L

Locally Weighted Regression for Control

Jo-Anne Ting[1], Franziska Meier[2], Sethu Vijayakumar[1,2], and Stefan Schaal[3,4]
[1]University of Edinburgh, Edinburgh, UK
[2]University of Southern California, Los Angeles, CA, USA
[3]Max Planck Institute for Intelligent Systems, Stuttgart, Germany
[4]Computer Science, University of Southern California, Los Angeles, CA, USA

Synonyms

Kernel shaping; Lazy learning; Locally weighted learning; Local distance metric adaptation; LWR; LWPR; Nonstationary kernels; Supersmoothing

Definition

This entry addresses two topics: learning control and locally weighted regression.

Learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. It is usually distinguished from adaptive control (Aström and Wittenmark 1989) in that the learning system is permitted to fail during the process of learning, resembling how humans and animals acquire new movement strategies. In contrast, adaptive control emphasizes single-trial convergence without failure, fulfilling stringent performance constraints, e.g., as needed in life-critical systems like airplanes and industrial robots.

Locally weighted regression refers to supervised learning of continuous functions (otherwise known as function approximation or regression) by means of spatially localized algorithms, which are often discussed in the context of kernel regression, nearest neighbor methods, or lazy learning (Atkeson et al. 1997). Most regression algorithms are global learning systems. For instance, many algorithms can be understood in terms of minimizing a global loss function such as the expected sum squared error:

$$J = E\left[\frac{1}{2}\sum_{i=1}^{N}(\mathbf{t}_i - \mathbf{y}_i)^2\right] = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \phi(\mathbf{x}_i)^T\boldsymbol{\beta}\right)^2\right] \quad (1)$$

where $E[\cdot]$ denotes the expectation operator, $\mathbf{t}_i$ the noise-corrupted target value for an input $\mathbf{x}_i$—which is expanded by basis functions into a basis function vector $\phi(\mathbf{x}_i)$-and $\boldsymbol{\beta}$ is the vector of (usually linear) regression coefficients. Classical feedforward neural networks, radial basis function networks, mixture models, or Gaussian process regression are all global function approximators in the spirit of Eq. (1).

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7502-7_493-1

In contrast, local learning systems conceptually split up the global learning problem into multiple simpler learning problems. Traditional locally weighted regression approaches achieve this by dividing up the cost function into multiple independent local cost functions,

$$J = E\left[\frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N} w_{k,i}\left(\mathbf{t}_i - \mathbf{x}_i^T\boldsymbol{\beta}_k\right)^2\right] = \frac{1}{2}\sum_{k=1}^{K} E\left[\sum_{i=1}^{N} w_{k,i}\left(\mathbf{t}_i - \mathbf{x}_i^T\boldsymbol{\beta}_k\right)^2\right] = \frac{1}{2}\sum_{k=1}^{K} J_k. \tag{2}$$

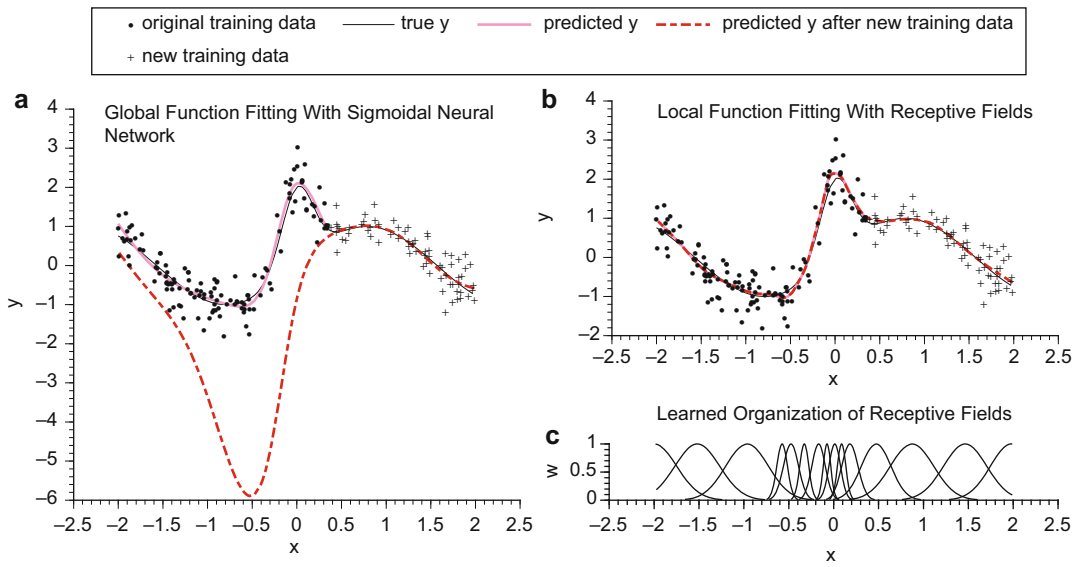
resulting in K (independent) local model learning problems. A different strategy for local learning starts out with the global objective (Eq. 1) and reformulates it to capture the idea of local models that cooperate to generate a (global) function fit. This is achieved by assuming there are K feature functions ϕ_k, such that the kth feature function $\phi_k(\mathbf{x}_i) = w_{k,i}\mathbf{x}_i$, resulting in

$$J = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \phi(\mathbf{x}_i)^T\boldsymbol{\beta}\right)^2\right] = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \sum_{k=1}^{K} w_{k,i}(\mathbf{x}_i^T\boldsymbol{\beta}_k)\right)^2\right]. \tag{3}$$

In this setting, local models are initially coupled and approximations are found to decouple the learning of the local models parameters.

Motivation and Background

Figure 1 illustrates why locally weighted regression methods are often favored over global methods when it comes to learning from incrementally arriving data, especially when dealing with nonstationary input distributions. The figure shows the division of the training data into two sets: the "original training data" and the "new training data" (in dots and crosses, respectively).



