5 Approximate Nearest Neighbor Regression in Very High Dimensions

Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal

Fast and approximate nearest-neighbor search methods have recently become popular for scaling nonparameteric regression to more complex and high-dimensional applications. As an alternative to fast nearest neighbor search, training data can also be incorporated online into appropriate sufficient statistics and adaptive data structures, such that approximate nearest-neighbor predictions can be accelerated by orders of magnitude by means of exploiting the compact representations of these sufficient statistics. This chapter describes such an approach for locally weighted regression with locally linear models. Initially, we focus on *local* dimensionality reduction techniques in order to scale locally weighted learning to domains with very high dimensional input data. The key issue here revolves around obtaining a statistically robust and computationally inexpensive estimation of local linear models in such large spaces, despite potential irrelevant and redundant inputs. We develop a local version of partial least squares regression that fulfills all of these requirements, and embed it in an incremental nonlinear regression algorithm that can be shown to work efficiently in a number of complex applications. In the second part of the chapter, we introduce a novel Bayesian formulation of partial least squares regression that converts our nonparametric regression approach to a probabilistic formulation. Some of the heuristic components inherent in partial least squares can be eliminated with this new algorithm by means of efficient Bayesian regularization techniques. Evaluations are provided for all algorithms on various synthetic data sets and real-time learning examples with anthropomorphic robots and complex simulations.

5.1 Introduction

Despite the recent progress in statistical learning, nonlinear function approximation with high-dimensional input data remains a nontrivial problem, especially in incremental and real-time formulations. There is, how-

ever, an increasing number of problem domains where both these properties are important. Examples include the online 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 models for learning control, particularly in the context of high-dimensional movement systems like humans or humanoid robots. An ideal algorithm for such tasks needs to avoid potential numerical problems from redundancy in the input data, eliminate irrelevant input dimensions, keep the computational complexity of learning updates low while remaining data efficient, allow for online incremental learning, and, of course, achieve accurate function approximation and adequate generalization.

When looking for a learning framework to address these goals, one can identify two broad classes of function approximation methods: (i) methods which fit nonlinear functions *globally*, typically by input space expansions with predefined or parameterized basis functions and subsequent linear combinations of the expanded inputs, and (ii) methods which fit nonlinear functions *locally*, usually by using spatially localized simple (e.g., low-order polynomial) models in the original input space and automatically adjusting the complexity (e.g., number of local models and their locality) to accurately account for the nonlinearities and distributions of the target function. Interestingly, the current trends in statistical learning have concentrated on methods that fall primarily in the first class of global nonlinear function approximators, for example, Gaussian process regression(GPR)[48], support vector machine regression(SVMR)[40] and variational Bayes for mixture models[1](VBM)[13]. In spite of the solid theoretical foundations that these approaches possess in terms of generalization and convergence, they are not necessarily the most suitable for online learning in high-dimensional spaces. First, they require an a priori determination of the right modeling biases. For instance, in the case of GPR and SVMR, these biases involve selecting the right function space in terms of the choice of basis or kernel functions[44], and in VBM the biases are concerned with the the right number of latent variables and proper initialization.[2] Second, all these recent function approximator methods were developed primarily for batch data analysis and are not easily or efficiently adjusted for incrementally arriving data. For instance, in SVMR, adding a new data point can drastically change the outcome of the global optimization problem in terms of which data points actually become support vectors, such that all (or a carefully selected subset) of data has to be kept in memory for reevaluation. Thus, adding a new data point in SVMR is computationally rather expensive, a property that is also shared by GPR. VBM suffers from similar problems due to the need for storing and reevaluating data when adding new mixture components[43]. In general, it seems that most suggested Bayesian learning algorithms are computationally too expensive for real-time learning because they tend to represent the complete joint distribution of the data, albeit

as a conditionally independent factored representation. As a last point, incremental approximation of functions with global methods is prone to lead to negative interference when input distributions change[35]; such changes are, however, a typical scenario in many online learning tasks.

In contrast to the global learning methods described above, function approximation with spatially localized models, that is, nearest-neighbor(NN) techniques, are rather well suited for incremental and real-time learning[2]. Such nonparametric methods are very useful when there is limited knowledge about the model complexity such that the model resources need to be increased in a purely incremental and data-driven fashion, as demonstrated in previous work[35]. However, since these techniques allocate resources to cover the input space in a localized fashion, in general, with an increasing number of input dimensions, they encounter an exponential explosion in the number of local models required for accurate approximation, often referred to as the "curse of dimensionality"[31]. Hence, at the outset, high-dimensional function approximation seems to be computationally infeasible for local nonparametric learning.

Nonparametric learning in high-dimensional spaces with *global* methods, however, has been employed successfully by using techniques of projection regression (PR). PR copes with high-dimensional inputs by decomposing multivariate regressions into a superposition of single variate regressions along a few selected projections in input space. The major difficulty of PR lies in the selection of efficient projections, that is, how to achieve the best fitting result with as few univariate regressions as possible. Among the best known PR algorithms are projection pursuit regression [11], and its generalization in the form of generalized additive models[16]. Sigmoidal neural networks can equally be conceived of as a method of projection regression, in particular when new projections are added sequentially, e.g., as in cascade correlation[9].

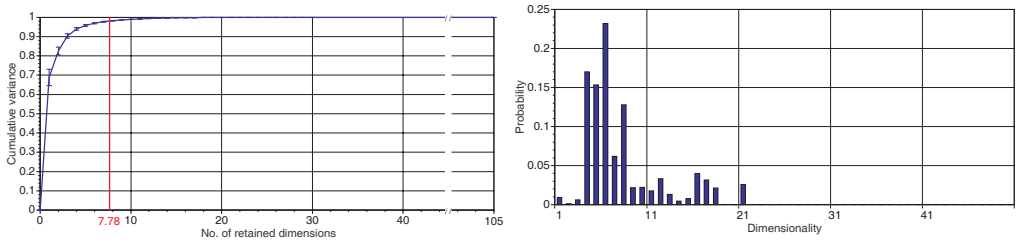
In this chapter we suggest a method of extending the beneficial properties of *local nonparametric learning* to high-dimensional function approximation problems. The prerequisite of our approach is that the high-dimensional learning problems we address have locally low dimensional distributions, an assumption that holds for a large class of real-world data (see below). If distributions are locally low-dimensional, the allocation of local models can be restricted to these thin distributions, and only a tiny part of the entire high dimensional space needs to be filled with local models. Thus, the curse of dimensionality of spatially localized model fitting can be avoided. Under these circumstances, an alternative method of projection regression can be derived, focusing on finding efficient *local* projections. Local projections can be used to accomplish local function approximation in the neighborhood of a given query point with traditional local nonparametric approaches, thus inheriting most of the statistical properties from the well established methods of locally weighted learning and nearest-neighbor regression[15, 2]. As

this chapter will demonstrate, the resulting learning algorithm combines the fast, efficient, and incremental capabilities of the nonparametric techniques while alleviating the problems faced due to high-dimensional input domains through local projections.

In the following sections, we first motivate why many high dimensional learning problems have locally low-dimensional data distributions such that the prerequisites of our local learning system are justified. Second, we address the question of how to find good local projections by looking into various schemes for performing dimensionality reduction for regression. Third, we embed the most efficient and robust of these projection algorithms in an incremental nonlinear function approximator[45] capable of automatically adjusting the model complexity in a purely data-driven fashion. Finally, a new Bayesian approach is suggested to reformulate our algorithms in a probabilistic framework, thus removing several levels of open parameters in the techniques. In several evaluations, in both on synthetic and real world data, in the resulting incremental learning system demonstrates high accuracy for function fitting in very high-dimensional spaces, robustness toward irrelevant and redundant inputs, as well as low computational complexity. Comparisons will prove the competitiveness with other state-of-the-art learning systems.

5.2 Evidence for Low-Dimensional Distributions

The development of our learning system in the next sections relies on the assumption that high-dimensional data sets have locally low dimensional distributions, an assumption that requires some clarification. Across domains like vision, speech, motor control, climate patterns, human gene distributions, and a range of other physical and biological sciences, various researchers have reported evidence that corroborate the fact that the true intrinsic dimensionality of high-dimensional data is often very low[42, 27, 47]. We interpret these findings as evidence that the physical world has a significant amount of coherent structure that expresses itself in terms of a strong correlations between different variables that describe the state of the world at a particular moment in time. For instance, in computer vision it is quite obvious that neighboring pixels of an image of a natural scene have redundant information. Moreover, the probability distribution of natural scenes in general has been found to be highly structured such that it lends itself to a sparse encoding in terms of set of basis functions[24, 3]. Another example comes from our own research on human motor control. In spite of the fact that humans can accomplish movement tasks in almost arbitrary ways, thus possibly generating arbitrary distributions of the variables that describe their movements, behavioral research has discovered a tremendous amount of regularity within and across individuals[20, 32]. These regular-



(a) (b)

Figure 5.1 Dimensionality Analysis. (*a*) The cumulative variance accounted vs. the local dimensionality (averaged across all mixture models). (*b*) The distribution of the effective dimensionality across all mixture models.

ities lead to a locally low-dimensional data distribution, as illustrated in the example in figure5.1. In this analysis[6], we assessed the intrinsic dimensionality of data collected from full body movement of several human subjects, collected with a special full-body exoskeleton that recorded simultaneously 35 degrees of freedom(DOF) of the joint angular movement of the subjects at 100Hz sampling frequency. Subjects performed a variety of daily-life tasks (e.g., walking, object manipulation, reaching, etc.) until about a gigabyte of data was accumulated. Our analysis examined the local dimensionality of the joint distribution of positions, velocities, and accelerations of the collected data, that is, a 105-dimensional data set, as would be needed as inputs to learn an inverse dynamics model for motor control[20]. As an analysis tool, we employed a variational Bayesian mixture of factor analyzers that automatically estimated the required number of mixture components[13]. As demonstrated in figure 5.1(a), the local dimensionality was around five to eight dimensions, computed based on the average number of significant latent variables per mixture component. Figure 5.1(b) shows the distribution of the effective dimensionality across all mixture models.

In summary, the results from our analysis and other sources in the literature show that there is a large class of high-dimensional problems that can be treated locally in much lower dimensions if one can determine appropriate regions of locality and the local projections that model the corresponding low-dimensional distributions. As a caveat, however, it may happen that such low dimensional distributions are embedded in additional dimensions that are irrelevant to the problem at hand but have considerable variance. In the context of regression, it will thus be important to only model those local dimensions that carry information that is important for the regression and eliminate all other dimensions, that is, to perform local dimensionality reduction with regression in mind and not just based on input or joint input-output distributions, as discussed in the next section.

5.3 Local Dimensionality Reduction

Assuming that data are characterized by locally low dimensional distributions, efficient algorithms are needed to exploit this property. For this purpose, we will focus on locally weighted learning (LWL) methods[2] because they allow us to adapt a variety of linear dimensionality reduction techniques for the purpose of nonlinear function approximation (see section 5.4) and because they are easily modified for incremental learning. LWL-related methods have also found widespread application in mixture models [19, 50, 13] such that the results of this section can contribute to this field, too.

In pursuit of the question of what is the "right" method to perform local dimensionality reduction for regression, Schaal et al.[36] examined several candidate techniques theoretically and empirically. In particular, the authors focused on locally weighted versions of principal component regression (LWPCR), joint data principal component analysis (LWPCA), factor analysis (LWFA), and partial least squares (LWPLS). We will briefly revisit some main insights of this work before moving on to the development of our key learning algorithm.

5.3.1 The Locally Weighted Regression Model

The learning problems considered here assume the standard regression model:

$$y = f(\mathbf{x}) + \epsilon,$$

where $\mathbf{x}$ denotes the d-dimensional input vector, y the (for simplicity) scalar output, and ϵ a mean-zero random noise term. When only a local subset of data in the vicinity of a point $\mathbf{x}_c$ is considered and the locality is chosen appropriately, a low-order polynomial can be employed to model this local subset. Due to a favorable compromise between computational complexity and quality of function approximation[15], we choose linear models

$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon.$$

A measure of locality for each data point, the weight w_i, is computed from a Gaussian kernel:

$$w_i = \exp(-0.5(\mathbf{x}_i - \mathbf{x}_c)^T \mathbf{D}(\mathbf{x}_i - \mathbf{x}_c)), \quad \mathbf{W} \equiv diag\{w_1, \ldots, w_M\}, \tag{5.1}$$

where $\mathbf{D}$ is a positive semidefinite distance metric which determines the size and shape of the neighborhood contributing to the local model[2]. The weights w_i will enter all following algorithms to assure spatial localization in input space. In particular, zero mean prerequisites of several algorithms are ensured by subtracting the weighted mean $\overline{\mathbf{x}}$ or $\overline{y}$ from the data, where

$$\overline{\mathbf{x}} = \sum_{i=1}^{M} w_i \mathbf{x}_i / \sum_{i=1}^{M} w_i, \text{ and } \overline{y} = \sum_{i=1}^{M} w_i y_i / \sum_{i=1}^{M} w_i, \tag{5.2}$$

and M denotes the number of data points. For simplicity, and without loss of generality, we will thus assume in the derivations in the next sections that all input and output data have a weighted mean of zero with respect to a particular weighting kernel. The input data are summarized in the rows of the matrix $\mathbf{X}=[\mathbf{x}_1\ \mathbf{x}_2\ ...\mathbf{x}_M]^T$, the corresponding outputs are the coefficients of the vector $\mathbf{y}$, and the corresponding weights, determined from (5.1), are in the diagonal matrix $\mathbf{W}$. In some cases, we need the joint input and output data, denoted as $\mathbf{Z}=[\mathbf{z}_1\ \mathbf{z}_2...\mathbf{z}_M]^T=[\mathbf{X}\ \mathbf{y}]$.

