Incremental Local Gaussian Regression

Franziska Meier[1]
fmeier@usc.edu

Philipp Hennig[2]
phennig@tue.mpg.de

Stefan Schaal[1,2]
sschaal@usc.edu

[1]**University of Southern California**
Los Angeles, CA 90089, USA

[2]**Max Planck Institute for Intelligent Systems**
Spemannstraße 38, Tübingen, Germany

Abstract

Locally weighted regression (LWR) was created as a nonparametric method that can approximate a wide range of functions, is computationally efficient, and can learn continually from very large amounts of incrementally collected data. As an interesting feature, LWR can regress on non-stationary functions, a beneficial property, for instance, in control problems. However, it does not provide a proper generative model for function values, and existing algorithms have a variety of manual tuning parameters that strongly influence bias, variance and learning speed of the results. Gaussian (process) regression, on the other hand, does provide a generative model with rather black-box automatic parameter tuning, but it has higher computational cost, especially for big data sets and if a non-stationary model is required. In this paper, we suggest a path from Gaussian (process) regression to locally weighted regression, where we retain the best of both approaches. Using a localizing function basis and approximate inference techniques, we build a Gaussian (process) regression algorithm of increasingly local nature and similar computational complexity to LWR. Empirical evaluations are performed on several synthetic and real robot datasets of increasing complexity and (big) data scale, and demonstrate that we consistently achieve on par or superior performance compared to current state-of-the-art methods while retaining a principled approach to fast incremental regression with minimal manual tuning parameters.

1 Introduction

Besides accuracy and sample efficiency, computational cost is a crucial design criterion for machine learning algorithms in real-time settings, such as control problems. An example is the modeling of robot dynamics: The sensors in a robot can produce thousands of data points per second, quickly amassing a coverage of the task related workspace, but what really matters is that the learning algorithm incorporates this data in real time, as a physical system can not necessarily stop and wait in its control – e.g., a biped would simply fall over. Thus, a learning method in such settings should produce a good local model in fractions of a second, and be able to extend this model as the robot explores new areas of a very high dimensional workspace that can often not be anticipated by collecting "representative" training data. Ideally, it should rapidly produce a good (local) model from a large number N of data points by adjusting a small number M of parameters. In robotics, local learning approaches such as locally weighted regression [1] have thus been favored over global approaches such as Gaussian process regression [2] in the past.

Local regression models approximate the function *in the neighborhood* of a query point $\boldsymbol{x}_*$. Each local model's region of validity is defined by a kernel. Learning the shape of that kernel [3] is the key component of locally weighted learning. Schaal & Atkeson [4] introduced a non-memory-based version of LWR to compress large amounts of data into a small number of parameters. Instead of keeping data in memory and constructing local models around query points on demand, their

algorithm incrementally compresses data into M local models, where M grows automatically to cover the experienced input space of the data. Each local model can have its own distance metric, allowing local adaptation to local characteristics like curvature or noise. Furthermore, each local model is trained independently, yielding a highly efficient parallelizable algorithm. Both its local adaptiveness and its low computation cost (linear, $\mathcal{O}(NM)$) has made LWR feasible and successful in control learning. The downside is that LWR requires several tuning parameters, whose optimal values can be highly data dependent. This is at least partly a result of the strongly localized training, which does not allow models to 'coordinate', or to benefit from other local models in their vicinity.

Gaussian process regression (GPR) [2], on the other hand, offers principled inference for hyperparameters, but at high computational cost. Recent progress in sparsifying Gaussian processes [5, 6] has resulted in computationally efficient variants of GPR . Sparsification is achieved either through a subset selection of support points [7, 8] or through sparsification of the spectrum of the GP [9, 10]. Online versions of such sparse GPs [11, 12, 13] have produced a viable alternative for real-time model learning problems [14]. However, these sparse approaches typically learn one global distance metric, making it difficult to fit the non-stationary data encountered in robotics. Moreover, restricting the resources in a GP also restricts the function space that can be covered, such that with the need to cover a growing workspace, the accuracy of learning with naturally diminish.

Here we develop a probabilistic alternative to LWR that, like GPR, has a global generative model, but is locally adaptive and retains LWRs fast incremental training. We start in the batch setting, where rethinking LWRs localization strategy results in a loss function coupling local models that can be modeled within the Gaussian regression framework (Section 2). Modifying and approximating the global model, we arrive at a localized batch learning procedure (Section 3), which we term *Local Gaussian Regression* (LGR). Finally, we develop an incremental version of LGR that processes streaming data (Section 4). Previous probabilistic formulations of local regression [15, 16, 17] are bottom-up constructions—generative models for one local model at a time. Ours is a top-down approach, approximating a global model to give a localized regression algorithm similar to LWR.

2 Background

Locally weighted regression (LWR) with a fixed set of M local models minimizes the loss function

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N}\sum_{m=1}^{M} \eta_m(\boldsymbol{x}_n)(y_n - \boldsymbol{\xi}_m(\boldsymbol{x}_n)^T \boldsymbol{w}_m)^2 = \sum_{m=1}^{M} \mathcal{L}(\boldsymbol{w}_m). \tag{1}$$

The right hand side decomposes $\mathcal{L}(\boldsymbol{w})$ into independent losses for M models. We assume each model has K local feature functions $\xi_{mk}(\boldsymbol{x})$, so that the m-th model's prediction at $\boldsymbol{x}$ is

$$f_m(\boldsymbol{x}) = \sum_{k=1}^{K} \xi_{mk}(\boldsymbol{x}) w_{mk} = \boldsymbol{\xi}_m(\boldsymbol{x})^\top \boldsymbol{w}_m \tag{2}$$