Locally Weighted Regression for Control, Fig. 1 Function approximation results for the function $y = \sin(2x) + 2\exp(-16x^2) + N(0, 0.16)$ with (**a**) a sigmoidal neural network, (**b**) a locally weighted regression algorithm (note that the data traces "true y," "predicted y," and "predicted y after new training data" largely coincide), and (**c**) the organization of the (Gaussian) kernels of (**b**) after training. See Schaal and Atkeson 1998 for more details

Initially, a sigmoidal neural network and a locally weighted regression algorithm are trained on the "original training data," using 20 % of the data as a cross validation set to assess convergence of the learning. In a second phase, both learning systems are trained solely on the "new training data" (again with a similar cross-validation procedure), but without using any data from the "original training data." While both algorithms generalize well on the "new training data," the global learner incurred catastrophic interference, unlearning what was learned initially, as seen in Fig. 1a. Figure 1b shows that the locally weighted regression algorithm does not have this problem since learning (along with generalization) is restricted to a local area.

Appealing properties of locally weighted regression include the following:

- Function approximation can be performed incrementally with nonstationary input and output distributions and without significant danger of interference. Locally weighted regression can provide posterior probability distributions, offer confidence assessments, and deal with heteroscedastic data.
- Locally weighted learning algorithms are computationally inexpensive to compute. It is well suited for online computations (e.g., for online and incremental learning) in the fast control loop of a robot—typically on the order of 100–1000 Hz.
- Locally weighted regression methods can implement continual learning and learning from large amounts of data without running into severe computational problems on modern computing hardware.
- Locally weighted regression is a nonparametric method (i.e., it does not require that the user determine a priori the number of local models in the learning system), and the learning systems grow with the complexity of the data it tries to model.
- Locally weighted regression can include feature selection, dimensionality reduction, and Bayesian inference—all which are required for robust statistical inference.
- Locally weighted regression works favorably with locally linear models (Hastie and Loader 1993), and local linearizations are of ubiquitous use in control applications.

Background

Returning to Eqs. (1) to (3), the main differences between global methods that directly solve Eq. (1) and local methods that solve either Eqs. (2) or (3) are listed below:

(i) A weight $w_{i,k}$ is introduced that focuses:
- either the function approximation fit in Eq. (2)
- or a local models contribution toward the global function fit in Eq. (3)

on only a small neighborhood around a point of interest $\mathbf{c}_k$ in input space (see Eq. 4 below).

(ii) The learning problem is split into K independent optimization problems.

(iii) Due to the restricted scope of the function approximation problem, we do not need a nonlinear basis function expansion and can, instead, work with simple local functions or local polynomials (Hastie and Loader 1993).

The weights $w_{k,i}$ in Eq. (2) are typically computed from some kernel function (Atkeson et al. 1997) such as a squared exponential kernel:

$$w_{k,i} = \exp\left(-\frac{1}{2}\left(\mathbf{x}_i - \mathbf{c}_k\right)^T \mathbf{D}_k \left(\mathbf{x}_i - \mathbf{c}_k\right)\right) \quad (4)$$

with $\mathbf{D}_k$ denoting a positive semidefinite distance metric and $\mathbf{c}_k$ the center of the kernel. The number of kernels K is not finite. In many local learning algorithms, the kernels are never maintained in memory. Instead, for every query point $\mathbf{x}_q$, a new kernel is centered at $\mathbf{c}_k = \mathbf{x}_q$, and the localized function approximation is solved with weighted regression techniques (Atkeson et al. 1997).

Locally weighted regression should not be confused with mixture of experts models (Jordan and Jacobs 1994). Mixture models are *global* learning systems since the experts compete globally to cover training data. Mixture models

address the bias-variance dilemma (Intuitively, the bias-variance dilemma addresses how many parameters to use for a function approximation problem to find an optimal balance between overfitting and oversmoothing of the training data.) by finding the right number of local experts. Locally weighted regression addresses the bias-variance dilemma in a local way by finding the optimal distance metric for computing the weights in the locally weighted regression (Schaal and Atkeson 1998).

Structure of Learning System

All local learning approaches have three critical components in common:

(i) Optimizing the regression parameters $\boldsymbol{\beta}_k$
(ii) Learning the distance metric $\mathbf{D}_k$ that defines a local model neighborhood
(iii) Choosing the location $\mathbf{c}_k$ of receptive field(s)

Local learning methods can be separated into "lazy" approaches that require all training data to be stored and "memoryless" approaches that compress data into a several local models and thus do not require storage of data points.

In the "lazy" approach, the computational burden of a prediction is deferred until the last moment, i.e., when a prediction is needed. Such a "compute-the-prediction-on-the-fly" approach is often called lazy learning and is a memory-based learning system where all training data is kept in memory for making predictions. A prediction is formed by optimizing the parameters $\boldsymbol{\beta}_q$ and distance metric $\mathbf{D}_q$ of one local model centered at the query point $\mathbf{c}_q = \mathbf{x}_q$.

Alternatively, in the "memoryless" approach, multiple kernels are created as needed to cover the input space, and the sufficient statistics of the weighted regression are updated incrementally with recursive least squares (Schaal and Atkeson 1998). This approach does not require storage of data points in memory. Predictions of neighboring local models can be blended, improving function fitting results in the spirit of committee machines.

We describe some algorithms of both flavors next.

Memory-Based Locally Weighted Regression (LWR)

The original locally weighted regression algorithm was introduced by Cleveland (1979) and popularized in the machine learning and learning control community by Atkeson (1989). The algorithm – categorized as a "lazy" approach – can be summarized as follows below (for algorithmic pseudo-code, see Schaal et al. 2002):

- All training data is collected in the rows of the matrix $\mathbf{X}$ and the vector (For simplicity, only functions with a scalar output are addressed. Vector-valued outputs can be learned either by fitting a separate learning system for each output or by modifying the algorithms to fit multiple outputs (similar to multi-output linear regression).) $\mathbf{t}$.
- For every query point $\mathbf{x}_q$, the weighting kernel is centered at the query point.
- The weights are computed with Eq. (4), and all data points' weights are collected in the diagonal weight matrix $\mathbf{W}_q$
- The local regression coefficients are computed as

$$\boldsymbol{\beta}_q = \left(\mathbf{X}^T \mathbf{W}_q \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W}_q \mathbf{t} \qquad (5)$$

- A prediction is formed with $y_q = \left[\mathbf{x}_q^T \;\; 1\right] \boldsymbol{\beta}_q$.