5.3.2 Locally Weighted Factor Analysis

Factor analysis[8] is a density estimation technique that assumes that the observed data $\mathbf{z}$ were actually generated from a lower-dimensional process, characterized by *k-latent* or *hidden* variables $\mathbf{v}$ that are all independently distributed with mean zero and unit variance. The observed variables are generated from the latent variables through the transformation matrix $\mathbf{U}$ and additive mean zero independent noise $\boldsymbol{\epsilon}$ with diagonal covariance matrix $\boldsymbol{\Omega}$:

$$\mathbf{z} = \mathbf{U}\mathbf{v} + \boldsymbol{\epsilon}, \tag{5.3}$$

where

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ y \end{bmatrix}, \boldsymbol{\epsilon} = \begin{bmatrix} \boldsymbol{\epsilon}_x \\ \epsilon_y \end{bmatrix}, E\{\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\} = \boldsymbol{\Omega}, \tag{5.4}$$

and $E\{.\}$ denotes the expectation operator. If both $\mathbf{v}$ and $\boldsymbol{\epsilon}$ are normally distributed, the parameters $\boldsymbol{\Omega}$ and $\mathbf{U}$ can be obtained iteratively by the expectation-maximization (EM) algorithm[28].

Factor analysis can be adapted for linear regression problems by assuming that $\mathbf{z}$ was generated with

$$\mathbf{U} = [\mathbf{I}^d\ \boldsymbol{\beta}]^T, \tag{5.5}$$

$$\mathbf{v} = \mathbf{x}, \tag{5.6}$$

where $\boldsymbol{\beta}$ denotes the vector of regression coefficients of the linear model $y = \boldsymbol{\beta}^T\mathbf{x}$ and $\mathbf{I}^d$ the d-dimensional identity matrix. For the standard regression model, $\boldsymbol{\epsilon}_x$ would be zero, that is, we consider noise contamination in the output only; for numerical reasons, however, some remaining variance needs to be permitted and we prefer to leave it to the EM algorithm to find the optimal values of the covariance $\boldsymbol{\Omega}$. After calculating $\boldsymbol{\Omega}$ and $\mathbf{U}$ with EM in joint data space as formulated in (5.3), an estimate of $\boldsymbol{\beta}$ can be derived from the conditional probability $p(y|\mathbf{x})$. Let us denote $\mathbf{Z}=[\mathbf{z}_1\ \mathbf{z}_2...\mathbf{z}_M]^T$ and $\mathbf{V}=[\mathbf{v}_1\ \mathbf{v}_2...\mathbf{v}_M]^T$. The locally weighted version (LWFA) of $\boldsymbol{\beta}$ can be obtained together with an estimate of the factors $\mathbf{v}$ from the joint weighted covariance matrix $\boldsymbol{\Psi}$ of $\mathbf{z}$ and $\mathbf{v}$ as

$$E\{\begin{bmatrix} y \\ \mathbf{v} \end{bmatrix}|\mathbf{x}\} = \begin{bmatrix} \boldsymbol{\beta}^T \\ \mathbf{B} \end{bmatrix}\mathbf{x} = \Psi_{21}\Psi_{11}^{-1}\mathbf{x}, \tag{5.7}$$

1. Initialize: $\mathbf{X}_{res} = \mathbf{X}$, $\mathbf{y}_{res} = \mathbf{y}$
2. **for** $i = 1$ **to** k **do**
   (a) $\mathbf{u}_i = \mathbf{X}_{res}^T \mathbf{y}_{res}$.
   (b) $\beta_i = \mathbf{s}_i^T \mathbf{y}_{res} / (\mathbf{s}_i^T \mathbf{s}_i)$ where $\mathbf{s}_i = \mathbf{X}_{res} \mathbf{u}_i$.
   (c) $\mathbf{y}_{res} = \mathbf{y}_{res} - \mathbf{s}_i \beta_i$, $\mathbf{X}_{res} = \mathbf{X}_{res} - \mathbf{s}_i {\mathbf{p}_i}^T$ where $\mathbf{p}_i = \mathbf{X}_{res}^T \mathbf{s}_i / (\mathbf{s}_i^T \mathbf{s}_i)$.

Algorithm 5.1: Outline of PLS regression algorithm

where

$$\mathbf{\Psi} = [\mathbf{Z}^T \ \mathbf{V}^T]\mathbf{W} \begin{bmatrix} \mathbf{Z} \\ \mathbf{V} \end{bmatrix} / \sum_{i=1}^{M} w_i = \begin{bmatrix} \Omega + \mathbf{U}\mathbf{U}^T & \mathbf{U} \\ \mathbf{U}^T & \mathbf{I}^d \end{bmatrix} = \begin{bmatrix} \Psi_{11} & \Psi_{12} \\ \Psi_{21} & \Psi_{22} \end{bmatrix},$$

and $\mathbf{B}$ is a matrix of coefficients involved in estimating the factors $\mathbf{v}$. Note that unless the noise $\boldsymbol{\epsilon}_x$ is zero, the estimated $\boldsymbol{\beta}$ is different from the true $\boldsymbol{\beta}$ as it tries to optimally average out the noise in the data. Thus, factor analysis can also be conceived of as a tool for linear regression with noise-contaminated inputs.

5.3.3 Partial Least Squares

Partial least squares (PLS)[49, 10], a technique used extensively in chemometrics, recursively computes orthogonal projections of the input data and performs single variable regressions along these projections on the residuals of the previous iteration step. It is outlined in algorithm 5.1. 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 $\mathbf{s}$ in order to ensure the orthogonality of all the projections $\mathbf{u}$ (step 2c). Actually, this additional regression could be avoided by replacing $\mathbf{p}$ with $\mathbf{u}$ in step 2c, similar to techniques used in PCA[30]. However, using this regression step leads to better performance of the algorithm as PLS chooses the most effective projections if the input data have a spherical distribution: in the spherical case, with only one projection, PLS will find the direction of the gradient and achieve optimal regression results. The regression step in 2(c) in algorithm 5.1 chooses the reduced input data $\mathbf{X}_{res}$ such that the resulting data vectors have minimal norms and, hence, push the distribution of $\mathbf{X}_{res}$ to become more spherical. An additional consequence of 2(c) is that all the projections $\mathbf{s}_i$ become uncorrelated, that is, $\mathbf{s}_j^T \mathbf{s}_i = 0 \ \forall i \neq j$, a property which will be important in the derivations below.

5.3.4 Other Techniques

There are several other approaches to local dimensionality reduction, including principal component regression[22, 45], and principal component in joint space(LWPCA_J). Both of these methods can be regarded as locally weighted factor analysis models with specific constraints on the structure of the data generating model, and restrictive assumptions about the generative probabilistic model. As shown in [36], the methods are always inferior to the full formulation of factor analysis from the previous section.

5.3.5 Which Approach to Choose?

Schaal et al.[36] demonstrated that both LWPLS and LWFA perform approximately the same under a large variety of evaluation sets. This result was originally slightly surprising, as LWPLS has more of a heuristic component than the theoretically principled factor analysis, a fact that leads to naturally favoring factor analysis models. However, there are two components that lift LWPLS above LWFA for our approximate nearest-neighbor approach. First, we wish to learn incrementally and remain computationally inexpensive in high dimensions. For this purpose, one would like to have a constructive approach to factor analysis, that is, an approach that adds latent variables as needed, a pruning approach, starting from the maximal latent variable dimensionality, would be unacceptably expensive. Empirically, [36] found that LWFA performs a lot worse if the latent variable space is underestimated, meaning that a constructive approach to LWFA would have transients of very bad performance until enough data were encountered to correctly estimate the latent space. For applications in robot control, for instance, such behavior would be inappropriate.

Second, besides redundant variables, we also expect a large number of irrelevant variables in our input data. This scenario, however, is disadvantageous for LWFA, as in the spirit of a density estimation technique, it needs to model the full dimensionality of the latent space, and not just those dimensions that matter for regression. In empirically evaluations (not shown here) similar to [36], we confirmed that LWFA's performance degrades strongly if it does not have enough latent variables to represent the irrelevant inputs. LWPLS, in contrast, uses the projection step based on input-output correlation to exclude irrelevant inputs, which exhibits very reliable and good performance even if there are many irrelevant and redundant inputs.

These considerations make LWPLS a superior choice for approximate nearest-neighbors in high dimensions. In the next section, we embed LWPLS in an incremental nonlinear function approximator to demonstrate its abilities for nontrivial function-fitting problems.

Notation	Affectation
M	No. of training data points
N	Input dimensionality (i.e., dim. of $\mathbf{x}$)
$k = (1 : K)$	No. of local models
$r = (1 : R)$	No. of local projections used by PLS
$\{\mathbf{x}_i, y_i\}_{i=1}^M$	Training data
$\{\mathbf{z}_i\}_{i=1}^M$	Lower-dimensional projection of input data $\mathbf{x}_i$
$\{z_{i,r}\}_{r=1}^R$	Elements of projected input
$\mathbf{X}$, $\mathbf{Z}$	Batch representations of input and projected data
w	Weight or activation of data $(\mathbf{x}, y)$ with respect to local model or receptive field(RF) center $\mathbf{c}$
$\mathbf{W}$	Weight matrix: $\mathbf{W} \equiv diag\{w_1, \ldots, w_N\}$
W^n	Cumulative weights seen by local model
$a_{var,r}^n$	Trace variable for incremental computation of rth dimension of variable var after seeing n data points

Table 5.1 Legend of indexes and symbols used for LWPR

5.4 Locally Weighted Projection Regression

For nonlinear function approximation, the core concept of our learning system - locally weighted projection regression (LWPR)- is to find approximations by means of piecewise linear models[2]. Learning involves automatically determining the appropriate *number* of local models K, the parameters $\boldsymbol{\beta}_k$ of the *hyperplane* in each model, and also the *region of validity*, called receptive field (RF), parameterized as a distance metric $\mathbf{D}_k$ in a Gaussian kernel:

$$w_k = exp(-\frac{1}{2}(\mathbf{x} - \mathbf{c}_k)^T \mathbf{D}_k (\mathbf{x} - \mathbf{c}_k)). \tag{5.8}$$

Given a query point $\mathbf{x}$, every linear model calculates a prediction $\hat{y}_k(\mathbf{x})$. The total output of the learning system is the normalized weighted mean of all K linear models:

$$\hat{y} = \sum_{k=1}^{K} w_k \hat{y}_k / \sum_{k=1}^{K} w_k, \tag{5.9}$$

also illustrated in Figure 5.2. The centers $\mathbf{c}_k$ of the RFs remain fixed in order to minimize negative interference during incremental learning that could occur due to changing input distributions[35]. Local models are created on an "as-needed" basis as described in sub-section 5.4.2. Table 5.1 provides a reference list of indices and symbols that are used consistently across the description of the LWPR algorithm.

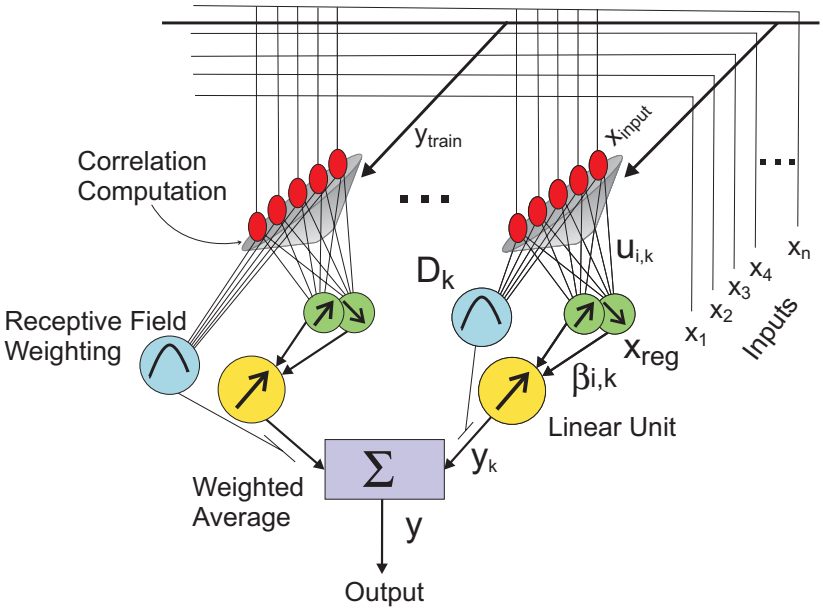


Figure 5.2 Information processing unit of LWPR.