$K = 2, \xi_{m1}(\boldsymbol{x}) = 1, \xi_{m2}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{c}_m)$ gives a linear model around $\boldsymbol{c}_m$. Higher polynomials can be used, too, but linear models have a favorable bias-variance trade-off [18]. The models are localized by a non-negative, symmetric and integrable weighting $\eta_m(\boldsymbol{x})$, typically the radial basis function

$$\eta_m(x) = \exp\left[-\frac{(x - c_m)^2}{2\lambda_m^2}\right], \quad \text{or} \quad \eta_m(\boldsymbol{x}) = \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{c}_m)\Lambda_m^{-1}(\boldsymbol{x} - \boldsymbol{c}_m)^\top\right] \tag{3}$$

for $\boldsymbol{x} \in \mathbb{R}^D$, with center $\boldsymbol{c}_m$ and length scale λ_m or positive definite metric Λ_m. $\eta_m(\boldsymbol{x}_n)$ localizes the effect of errors on the least-squares estimate of $\boldsymbol{w}_m$—data points far away from $\boldsymbol{c}_m$ have little effect. The prediction y_* at a test point $\boldsymbol{x}_*$ is a normalized weighted average of the local predictions $y_{*,m}$:

$$y_* = \frac{\sum_{m=1}^{M} \eta_m(\boldsymbol{x}_*) f_m(\boldsymbol{x}_*)}{\sum_{m=1}^{M} \eta_m(\boldsymbol{x}_*)} \tag{4}$$

LWR effectively trains M linear models on M *separate* datasets $y_m(\boldsymbol{x}_n) = \sqrt{\eta_m(x_n)}y_n$. These models differ from the one of Eq. (4), used at test time. This smoothes discontinuous transitions between models, but also means that LWR can not be cast probabilistically as one generative model for training and test data simultaneously. (This holds for any bottom-up construction that learns local

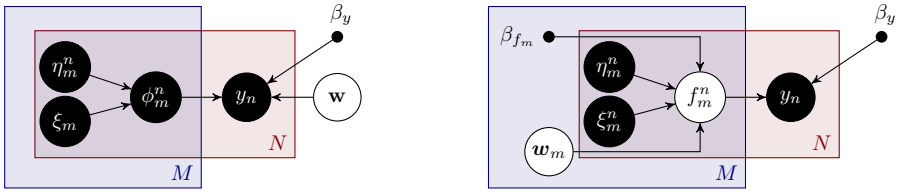


Figure 1: **Left:** Bayesian linear regression with M feature functions $\phi_m^n = \phi_m(\boldsymbol{x}_n) = \eta_m^n \xi_m^n$, where η_m^n can be a function localizing the effect of the m^{th} input function ξ_m^n towards the prediction of y_n. **Right:** Latent variables f_m^n placed between the features and y_n decouple the M regression parameters $\boldsymbol{w}_m$ and effectively create M local models connected only through the latent f_m^n.

models independently and combines them as above, e.g., [15, 16]). The independence of local models is key to LWR's training: changing one local model does not affect the others. While this lowers cost, we believe it is also partially responsible for LWR's sensitivity to manually tuned parameters.

Here, we investigate a different strategy to achieve localization, aiming to retain the computational complexity of LWR, while adding a sense of globality. Instead of using η_m to localize the training error of data points, we localize a model's contribution $\hat{y}_m = \boldsymbol{\xi}(x)^T \boldsymbol{w}_m$ towards the global fit of training point y, similar to how LWR operates during test time (Eq.4). Thus, already during training, local models must collaborate to fit a data point $\hat{y} = \sum_{m=1} \eta_m(\boldsymbol{x}) \boldsymbol{\xi}(x)^T \boldsymbol{w}_m$. Our loss function is

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} \left(y_n - \sum_{m=1}^{M} \eta_m(\boldsymbol{x}_n) \boldsymbol{\xi}_m(\boldsymbol{x}_n)^T \boldsymbol{w}_m \right)^2 = \sum_{n=1}^{N} \left(y_n - \sum_{m=1}^{M} \phi_m(\boldsymbol{x}_n)^T \boldsymbol{w}_m \right)^2, \tag{5}$$

combining the localizer $\eta_m(\boldsymbol{x}_n)$ and the m^{th} input function $\xi_m(\boldsymbol{x}_n)$ to form the feature $\boldsymbol{\phi}_m(\boldsymbol{x}_n) = \eta_m(\boldsymbol{x}_n)\xi_m(\boldsymbol{x}_n)$. This form of localization couples all local models, as in classical radial basis function networks [19]. At test time, all local predictions form a joined prediction

$$y_* = \sum_{m=1}^{M} y_{*m} = \sum_{m=1}^{M} \boldsymbol{\phi}_m(\boldsymbol{x}_*)^T \boldsymbol{w}_m \tag{6}$$

This loss can be minimized through a regularized least-square estimator for $\boldsymbol{w}$ (the concatenation of all $\boldsymbol{w}_m$). We follow the probabilistic interpretation of least-squares estimation as inference on the weights $\boldsymbol{w}$, from a Gaussian prior $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_0, \Sigma_0)$ and likelihood $p(\boldsymbol{y} \,|\, \boldsymbol{\phi}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{\phi}^\top \boldsymbol{w}, \beta_y^{-1} \boldsymbol{I})$. The probabilistic formulation has additional value as a generative model for all (training and test) data points y, which can be used to learn hyperparameters (Figure 1, left). The posterior is

$$p(\boldsymbol{w} \,|\, \boldsymbol{y}, \boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_N, \Sigma_N) \quad \text{with} \tag{7}$$

$$\boldsymbol{\mu}_N = (\Sigma_0^{-1} + \beta_y \Phi^\top \Phi)^{-1} (\beta_y \Phi^\top \boldsymbol{y} - \Sigma_0^{-1} \boldsymbol{\mu}_0) \quad \text{and} \quad \Sigma_N = (\Sigma_0^{-1} + \beta_y \Phi^\top \Phi)^{-1} \tag{8}$$