As in all kernel methods, it is important to optimize the kernel parameters in order to get optimal function fitting quality. For LWR, the critical parameter determining the bias-variance trade-off is the distance metric $\mathbf{D}_q$. If the kernel is too narrow, it starts fitting noise. If it is too broad, oversmoothing will occur. $\mathbf{D}_q$ can be optimized with leave-one-out cross validation to obtain a *globally* optimal value, i.e., the same $\mathbf{D}_q = \mathbf{D}$ is used throughout the entire input space of the data. Alternatively, $\mathbf{D}_q$ can be *locally* optimized as a function of the query point, i.e., obtain a $\mathbf{D}_q$

(as indicated by the subscript "q"). In the recent machine learning literature (in particular, work related to kernel methods), such input-dependent kernels are referred to as nonstationary kernels.

Locally Weighted Projection Regression (LWPR)

Schaal and Atkeson (1998) suggested a memoryless version of LWR, called RFWR, in order to avoid the expensive nearest neighbor computations—particularly for large training data sets—of LWR and to have fast real-time (In most robotic systems, "real time" means on the order of maximally 1–10 ms computation time, corresponding to a 1000 to 100 Hz control loop.) prediction performance. The main ideas of the RFWR algorithm (Schaal and Atkeson 1998) are listed below:

- Create new kernels only if no existing kernel in memory covers a training point with some minimal activation weight.
- Keep all created kernels in memory and update the weighted regression with weighted recursive least squares for new training points $\{\mathbf{x}, t\}$:

$$\boldsymbol{\beta}_k^{n+1} = \boldsymbol{\beta}_k^n + w\mathbf{P}^{n+1}\tilde{\mathbf{x}}\left(t - \tilde{\mathbf{x}}^T\boldsymbol{\beta}_k^n\right)$$

$$\text{where } \mathbf{P}_k^{n+1} = \frac{1}{\lambda}\left(\mathbf{P}_k^n - \frac{\mathbf{P}_k^n\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\mathbf{P}_k^n}{\frac{\lambda}{w} + \tilde{\mathbf{x}}^T\mathbf{P}_k^n\tilde{\mathbf{x}}}\right) \text{ and } \tilde{\mathbf{x}} = \left[\mathbf{x}^T \; 1\right]^T. \tag{6}$$

- Adjust the distance metric $\mathbf{D}_q$ for each kernel with a gradient descent technique using leave-one-out cross validation.
- Make a prediction for a query point taking a weighted average of predictions from all local models:

$$\mathbf{y}_q = \frac{\sum_{k=1}^K w_{q,k}\hat{\mathbf{y}}_{q,k}}{\sum_{k=1}^K w_{q,k}} \tag{7}$$

Adjusting the distance metric $\mathbf{D}_q$ with leave-one-out cross validation *without* keeping all training data in memory is possible due to the PRESS residual. The PRESS residual allows the leave-one-out cross validation error to be computed in closed form without needing to actually exclude a data point from the training data.

Another deficiency of LWR is its inability to scale well to high-dimensional input spaces since the covariance matrix inversion in Eq. (5) becomes severely ill-conditioned. Additionally, LWR becomes expensive to evaluate as the number of local models to be maintained increases. Vijayakumar et al. (2005) suggested local dimensionality reduction techniques to handle this problem. Partial least squares (PLS) regression is a useful dimensionality reduction method that is used in the LWPR algorithm (Vijayakumar et al. 2005). In contrast to PCA methods, PLS performs dimensionality reduction for regression, i.e., it eliminates subspaces of the input space that minimally correlates with the outputs, not just parts of the input space that have low variance.

While LWPR is typically used in conjunction with linear local models, the use of local non-parametric models, such as Gaussian processes, has also been explored (Nguyen-Tuong et al. 2008). Finally, LWPR is currently one of the best developed locally weighted regression algorithms for control (Klanke et al. 2008) and has been applied to learning control problems with over 100 input dimensions.

A Full Bayesian Treatment of Locally Weighted Regression

Ting et al. (2008) proposed a fully probabilistic treatment of LWR in an attempt to avoid cross-validation procedures and minimize any manual parameter tuning (e.g., gradient descent rates, kernel initialization, forgetting rates, etc.). The resulting Bayesian algorithm learns the distance metric of local linear model (For simplicity, a local linear model is assumed, although local polynomials can be used as well.) probabilistically, can cope with high input dimensions, and rejects data outliers automatically. The main ideas of Bayesian LWR are listed below (please see Ting 2009 for details):

- Introduce hidden variables $\mathbf{z}$ to the local linear model to decompose the statistical estimation problem into d individual estimation problems (where d is the number of input dimensions). The result is an iterative expectation-maximization (EM) algorithm that is of linear computational complexity in d and the number of training data samples N, i.e., $O(Nd)$.
- Associate a scalar weight w_i with each training data sample $\{\mathbf{x}_i, t_i\}$, placing a Bernoulli prior probability distribution over a weight w_{im} *for each input dimension* m so that the weights are positive and between 0 and 1:

$$w_i = \prod_{m=1}^{d} w_{im} \text{ where}$$
$$w_{im} \sim \text{Bernoulli}\,(q_{im}) \text{ for } i = 1, .., N;$$
$$m = 1, .., d \tag{8}$$