5.4.1 Learning with Locally Weighted Projection Regression

Despite its appealing simplicity, the "piecewise linear modeling" approach becomes numerically brittle and computationally too expensive in high-dimensional input spaces when using ordinary linear regression to determine the local model parameters[35]. Given the empirical observation (cf. section 5.2) that high-dimensional data often lie on locally low-dimensional distributions, and given the algorithmic results in section 5.3, we will thus use local projection regression, that is, LWPLS, within each local model to fit the hyperplane. As a significant computational advantage, we expect that far fewer projections than the actual number of input dimensions are needed for accurate learning. The next sections will describe the necessary modifications of LWPLS for this implementation, embed the local regression into the LWL framework, explain a method of automatic distance metric adaptation, and finish with a complete nonlinear learning scheme, called locally weighted projection regression (LWPR).

Incremental Computation of Projections and Local Regression

For incremental learning, that is, a scheme that does not explicitly store any training data, the sufficient statistics of the learning algorithm need to be accumulated in appropriate variables. Algorithm 5.2[46] provides suitable incremental update rules. The variables $a_{zz,r}$, $a_{zres,r}$, and $\mathbf{a}_{xz,r}$ are sufficient statistics that enable us to perform the univariate regressions in step 2c.1.2 and step 2c.2.2, similar to recursive least squares, that is, a fast Newton-like incremental learning technique. $\lambda \in [0, 1]$ denotes a forgetting factor that allows exponential forgetting of older data in the sufficient statistics.

1. Initialization: (# data points seen $n = 0$)
$\mathbf{x}_0^0 = \mathbf{0}$, $\beta_0^0 = 0$, $W^0 = 0$, $\mathbf{u}_r^0 = \mathbf{0}$; $r = 1 : R$

2. Incorporating new data: Given training point $(\mathbf{x}, y)$

2a. Compute activation and update the means

1. $w = exp(-\frac{1}{2}(\mathbf{x} - \mathbf{c})^T \mathbf{D}(\mathbf{x} - \mathbf{c}))$; $W^{n+1} = \lambda W^n + w$
2. $\mathbf{x}_0^{n+1} = (\lambda W^n \mathbf{x}_0^n + w\mathbf{x})/W^{n+1}$; $\beta_0^{n+1} = (\lambda W^n \beta_0^n + wy)/W^{n+1}$

2b. Compute the current prediction error

$\mathbf{x}_{res,1} = \mathbf{x} - \mathbf{x}_0^{n+1}$, $\hat{y} = \beta_0^{n+1}$

Repeat for $r = 1 : R$ (# projections)

1. $z_r = \mathbf{x}_{res,r}^T \mathbf{u}_r^n / \sqrt{\mathbf{u}_r^{n\,T} \mathbf{u}_r^n}$
2. $\hat{y} \leftarrow \hat{y} + \beta_r^n z_r$
3. $\mathbf{x}_{res,r+1} = \mathbf{x}_{res,r} - z_r \mathbf{p}_r^n$
4. $MSE_r^{n+1} = \lambda MSE_r^n + w\ (y - \hat{y})^2$

$e_{cv} = y - \hat{y}$

2c. Update the local model

$res_1 = y - \beta_0^{n+1}$

For $r = 1 : R$ (# projections)

2c.1 *Update the local regression and compute residuals*

1. $a_{zz,r}^{n+1} = \lambda\ a_{zz,r}^n + w\ z_r^2$; $a_{zres,r}^{n+1} = \lambda\ a_{zres,r}^n + w\ z_r\ res_r$
2. $\beta_r^{n+1} = a_{zres,r}^{n+1} / a_{zz,r}^{n+1}$
3. $res_{r+1} = res_r - z_r \beta_r^{n+1}$
4. $\mathbf{a}_{xz,r}^{n+1} = \lambda\ \mathbf{a}_{xz,r}^n + w\mathbf{x}_{res,r} z_r$

2c.2 *Update the projection directions*

1. $\mathbf{u}_r^{n+1} = \lambda\ \mathbf{u}_r^n + w\mathbf{x}_{res,r}\ res_r$
2. $\mathbf{p}_r^{n+1} = \mathbf{a}_{xz,r}^{n+1} / a_{zz,r}^{n+1}$

$e = res_{r+1}$

3. Predicting with novel data ($\mathbf{x}_q$):
Initialize: $y_q = \beta_0, \mathbf{x}_q = \mathbf{x}_q - \mathbf{x}_0$
Repeat for $r = 1 : R$

1. $y_q \leftarrow y_q + \beta_r s_r$ where $s_r = \mathbf{u}_r^T \mathbf{x}_q$
2. $\mathbf{x}_q \leftarrow \mathbf{x}_q - s_r \mathbf{p}_r^n$

Note: The subscript k referring to the kth local model is omitted throughout this table since we are referring to updates in one local model or RF.

Algorithm 5.2: Incremental locally weighted PLS for one RF centered at $\mathbf{c}$

Forgetting is necessary in incremental learning since a change of some learning parameters will affect a change in the sufficient statistics; such forgetting factors are a standard technique in recursive system identification [21]. It can be shown that the prediction error of step 2b corresponds to the leave-one-out cross-validation error of the current point *after* the regression parameters were updated with the data point; hence, it is denoted by e_{cv}.

In algorithm 5.2, for $R = N$, that is, the same number of projections as the input dimensionality, the entire input space would be spanned by the projections $\mathbf{u}_r$ and the regression results would be identical to that of ordinary linear regression[49]. However, once again, we would like to emphasize the important properties of the local projection scheme. First, if all the input variables are statistically independent and have equal variance,[3] PLS will find the optimal projection direction $\mathbf{u}_r$ in roughly a single sweep through the training data; the optimal projection direction corresponds to the gradient of the local linearization parameters of the function to be approximated. Second, choosing the projection direction from correlating the input and the output data in step 2b.1 automatically excludes irrelevant input dimensions. And third, there is no danger of numerical problems due to redundant input dimensions as the univariate regressions can easily be prevented from becoming singular.

Given that we will adjust the distance metric to optimize the local model approximation (see below), it is also possible to perform LWPR with only *one* projection direction (denoted as LWPR-1). In this case, this distance metric will have to be adjusted to find the optimal receptive field size for a local linearization as well as to make the locally weighted input distribution spherical. An appropriate learning rule of the distance metric can accomplish this adjustment, as explained below. It should be noted that LWPR-1 is obtained from algorithm5.2 by setting $R = 1$.

Adjusting the Shape and Size of the Receptive Field

The distance metric $\mathbf{D}$ and hence the locality of the receptive fields can be learned for each local model individually by stochastic gradient descent in a penalized leave-one-out cross-validation cost function[35]:

$$J = \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} w_i (y_i - \hat{y}_{i,-i})^2 + \frac{\gamma}{N} \sum_{i,j=1}^{N} D_{ij}^2, \tag{5.10}$$

where M denotes the number of data points in the training set. The first term of the cost function is the mean leave-one-out cross-validation error of the local model (indicated by the subscript $i, -i$) which ensures proper generalization[35]. The second term, the penalty term, makes sure that receptive fields cannot shrink indefinitely in case of large amounts of training data; such shrinkage would be statistically correct for asymptotically unbiased function approximation, but it would require maintaining an ever

Table 5.2 Derivatives for distance metric update

For the current data point $\mathbf{x}$*, its PLS projection* $\mathbf{z}$ *and activation* w*:*
(Refer to table 5.2 for some of the variables)

$$
\begin{aligned}
\frac{\partial J}{\partial \mathbf{M}} &\approx (\sum_{i=1}^{M} \frac{\partial J_1}{\partial w}) \frac{\partial w}{\partial \mathbf{M}} + \frac{w}{W^{n+1}} \frac{\partial J_2}{\partial \mathbf{M}} \text{ [stochastic update of (5.12)]} \\
\frac{\partial w}{\partial M_{kl}} &= -\frac{1}{2} w (\mathbf{x}-\mathbf{c})^T \frac{\partial \mathbf{D}}{\partial M_{kl}} (\mathbf{x}-\mathbf{c}); \quad \frac{\partial J_2}{\partial M_{kl}} = 2\frac{\gamma}{N} \sum_{i,j=1}^{N} D_{ij} \frac{\partial D_{ij}}{\partial M_{kl}} \\
\frac{\partial D_{ij}}{\partial M_{kl}} &= M_{kj}\delta_{il} + M_{ki}\delta_{jl}; \text{ where } \delta_{ij} = 1 \text{ if } i = j, \text{ else } \delta_{ij} = 0. \\
\sum_{i=1}^{M} \frac{\partial J_1}{\partial w} &= \frac{e_{cv}^2}{W^{n+1}} - \frac{2\ e}{W^{n+1}} \mathbf{q}^T \mathbf{a}_H^n - \frac{2}{W^{n+1}} {\mathbf{q}^2}^T \mathbf{a}_G^n - \frac{a_E^{n+1}}{(W^{n+1})^2} \\
& \quad where\ \ \mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_R \end{bmatrix} \mathbf{z}^2 = \begin{bmatrix} z_1^2 \\ \vdots \\ z_R^2 \end{bmatrix} \mathbf{q} = \begin{bmatrix} z_1/a_{zz,1}^{n+1} \\ \vdots \\ z_R/a_{zz,R}^{n+1} \end{bmatrix} \\
\mathbf{a}_H^{n+1} &= \lambda \mathbf{a}_H^n + \frac{w\ e_{cv} \mathbf{z}}{(1-h)}; \quad \mathbf{a}_G^{n+1} = \lambda \mathbf{a}_G^n + \frac{w^2 e_{cv}^2 \mathbf{z}^2}{(1-h)} \\
& \quad \text{where } h = w\mathbf{z}^T\mathbf{q} \\
a_E^{n+1} &= \lambda a_E^n + w e_{cv}^2
\end{aligned}
$$

increasing number of local models in the learning system, which is computationally too expensive. The tradeoff parameter γ can be determined either empirically or from assessments of the maximal local curvature of the function to be approximated[34]; in general, results are not very sensitive to this parameter[35] as it primarily affects resource efficiency.

It should be noted that due to the *local* cost function in (5.10), learning becomes entirely localized, too, that is, no parameters from other local models are needed for updates as, for instance, in competitive learning with mixture models. Moreover, minimizing (5.10) can be accomplished in an incremental way *without* keeping data in memory[35]. This property is due to a reformulation of the leave-one-out cross-validation error as the PRESS residual error[4]. As detailed in[35] the bias-variance tradeoff is thus resolved for every local model *individually* such that an increasing number of local models will not lead to overfitting; indeed, it leads to better approximation results due to model averaging [e.g. (5.9)]in the sense of committee machines[25].

In ordinary weighted linear regression, expanding (5.10) with the PRESS residual error results in

$$J = \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} \frac{w_i (y_i - \hat{y}_i)^2}{(1 - w_i \mathbf{x}_i^T \mathbf{P} \mathbf{x}_i)^2} + \frac{\gamma}{N} \sum_{i,j=1}^{N} D_{ij}^2, \tag{5.11}$$

where $\mathbf{P}$ corresponds to the inverted weighted covariance matrix of the input data. Interestingly, the PRESS residuals of (5.11) can be *exactly* formulated in terms of the PLS projected inputs $\mathbf{z}_i \equiv [z_{i,1} \dots z_{i,R}]^T$ (algorithm 5.2) as

$$\begin{aligned} J &= \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} \frac{w_i (y_i - \hat{y}_i)^2}{(1 - w_i \mathbf{z}_i^T \mathbf{P}_z \mathbf{z}_i)^2} + \frac{\gamma}{N} \sum_{i,j=1}^{N} D_{ij}^2 \\ &\equiv \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} J_1 + \frac{\gamma}{N} J_2, \end{aligned} \tag{5.12}$$

where $\mathbf{P}_z$ corresponds to the covariance matrix computed from the projected inputs $\mathbf{z}_i$ for $R = N$, that is, the $\mathbf{z}_i$'s span the same full-rank input space[4] as the $\mathbf{x}_i$'s in (5.11). It can also been deduced that $\mathbf{P}_z$ is diagonal, which greatly contributes to the computational efficiency of our update rules. Based on this cost function, the distance metric in LWPR is learned by gradient descent:

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

where $\mathbf{M}$ is an upper triangular matrix resulting from a Cholesky decomposition of $\mathbf{D}$. Following[35], a stochastic approximation of the gradient $\frac{\partial J}{\partial \mathbf{M}}$ of (5.12) can be derived by keeping track of several sufficient statistics as shown in table 5.2. It should be noted that in these update laws, we treated the PLS projection direction and hence $\mathbf{z}$ as if it were independent of the distance metric, such that chain rules need not be taken throughout the entire PLS recursions. Empirically, this simplification did not seem to have any negative impact and reduced the update rules significantly.