(Heteroscedastic data will be addressed below). The prediction for $f(\boldsymbol{x}_*)$ with features $\boldsymbol{\phi}(\boldsymbol{x}_*) =: \boldsymbol{\phi}_*$ is also Gaussian, with $p(f(\boldsymbol{x}_*) \,|\, \boldsymbol{y}, \boldsymbol{\phi}) = \mathcal{N}(f(\boldsymbol{x}_*); \boldsymbol{\phi}_* \boldsymbol{\mu}_N, \boldsymbol{\phi}_* \Sigma_N \boldsymbol{\phi}_*^\top)$. As is widely known, this framework can be extended nonparametrically by a limit that replaces all inner products $\boldsymbol{\phi}(\boldsymbol{x}_i) \Sigma_0 \boldsymbol{\phi}(\boldsymbol{x}_j)^\top$ with a Mercer (positive semi-definite) kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, corresponding to a Gaussian process prior. The direct connection between Gaussian regression and the elegant theory of Gaussian processes is a conceptual strength. The main downside, relative to LWR, is computational cost: Calculating the posterior (7) requires solving the least-squares problem for all F parameters $\boldsymbol{w}$ jointly, by inverting the Gram matrix $(\Sigma_0^{-1} + \beta_y \Phi^\top \Phi)$. In general, this requires $\mathcal{O}(F^3)$ operations. Below we propose approximations to lower the computational cost of this operation to a level comparable to LWR, while retaining the probabilistic interpretation, and the modeling robustness of the full Gaussian model.

3 Local Parametric Gaussian Regression

The above shows that Gaussian regression with features $\boldsymbol{\phi}_m(\boldsymbol{x}) = \eta_m(\boldsymbol{x}) \boldsymbol{\xi}_m(\boldsymbol{x})$ can be interpreted as global regression with M models, where $\eta_m(\boldsymbol{x}_n)$ localizes the contribution of the model $\boldsymbol{\xi}_m(\boldsymbol{x})$ towards the joint prediction of y_n. The choice of local parametric model $\boldsymbol{\xi}_m$ is essentially free. Local

linear regression in a K-dimensional input space takes the form $\boldsymbol{\xi}_m(\boldsymbol{x}_n) = \boldsymbol{x}_n - \boldsymbol{c}_m$, and can be viewed as the analog of locally weighted linear regression. Locally constant models $\xi_m(\boldsymbol{x}) = 1$ correspond to Gaussian regression with RBF features. Generalizing to M local models with K parameters each, feature function ϕ^n_{mk} combines the k^{th} component of the local model $\xi_{km}(\boldsymbol{x}_n)$, localized by the m-th weighting function $\eta_m(\boldsymbol{x}_n)$

$$\phi^n_{mk} := \phi_{mk}(\boldsymbol{x}_n) = \eta_m(\boldsymbol{x}_n)\xi_{km}(\boldsymbol{x}_n). \tag{9}$$

Treating mk as indices of a vector $\in \mathbb{R}^{MK}$, Equation (7) gives localized linear Gaussian regression. Since it will become necessary to prune the model, we adopt the classic idea of automatic relevance determination [20, 21] using a factorizing prior

$$p(\boldsymbol{w}|A) = \prod_{m=1}^{M} \mathcal{N}(\boldsymbol{w}_m; 0, A_m^{-1}) \quad \text{with} \quad A_m = \mathrm{diag}(\alpha_{m1}, \ldots, \alpha_{mK}). \tag{10}$$

Thus every component k of local model m has its own precision, and can be pruned out by setting $\alpha_{mk} \to \infty$. Section 3.1 assumes a fixed number M of local models with fixed centers $\boldsymbol{c}_m$. The parameters are $\boldsymbol{\theta} = \{\beta_y, \{\alpha_{mk}\}, \{\lambda_{md}\}\}$, where K is the dimension of local model $\boldsymbol{\xi}(\boldsymbol{x})$ and D is the dimension of input $\boldsymbol{x}$. We propose an approximation for estimating $\boldsymbol{\theta}$. Section 4 then describes an incremental algorithm allocating local models as needed, adapting M and $\boldsymbol{c}_m$.

3.1 Learning in Local Gaussian Regression

Exact Gaussian regression with localized features still has cubic cost. However, because of the localization, correlation between distant local models approximately vanishes, and inference is approximately independent between local models. To use this near-independence for cheap local approximate inference, similar to LWR, we introduce a latent variable f^n_m for each local model m and datum $\boldsymbol{x}_n$, as in probabilistic backfitting [22]. Intuitively, the $\boldsymbol{f}$ form approximate local targets, against which the local parameters fit (Figure 1, right). Moreover, as formalized below, each f^n_m has its own variance parameter, which re-introduces the ability to model hetereoscedastic data.

This modified model motivates a factorizing variational bound (Section 3.1.1). Rendering the local models computationally independent, it allows for fast approximate inference in the local Gaussian model. Hyperparameters can be learned by approximate maximum likelihood (Section 3.1.2), i.e. iterating between constructing a bound $q(\boldsymbol{z}\,|\,\boldsymbol{\theta})$ on the posterior over hidden variables $\boldsymbol{z}$ (defined below) given current parameter estimates $\boldsymbol{\theta}$ and optimizing q with respect to $\boldsymbol{\theta}$.