  The weight w_i indicates a training sample's contribution to the local model. The formulation of the parameter q_{im} determines the shape of the weighting function applied to the local model. The weighting function q_{im} used in Bayesian LWR is listed below:

$$q_{im} = \frac{1}{1 + \left(x_{im} - x_{qm}\right)^2 h_m} \text{ for } i = 1, .., N;$$
$$m = 1, .., d \tag{9}$$

  where $\mathbf{x}_q \in \Re^{d \times 1}$ is the query input point and h_m is the bandwidth parameter/distance metric of the local model in the m-th input dimension.
- Place a gamma prior probability distribution over the distance metric h_m:

$$h_m \sim \text{Gamma}\,(a_{hm0}, b_{hm0}) \tag{10}$$

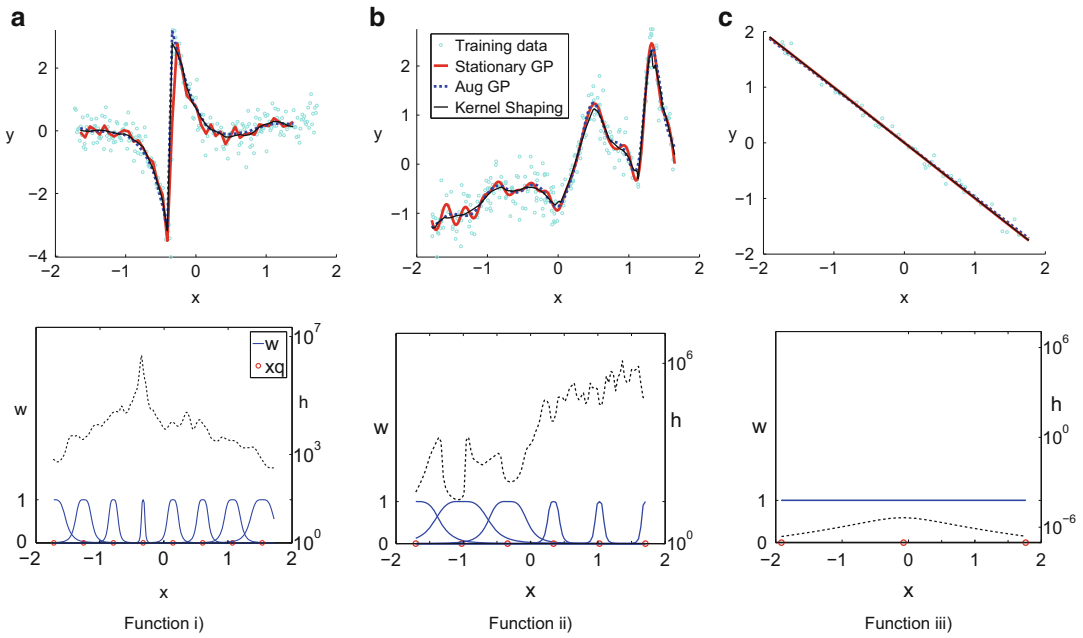
  where $\{a_{hm0}, b_{hm0}\}$ are the prior parameter values of the gamma distribution.
- Treat the model as an EM-like regression problem, using variational approximations to achieve analytically tractable inference of the posterior probability distributions.

This Bayesian method can also be applied as general kernel shaping algorithm for global kernel learning methods that are linear in the parameters (e.g., to realize nonstationary Gaussian processes (Ting et al. 2008), resulting in an augmented nonstationary Gaussian process).

Figure 2 illustrates Bayesian kernel shaping's bandwidth adaptation abilities on several synthetic data sets, comparing it to a stationary Gaussian process and the augmented nonstationary Gaussian process. For the ease of visualization, the following one-dimensional functions are considered: (i) a function with a discontinuity, (ii) a spatially inhomogeneous function, and (iii) a straight line function. Figure 2 shows the predicted outputs of all three models trained on noisy data drawn from data sets (i)–(iii). The local kernel shaping algorithm smoothens over regions where a stationary Gaussian process overfits, and yet, it still manages to capture regions of highly varying curvature, as seen in Fig. 2a, b. It correctly adjusts the bandwidths h with the curvature of the function. When the data looks linear, the algorithm opens up the weighting kernel so that all data samples are considered, as Fig. 2c shows.

From the viewpoint of learning control, overfitting—as seen in the Gaussian process in Fig. 2—can be detrimental since learning control often relies on extracting local linearizations to derive controllers. Obtaining the wrong sign on a slope in a local linearization may destabilize a controller.

In contrast to LWPR, the Bayesian LWR method is a "lazy" learner, although memoryless versions could be derived. Future work will also have to address how to incorporate dimensionality reduction methods for robustness in high dimensions. Nevertheless, it is a first step toward a probabilistic locally weighted regression method with minimal parameter tuning required by the user.



Locally Weighted Regression for Control, Fig. 2 Predicted outputs using a stationary Gaussian process (GP), the augmented nonstationary GP, and local kernel shaping on three different data sets. Figures on the *bottom row* show the bandwidths learned by local kernel shaping and the corresponding weighting kernels (in *dotted black lines*) for various input query points (shown in *red circles*)

L

From Global to Local: Local Regression with Coupling Between Local Models

Meier et al. (2014) offer an alternative approach to local learning. They start out with the global objective (Eq. 3) and reformulate it to capture the idea of local models that cooperate to generate a function fit, resulting in

$$J = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \sum_{k=1}^{K} w_{k,i}(\mathbf{x}_i^T\boldsymbol{\beta}_k)\right)^2\right]. \tag{11}$$

With this change, a local models' contribution $\hat{y}_k = \mathbf{x}_i^T\boldsymbol{\beta}_k$ toward the fit of target $\mathbf{t}_i$ is localized through weight $w_{k,i}$. However, this form of localization couples all local models. For efficient learning, local Gaussian regression (LGR) thus employs approximations to decouple learning of parameters. The main ideas of LGR are:

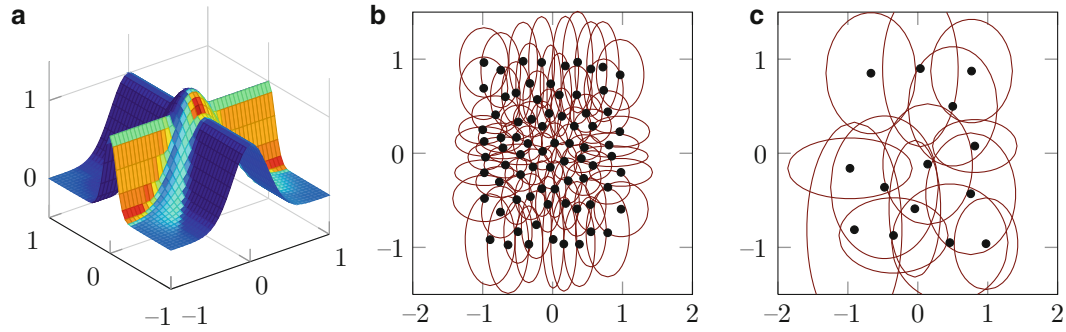
- Introduce Gaussian hidden variables $\mathbf{f}_k$ that form virtual targets for the weighted contribution of the kth local model:

$$f_{k,i} = \mathcal{N}\left(w_{k,i}(\mathbf{x}_i^T\boldsymbol{\beta}_k),\ \beta_m^{-1}\right) \tag{12}$$

  Assume that the target t is observed with Gaussian noise and that the hidden variables f_k need to sum up to noisy target t_i

$$t_i = \mathcal{N}\left(\sum_k f_{k,i},\ \beta_y^{-1}\right) \tag{13}$$

  In its exact form, this model learning procedure will couple all local models parameters.
- Employ a variational approximation to decouple local models. This results in an iterative (EM style) learning procedure, between updating posteriors over hidden variables $\mathbf{f}_k$ followed by posterior updates for regression parameters $\boldsymbol{\beta}_k$, for all local models $k = 1, \ldots, K$.
- The updates over the hidden variables $\mathbf{f}_k$ turn out to be a form of message passing between local model predictions. This step allows the redistribution of virtual target values for each



Locally Weighted Regression for Control, Fig. 3 Local models trained on data from the 2D cross function for LWPR and LGR. Local models trained via LWPR (visualized in (**b**)) do not know of each other, while local models trained by LGR (visualized in (**c**)) *collaborate* to generate a function fit

local model. This *communication* between local models is what distinguishes LGR from typical LWR approaches. This update is linear in the number of local models and in the number of data points.

- The parameter updates ($\boldsymbol{\beta}_k$ and $\mathbf{D}_k$) per local model become completely independent through the variational approximation, resulting in a localized learning algorithm, similar in spirit to LWR.
- Place Gaussian priors over regression parameters $\boldsymbol{\beta}_k \sim \mathcal{N}(\boldsymbol{\beta}_k; 0, \text{diag}(\boldsymbol{\alpha}_k))$ that allow for automatic relevance determination of the input dimensions.
- For incrementally incoming data, apply recursive Bayesian updates that utilize the posterior over parameters at time step $t-1$ to be the prior over parameters at time step t. Furthermore, new local models are added if no existing local model is activated with some minimal activation weight, similar to LWPR.
- Prediction for a query input $\mathbf{x}_q$ becomes a weighted average of local models predictions

$$y_q = \sum_{k=1}^{K} w_{k,q}(\mathbf{x}_q^T \boldsymbol{\beta}_k)$$

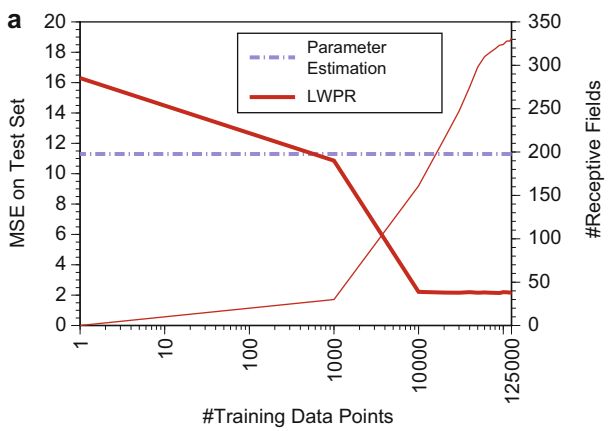
More details and a pseudo-algorithm for incremental LGR can be found in Meier et al. (2014). Figure 3 illustrates the different shapes of local models being learned by LWPR and LGR. Local models learned by LGR *collaborate* to generate a good fit, as visualized in Fig. 3c. Compared to LWPR, this often allows LGR to achieve similar predictive performance while using fewer local models.

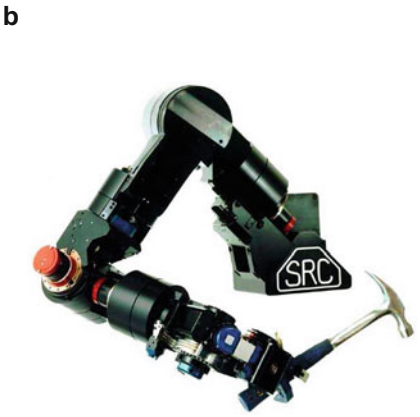
Finally, an interesting structural feature of local Gaussian regression is that it easily extends to a model with finitely many local nonparametric Gaussian process models.

Applications

Learning Internal Models with LWPR

Learning an internal model is one of most typical applications of local regression methods for control. The model could be a forward model (e.g., the nonlinear differential equations of robot dynamics), an inverse model (e.g., the equations that predict the amount of torque to achieve a change of state in a robot), or any other function that models associations between input and output data about the environment. The models are used, subsequently, to compute a controller, e.g., an inverse dynamics controller similar to Eq. (16). Models for complex robots such as like humanoids exceed easily a hundred input dimensions. In such high-dimensional spaces, it is hopeless to assume that a representative data set can be collected for offline training that





Locally Weighted Regression for Control, Fig. 4 Learning an inverse dynamics model in real time with a high-performance anthropomorphic robot arm. (**a**) Learning curve LWPR online learning. (**b**) Seven degree-of-freedom Sarcos robot arm

can generalize sufficiently to related tasks. Thus, the local regression philosophy involves having a learning algorithm that can learn rapidly when entering a new part of the state space such that it can achieve acceptable generalization performance almost instantaneously. Both LWPR (Vijayakumar et al. 2005) and incremental LGR (Meier et al. 2014) have been applied to inverse dynamics learning tasks.