5.4.2 The Complete LWPR Algorithm

All update rules can be combined in an incremental learning scheme that automatically allocates new locally linear models as needed. The concept of the final learning network is illustrated in figure 5.2 and an outline of the final LWPR algorithm is shown in algorithm 5.3.

In this pseudocode, w_{gen} is a threshold that determines when to create a new receptive field, as discussed in [35], w_{gen} is a computational efficiency parameter and not a complexity parameter as in mixture models. The closer w_{gen} is set to 1, the more overlap local models will have, which is beneficial in the spirit of committee machines (cf. [35, 25]) but more costly to compute; in general, the more overlap is permitted, the better the function fitting results, without any danger that the increase in overlap can lead to overfitting. $\mathbf{D}_{def}$ is the initial (usually diagonal) distance metric in (5.8). 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

```
1:  Initialize the LWPR with no receptive field (RF)
2:  for every new training sample (x,y) do
3:     for k=1 to K(# of receptive fields) do
4:        calculate the activation from (5.8)
5:        update projections and regression (algorithm 5.2) and distance metric (table 5.2),
6:        check if number of projections needs to be increased (cf. subsection 5.4.2).
7:     end for
8:     if no RF was activated by more than w_gen then
9:        create a new RF with R = 2, c = x, D = D_def.
10:    end if
11: end for
```

Algorithm 5.3: Pseudocode of the complete LWPR algorithm

included in a local model, that is, step 2b.4 in algorithm 5.2. If the MSE at the next projection does not decrease more than a certain percentage of the previous MSE, that is, $\frac{MSE_{r+1}}{MSE_r} > \phi$, where $\phi \in [0, 1]$, the algorithm will stop adding new projections locally. As MSE_r can be interpreted as an approximation of the leave-one-out cross-validation error of each projection, this threshold criterion avoids problems due to overfitting. Due to the need to compare the MSE of two successive projections, LWPR needs to be initialized with at least two projection dimensions.

Speedup for Learning from Trajectories

If in incremental learning, training data are generated from trajectories, that is, data are temporally correlated, it is possible to accelerate lookup and training times by taking advantage of the the fact that two consecutively arriving training points are close neighbors in input space. For such cases, we added a special data structure to LWPR that allows restricting updates and lookups only to a small fraction of local models instead of exhaustively sweeping through all of them. For this purpose, each local model maintains a list of all other local models that overlap sufficiently with it. Sufficient overlap between two models 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 (5.8) of one of the local models gives the activation w due to this point. The two local models are listed as sufficiently overlapping if $w \geq w_{gen}$ (cf. algorithm 5.3). 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 counter for each local model ensures that overlap with all other local models is checked exhaustively. Given this "nearest-neighbor" data structure, lookup and learning can be confined to only a 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 shown that this method per-

forms as well as an exhaustive lookup (update) strategy that excludes RFs that are activated below a certain threshold w_{cutoff}.

Pruning of Local Models

As in the RFWR algorithm[35], it is possible to prune local models depending upon the level of overlap between two local models and the accumulated locally weighted mean-squared error; the pruning strategy is virtually identical as in [[35], section 3.14]. However, due to the numerical robustness of PLS, we have noticed that the need for pruning or merging is almost nonexistent in the LWPR implementation, such that we do not expand on this possible feature of the algorithm.

Computational Complexity

For a diagonal distance metric $\mathbf{D}$ and under the assumption that the number of projections R remains small and bounded, the computational complexity of one incremental update of all parameters of LWPR is linear in the number of input dimensions N. To the best of our knowledge, this property makes LWPR one of the computationally most efficient algorithms that have been suggested for high-dimensional function approximation. This low computational complexity sets LWPR apart from our earlier work on the RFWR algorithm[35], which was cubic in the number of input dimensions. We thus accomplished one of our main goals, that is, maintaining the appealing function approximation properties of RFWR while eliminating its problems in high-dimensional learning problems.

Confidence Intervals

Under the classical probabilistic interpretation of weighted least squares[12], that is, that each local model's conditional distribution is normal with heteroscedastic variances $p(y|\mathbf{x}; w_k) \sim N({\mathbf{z}_k}^T \beta_k, {s_k}^2/w_k)$, it is possible to derive the predictive variances $\sigma^2_{pred,k}$ for a new query point $\mathbf{x}_q$ for each local model in LWPR.[5] The derivation of this measure is in analogy with ordinary linear regression[33, 23] and is also consistent with the Bayesian formulation of predictive variances [12]. For each individual local model, $\sigma^2_{pred,k}$ can be estimated as (see table 5.2 and algorithm 5.2 for variable definitions):

$$\sigma^2_{pred,k} = s^2_k(1 + w_k \mathbf{z}^T_{q,k} \mathbf{q}_k), \tag{5.13}$$

where $\mathbf{z}_{q,k}$ is the projected query point $\mathbf{x}_q$ under the kth local model, and

$$\begin{aligned} {s_k}^2 &\approx MSE_{k,R}^{n=M}/(M'_k - p'_k); \quad M'_k \equiv \sum_{i=1}^{M} w_{k,i} \approx {W_k}^{n=M}, \\ p'_k &\equiv \sum_{i=1}^{M} w_{k,i}^2 \mathbf{z}_{k,i}^T \mathbf{q}_{k,i} \approx a_{p'_k}^{n=M}, \\ &\quad \text{with incremental update of } \; a_{p'_k}^{n+1} = \lambda a_{p'_k}^{n} + {w_k}^2 {\mathbf{z}_k}^T \mathbf{q}_k. \end{aligned}$$

The definition of M' in terms of the sum of weights reflects the effective number of data points entering the computation of the local variance ${s_k}^2$[33] after an update of M training points has been performed. The definition of p', also referred to the as the *local* degrees of freedom, is analogous to the global degrees of freedom of linear smoothers [16, 33].

In order to obtain a predictive variance measure for the averaging formula (5.9), one could just compute the weighed average of the predictive variance in (5.13). While this approach is viable, it nevertheless ignores important information that can be obtained from variance of the individual predictions $\hat{y}_{q,k}$ and is thus potentially too optimistic. To remedy this issue, we postulate that from the view of combining individual $\hat{y}_{q,k}$, each contributing $y_{q,k}$ was generated from the process

$$y_{q,k} = y_q + \epsilon_1 + \epsilon_{2,k},$$

where we assume two separate noise processes: (i) one whose variance σ^2 is independent of the local model, that is, $\epsilon_1 \sim N(0, \sigma^2/w_k)$ (and accounts for the differences between the predictions of the local models), and (ii) another, which is the noise process $\epsilon_{2,k} \sim N(0, \sigma_{pred,k}^2/w_k)$ of the individual local models. It can be shown that (5.9) is a consistent way of combining prediction from multiple models under the noise model we just described and that the combined predictive variance over all models can be approximated as

$$\sigma_{pred}^2 = \frac{\sum_k w_k \; \sigma^2}{(\sum_k w_k)^2} + \frac{\sum_k w_k \; \sigma_{pred,k}^2}{(\sum_k w_k)^2}. \tag{5.14}$$

The estimate of $\sigma_{pred,k}$ is given in (5.13). The global variance across models can be approximated as $\sigma^2 = \sum_k w_k (\hat{y}_q - \hat{y}_{k,q})^2 / \sum_k w_k$. Inserting these values in (5.14), we obtain

$$\sigma_{pred}^2 = \frac{1}{(\sum_k w_k)^2} \sum_{k=1}^{K} w_k [(\hat{y}_q - \hat{y}_{k,q})^2 + s_k^2 (1 + w_k \mathbf{z}_k^T \mathbf{q}_k)]. \tag{5.15}$$

A one-standard-deviation-based confidence interval would thus be

$$I_c = \hat{y}_q \pm \sigma_{pred}. \tag{5.16}$$

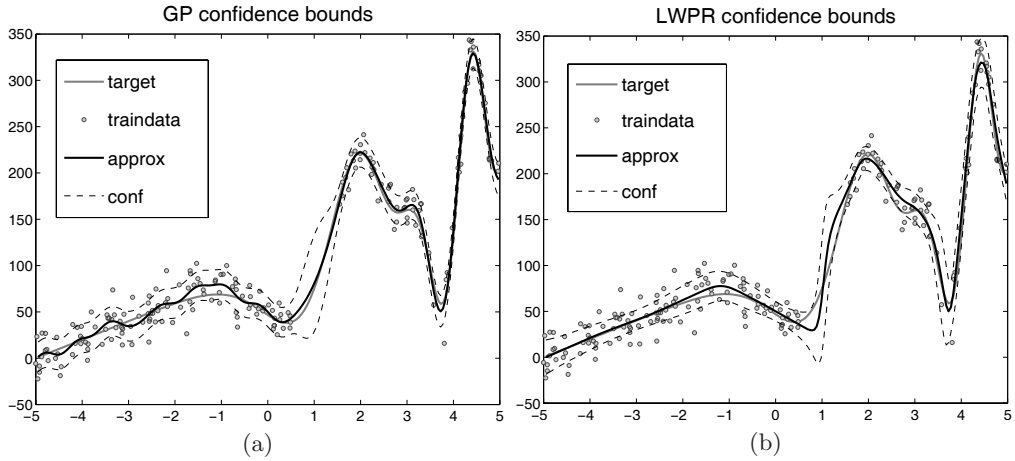


Figure 5.3 Function approximation with 200 noisy data points along with plots of confidence intervals for *(a)* Gaussian Process Regression and *(b)* LWPR algorithms. Note the absence of data in the range [0.5 1.5]

The variance estimate in (5.14) is consistent with the intuitive requirement that when only one local model contributes to the prediction, the variance is entirely attributed to the predictive variance of that single model. Moreover, a query point that does not receive a high weight from any local model will have a large confidence interval due to the small squared sum-of-weight value in the denominator. Figure 5.3 illustrates comparisons of confidence interval plots on a toy problem with 200 noisy data points. Data from the range [0.5 1.5] was excluded from the training set. Both GPR and LWPR show qualitatively similar confidence interval bounds and fitting results.

5.5 Empirical Evaluation

The following sections provide an evaluation of our proposed LWPR learning algorithm over a range of artificial and real-world data sets. Whenever useful and feasible, comparisons to state-of-the-art alternative learning algorithms are provided, in particular SVMR and GPR. SVMR and GPR were chosen due to their generally acknowledged excellent performance in nonlinear regression on finite data sets. However, it should be noted, that both SVMR and GPR are batch learning systems, while LWPR was implemented as a fully incremental algorithm, as described in the previous sections.