3.1.1 Variational Bound

The complete data likelihood of the modified model (Figure 1, right) is

$$p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{w}\,|\,\Phi, \theta) = \prod_{n=1}^{N} \mathcal{N}(y_n; \boldsymbol{f}^n, \beta_y) \prod_{n=1}^{N} \prod_{m=1}^{M} \mathcal{N}(f^n_m; \boldsymbol{\phi}^n_m \boldsymbol{w}_m, \beta_{fm}) \prod_{m=1}^{M} \mathcal{N}(\boldsymbol{w}_m; 0, A_m) \tag{11}$$

Our Gaussian model involves the latent variables $\boldsymbol{w}$ and $\boldsymbol{f}$, the precisions $\beta = \{\beta_y, \beta_{f1}, \ldots, \beta_{fM}\}$ and the model parameters $\boldsymbol{\lambda}_m, \boldsymbol{c}_m$. We treat $\boldsymbol{w}$ and $\boldsymbol{f}$ as probabilistic variables and estimate $\theta = \{\beta, \boldsymbol{\lambda}, \boldsymbol{c}\}$. On $\boldsymbol{w}, \boldsymbol{f}$, we construct a variational bound $q(\boldsymbol{w}, \boldsymbol{f})$ imposing factorization $q(\boldsymbol{w}, \boldsymbol{f}) = q(\boldsymbol{w})q(\boldsymbol{f})$. The variational free energy is a lower bound on the log evidence for the observations y:

$$\log p(\boldsymbol{y}\,|\,\theta) \geq \int q(\boldsymbol{w}, \boldsymbol{f}) \log \frac{p(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{f}\,|\,\theta)}{q(\boldsymbol{w}, \boldsymbol{f})}. \tag{12}$$

This bound is maximized by the $q(\boldsymbol{w}, \boldsymbol{f})$ minimizing the relative entropy $D_{\mathrm{KL}}[q(\boldsymbol{w}, \boldsymbol{f}) \| p(\boldsymbol{w}, \boldsymbol{f}\,|\,\boldsymbol{y}, \theta)]$, the distribution for which $\log q(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{f}}[\log p(\boldsymbol{y}\,|\,\boldsymbol{f}, \boldsymbol{w}) p(\boldsymbol{w}, \boldsymbol{f})]$ and $\log q(\boldsymbol{f}) = \mathbb{E}_{\boldsymbol{w}}[\log p(\boldsymbol{y}\,|\,\boldsymbol{f}, \boldsymbol{w}) p(\boldsymbol{w}, \boldsymbol{f})]$. It is relatively easy to show (e.g. [23]) that these distributions are Gaussian in both $\boldsymbol{w}$ and $\boldsymbol{f}$.The approximation on $\boldsymbol{w}$ is

$$\log q(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{f}} \left[\sum_{n=1}^{N} \log p(\boldsymbol{f}^n\,|\,\boldsymbol{\phi}^n, \boldsymbol{w}) + \log p(\boldsymbol{w}\,|\,A) \right] = \log \prod_{m=1}^{M} \mathcal{N}(\boldsymbol{w}_m; \boldsymbol{\mu}_{w_m}, \Sigma_{w_m}) \tag{13}$$

where

$$\Sigma_{w_m} = \left(\beta_{fm} \sum_{n=1}^{N} \boldsymbol{\phi}^n_m \boldsymbol{\phi}^{n\,T}_m + A_m \right)^{-1} \in \mathbb{R}^{K \times K} \quad \text{and} \quad \boldsymbol{\mu}_{w_m} = \beta_{fm} \Sigma_{w_m} \left(\sum_{n=1}^{N} \boldsymbol{\phi}^n_m \mathbb{E}\left[f^n_m \right] \right) \in \mathbb{R}^{K \times 1} \tag{14}$$

The posterior update equations for the weights are *local*: each of the local models updates its parameters independently. This comes at the cost of having to update the belief over the variables f_m^n, which achieves a coupling between the local models. The Gaussian variational bound on $\boldsymbol{f}$ is

$$\log q(\boldsymbol{f}^n) = \mathbb{E}_{\boldsymbol{w}}\left[\log p(y_n \mid \boldsymbol{f}^n, \beta_y) + \log p(\boldsymbol{f}^n \mid \boldsymbol{\phi}_m^n, \boldsymbol{w})\right] = \mathcal{N}(\boldsymbol{f}^n; \boldsymbol{\mu}_{fn}, \Sigma_f), \tag{15}$$

where

$$\Sigma_f = B^{-1} - B^{-1}\mathbf{1}(\beta_y^{-1} + \mathbf{1}^T B^{-1}\mathbf{1})^{-1}\mathbf{1}^T B^{-1} = B^{-1} - \frac{B^{-1}\mathbf{1}\mathbf{1}^T B^{-1}}{\beta_y^{-1} + \mathbf{1}^T B^{-1}\mathbf{1}} \tag{16}$$

$$\mu_{f_m^n} = \mathbb{E}_{\boldsymbol{w}}\left[\boldsymbol{w}_m\right]^T \boldsymbol{\phi}_m^n \frac{\beta_{fm}^{-1}}{\beta_y^{-1} + \sum_{m=1}^M \beta_{fm}^{-1}} \left(y_n - \sum_{m=1}^M \mathbb{E}_{\boldsymbol{w}}\left[\boldsymbol{w}_m\right]^T \boldsymbol{\phi}_m^n\right) \tag{17}$$

and $B = \mathrm{diag}\,(\beta_{f1}, \ldots, \beta_{fM})$. $\mu_{f_m^n}$ is the posterior mean of the m-th model's virtual target for data point n. These updates can be performed in $\mathcal{O}(MK)$. Note how the posterior over hidden variables $\boldsymbol{f}$ couples the local models, allowing for a form of message passing between local models.