Figure 4 demonstrates online learning of an inverse dynamics model for the elbow joint (cf. Eq. 16) for a Sarcos Dexterous Robot Arm. The robot starts with no knowledge about this model, and it tracks some randomly varying desired trajectories with a proportional-derivative (PD) controller. During its movements, training data consisting of tuples $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \tau)$—which model a mapping from joint position, joint velocities, and joint accelerations $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ to motor torques τ—are collected (at about every 2 ms). Here, every data point is used to train a LWPR function approximator, which generates a feedforward command for the controller. The learning curve is shown in Fig. 4a.

Using a test set created beforehand, the model predictions of LWPR are compared every 1000 training points with that of a parameter estimation method. The parameter estimation approach fits the minimal number of parameters to an analytical model of the robot dynamics under an idealized rigid body dynamics (RBD) assumptions, using all training data (i.e., not incrementally). Given that the Sarcos robot is a hydraulic robot, the RBD assumption is not very suitable, and, as Fig. 4a shows, LWPR (in thick red line) outperforms the analytical model (in dotted blue line) after a rather short amount of training. After about 5 min of training (about 125,000 data points), very good performance is achieved, using about 350 local models. This example demonstrates (i) the quality of function approximation that can be achieved with LWPR and (ii) the online allocation of more local models as needed.

Learning Paired Inverse-Forward Models

Learning inverse models (such as inverse kinematics and inverse dynamics models) can be challenging since the inverse model problem is often a relation, not a function, with a one-to-many mapping. Applying any arbitrary nonlinear function approximation method to the inverse model problem can lead to unpredictably bad performance, as the training data can form non-convex solution spaces, in which averaging is inappropriate. Architectures such as mixture models (in particular, mixture density networks) have been proposed to address problems with non-convex solution spaces. A particularly interesting approach in control involves learning lin-

Locally Weighted Regression for Control, Fig. 5 SensAble Phantom haptic robotic arm

earizations of a forward model (which is proper function) and learning an inverse mapping within the local region of the forward model.

Ting et al. (2008) demonstrated such a forward-inverse model learning approach with Bayesian LWR to learn an inverse kinematics model for a haptic robot arm (shown in Fig. 5) in order to control the end effector along a desired trajectory in task space. Training data was collected while the arm performed random sinusoidal movements within a constrained box volume of Cartesian space. Each sample consists of the arm's joint angles $\mathbf{q}$, joint velocities $\dot{\mathbf{q}}$, end-effector position in Cartesian space $\mathbf{x}$, and end-effector velocities $\dot{\mathbf{x}}$. From this data, a forward kinematics model is learned:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{14}$$

where $\mathbf{J}(\mathbf{q})$ is the Jacobian matrix. The transformation from $\dot{\mathbf{q}}$ to $\dot{\mathbf{x}}$ can be assumed to be locally linear at a particular configuration $\mathbf{q}$ of the robot arm. Bayesian LWR is used to learn the forward model, and, as in LWPR, local models are only added if a training point is not already sufficiently covered by an existing local model. Importantly, the kernel functions in LWR are localized only with respect to $\mathbf{q}$, while the regression of each model is trained only on a mapping from $\dot{\mathbf{q}}$ to $\dot{\mathbf{x}}$—these geometric insights are easily incorporated as priors in Bayesian LWR, as they are natural to locally linear models. Incorporating these priors in other function approximators, e.g., Gaussian process regression, is not straightforward.

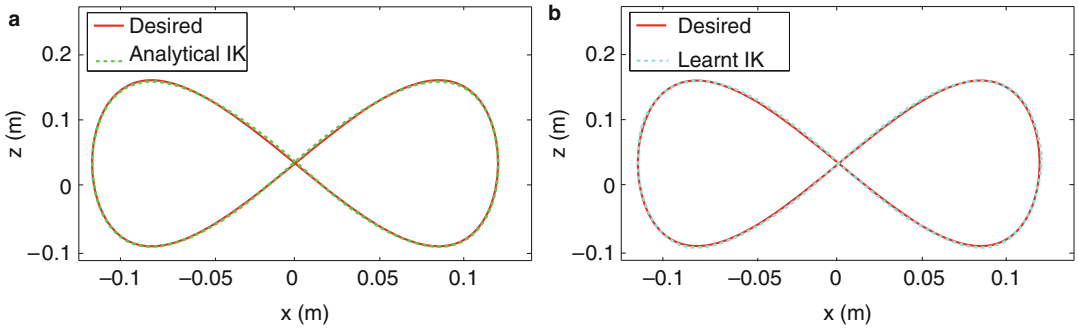
The goal of the robot task is to track a desired trajectory $(\mathbf{x}, \dot{\mathbf{x}})$ specified only in terms of x and z positions and velocities, i.e., the movement is supposed to be in a vertical plane in front of the robot, but the exact position of the vertical plane is not given. Thus, the task has one degree of redundancy, and the learning system needs to generate a mapping from $\{\mathbf{x}, \dot{\mathbf{x}}\}$ to $\dot{\mathbf{q}}$. Analytically, the inverse kinematics equation is

$$\dot{\mathbf{q}} = \mathbf{J}^{\#}(\mathbf{q})\dot{\mathbf{x}} - \alpha(\mathbf{I} - \mathbf{J}^{\#}\mathbf{J})\frac{\partial g}{\partial \mathbf{q}} \tag{15}$$