5.5.1 Function Approximation with Redundant and Irrelevant Data

We implemented the LWPR algorithm as outlined in section 5.4. In each local model, the projection regressions are performed by (locally weighted) PLS, and the distance metric $\mathbf{D}$ is learned by stochastic incremental cross-validation; all learning methods employed second-order learning techniques,

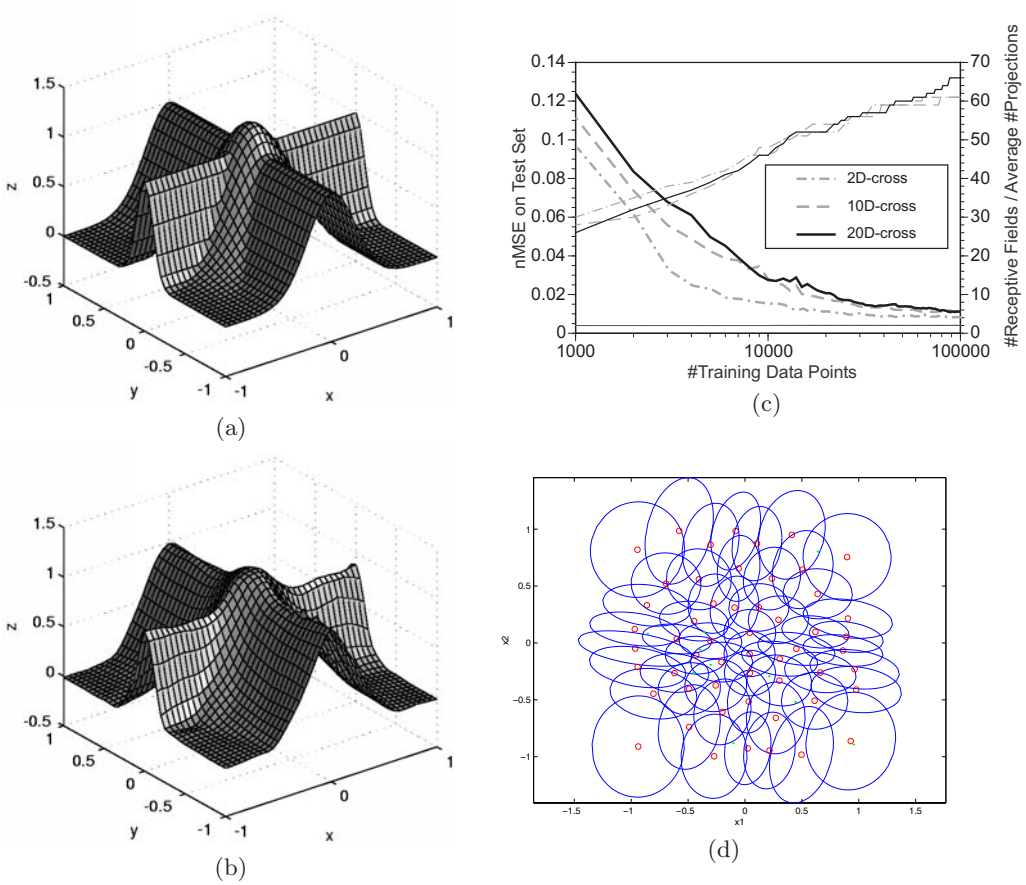


Figure 5.4 *(a)* Target and *(b)* learned nonlinear cross-function.*(c)* Learning curves for 2D, 10D and 20D data. *(d)* The automatically tuned distance metric.

that is, incremental PLS uses recursive least squares, and gradient descent in the distance metric was accelerated as described in [35]. In all our evaluations, an initial (diagonal) distance metric of $\mathbf{D}_{def} = 30\mathbf{I}$ was chosen; the activation threshold for adding local models was $w_{gen} = 0.2$, and the threshold for adding new projections was $\phi = 0.9$ (cf. subsection 5.4.2).
As a first test, we ran LWPR on 500 noisy training data drawn from the two-dimensional function (cross 2D) generated from

$$y = \max\{\exp(-10x_1^2), \exp(-50x_2^2, 1.25\exp(-5(x_1^2 + x_2^2)))\} + \mathrm{N}(0, 0.01),$$

as shown in figure 5.4(a). This function has a mixture of areas of rather high and rather low curvature and is an interesting test of the learning and generalization capabilities of a learning algorithm: learning models with low complexity find it hard to capture the nonlinearities accurately, while more complex models easily overfit, especially in linear regions. A second test added eight constant (i.e., redundant) dimensions to the inputs and rotated this new input space by a random 10D rotation matrix to create a 10D input space with high rank deficiency (cross 10D). A third test added another ten (irrelevant) input dimensions to the inputs of the

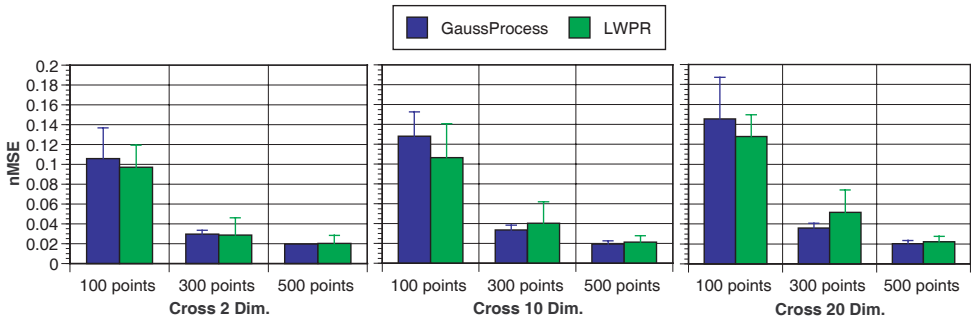


Figure 5.5 Normalized mean squared error comparisons between LWPR and Gaussian Processes for 2D, 10D and 20D Cross data sets

second test, each having $N(0, 0.05^2)$ Gaussian noise, thus obtaining a data set with 20D input space (cross 20D). Typical learning curves with these data sets are illustrated in figure 5.4(c). In all three cases, LWPR reduced the normalized mean squared error (thick lines) on a noiseless test set (1681 points on a $41x41$ grid in the unit-square in input space) rapidly in ten to twenty epochs of training to less than $nMSE = 0.05$, and it converged to the excellent function approximation result of $nMSE = 0.015$ after 100,000 data presentations or 200 epochs.[6] Figure 5.4(b) illustrates the reconstruction of the original function from the 20D test data, visualized in 3D - a highly accurate approximation. The rising lines in figure 5.4(c) show the number of local models that LWPR allocated during learning. The lines at the bottom of the graph indicate the average number of projections that the local models allocated: the average settled at a value of around two local projections, as is appropriate for this originally 2D data set. This set of tests demonstrate that LWPR is able to recover a low-dimensional nonlinear function embedded in high-dimensional space despite irrelevant and redundant dimensions, and that the data efficiency of the algorithm does not degrade in higher-dimensional input spaces. The computational complexity of the algorithm only increased linearly with the number of input dimensions, as explained in section 5.4.

The results of these evaluations can be directly compared with our earlier work on the RFWR algorithm[35], in particular figures 4 and 5 of this earlier paper. The learning speed and the number of allocated local models for LWPR is essentially the same as for RFWR in the 2D test set. Applying RFWR to the 10D and 20D data set of this paper, however, is problematic, as it requires a careful selection of initial ridge regression parameters to stabilize the highly rank-deficient full covariance matrix of the input data, and it is easy to create too much bias or too little numerical stabilization initially, which can trap the local distance metric adaptation in local minim. While the LWPR algorithm just computes about a factor ten times longer for the 20D experiment in comparison to the 2D experiment, REFER requires a 1000-fold increase of computation time, thus rendering this algorithm unsuitable for high-dimensional regression.

In order to compare LWPR's results to other popular regression methods, we evaluated the 2D, 10D, and 20D cross data sets with GPR and SVMR in addition to our LWPR method. It should be noted that neither SVMR nor GPR methods is an incremental method, although they can be considered the state-of-the-art for batch regression under relatively small number of training data and reasonable input dimensionality. The computational complexity of these methods are prohibitively high for realtime applications. The GPR algorithm[14] used a generic covariance function and optimized over the hyperparameters. The SVMR was performed using a standard available package[29] and optimized for kernel choices.

Figure 5.5 compares the performance of LWPR and GPR for the above mentioned data sets using 100, 300, and 500 training data points.[7] As in figure 5.4, the test data set consisted of 1681 data points corresponding to the vertices of a $41x41$ grid over the unit square; the corresponding output values were the exact function values. The approximation error was measured as a normalized weighted mean squared error, $nMSE$, i.e, the weighted MSE on the test set normalized by the variance of the outputs of the test set; the weights were chosen as $1/\sigma^2_{pred,i}$ for each test point $\mathbf{x}_i$. Using such a weighted $nMSE$ was useful to allow the algorithms to incorporate their confidence in the prediction of a query point, which is especially useful for training data sets with few data points where query points often lie far away from any training data and require strong extrapolation to form a prediction. Multiple runs on ten randomly chosen training data sets were performed to accumulate the statistics.

As can be seen from figure 5.5, the performance differences between LWPR and GPR were largely statistically insignificant across training data sizes and input dimensionality. LWPR had a tendency to perform slightly better on the 100-point data sets, most likely due to its quickly decreasing confidence when significant extrapolation is required for a test point. For the 300-point data sets, GPR had a minor advantage and less variance in its predictions, while for 500-point data sets both algorithms achieved equivalent results. While GPRs used all the input dimensions for predicting the output (deduced from the final converged coefficients of the covariance matrix), LWPR stopped at an average of two local projections, reflecting that it exploited the low dimensional distribution of the data. Thus, this comparison illustrates that LWPR is a highly competitive learning algorithm in terms of its generalization capabilities and accuracy of results, despite it being a truly incremental, computationally efficient and real-time implementable algorithm.

5.5.2 Comparisons on Benchmark Regression Data Sets

While LWPR is specifically geared toward real-time incremental learning in high dimensions, it can nevertheless also be employed for traditional batch

Table 5.3 Comparison of $nMSE$ on Boston and Abalone data sets

	Gaussian Process	**Support Vectors**	**LWPR**
Boston	0.0806 ± 0.0195	0.1115 ± 0.09	0.0846 ± 0.0225
Abalone	0.4440 ± 0.0209	0.4830 ± 0.03	0.4056 ± 0.0131

data analysis. Here we compare its performance on two natural real-world benchmark datasets, using again GPR and SVMR as competitors.

The data sets we used were the Boston housing data and the Abalone data set, both available from the UCI Machine Learning Repository[18]. The Boston housing data, which had fourteen attributes, was split randomly (10 random splits) into disjoint sets of 404 training and 102 testing data. The Abalone data set, which had nine attributes, was downsampled to yield ten disjoint sets of 500 training data points and 1177 testing points.[8]

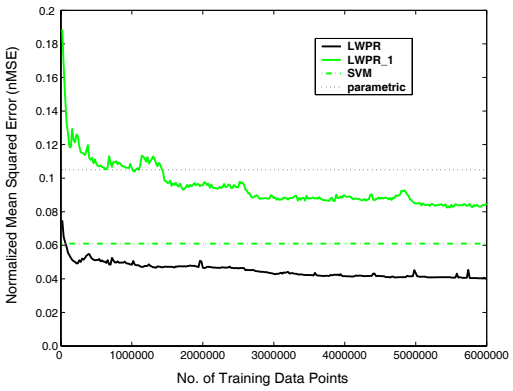
The GPR used hyperparameter estimation for the open parameters of the covariance matrix while for SVMR, the results were obtained by employing a Gaussian kernel of width 3.9 and 10 for the Boston and Abalone data sets, respectively, based on the optimized values suggested in [38]. Table 5.3 shows the comparisons of the $nMSE$ achieved by GPR, SVMR and LWPR on both these data sets. Once again, LWPR was highly competitive on these real-world data sets, consistently outperforming SVMR and achieving very similar $nMSE$ results as GPR.

5.5.3 Sensorimotor Learning in High Dimensional Space

In this section, we look at the application of LWPR to realtime learning in high-dimensional spaces in a data-rich environment - an example of which is learning for robot control. In such domains, LWPR is -to the best of our knowledge - one of the only viable and practical options for principled statistical learning. The goal of learning in this evaluation is to estimate the inverse dynamics model (also referred to as an *internal model*) of the robotic system such that it can be used as a component of a feedforward controller for executing fast accurate movements.

Before demonstrating the applicability of LWPR in realtime, 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 patterns from a 7DOF anthropomorphic robot arm [figure 5.6(a)], sampled at 50 Hz. The learning task was to fit the the inverse dynamics model of the robot, a mapping from seven joint positions, seven joint velocities, and seven joint accelerations to the corresponding seven joint torques (i.e, a 21D to 7D function). Ten percent of these data were excluded from training as a test set. The training data were approximated by four different methods: (i) parameter estimation based on an analytical rigid-body dynamics model[1], (ii) SVMR[29] (using a ten-fold downsampled

(a)

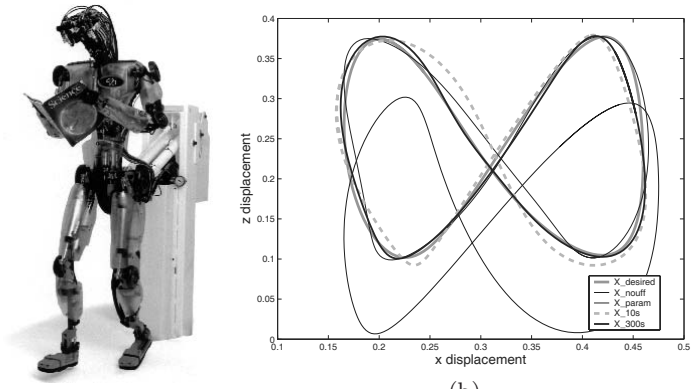


(b)

Figure 5.6 *(a)* Sarcos dextrous arm. *(b)* Comparison of $nMSE$ learning curves for learning the robot's inverse dynamics model for the shoulder DOF.

training set for computational feasibility), (iii) LWPR-1, that is, LWPR that used only one single projection (cf. 5.4.1), and (iv) full LWPR. It should be noted that neither (i) nor (ii) is an incremental method. Using a parametric rigid-body dynamics model as suggested in (i) and just approximating its open parameters from data results in a global model of the inverse dynamics that is theoretically the most powerful method. However, given that our robot is actuated hydraulically and is 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, that is, all models had a univariate output and twenty-one inputs. LWPR employed a diagonal distance metric.