3.1.2 Optimizing Hyperparameters

To set the parameters $\theta = \{\beta_y, \{\beta_{fm}, \lambda_m\}_{m=1}^M, \{\alpha_{mk}\}\}$, we maximize the expected complete log likelihood under the variational bound

$$\mathbb{E}_{\boldsymbol{f},\boldsymbol{w}}[\log p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{w} \mid \Phi, \theta)] = \mathbb{E}_{\boldsymbol{f},\boldsymbol{w}}\Bigg\{ \sum_{n=1}^N \Bigg[\log \mathcal{N}\left(y_n; \sum_{m=1}^M f_m^n, \beta_y^{-1}\right) + \sum_{m=1}^M \log \mathcal{N}(f_m^n; \boldsymbol{w}_m^T \boldsymbol{\phi}_m^n, \beta_{fm}^{-1}) \Bigg] + \sum_{m=1}^M \log \mathcal{N}(\boldsymbol{w}_m; 0, A_m^{-1}) \Bigg\}. \tag{18}$$

Setting the gradient of this expression to zero leads to the following update equations for the variances

$$\beta_y^{-1} = \frac{1}{N}\sum_{n=1}^N (y_n - \mathbf{1}\boldsymbol{\mu}_{fn})^2 + \mathbf{1}^T \Sigma_f \mathbf{1} \tag{19}$$

$$\beta_{fm}^{-1} = \frac{1}{N}\sum_{n=1}^N \left[(\mu_{fnm} - \boldsymbol{\mu}_{w_m}\boldsymbol{\phi}_m^n)^2 + {\boldsymbol{\phi}_m^n}^T \Sigma_{w_m} \boldsymbol{\phi}_m^n\right] + \sigma_{fm}^2 \tag{20}$$

$$\alpha_{mk}^{-1} = \mu_{w_{mk}}^2 + \Sigma_{w,kk} \tag{21}$$

The gradient with respect to the scales of each local model is completely localized

$$\frac{\partial \mathbb{E}_{\boldsymbol{f},\boldsymbol{w}}\left[\log p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{w} \mid \Phi, \theta)\right]}{\partial \lambda_{md}} = \frac{\partial \mathbb{E}_{\boldsymbol{f},\boldsymbol{w}}\left[\sum_{n=1}^N \mathcal{N}(f_m^n; \boldsymbol{w}_m^T \boldsymbol{\phi}_m^n, \beta_{fm}^{-1})\right]}{\partial \lambda_{md}} \tag{22}$$

We use gradient ascent to optimize the length scales λ_{md}. All necessary equations are of low cost and, with the exception of the variance $1/\beta_y$, all hyper-parameter updates are solved independently for each local model, similar to LWR. In contrast to LWR, however, these local updates *do not cause a potential catastrophic shrinking in the length scales*: In LWR, both inputs and outputs are weighted by the localizing function, thus reducing the length scale improves the fit. The localization in Equation (22) only affects the influence of regression model m, but the targets still need to be fit accordingly. Shrinking of local models only happens if it actually improves the fit against the unweighted targets f_{nm} such that no complex cross validation procedures are required.

3.1.3 Prediction

Predictions at a test point $\boldsymbol{x}_*$ arise from marginalizing over both $\boldsymbol{f}$ and $\boldsymbol{w}$, using

$$\int \left[\int \mathcal{N}(y_*; \mathbf{1}^T \boldsymbol{f}_*, \beta_y^{-1}) \mathcal{N}(\boldsymbol{f}_*; W^T \boldsymbol{\phi}(x_*), B^{-1}) \mathrm{d}\boldsymbol{f}_*\right] \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}_w, \Sigma_w) \mathrm{d}\boldsymbol{w} = \mathcal{N}\left(y_*; \sum_m \boldsymbol{w}_m^T \boldsymbol{\phi}_m^*, \sigma^2(x^*)\right) \tag{23}$$

where $\sigma^2(x^*) = \beta_y^{-1} + \sum_{m=1}^M \beta_{fm}^{-1} + \sum_{m=1}^M {\boldsymbol{\phi}_m^*}^T \Sigma_{w_m} \boldsymbol{\phi}_m^*$, which is linear in M and K.

4 Incremental Local Gaussian Regression

The above approximate posterior updates apply in the batch setting, assuming the number M and locations $\boldsymbol{c}$ of local models are fixed. This section constructs an online algorithm for incrementally incoming data, creating new local models when needed. There has been recent interest in variational online algorithms for efficient learning on large data sets [24, 25]. Stochastic variational inference [24] operates under the assumption that the data set has a fixed size N and optimizes the variational lower bound for N data points via stochastic gradient descent. Here, we follow algorithms for streaming datasets of unknown size. Probabilistic methods in this setting typically follow a Bayesian filtering approach [26, 25, 27] in which the posterior after $n-1$ data points becomes the prior for the n-th incoming data point. Following this principle we extend the model presented in Section 3 and treat precision variables $\{\beta_{fm}, \alpha_{mk}\}$ as random variables, assuming Gamma priors $p(\beta_{fm}) = \mathcal{G}(\beta_{fm} \,|\, a_0^\beta, b_0^\beta)$ and $p(\boldsymbol{\alpha}_m) = \prod_{k=1}^K \mathcal{G}(\alpha_{mk} \,|\, a_0^\alpha, b_0^\alpha)$. Thus, the factorized approximation on the posterior $q(\boldsymbol{z})$ over all random variables $\boldsymbol{z} = \{\boldsymbol{f}, \boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}_f\}$ is changed to

$$q(\boldsymbol{z}) = q(\boldsymbol{f}, \boldsymbol{w}, \boldsymbol{\beta}_f, \boldsymbol{\alpha}) = q(\boldsymbol{f}) q(\boldsymbol{w}) q(\boldsymbol{\beta}_f) q(\boldsymbol{\alpha}) \tag{24}$$

A batch version of this was introduced in [28]. Given that, the recursive application of Bayes' theorem results in the approximate posterior

$$p(\boldsymbol{z} | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \approx p(\boldsymbol{x}_n \,|\, \boldsymbol{z}) q(\boldsymbol{z} \,|\, \boldsymbol{x}_1, \ldots \boldsymbol{x}_{n-1}) \tag{25}$$

after n data points. In essence, this formulates the (approximate) posterior updates in terms of sufficient statistics, which are updated with each new incoming data point. The batch updates (listed in [28]) can be rewritten such that they depend on the following sufficient statistics $\sum_{n=1}^N \boldsymbol{\phi}_m^n \boldsymbol{\phi}_m^{n\,\top}, \sum_{n=1} \boldsymbol{\phi}_m^n \mu_{fm}^n$ and $\sum_{n=1} (\mu_{fm}^n)^2$. Although the length-scales $\boldsymbol{\lambda}_m$ could be treated as random variables too, here we update them using the noisy (stochastic) gradients produced by each incoming data point. Due to space limitations, we only summarize these update equations in the algorithm below, where we have replaced the expectation operator by $\langle \cdot \rangle$.