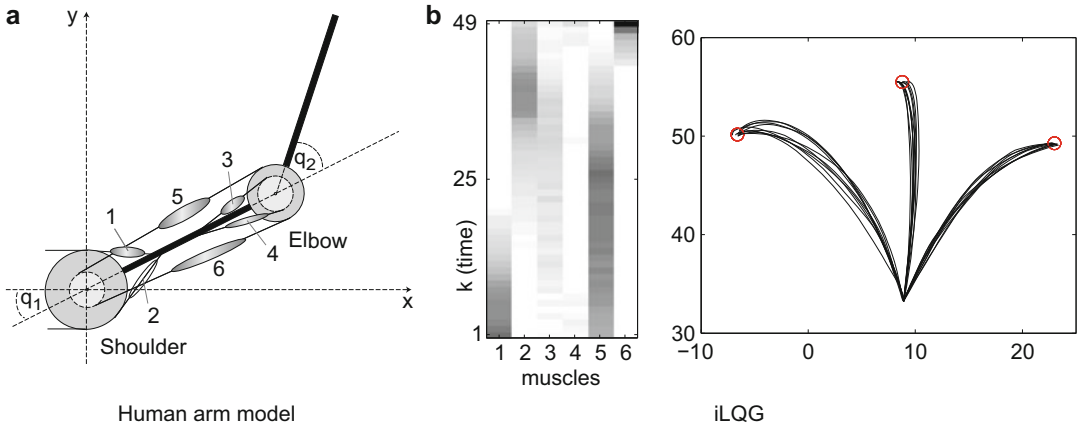
where $J^{\#}(\mathbf{q})$ is the pseudo-inverse of the Jacobian. The second term is a gradient descent optimization term for redundancy resolution, specified here by a cost function g in terms of joint angles $\mathbf{q}$.

To learn an inverse kinematics model, the local regions of $\mathbf{q}$ from the forward model can be reused since any inverse of $\mathbf{J}$ is locally linear within these regions. Moreover, for locally linear models, all solution spaces for the inverse model are locally convex, such that an inverse can be learned without problems. The redundancy issue can be solved by applying an additional weight to each data point according to a reward function. Since the experimental task is specified in terms of $\{\dot{x}, \dot{z}\}$, a reward is defined, based on a desired y coordinate, y_{des}, and enforced as a soft constraint. The resulting reward function is $g = e^{-\frac{1}{2}h(k(y_{des}-y)-\dot{y})^2}$, where k is a gain and h specifies the steepness of the reward. This ensures that the learned inverse model chooses a solution that pushes $\dot{y}$ toward y_{des}. Each forward local model is inverted using a weighted linear regression, where each data point is weighted by the kernel weight from the forward model and additionally weighted by the reward. Thus, a piecewise locally linear solution to the inverse problem can be learned efficiently.

Figure 6 shows the performance of the learned inverse model (Learnt IK) in a figure-eight track-



Locally Weighted Regression for Control, Fig. 6 Desired versus actual trajectories for SensAble Phantom robot arm. (**a**) Analytical solution. (**b**) Learned solution



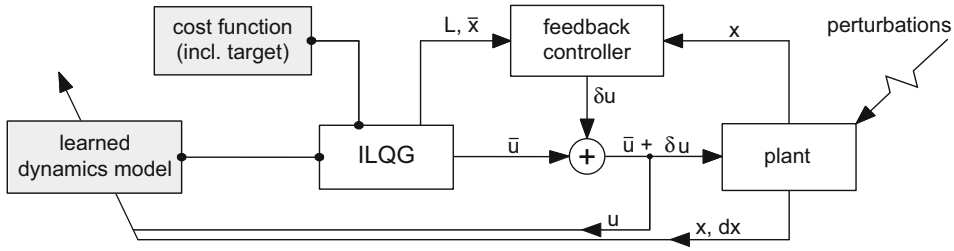
Locally Weighted Regression for Control, Fig. 7 (**a**) Human arm model with six muscles; (**b**) Optimized control sequence (*left*) and resulting trajectories (*right*) using the known analytic dynamics model. The control sequences (left target only) for each muscle (1–6) are drawn from bottom to top, with *darker gray* levels indicating stronger muscle activation

ing task. The learned model performs as well as the analytical inverse kinematics solution (Analytical IK), with root-mean-squared tracking errors in positions and velocities very close to that of the analytical solution.

Learning Trajectory Optimizations

Mitrovic et al. (2008) have explored a theory for sensorimotor adaptation in humans, i.e., how humans replan their movement trajectories in the presence of perturbations. They rely on the iterative Linear Quadratic Gaussian (iLQG) algorithm (Todorov and Li 2004) to deal with the nonlinear and changing plant dynamics that may result from altered morphology, wear and tear, or external perturbations. They take advantage of the "on-the-fly" adaptation of locally weighted regression methods like LWPR to learn the forward dynamics of a simulated arm for the purpose of optimizing a movement trajectory between a start point and an end point.

Figure 7a shows the diagram of a two degrees-of-freedom planar human arm model, which is actuated by four single-joint and two double-joint antagonistic muscles. Although kinematically simple, the system is over-actuated and, therefore, an interesting test bed because large redundancies in the dynamics have to be resolved. The dimensionality of the control signals makes adaptation processes (e.g., to external force fields) quite demanding.



Locally Weighted Regression for Control, Fig. 8 Illustration of learning and control scheme of the iterative Linear Quadratic Gaussian (iLQG) algorithm with learned dynamics

The dynamics of the arm is, in part, based on standard RBD equations of motion:

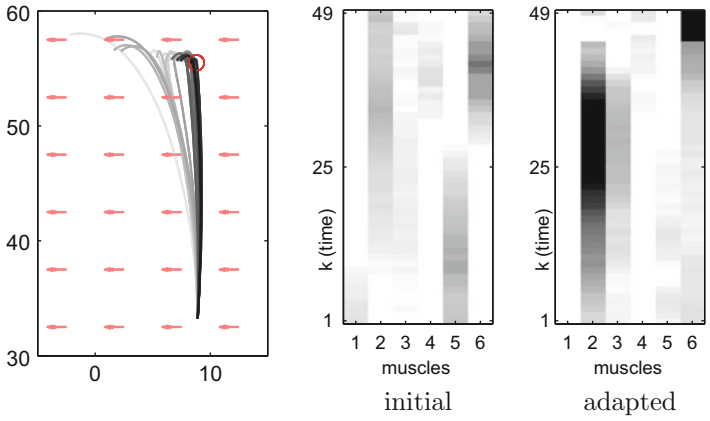
$$\tau = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} \tag{16}$$

where τ are the joint torques; $\mathbf{q}$ and $\dot{\mathbf{q}}$ are the joint angles and velocities, respectively; $\mathbf{M}(\mathbf{q})$ is the two-dimensional symmetric joint space inertia matrix; and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ accounts for Coriolis and centripetal forces. Given the antagonistic muscle-based actuation, it is not possible to command joint torques directly. Instead, the effective torques from the muscle activations $\mathbf{u}$—which happens to be quadratic in $\mathbf{u}$—should be used. As a result, in contrast to standard torque-controlled robots, the dynamics equation in Eq. (16) is *nonlinear in the control signals* $\mathbf{u}$.

The iLQG algorithm (Todorov and Li 2004) is used to calculate solutions to "localized" linear and quadratic approximations, which are iterated to improve the global control solution. However, it relies on an analytical forward dynamics model $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and finite difference methods to compute gradients. To alleviate this requirement and to make iLQG adaptive, LWPR can be used to learn an approximation of the plant's forward dynamics model. Figure 8 shows the control diagram, where the "learned dynamics model" (the forward model learned by LWPR) is then updated in an online fashion with every iteration to cope with changes in dynamics. The resulting framework is called iLQG-LD (iLQG with learned dynamics).

Movements of the arm model in Fig. 7a are studied for fixed time horizon reaching movement. The manipulator starts at an initial position $\mathbf{q}_0$ and reaches toward a target $\mathbf{q}_{tar}$. The cost function to be optimized during the movement is a combination of target accuracy and amount of muscle activation (i.e., energy consumption). Figure 7b shows trajectories of generated movements for three reference targets (shown in red circles) using the feedback controller from iLQG with the analytical plant dynamics. The trajectories generated with iLQG-LD (where the forward plant dynamics are learned with LWPR) are omitted as they are hardly distinguishable from the analytical solution.

A major advantage of iLQG-LD is that it does not rely on an accurate analytic dynamics model; this enables the framework to predict adaptation behavior under an ideal observer planning model. Reaching movements were studied where a constant unidirectional force field acting perpendicular to the reaching movement was generated as a perturbation (see Fig. 9 (left)). Using the iLQG-LD model, the manipulator gets strongly deflected when reaching for the target because the learned dynamics model cannot yet account for the "spurious" forces. However, when the deflected trajectory is used as training data and the dynamics model is updated online, the tracking improves with each new successive trial (Fig. 9 (left)). Please refer to Mitrovic et al. (2008) for more details. Aftereffects upon removing the force field, very similar to those observed in human experiments, are also observed.



Locally Weighted Regression for Control, Fig. 9 Adaptation to a unidirectional constant force field (indicated by the *arrows*). *Darker lines* indicate better trained models. In particular, the left-most trajectory corresponds to the "initial" control sequence, which was calculated using the LWPR model *before* the adaptation process. The fully "adapted" control sequence results in a nearly straight line reaching movement

Cross References

- Bayesian Inference
- Bias and Variance
- Dimensionality Reduction
- Direct and Indirect Control
- Incremental Learning
- Kernel Function
- Kernel Methods
- Lazy Learning
- Linear Regression
- Mixture Models
- On-line Learning
- Overfitting
- Radial Basis Functions
- Regression
- Supervised Learning
- Variational Approximations

Programs and Data

http://www-clmc.usc.edu/software
http://www.ipab.inf.ed.ac.uk/slmc/software/

Recommended Reading

Aström KJ, Wittenmark B (1989) Adaptive control. Addison-Wesley, Reading

Atkeson C (1989) Using local models to control movement. In: Proceedings of the advances in neural information processing systems, vol 1. Morgan Kaufmann, San Mateo, pp 157–183

Atkeson C, Moore A, Schaal S (1997) Locally weighted learning. AI Rev 11:11–73

Cleveland WS (1979) Robust locally weighted regression and smoothing scatterplots. J Am Stat Assoc 74:829–836

Hastie T, Loader C (1993) Local regression: automatic kernel carpentry. Stat Sci 8:120–143

Jordan MI, Jacobs R (1994) Hierarchical mixtures of experts and the EM algorithm. Neural Comput 6:181–214

Klanke S, Vijayakumar S, Schaal S (2008) A library for locally weighted projection regression. J Mach Learn Res 9:623–626

Meier F, Hennig P, Schaal S (2014) Incremental local Gaussian regression. In: Proceedings of advances in neural information processing systems, Montreal, vol 27

Mitrovic D, Klanke S, Vijayakumar S (2008) Adaptive optimal control for redundantly actuated arms. In: Proceedings of the 10th international conference on the simulation of adaptive behavior, Osaka. Springer, pp 93–102

Nguyen-Tuong D, Peters J, Seeger M (2008) Local Gaussian process regression for real-time online model learning. In: Proceedings of advances in neural information processing systems, Vancouver, vol 21

Schaal S, Atkeson CG (1998) Constructive incremental learning from only local information Neural Comput 10(8):2047–2084

Schaal S, Atkeson CG, Vijayakumar S (2002) Scalable techniques from nonparametric statistics. Appl Intell 17:49–60

Ting J (2009) Bayesian methods for autonomous learning systems. Phd Thesis, Department of Computer Science, University of Southern California

Ting J, Kalakrishnan M, Vijayakumar S, Schaal S (2008) Bayesian kernel shaping for learning control. In: Proceedings of advances in neural information processing systems, Vancouver, vol 21. MIT Press, pp 1673–1680

Todorov E, Li W (2004) A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In: Proceedings of 1st international conference of informatics in control, automation and robotics, Setúbal

Vijayakumar S, D'Souza A, Schaal S (2005) Incremental online learning in high dimensions. Neural Comput 17:2602–2634