Gaussian Process Regression in Latent Space

Anonymous Author(s)
Affiliation
Address
email

Abstract

Training the hyperparameters of Gaussian process regression (GPR) is expensive. However, even after the initial training, data analysts often need to repeatedly adjust the hyperparameters until the regression curve achieves a desirable smoothness. These subsequents adjustments are quite slow, each requiring $O(n^3)$ operations given n data points. We propose an alternative model construction for GPR that allows for much faster adjustments of the smoothness. Our construction maintains orthogonal non-linear regressors, each associated with a different smoothness level. Those orthogonal regressors can be combined almost instantly according to the requested smoothness level. Our empirical study shows that for varying levels of requested smoothness, the quality of our model's regression curve is nearly identical to those generated by standard GPR. Our approach is also amenable to sparse approximations; thus, it can scale to datasets with millions of data points.

1 Introduction

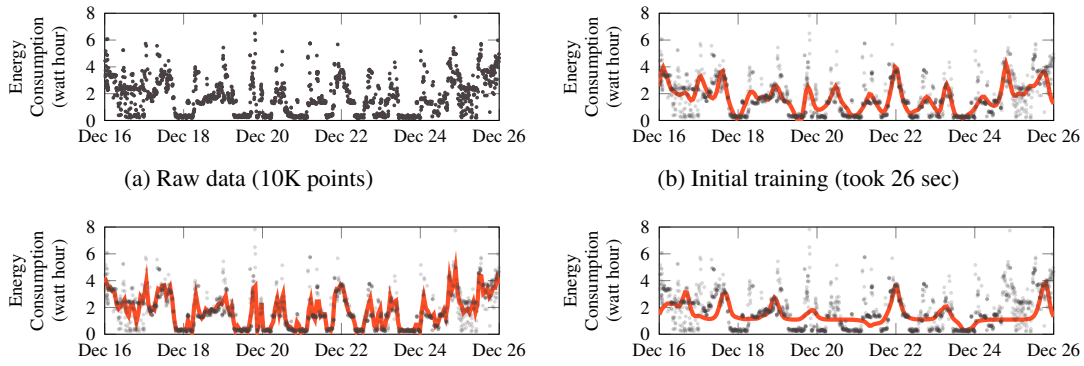
Gaussian process regression (GPR) is a popular non-linear method for probabilistic inference [14]. Unlike linear regression, GPR does not assume any parametric functional forms (e.g., linear, quadratic, etc.), which allows GPR to extract an underlying non-linear pattern (i.e., regression curve) for a dataset drawn from an arbitrary distribution. However, the smoothness of the underlying pattern must be determined using its hyperparameters.[1]

Despite its expressiveness, GPR has a few drawbacks that have precluded it from becoming a user-friendly, real-time data exploration tool. GPR's typical workflow is as follows. When the analyst initially invokes a training algorithm, it first finds the optimal hyperparameters (a.k.a. *hyperparameter tuning*) according to some goodness criteria (e.g., marginal likelihood, variational free energy [2]), and then trains a *single* model based on those hyperparameters. The model with its optimal hyperparameters is expected to yield the lowest error on new (unseen) data. This *initial training* process takes $O(t\,n^3)$ time, where n is the number of data points and t is the number of iterations the algorithm performs to find the optimal hyperparameters. The analyst then decides whether the regression curve overfits (or underfits) the data, and if so, adjusts the hyperparameters manually and invokes the training algorithm again to obtain a new regression curve based on those manually specified hyperparameters. The analyst may continue to repeatedly adjust and retry until s/he finds the most desired model. Each of these *model adjustments* takes $O(n^3)$. As a result, GPR is not an ideal candidate for data warehouse dashboards and business intelligent (BI) tools, where users expect fast (i.e., real-time) exploration of data with different model granularities.

In this paper, we propose a data-analyst-friendly extension of GPR, which we call *Gaussian Process regression in latent space* (LATENTGP). The key advantage of LATENTGP is its ability to simultaneously train multiple submodels with different smoothness levels, rather than training a single model.

[1] This is true for the GPR with a squared exponential covariance function, a Matérn covariance function, or a γ-exponential covariance function, which we focus on in this paper.

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.



(a) Raw data (10K points)

(b) Initial training (took 26 sec)

(c) A more data-sensitive curve (took 0.10 sec)

(d) A smoother curve (took 0.10 sec)

Figure 1: Unlike GPR, after the initial training, the subsequent adjustments of the regression model (e.g., to avoid overfitting/underfitting) can be performed almost immediately.

LATENTGP can then simply sum up those submodels to produce models with different smoothness levels.