Finally, we use an extension analogous to incremental training of the relevance vector machine [29] to iteratively add local models at new, greedily selected locations $\boldsymbol{c}_{M+1}$. Starting with one local model, each iteration adds one local model in the variational step, and prunes out existing local models for which all components $\alpha_{mk} \rightarrowtail \infty$. This works well in practice, with the caveat that the model number M can grow fast initially, before the pruning becomes effective. Thus, we check for each selected location $\boldsymbol{c}_{M+1}$ whether any of the existing local models $\boldsymbol{c}_{1:M}$ produces a localizing weight $\eta_m(\boldsymbol{c}_{M+1}) \geq w_{\text{gen}}$, where w_{gen} is a parameter between 0 and 1 and regulates how many parameters are added. Algorithm 1 gives an overview of the entire incremental algorithm.

Algorithm 1 Incremental LGR

$M = 0; C = \{\}, a_0^\alpha, b_0^\alpha, a_0^\beta, \beta_0^\beta$, forgetting rate κ, learning rate ν
for all $(\boldsymbol{x}_n, y_n)$ **do** // for each data point
  if $\eta_m(\boldsymbol{x}_n) < w_{\text{gen}}, \forall m = 1, \ldots, M$ **then** $c_m \leftarrowtail x_n$; $C \leftarrowtail C \cup \{c_m\}$; $M = M + 1$ **end if**
  $\Sigma_f = B^{-1} - \frac{B^{-1}\mathbf{1}\mathbf{1}^T B^{-1}}{\beta_y^{-1} + \sum_m \langle\beta\rangle_{fm}}$, $\mu_{f_m^n} = \mu_{w_m}^T \boldsymbol{\phi}_m^n \frac{\beta_{fm}^{-1}}{\beta_y^{-1} + \sum_{m=1}^M \langle\beta\rangle_{fm}^{-1}} \left(y_n - \sum_{m=1}^M \mu_{w_m}^T \boldsymbol{\phi}_m^n\right)$
  for $m = 1$ **to** M **do**
    if $\eta_m(\boldsymbol{x}_n) < 0.01$ **then** continue **end if**
    $S_{\boldsymbol{\phi}_m \boldsymbol{\phi}_m^T} \leftarrowtail \kappa S_{\boldsymbol{\phi}_m \boldsymbol{\phi}_m^\top} + \boldsymbol{\phi}_m^n \boldsymbol{\phi}_m^{n\,\top}$, $S_{\boldsymbol{\phi}_m \mu_{fm}} \leftarrowtail \kappa S_{\boldsymbol{\phi}_m \mu_{fm}} + \phi_m^n \mu_{f_m^n}$, $S_{\mu_{f_m}^2} \leftarrowtail \kappa S_{\mu_{f_m}^2} + \mu_{f_m^n}^2$
    $\Sigma_{w_m} = \left(\langle\beta\rangle_{fm} S_{\boldsymbol{\phi}_m \boldsymbol{\phi}_m^T} + \langle A\rangle_m\right)^{-1}$, $\mu_{w_m} = \langle\beta\rangle_{fm} \Sigma_{w_m} S_{\boldsymbol{\phi}_m \mu_{fm}}$
    $N_m = \kappa N_m + 1$, $a_{Nm}^\beta = a_0^\beta + N_m$, $a_{Nm}^\alpha = a_0^\alpha + 0.5$
    $b_{Nm}^\beta = S_{\mu_{f_m^n}^2} - 2\mu_{w_m}^\top S_{\boldsymbol{\phi}_m \mu_{fm}} + \operatorname{tr}\left[S_{\boldsymbol{\phi}_m \boldsymbol{\phi}_m^\top} (\Sigma_{w_m} + \mu_{w_m} \mu_{w_m}^\top)\right] + N_m \sigma_{fm}^2$
    $b_{Nmk}^\alpha = \mu_{w_m,k}^2 + \Sigma_{w_m,kk}$
    $\langle\beta\rangle_{fm} = a_{Nm}^\beta / b_{Nm}^\beta$, $\langle A\rangle_m = \operatorname{diag}\left(a_{Nmk}^\alpha / b_{Nmk}^\alpha\right)$
    $\lambda_m = \lambda_m + \nu \left(\partial / \partial \lambda_m \mathcal{N}(\langle f^n\rangle_m; \langle \boldsymbol{w}\rangle_m^T \boldsymbol{\phi}_m^n, \langle\beta\rangle_{fm}^{-1})\right)$
    if $\langle\alpha\rangle_{mk} > 1e3 \quad \forall k = 1, \ldots, K$ **then** prune local model m, $\quad M \leftarrowtail M - 1$ **end if**
  end for
end for

Table 1: Datasets for inverse dynamics tasks: KUKA_1, KUKA_2 are different splits of the same data. Rightmost column indicates the overlap in input space coverage between offline ($\text{IS}_{\text{offline}}$) and online training ($\text{IS}_{\text{online}}$) sets.

