock, S. Grossberg, and F. H. Guenther. A self-organizing neural model of motor equivalent reaching and tool use by a multijoint arm. *Journal of Cognitive Neuroscience*, 5(4):408–435, 1993.

[10] W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74:829–836, 1979.

[11] W. S. Cleveland and C. Loader. Smoothing by local regression: Principles and methods, 1995. Technical Report.

[12] J. J. Craig. *Introduction to robotics*. Addison-Wesley, Reading, MA, 1986.

[13] A. D'Souza, S. Vijayakumar, and S. Schaal. Learning inverse kinematics. In *Proc. Intl. Conf. on Intelligence in Robotics and Autonomous Systems (IROS 2001)*, (submitted).

[14] I. E. Frank and J. H. Friedman. A statistical view of some chemometric regression tools. *Technometrics*, 35(2):109–135, 1993.

[15] T. Hastie and C. Loader. Local regression: Automatic kernel carpentry. *Statistical Science*, 8:120–143, 1993.

[16] T. J. Hastie and R. J. Tibshirani. Nonparametric regression and classification: Part i: Nonparametric regression. In V. Cherkassky, J. H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications.*, pages 120–143. Springer, 1994.

[17] I. M. Jordan and Rumelhart. Supervised learning with a distal teacher. In *Cognitive Science*, volume 16, pages 307–354, 1992.

[18] M. Kawato. Internal models for motor control and trajectory planning. *Curr Opin Neurobiol*, 9(6):718–727, 1999.

[19] L. Ljung and T. Soderstrom. *Theory and practice of recursive identification*. Cambridge MIT Press, 1986.

[20] D. G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, 1995.

[21] A. W. Moore. Efficient memory-based learning for robot control. PhD. Thesis; Technical Report No. 229.

[22] W. P. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes in C: The art of scientific computing*. Press Syndicate, Cambridge, MA, 1989.

[23] S. Schaal. Is imitation learning the route to humanoid robots ? *Trends in Cognitive Science*, 3:233–242, 1999.

[24] S. Schaal and C. G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

[25] S. Schaal, S. Vijayakumar, and C. G. Atkeson. Local dimensionality reduction. In *Advances in Neural Information Processing Systems 10*, pages 633–639. MIT Press, Cambridge, MA, 1998.

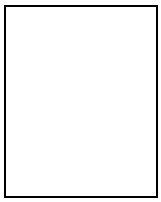
[26] G. Tevatia and S. Schaal. Inverse kinematics for humanoid robots. In *International Conference on Robotics and Automation (ICRA2000)*. San Fransisco, 2000.

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

[28] S. Vijayakumar, A. D'Souza, S. Schaal, T. Shibata, and J. Conradt. Statistical learning for humanoid robots. *Autonomous Robots*, (in press).

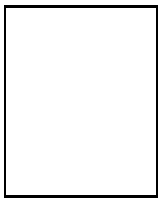
[29] S. Vijayakumar and S. Schaal. Locally weighted projection regression : An o(n) algorithm for incremental real time learning in high dimensional spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, volume 1, pages 288–293. Stanford, CA, 2000.

[30] H. Wold. Soft modeling by latent variables: the nonlinear iterative partial least squares approach. In J. Gani, editor, *Perspectives in Probability and Statistics, Papers in Honour of M. S. Bartlett*, pages 520–540. Academic Press, London, 1975.



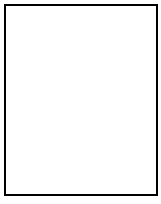
Stefan Schaal

Dr. Schaal is an Assistant Professor at the Department of Computer Science and the Neuroscience Program at the University of Southern California. He also holds additional appointments as Head of the Computational Learning Group of the Kawato Dynamic Brain Project (ERATO/JST) and as an Adjunct Assistant Professor at the Department of Kinesiology of the Pennsylvania State University. Before joining USC, Dr. Schaal was a postdoctoral fellow at the Department of Brain and Cognitive Sciences and the Artificial Intelligence Laboratory at MIT, an Invited Researcher at the ATR Human Information Processing Research Laboratories in Japan, and an Adjunct Assistant Professor at the Georgia Institute of Technology.



Sethu Vijayakumar

Dr. Vijayakumar is a Research Assistant Professor in the Department of Computer Science and the Neuroscience at the University of Southern California. He also holds additional appointments as a Research Scientist of the Kawato Dynamic Brain Project (ERATO/JST) and a Visiting Scientist of the RIKEN Brain Science Institute. His interests spans basic research in the field of statistical learning, motor control and computational neuroscience. Dr. Vijayakumar is a recipient of the IEEE Vincent Bendix Awarded, the 1995 ICNN Best Student Paper Award and the 1996 IEEE R.K.Wilson Award.



Aaron D'Souza

Aaron is graduate student in the Department of Computer Science at the University of Southern California. His research interests include machine learning, specifically statistical learning and artifical neural networks with applications in Humanoid robot control.