Figure 5.6 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 SVMR. Full LWPR performed the best. The results for all other DOFs were analogous and are not shown here. 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. As in the previous sections, these results demonstrate that LWPR is a competitive function approximation technique that can be applied successfully in real world applications.



(a) (b)

Figure 5.7 *(a)* The 30-DOF Sarcos humanoid robot. *(b)* Results of online learning of the inverse dynamics with LWPR on the humanoid robot.

Online Learning for Humanoid Robots

We implemented LWPR on the realtime operating system (vxWorks) for two of our robotic setups, the 7DOF Sarcos dextrous arm mentioned above in figure 5.6(a), and the Sarcos humanoid robot in figure 5.7(a), a 30DOF system. Out of the four parallel processors of the system, one 366 MHz PowerPC processor was completely devoted to lookup and learning with LWPR.

For the dexterous arm, each DOF had its own LWPR learning system, resulting in seven parallel learning modules. In order to accelerate lookup and training times, the nearest-neighbor data lookup described on page 118 was utilized. The LWPR models were trained online while the robot performed a randomly drifting figure-eight pattern in front of its body. Lookup proceeded at 480 Hz, while updating the learning model was achieved at about 70 Hz. At 10-second intervals, learning was stopped and the robot attempted to draw a planar figure eight in the x-z plane of the robot end effector at 2 Hz frequency for the entire pattern. The quality of these drawing patterns is illustrated in figure 5.8. In figure 5.8(a), X_{des} denotes the desired figure eight pattern, X_{sim} illustrates the figure eight 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 slightly better than the simulator. Figure 5.8(b) illustrates the speed of LWPR learning. The X_{nouff} trace demonstrates the figure eight patterns performed without any inverse dynamics model, just using a low-gain PD controller. The other traces show how rapidly LWPR learned the figure eight pattern during training: they

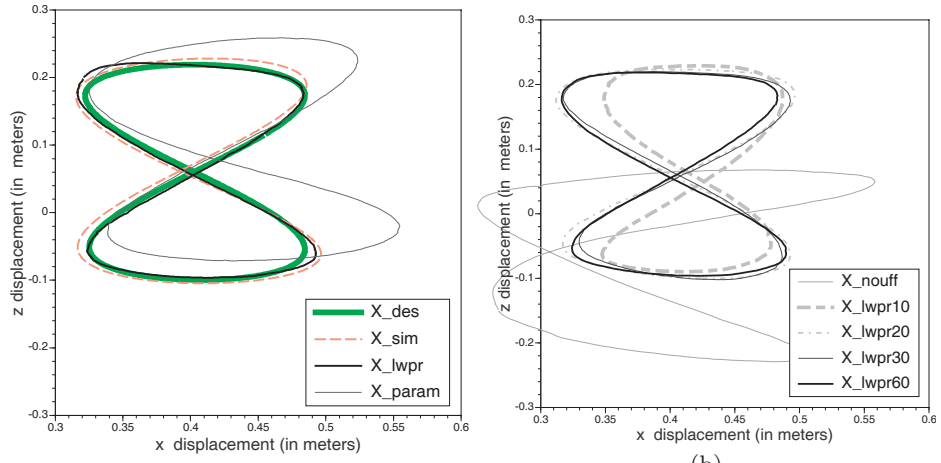


Figure 5.8 *(a)* Trajectories of robot end effector: X_{des} is the desired trajectory, X_{sim} is a trajectory obtained from a robot simulator that had a perfect controller but numerical inaccuracy due to the integration of the equations of motion, X_{lwpr} is the result of a computed torque controller using LWPR's approximated inverse model, and X_{param} is the trajectory using an inverse model due to rigid body parameter estimation. *(b)* Results of online learning with LWPR starting from scratch, that is, initially with no functional inverse model; the improvement of control due to LWPR learning is shown at intervals of 10 seconds over the first minute of learning.

denote performance after 10, 20, 30, and 60 *seconds* of training. After 60 seconds, the figure eight is hardly distinguishable from the desired trace.

In order to demonstrate the complexity of functions that can be learned in realtime with LWPR, we repeated the same training and evaluation procedure with the Sarcos humanoid robot, which used its right hand to draw a lying figure eight pattern. In this case, learning of the inverse dynamics model required learning in a 90D input space, and the outputs were the thirty torque commands for each of the DOFs. As the learning of thirty parallel LWPR models would have exceeded the computational power of our 366 MHz real-time processors, we chose to learn one single LWPR model with a 30D output vector, that is, each projection of PLS in LWPR regressed all outputs vs. the projected input data. The projection direction was chosen as the mean projection across all outputs at each projection stage of PLS. This approach is suboptimal, as it is quite unlikely that all output dimensions agree on one good projection direction; essentially, one assumes that the gradients of all outputs point roughly in the same direction. On the other hand, section 5.2 demonstrated that movement data of actual physical systems lie on locally low-dimensional distributions, such that one can hope that LWPR with multiple outputs can still work successfully by simply spanning this locally low-dimensional input space with all projections. Figure 5.7(b) demonstrates the result of learning in a similar way as figure 5.8(b); the notation for the different trajectories in this figure follow as explained above for the 7DOF robot. Again, LWPR very rapidly improves

over a control system with no inverse dynamics controller, that is, within 10 seconds of movement, the most significant inertial perturbation have been compensated. Convergence to low error tracking of the figure eight takes slightly longer, that is, about 300 seconds [X_{300} in figure 5.7(b)], but is reliably achieved. About fifty local models were created for this task. The learned inverse dynamics outperformed a model estimated by rigid-body dynamics methods significantly [cf. X_{param} in figure 5.7(b)].

Online Learning for Autonomous Airplane Control

The online learning abilities of LWPR are ideally suited to be incorporated in algorithms of provably stable adaptive control. The control theoretic development of such an approach was presented in Nakanishi et al.[26]. In essence, the problem formulation begins with a specific class of equations of motion of the form

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\,\mathbf{u}, \tag{5.17}$$

where $\mathbf{x}$ denotes the state of the control system, the control inputs, and $f(\mathbf{x})$ and $g(\mathbf{x})$ are nonlinear function to approximated. A suitable control law for such a system is

$$\mathbf{u} = \hat{g}(\mathbf{x})^{-1}\left(-\hat{f}(\mathbf{x}) + \dot{\mathbf{x}}_c + \mathbf{K}(\mathbf{x}_c - \mathbf{x})\right), \tag{5.18}$$

where $\mathbf{x}_c, \dot{\mathbf{x}}_c$ are a desired reference trajectory to be tracked, and the "hat" notation indicates that these are the approximated version of the unknown function.

We applied LWPR in this control framework to learn the unknown function f and g for the problem of autonomous airplane control on a high-fidelity simulator. For simplicity, we only considered a planar version of the airplane, governed by the differential equation[41]:

$$\begin{aligned} \dot{V} &= \tfrac{1}{m}(T\cos\alpha - D) - g\sin\gamma, \\ \dot{\alpha} &= -\tfrac{1}{mV}(L + T\sin\alpha) + \tfrac{g\cos\gamma}{V} + Q, \\ \dot{Q} &= cM. \end{aligned} \tag{5.19}$$

In these equations, V denotes the forward speed of the airplane, m the mass, T the thrust, α the angle of attack, g the gravity constant, γ the flight path angle with respect to the horizontal world coordinate system axis, Q the pitch rate, and c an inertial constant. The complexity of these equations is hidden in D,L, and M, which are the unknown highly nonlinear aerodynamic lift force, drag force, and pitch moment terms, which are specific to every airplane.

While we will not go into the details of provably stable adaptive control with LWPR in this chapter and how the control law (5.18) is applied to for airplane control, from the viewpoint of learning the main components

to learn are the lift and drag forces and the pitch moment. These can be obtained by rearranging (5.19) to

$$\begin{aligned}
&D = T\cos\alpha - \left(\dot{V} + g\sin\gamma\right) m = \\
&f_D\left(\alpha, Q, V, M, \gamma, \delta_{OFL}, \delta_{OFR}, \delta_{MFL}, \delta_{MFR}, \delta_{SPL}, \delta_{SPR}\right) \\
&L = \left(\tfrac{g\cos\gamma}{V} + Q - \dot{\alpha}\right) mV - T\sin\alpha = \\
&f_L\left(\alpha, Q, V, M, \gamma, \delta_{OFL}, \delta_{OFR}, \delta_{MFL}, \delta_{MFR}, \delta_{SPL}, \delta_{SPR}\right) \\
&M = \tfrac{Q}{c} = f_M\left(\alpha, Q, V, M, \gamma, \delta_{OFL}, \delta_{OFR}, \delta_{MFL}, \delta_{MFR}, \delta_{SPL}, \delta_{SPR}\right).
\end{aligned} \tag{5.20}$$

The δ terms denote the control surface angles of the airplane, with indices midboard-flap-left/right (MFL,MFR), outboard-flap-left/right(OFL,OFR), and left and right spoilers (SPL,SPR). All terms on the right hand side of (5.20) are known, such that we have to cope with three simultaneous function approximation problems in an 11D input space, an ideal application for LWPR.

We implemented LWPR for the three functions above in a high-fidelity simulink simulation of an autonomous airplane using the adaptive control approach of[26]. The airplane started with no initial knowledge, just the proportional controller term in (5.18) (i.e., the term multiplied by $\mathbf{K}$). The task of the controller was to fly doublets, that is, up-and-down trajectories, which are essentially sinusoid like variations of the flight path angle γ

Figure 5.9 demonstrates the results of this experiment. Figure 5.9(a) shows the desired trajectory in γ and its realization by the controller. Figure 5.9(b-d) illustrate the online function approximation of D, L, and M. As can be seen, the control of γ achieves almost perfect tracking after just a very few seconds. The function approximation of D and L is very accurate after a very short time. The approximation M requires a longer time for convergence, but progresses fast. About ten local models were needed for learning f_D and f_L, while about twenty local models were allocated for f_M.

An interesting element of figure 5.9 happens after 400 seconds of flight, where we simulated a failure of the airplane mechanics by locking the MFR to 17-degree deflection. As can be seen, the function approximators very quickly reorganize after this change, and the flight is successfully continued, although γ tracking has some error for a while until it converges back to good tracking performance. The strong signal changes in the first seconds after the failure are due to oscillations of the control surfaces, and not a problem in function approximation. Without adaptive control, the airplane would have crashed.

5.6 Bayesian Backfitting

The PLS algorithm described in subsection 5.3.3 has an attractive feature: rather than reduce the dimensionality of the input data to the most rel-

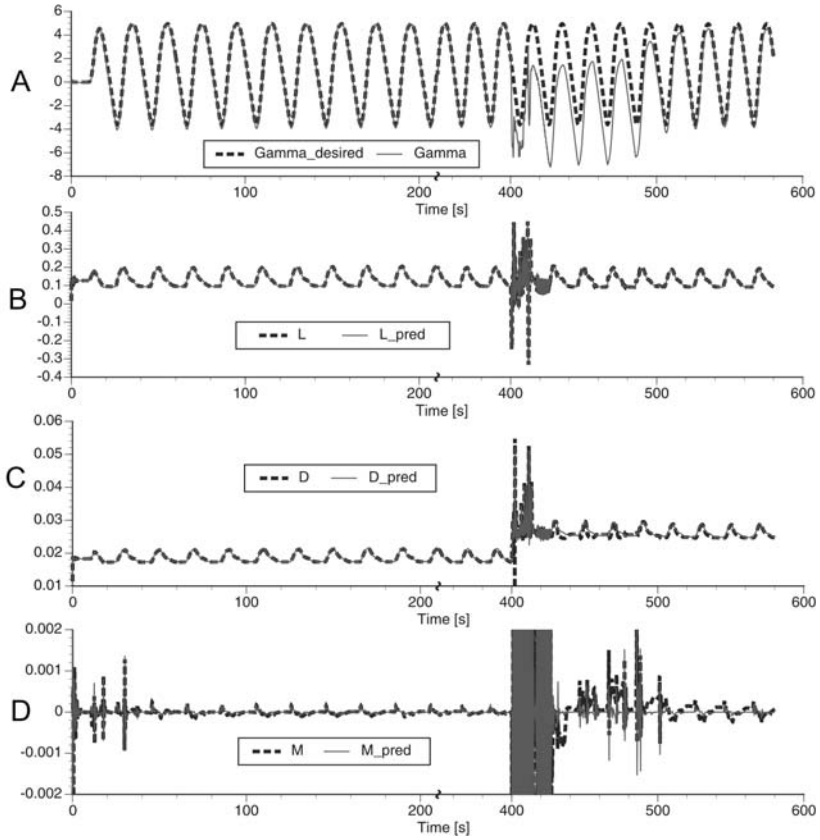


Figure 5.9 LWPR learning results for adaptive learning control on a simulated autonomous airplane *(a)* Tracking of flight angle γ. *(b)* Approximation of lift force. *(c)* Approximation of drag force, *(d)* Approximation of pitch moment. At 400 seconds into the flight, a failure is simulated that locks one control surface to a 17-degree angle. Note that for reasons of clearer illustration, an axis break was inserted after 200 seconds.

evant subspace, it deals with the complete dimensionality, and structures its computation efficiently such that successive computationally inexpensive univariate regressions suffice rather than expensive matrix inversion techniques. However, PLS also has two heuristic components, that is, the way the projection directions are selected by an input-output correlation analysis, and the decision on when to stop adding new projection directions. In this section we suggest a Bayesian algorithm to replace PLS.