Dataset	freq	Motion	$N_{\text{offline train}}$	$N_{\text{online train}}$	N_{test}	$\text{IS}_{\text{offline}} \cup \text{IS}_{\text{online}}$
Sarcos [2]	100	rhythmic	4449	44484	-	large overlap
KUKA_1	500	rhythmic at various speeds	17560	180360	-	small overlap
KUKA_2	500	rhythmic at various speeds	17560	180360	-	no overlap
KUKA_{sim}	500	rhythmic + discrete	-	1984950	20050	-

5 Experiments

We evaluate our LGR on inverse dynamics learning tasks, using data from two robotic platforms: a SARCOS anthropomorphic arm and a KUKA lightweight arm. For both robots, learning the inverse dynamics involves learning a map from the joint positions $\boldsymbol{q}$ (rad), velocities $\dot{\boldsymbol{q}}$ ($^{\text{rad}}/_{s}$) and accelerations $\ddot{\boldsymbol{q}}$ ($^{\text{rad}}/_{s^2}$), to torques τ (Nm) for each of 7 joints (degrees of freedom). We compare to two methods previously used for inverse dynamics learning: LWPR[1] – an extension of LWR for high dimensional spaces [31] – and I-SSGPR[2] [13] – an incremental version of Sparse Spectrum GPR. I-SSGPR differs from LGR and LWPR in that it is a global method and does not learn the distance metric online. Instead, I-SSGPR needs offline training of hyperparameters before it can be used online. We mimic the procedure used in [13]: An offline training set is used to learn an initial model and hyperparameters, then an online training set is used to evaluate incremental learning. Where indicated we use initial offline training for all three methods. I-SSGPR uses typical GPR optimization procedures for offline training, and is thus only available in batch mode. For LGR, we use the batch version for pre-training/hyperparameter learning. For all experiments we initialized the length scales to $\lambda = 0.3$, and used $w_{\text{gen}} = 0.3$ for both LWPR and LGR.

We evaluate on four different data sets, listed in Table 1. These sets vary in scale, types of motion and how well the offline training set represents the data encountered during online learning. All results were averaged over 5 randomly seeded runs, mean-squared error (MSE) and normalized mean-squared error (nMSE) are reported on the online training dataset. The nMSE is reported as the mean-squared error normalized by the variance of the outputs.

Table 2: Predictive performance on online training data of **Sarcos** after one sweep. I-SSGPR has been trained with $200(400)$ features, MSE for 400 features is reported in brackets.

	I-SSGPR$_{200(400)}$		**LWPR**			**LGR**		
Joint	MSE	nMSE	MSE	nMSE	# of LM	MSE	nMSE	# of LM
J1	13.699 (10.832)	0.033	19.180	0.046	461.4	11.434	0.027	321.4
J2	6.158 (4.788)	0.027	9.783	0.044	495.0	8.342	0.037	287.4
J3	1.803 (1.415)	0.018	3.595	0.036	464.6	2.237	0.023	298.0
J4	1.198 (0.857)	0.006	4.807	0.025	382.8	5.079	0.027	303.2
J5	0.034 (0.027)	0.036	0.071	0.075	431.2	0.031	0.033	344.2
J6	0.129 (0.096)	0.044	0.248	0.085	510.2	0.101	0.034	344.2
J7	0.093 (0.063)	0.014	0.231	0.034	378.8	0.170	0.025	348.8

Sarcos: Table 2 summarizes results on the popular Sarcos benchmark for inverse dynamics learning tasks [2]. The traditional test set is used as the offline training data to pre-train all three models. I-SSGPR is trained with 200 and 400 sparse spectrum features, indicated as $\text{I-SSGPR}_{200(400)}$, where 200 features is the optimal design choice according to $[13]$. We report the (normalized) mean-squared error on the online training data, after one sweep through it - i.e. each data point has been used once - has been performed. All three methods perform well on this data, with I-SSGPR and LGR having a slight edge over LWPR in terms of accuracy; and LGR uses fewer local models than LWPR. The Sarcos data offline training set represents the data encountered during online training very well. Thus, here online distance metric learning is not necessary to achieve good performance.

[1] we use the LWPR implementation found in the SL simulation software package [30]

[2] we use code from the learningMachine library in the RobotCub framework, from http:// eris.liralab.it/iCub

Table 3: Predictive performance on online training data of KUKA_1 and KUKA_2 after one sweep. KUKA_2 results are averages across joints. I-SSGPR was trained on 200 and 400 features (results for I-SSGPR_{400} shown in brackets).

		I-SSGPR$_{200(400)}$		**LWPR**			**LGR**		
data	Joint	MSE	nMSE	MSE	nMSE	# of LM	MSE	nMSE	# of LM
KUKA_1	J1	7.021 (7.680)	0.233	2.362	0.078	3476.8	2.238	0.074	3188.6
	J2	16.385 (18.492)	0.265	2.359	0.038	3508.6	2.738	0.044	3363.8
	J3	1.872 (1.824)	0.289	0.457	0.071	3477.2	0.528	0.082	3246.6
	J4	3.124 (3.460)	0.256	0.503	0.041	3494.6	0.571	0.047	3333.6
	J5	0.095 (0.143)	0.196	0.019	0.039	3512.4	0.017	0.036	3184.4
	J6	0.142 (0.296)	0.139	0.043	0.042	3561.0	0.029	0.029	3372.4
	J7	0.129 (0.198)	0.174	0.023	0.031	3625.6	0.033	0.044	3232.6
KUKA_2	-	9.740 (9.985)	0.507	1.064	0.056	3617.7	1.012	0.054	3290.2

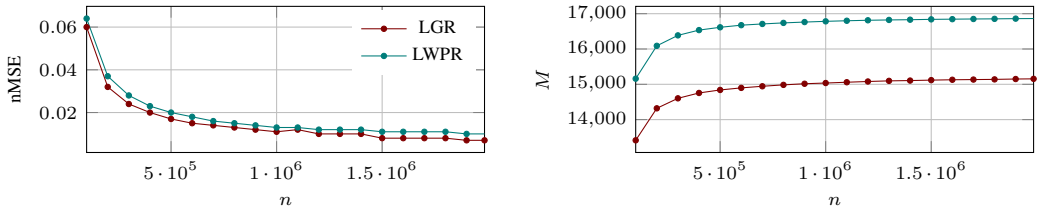


Figure 2: **Right:** nMSE on the first joint of simulated KUKA arm **Left:** average number of local models used.

KUKA_1 and KUKA_2: The two KUKA datasets consist of rhythmic motions at various speeds, and represent a more realistic setting in robotics: While one can collect some data for offline training, it is not feasible to cover the whole state-space. Offline data of KUKA_1 has been chosen to give partial coverage of the range of available speeds, while KUKA_2 consists of motion at only one speed. In this setting, both LWPR and LGR excel (Table 3). As they can learn local distance metrics on the fly, they adapt to incoming data in previously unexplored input areas. Performance of I-SSGPR_{200} degrades as the offline training data is less representative, while LGR and LWPR perform almost equally well on KUKA_1 and KUKA_2. While there is little difference in accuracy between LGR and LWPR, LGR consistently uses fewer local models and does not require careful manual meta-parameter tuning. Since both LGR and LWPR use more local models on this data (compared to the Sarcos data) we also tried increasing the feature space of I-SSGPR to 400 features. This did not improve I-SSGPRs performance on the online data (see Table 3). Finally, it is noteworthy that LGR processes both of these data sets at ~ 500Hz (C++ code, on a 3.4GHz Intel Core i7), making it a realistic alternative for real-time inverse dynamics learning tasks.

KUKA_{sim} : Finally, we evaluate LGRs ability to learn from scratch on KUKA_{sim}, a large data set of 2 million simulated data points, collected using [30]. We randomly drew 1% points as a test set, on which we evaluate convergence during online training. Figure 2 (left) shows convergence and number of local models used, averaged over 5 randomly seeded runs for joint 1. After the first $1e5$ data points, both LWPR and LGR achieve a normalized mean squared error below 0.07, and eventually converge to a nMSE of ~ 0.01. LGR converges slightly faster, while using fewer local models (Figure 2, right).

6 Conclusion

We proposed a top-down approach to probabilistic localized regression. Local Gaussian Regression decouples inference over M local models, resulting in efficient and principled updates for all parameters, including local distance metrics. These localized updates can be used in batch as well as incrementally, yielding computationally efficient learning in either case and applicability to big data sets. Evaluated on a variety of simulated and real robotic inverse dynamics tasks, and compared to I-SSGPR and LWPR, incremental LGR shows an ability to add resources (local models) and to update its distance metrics online. This is essential to consistently achieve high accuracy. Compared to LWPR, LGR matches or improves precision, while consistently using fewer resources (local models) and having significantly fewer manual tuning parameters.

References

[1] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, (1-5):75–113, 1997.

[2] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[3] Jianqing Fan and Irene Gijbels. Data-driven bandwidth selection in local polynomial fitting: variable bandwidth and spatial adaptation. *Journal of the Royal Statistical Society.*, pages 371–394, 1995.

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

[5] Joaquin Quiñonero Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *JMLR*, 6:1939–1959, 2005.

[6] Krzysztof Chalupka, Christopher KI Williams, and Iain Murray. A framework for evaluating approximation methods for Gaussian process regression. *JMLR*, 14(1):333–350, 2013.

[7] Michalis K Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 567–574, 2009.

[8] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257, 2006.

[9] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

[10] Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum Gaussian process regression. *JMLR*, 11:1865–1881, 2010.

[11] Marco F Huber. Recursive Gaussian process: On-line regression and learning. *Pattern Recognition Letters*, 45:85–91, 2014.

[12] Lehel Csató and Manfred Opper. Sparse on-line Gaussian processes. *Neural computation*, 2002.

[13] Arjan Gijsberts and Giorgio Metta. Real-time model learning using incremental sparse spectrum Gaussian process regression. *Neural Networks*, 41:59–69, 2013.

[14] James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *UAI*, 2013.

[15] Jo-Anne Ting, Mrinal Kalakrishnan, Sethu Vijayakumar, and Stefan Schaal. Bayesian kernel shaping for learning control. *Advances in neural information processing systems*, 6:7, 2008.

[16] Duy Nguyen-Tuong, Jan R Peters, and Matthias Seeger. Local Gaussian process regression for real time online model learning. In *Advances in Neural Information Processing Systems*, pages 1193–1200, 2008.

[17] Edward Snelson and Zoubin Ghahramani. Local and global sparse Gaussian process approximations. In *International Conference on Artificial Intelligence and Statistics*, pages 524–531, 2007.

[18] Trevor Hastie and Clive Loader. Local regression: Automatic kernel carpentry. *Statistical Science*, 1993.

[19] J. Moody and C. Darken. Learning with localized receptive fields. In *Proceedings of the 1988 Connectionist Summer School*, pages 133–143. San Mateo, CA, 1988.

[20] Radford M Neal. *Bayesian Learning for Neural Network*, volume 118. Springer, 1996.

[21] Michael E Tipping. Sparse Bayesian learning and the relevance vector machine. *The Journal of Machine Learning Research*, 1:211–244, 2001.

[22] Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal. The Bayesian backfitting relevance vector machine. In *ICML*, 2004.

[23] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 2008.

[24] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *J. Mach. Learn. Res.*, 14(1):1303–1347, May 2013.

[25] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael Jordan. Streaming variational Bayes. In *Advances in Neural Information Processing Systems*, pages 1727–1735, 2013.

[26] Jan Luts, Tamara Broderick, and Matt Wand. Real-time semiparametric regression. *arxiv*, 2013.

[27] Antti Honkela and Harri Valpola. On-line variational Bayesian learning. In *4th International Symposium on Independent Component Analysis and Blind Signal Separation*, pages 803–808, 2003.

[28] Franziska Meier, Philipp Hennig, and Stefan Schaal. Efficient Bayesian local model learning for control. In *Proceedings of the IEEE International Conference on Intelligent Robotics Systems (IROS)*, 2014.

[29] Joaquin Quiñonero-Candela and Ole Winther. Incremental Gaussian processes. In *NIPS*, 2002.

[30] Stefan Schaal. The SL simulation and real-time control software package. Technical report, 2009.

[31] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: Incremental real time learning in high dimensional space. In *ICML*, pages 1079–1086, 2000.