1: Init: $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_N]^T, \mathbf{y} = \left[y_1 \dots y_N\right]^T, g_{m,i} = g_m(\mathbf{x}_i; \theta_m), \mathbf{g}_m = [g_{m,1} \dots g_{m,N}]^T$
2: **repeat**
3: **for** $m = 1$ to d **do**
4: $\mathbf{r}_m \leftarrow \mathbf{y} - \sum_{k \neq m} \mathbf{g}_k$ {compute partial residual (fake target)}
5: $\theta_m \leftarrow \arg\min_{\theta_m} (\mathbf{g}_m - \mathbf{r}_m)^2$ {optimize to fit partial residual}
6: **end for**
7: **until** convergence of $\boldsymbol{\theta}_m$

Algorithm 5.4: Algorithm for backfitting.

Another algorithm similar in vein to PLS is *backfitting* [16]. The backfitting algorithm estimates additive models of the form

$$y(\mathbf{x}) = \sum_{m=1}^{d} g_m(\mathbf{x}; \theta_m),$$

where the functions g_m are adjustable basis functions (e.g., splines), parameterized by θ_m. As shown in algorithm 5.4, backfitting decomposes the statistical estimation problem into d individual estimation problems by using partial residuals as "fake supervised targets" for each function g_m. At the cost of an iterative procedure, this strategy effectively reduces the computational complexity of multiple input settings, and allows easier numerical robustness control since no matrix inversion is involved.

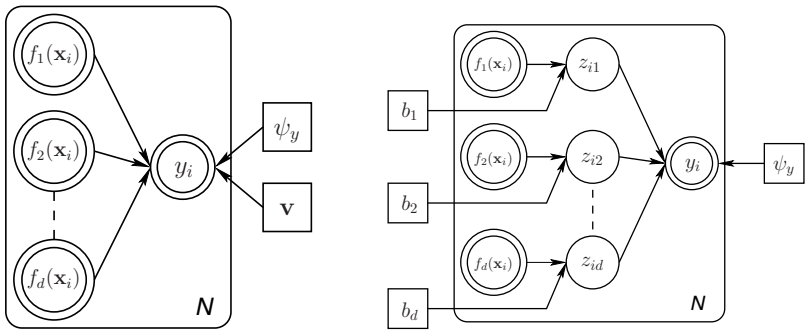
For all its computational attractiveness, backfitting presents two serious drawbacks. There is no guarantee that the iterative procedure outlined in algorithm 5.4 will converge as this is heavily dependent on the nature of the functions g_m. The updates have no probabilistic interpretation, making backfitting difficult to insert into the current framework of statistical learning which emphasizes confidence measures, model selection, and predictive distributions. It should be mentioned that a Bayesian version of backfitting has been proposed in [17]. This algorithm however, relies on Gibbs sampling, which is more applicable when dealing with the nonparametric spline models discussed there, and is quite useful when one wishes to generate samples from the posterior additive model.

5.6.1 A Probabilistic Derivation of Backfitting

Consider the graphical model shown in figure 5.10(a), which represents the statistical model for *generalized* linear regression (GLR)[16]:

$$y|\mathbf{x} \sim \text{Normal}\left(y; \sum_{m=1}^{d} b_m f_m\left(\mathbf{x}; \boldsymbol{\theta}_m\right), \psi_y\right)$$

Given a data set $\mathbf{x}_{\mathcal{D}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, we wish to determine the most likely regression vector $\mathbf{v} = \begin{bmatrix} b_1 & b_2 & \cdots & b_d \end{bmatrix}^T$ which linearly combines the basis functions f_m to generate the output y. Since computing the ordinary least squares (OLS) solution ($\mathbf{v} = \left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{y}$) is an $O(d^3)$ task that grows computationally expensive and numerically brittle as the dimensionality of the input increases, we introduce a simple modification of the graphical model of figure 5.10(a), which enables us to create the desired algorithmic decoupling of the predictor functions, and gives backfitting a probabilistic interpretation. Consider the introduction of random variables z_{im} as shown in figure 5.10(b). These variables are analogous to the output of the g_m function of algorithm 5.4, and can also be interpreted as an unknown



(a) Graphical model for generalized linear regression.

(b) Graphical model for probabilistic backfitting.

Figure 5.10 We modify the original graphical model for generalized linear regression by inserting hidden variables z_{im} in each branch of the fan-in. This modified model can be solved using the EM framework to derive a probabilistic version of backfitting.

fake target for each branch of the regression fan-in. For the derivation of our algorithm, we assume the following conditional distributions for each variable in the model:

$$\begin{aligned} y_i|\mathbf{z}_i &\sim \text{Normal}\left(y_i; \mathbf{1}^T\mathbf{z}_i, \psi_y\right) \\ z_{im}|\mathbf{x}_i &\sim \text{Normal}\left(z_{im}; b_m f_m(\mathbf{x}_i), \psi_{zm}\right) \end{aligned} \tag{5.21}$$

where $\mathbf{1} = [1, 1, \ldots, 1]^T$. With this modification in place, we are essentially in a situation where we wish to optimize the parameters

$$\boldsymbol{\phi} = \left\{ \{b_m, \psi_{zm}\}_{m=1}^d, \psi_y \right\},$$

given that we have observed variables $\mathbf{x}_\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ and that we have unobserved variables $\mathbf{x}_\mathcal{H} = \{\mathbf{z}_i\}_{i=1}^N$ in our graphical model. This situation fits very naturally into the framework of maximum-likelihood estimation via the EM algorithm, by maximizing the expected *complete* log likelihood $\langle \ln p(\mathbf{x}_\mathcal{D}, \mathbf{x}_\mathcal{H}; \boldsymbol{\phi}) \rangle$ which, from figure 5.10(b), can be expressed as

$$\begin{aligned} \ln p(\mathbf{x}_\mathcal{D}, \mathbf{x}_\mathcal{H}; \boldsymbol{\phi}) = &-\frac{N}{2}\ln\psi_y - \frac{1}{2\psi_y}\sum_{i=1}^N \left(y_i - \mathbf{1}^T\mathbf{z}_i\right)^2 \\ &- \sum_{m=1}^d \left[\frac{N}{2}\ln\psi_{zm} + \frac{1}{2\psi_{zm}}\sum_{i=1}^N \left(z_{im} - b_m f_m(\mathbf{x}_i; \boldsymbol{\theta}_m)\right)^2 \right] \\ &+ const. \end{aligned} \tag{5.22}$$

The resulting EM update equations are summarized below:

$$
\begin{aligned}
&\textbf{M-Step}: \\
b_m &= \frac{\sum_{i=1}^{N} \langle z_{im} \rangle \, f_m(\mathbf{x}_i)}{\sum_{i=1}^{N} f_m(\mathbf{x}_i)^2} \\
\psi_y &= \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \mathbf{1}^T \langle \mathbf{z}_i \rangle \right)^2 + \mathbf{1}^T \mathbf{\Sigma_z} \mathbf{1} \\
\psi_{zm} &= \frac{1}{N} \sum_{i=1}^{N} \left(\langle z_{im} \rangle - b_m f_m(\mathbf{x}_i) \right)^2 + \sigma_{zm}^2 \\
&\textbf{E-Step}: \\
\mathbf{1}^T \mathbf{\Sigma_z} \mathbf{1} &= \left(\sum_{m=1}^{d} \psi_{zm} \right) \left[1 - \frac{1}{s} \left(\sum_{m=1}^{d} \psi_{zm} \right) \right] \\
\sigma_{zm}^2 &= \psi_{zm} \left(1 - \frac{1}{s} \psi_{zm} \right) \\
\langle z_{im} \rangle &= b_m f_m(\mathbf{x}_i) + \frac{1}{s} \psi_{zm} \left(y_i - \mathbf{v}^T \mathbf{f}(\mathbf{x}_i) \right)
\end{aligned}
$$

where we define $s \equiv \psi_y + \sum_{m=1}^{d} \psi_{zm}$. In addition, the parameters $\boldsymbol{\theta}_m$ of each function f_m can be updated by setting:

$$
\sum_{i=1}^{N} \Big(\langle z_{im} \rangle - b_m f_m \left(\mathbf{x}_i; \boldsymbol{\theta}_m \right) \Big) \frac{\partial f_m \left(\mathbf{x}_i; \boldsymbol{\theta}_m \right)}{\partial \boldsymbol{\theta}_m} = 0 \tag{5.23}
$$

and solving for $\boldsymbol{\theta}_m$. As this step depends on the particular choice of f_m, e.g., splines, kernel smoothers, parametric models, etc., we will not pursue it any further and just note that *any* statistical approximation mechanism could be used. Importantly, all equations in both the expectation and maximization steps are algorithmically $O(d)$ where d is the number of predictor functions f_m, and no matrix inversion is required.

To understand our EM solution as probabilistic backfitting, we note that backfitting can be viewed as a formal Gauss-Seidel algorithm; an equivalence that becomes exact in the special case of linear models[16]. For the linear system $\mathbf{F}^T \mathbf{F} \mathbf{v} = \mathbf{F}^T \mathbf{y}$, the Gauss-Seidel updates for the individual b_m are

$$
b_m = \frac{\sum_{i=1}^{N} \Big(y_i - \sum_{k \neq m}^{d} b_k f_k(\mathbf{x}_i) \Big) f_m(\mathbf{x}_i)}{\sum_{i=1}^{N} f_m(\mathbf{x}_i)^2}. \tag{5.24}
$$

Note that (5.24) - if used naively - only guarantees convergence for very specially structured matrices. An extension to the Gauss-Seidel algorithm adds a fraction $(1 - \omega)$ of b_m to the update and gives us the well-known *relaxation* algorithms:

Table 5.4 Results on the neuron-muscle data set

	Bayesian backfitting	**PLS**	**Baseline**
neuron match	93.6%	18%	—
$nMSE$	0.8446	1.77	0.84

$$b_m^{(n+1)} = (1-\omega)b_m^{(n)} + \omega \frac{\sum_{i=1}^{N}\left(y_i - \sum_{k\neq m}^{d} b_k f_k(\mathbf{x}_i)\right) f_m(\mathbf{x}_i)}{\sum_{i=1}^{N} f_m(\mathbf{x}_i)^2}, \tag{5.25}$$

which has improved convergence rates for *overrelaxation* ($1 < \omega < 2$), or improved stability for *underrelaxation* ($0 < \omega < 1$). For $\omega = 1$, the standard Gauss-Seidel/backfitting of equation (5.24) is recovered. The appropriate value of ω, which allows the iterations to converge while still maintaining a reasonable convergence rate can only be determined by treating (5.24) as a discrete dynamical system, and analyzing the eigenvalues of its system matrix - an $O(d^3)$ task. If, however, we substitute the expression for $\langle z_{im} \rangle$ in the maximization equation for b_m, and set $\omega = \omega_m = \psi_{zm}/s$ in (5.25), it can be shown that (after some algebraic rearrangement,) the two equations are identical, that is, we indeed derive a probabilistic version of backfitting.

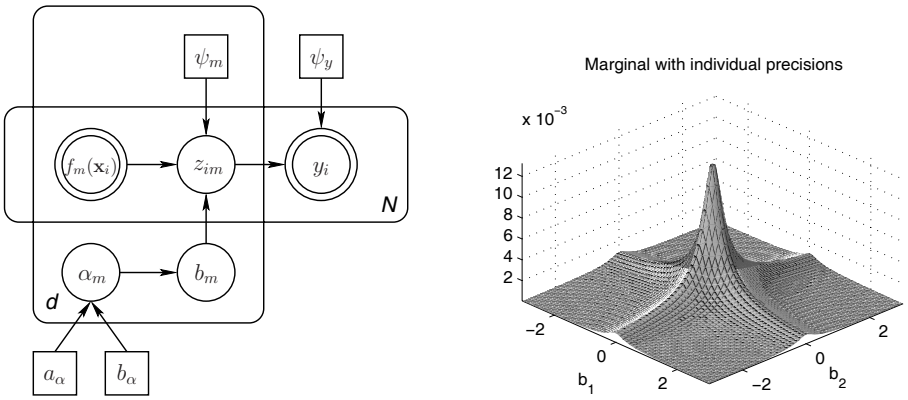
This allows us to now place backfitting within the wider context of Bayesian machine learning algorithms. In particular, we can place individual priors over the regression coefficients:

$$b_m \sim \text{Normal}\left(b_m; 0, 1/\alpha_m\right),$$
$$\alpha_m \sim \text{Gamma}\left(\alpha_m; a_\alpha, b_\alpha\right),$$

where a_α and b_α are small enough to select an uninformative Gamma prior over the precisions α_m. Figure 5.11(a) shows the graphical model with the added priors, while figure 5.11(b) shows the resulting marginal prior over $\mathbf{v}$. This prior structure favors solutions which have as few nonzero regression coefficients as possible, and thus performs an automatic relevance determination (ARD) sparsification of the input dimensions.

We compared the use of PLS and ARD Bayesian backfitting to analyze the following real-world data set collected from neuroscience. The data set consists of simultaneous recordings (2400 data points) of firing-rate coded activity in seventy-one motor cortical neurons and the electromyograms(EMGs) of eleven muscles. The goal is to determine which neurons are responsible for the activity of each muscle. The relationship between neural and muscle activity is assumed to be linear, such that the basis functions in backfitting are simply a copy of the respective input dimensions, that is $f_m(\mathbf{x}) = x_m$.

A brute-force study (conducted by our research collaborators) painstakingly considered every possible combination of neurons (up to groups of twenty for computational reasons; i.e., even this reduced analysis required



(a) Graphical model with ARD prior

(b) Resulting marginal prior over $\mathbf{v}$

Figure 5.11 By associating an individual gamma distributed precision with each regression coefficient, we create a marginal prior over $\mathbf{v}$ that favors sparse solutions which lie along the (hyper)-spines of the distribution.

several weeks of computation on a thirty-node cluster computer), to determine the optimal neuron-muscle correlation as measured on various validation sets. This study provided us with a baseline neuron-muscle correlation matrix that we hoped to duplicate with PLS and Bayesian backfitting, although with much reduced computational effort.

The results shown in table 5.4 demonstrate two points:

– The relevant neurons found by Bayesian backfitting contained over 93% of the neurons found by the baseline study, while PLS fails to find comparable correlations. The neuron match in backfitting is easily inferred from the resulting magnitude of the precision parameters α, while for PLS, the neuron match was inferred based on the subspace spanned by the projections that PLS employed.

– The regression accuracy of Bayesian backfitting (as determined by eight-fold crossvalidation), is comparable to that of the baseline study, while PLS's failure to find the correct correlations causes it to have significantly higher generalization errors. The analysis for both backfitting and PLS was carried out using the same validation sets as those used for the baseline analysis.

The performance of Bayesian backfitting on this particularly difficult data set shows that it is a viable alternative to traditional generalized linear regression tools. Even with the additional Bayesian inference for ARD, it maintains its algorithmic efficiency since no matrix inversion is required.

As an aside it is useful to note that Bayesian backfitting and PLS required of the order of 8 hours of computation on a standard PC (compared with

several weeks on a cluster for the brute-force study), and evaluated the contributions of all seventy-one neurons.

An alternative form of prior in which a *single* precision parameter is shared among the regression coefficients results in a shrinkage of the norm of the regression vector solution, similar to ridge regression. In this case, however, no additional crossvalidation is required to determine the ridge parameters, as these are automatically inferred. The Bayesian backfitting algorithm is also applicable within the framework of sparse Bayesian learning [7], and provides a competitive and robust nonlinear supervised learning tool.

Bayesian Backfitting can thus completely replace PLS in LWPR, thus reducing the number of open parameters in LWPR and facilitating its probabilistic interpretation.

5.7 Discussion

Nearest-neighbor regression with spatially localized models remains one of the most data efficient and computationally efficient methods for incremental learning with automatic determination of the model complexity. In order to overcome the curse of dimensionality of local learning systems, we investigated methods of linear projection regression and how to employ them in spatially localized nonlinear function approximation for high-dimensional input data that have redundant and irrelevant components. We compared various local dimensionality reduction techniques - an analysis that resulted in choosing a localized version of Partial Least Squares regression at the core of a novel nonparametric function approximator, Locally Weighted Projection Regression (LWPR). The proposed technique was evaluated on a range of artificial and real-world data sets in up to 90D input spaces. Besides showing fast and robust learning performance due to second-order learning methods based on stochastic leave-one-out cross-validation, LWPR excelled by its low computational complexity: updating each local model with a new data point remained *linear* in its computational cost in the number of inputs since the algorithm accomplishes good approximation results with only three to four projections irrespective of the number of input dimensions. To our knowledge, this is the first spatially localized incremental learning system that can efficiently work in high-dimensional spaces and that is thus suited for online and realtime applications. In addition, LWPR compared favorably in its generalization performance with state-of-the-art batch regression methods like Gaussian process regression, and can provide qualitatively similar estimates of confidence bounds and predictive variances. Finally, a new algorithm, Bayesian backfitting, was suggested to replace partial least squares in the future. Bayesian backfitting is a Bayesian treatment of linear regression with automatic relevance detection of inputs and a robust EM-like incremental updating technique. Future work will investigate this

algorithm in the nonlinear setting of LWPR on the way to a full Bayesian approach to approximate nearest-neighbor regression.

References

1. C.H. An, C. Atkeson, and J. Hollerbach. *Model Based Control of a Robot Manipulator.* Cambridge, MA, MIT Press, 1988.
2. C. Atkeson, A. Moore, and S.Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(4):76–113, 1997.
3. A. Bell and T. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338, 1997.
4. D.A. Belsley, E. Kuh and D. Welsch. *Regression Diagnostics.* New York, John Wiley & Sons, 1980.
5. C. Bishop *Neural Networks for Pattern Recognition.* Oxford, Oxford University Press, 1995.
6. A. D'Souza, S. Vijayakumar and S. Schaal. Are internal models of the entire body learnable? *Society for Neuroscience Abstracts.* Volume 27, Program No. 406.2, 2001.
7. A. D'Souza, S. Vijayakumar and S. Schaal. The Bayesian backfitting relevance vector machine. In *Proceedings of the Twenty-first International Conference on Machine Learning.* New York, ACM Press, 2004.
8. B.S. Everitt. *An Introduction to Latent Variable Models.* London, Chapman & Hall, 1984.
9. S.E. Fahlman and C. Lebiere. The cascade correlation learning architecture. *Advances in NIPS 2*, 1990.
10. I.E. Frank and J.H. Friedman. A statistical view of some chemometric regression tools. *Technometrics*, 35(2):109–135, 1993.
11. J.H. Friedman and W. Stutzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76:817-823, 1981.
12. A.B. Gelman, J.S. Carlin, H.S. Stern and D.B. Rubin. *Bayesian Data Analysis.* London, Chapman & Hall, 1995.
13. Z. Ghahramani and M.J. Beal. Variational inference for Bayesian mixtures of factor analysers. In editors, S. A. Solla, T. K. Leen and K. Muller, *Advances in Neural Information Processing Systems 12*, pages 449-455, Cambridge, MA, MIT Press, 2000.
14. M. Gibbs and D.J.C., MacKay. Efficient implementation of Gaussian processes. Technical Report, Cavendish Laboratory, Cambridge, UK, 1997.
15. T.J. Hastie and C. Loader. Local regression: Automatic kernel carpentry. *Statistical Science*, 8(2):120–143, 1993.
16. T.J. Hastie and R.J. Tibshirani. *Generalized Additive Models.* No. 43 in *Monographs on Statistics and Applied Probability.* London, Chapman & Hall, 1990.
17. T.J. Hastie and R.J. Tibshirani. Bayesian backfitting. *Statistical Science*, 15(3):196–213, August 2000.

18. S. Hettich and S. D. Bay. The UCI KDD archive. Irvine, CA, University of California, Dept. of Information and Computer Science, [http://kdd.ics.uci.edu], 1999.

19. M.I. Jordan and R. Jacobs. Hierarchical mixture of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994.

20. M. Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9:718–727, 1999.

21. L. Ljung and T. Soderstrom. *Theory and Practice of Recursive Identification*. Cambridge, MA, MIT Press, 1986.

22. W.F. Massy. Principal component regression in exploratory statistical research. *Journal of the American Statistical Association*, 60:234–246, 1965.

23. R.H. Myers. *Classical and Modern Regression with Applications* Boston: Duxbury Press, 1990.

24. B.A. Olshausen and D.J. Field. Emergence of simple cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607-609, 1996.

25. M.P. Perrone and L.N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In editor, R. J. Mammone, *Neural Networks for Speech and Image Processing*. London, Chapman & Hall, 1993.

26. J. Nakanishi, J.A. Farrell and S. Schaal, Learning composite adaptive control for a class of nonlinear systems. In *IEEE International Conference on Robotics and Automation*, pages 2647-2652, New Orleans, 2004.

27. S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323-2326, 2000.

28. D.B. Rubin and D.T. Thayer. EM algorithms for ML factor analysis. *Psychometrika*, 47(1):69–76, 1982.

29. C. Saunders, M.O. Stitson, J. Weston, L. Bottou, B. Schoelkopf and A. Smola. *Support Vector Machine - Reference Manual*. TR CSD-TR-98-03, Dept. of Computer Science, Royal Holloway, University of London, 1998.

30. T.D. Sanger. Optimal unsupervised learning in a single layer feedforward neural network. *Neural Networks*, 2:459–473, 1989.

31. D.W. Scott. *Multivariate Density Estimation*, Hoboken, NJ, John Wiley & Sons, 1992.

32. S. Schaal and D. Sternad. Origins and violations of the 2/3 power law in rhythmic 3D movements. *Experimental Brain Research*. 136:60-72, 2001.

33. S. Schaal, S. Vijayakumar and C.G. Atkeson. Assessing the quality of learned local models. *Advances in Neural Information Processing Systems 6*, pages 160-167. San Mateo, CA, Morgan Kaufmann, 1994.

34. S.Schaal & C.G.Atkeson Receptive field weighted regression, *Technical Report TR-H-209, ATR Human Information Processing Labs.*, Kyoto, Japan.

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

36. S. Schaal, S. Vijayakumar and C.G. Atkeson. Local dimensionality reduction. *Advances in NIPS, 10*, 1998.

37. S. Schaal, C.G. Atkeson and S. Vijayakumar. Realtime robot learning with locally weighted statistical learning. In *Proceedings of International Conference on Robotics and Automation ICRA2000*, San Francisco, CA, pages 288–293, 2000.

38. B. Scholkopf, A. Smola, R. Williamson, R and P. Bartlett. New Support Vector Algorithms. *Neural Computation*, (12)5:1207-1245, 2000.

39. B. Scholkopf, C. Burges and A. Smola. *Advances in Kernel Methods: Support Vector Learning*. Cambridge, MA, MIT Press, 1999.

40. A. Smola and B. Scholkopf. A tutorial on support vector regression. NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, 1998.

41. B.L. Stevens and F.L. Lewis. *Aircraft Control and Simulation*. Hoboken, NJ, John Wiley & Sons, 2003.

42. J. Tenenbaum, V. de Silva and J. Langford. A global geometric framework for nonlinear dimensionality reduction *Science*, 290:2319-2323, 2000.

43. , N. Ueda, R. Nakano, Z. Ghahramani and G. Hinton, SMEM Algorithm for Mixture Models, *Neural Computation*, 12, pp. 2109-2128, 2000.

44. S. Vijayakumar and H. Ogawa. RKHS based Functional Analysis for Exact Incremental Learning *Neurocomputing : Special Issue on Theoretical Analysis of Real Valued Function Classes*, 29(1-3):85-113, 1999.

45. S. Vijayakumar and S. Schaal. Local adaptive subspace regression. *Neural Processing Letters*, 7(3):139–149, 1998.

46. S. Vijayakumar and S. Schaal. Locally Weighted Projection Regression : An *O(n)* algorithm for incremental real time learning in high dimensional space. In *Proceedings of International Conference in Machine Learning (ICML)*, pages 1079-1086, 2000.

47. N. Vlassis, Y. Motomura and B. Krose. Supervised dimension reduction of intrinsically low-dimensional data. *Neural Computation*, Cambridge, MA, MIT Press, 2002.

48. C.K.I. Williams and C. Rasmussen. Gaussian processes for regression. In D.S. Touretsky, M. Mozer and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, Cambridge, MA, MIT Press, 1996.

49. H. Wold. Soft modeling by latent variables: The nonlinear iterative partial least squares approach. In J. Gani (Ed.), *Perspectives in probability and statistics, papers in honour of M. S. Bartlett*, pages 520–540. London, Academic Press, 1975.

50. L. Xu, M.I. Jordan and G.E. Hinton. An alternative model for mixtures of experts. In G. Tesauro, D. Touretzky, and T. Leen,

editors, *Advances in Neural Information Processing Systems 7*, pages 633–640. Cambridge, MA, MIT Press, 1995.