Figure 1 illustrates an example of the user interaction with LATENTGP. For the raw data depicted in Figure 1a, LATENTGP generates an initial regression curve shown in Figure 1b. Similar to GPR, the initial training of LATENTGP takes relatively long (i.e., 26 sec). After seeing the initial curve, the analyst may request a more data-sensitive curve or a smoother one. LATENTGP can generate each subsequent model almost immediately (Figure 1c and Figure 1d). The analyst can continue these adjustments until a desirable curve is obtained, without having to incur a significant cost for each adjustment.

Summary of our approach LATENTGP decomposes GPR's model into multiple components (i.e., submodels), each associated with a different smoothness level. According to the representer theorem [15], GPR's model $f(\boldsymbol{x})$ corresponds to a linear model in which each regressor is defined using the kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}_i)$ (or equivalently, the covariance function) that involves a different training data point $\boldsymbol{x}_i$. LATENTGP expresses this model using an alternative set of regressors $u_i(\boldsymbol{x})$, each with a different smoothness:

$$\underbrace{f(\boldsymbol{x}) = \sum_{i=1}^{n} w_i \, \kappa(\boldsymbol{x}, \boldsymbol{x}_i)}_{\text{GPR model}} \quad \Longleftrightarrow \quad \underbrace{f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \, u_i(\boldsymbol{x})}_{\text{LATENTGP model}} \tag{1}$$

where $\boldsymbol{x}_i$ for $i = 1, \ldots, n$ are the training data points, w_i for $i = 1, \ldots, n$ are the weighting parameters whose values are determined during GPR training, and α_i for $i = 1, \ldots, n$ are the weighting parameters whose values are determined during LATENTGP training.[2]

By combining k number of regressors (i.e., $u_1(\boldsymbol{x}), \ldots, u_k(\boldsymbol{x})$), we obtain the k-th submodel of LATENTGP. The submodel associated with a larger k (or lower smoothness level) is more sensitive to the training data points. For example, the regression curve in Figure 1c is associated with a larger value of k than the regression curve in Figure 1d. While there are different approaches to obtaining such regressors, LATENTGP uses the following approach: it *overloads* the expression on the left side of Equation (1) with the kernel functions of different hyperparameters; then, orthogonal functions (which we call *latent regressors*) are extracted from these overloaded regressors. This allows the linear combination of these latent regressors to be able to express both smooth and data-sensitive patterns. Although this construction is different from standard GPR in which only a single type of kernel (i.e., with the same hyperparameters) is used, LATENTGP is still equivalent to a GPR model with a redefined kernel.

[2]These weighting parameters are random variables with certain posterior distributions (see Section 2).

2 Background: Gaussian process regression

Given a training set D that consists of n pairs of a zero-meaned input vector and an observation, i.e., $\{(\boldsymbol{x}_i, y_i) \mid i = 1, \ldots, n\}$, Gaussian process regression (GPR) computes a posterior distribution based on the assumption that those n observations are noisy realizations of an underlying Gaussian process (GP) $\boldsymbol{f}$. That is, $y_i = f_i + \varepsilon_i$ where f_i is the output of the GP for $\boldsymbol{x}_i$, and ε_i is a zero-mean Gaussian noise with variance σ_n^2. The outputs of the GP, i.e., $(f_1, \ldots, f_n)$, jointly follow a multivariate normal distribution: $(f_1, \ldots, f_n) \sim \mathcal{N}(\mathbf{0},\, K_{n,n})$, where $K_{n,n}$ is a covariance matrix; the (i, j)-entry is the covariance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Typically, the covariance matrix is computed using a *kernel* function $\kappa(\,\cdot\,,\,\cdot\,)$ that returns the covariance between two data points.

Let $\boldsymbol{x}_*$ denote an unseen data point, and let f_* be the unknown function output at $\boldsymbol{x}_*$. Also, let S be a size-m subset of D. The data points in S are typically called *inducing variables*. Then, the posterior (or, the predictive) distribution of f_* is expressed as:

$$\mathcal{N}\left(\boldsymbol{k}_*^\top\, A^{-1}\, K_{n,m}^\top\, \boldsymbol{y},\quad \boldsymbol{k}_*^\top\, A^{-1}\, \boldsymbol{k}_*\right) \tag{2}$$

where $A = K_{n,m}^\top\, K_{n,m} + \sigma_n^2 K_{m,m}$. $K_{n,m}$ is the n-by-m covariance matrix between the data points in D and the data points in S, $K_{m,m}$ is the m-by-m covariance matrix of the data points in S, and $\boldsymbol{k}_*$ is the size-m column vector whose i-th element is the covariance between x_* and the i-th element of S. If $m = n$, we call the above model *full Gaussian process regression* (or `FullGP`). If $m < n$, we call the above model *sparse Gaussian process regression* (or `SparseGP`). In the literature, this particular sparse GPR is referred to as the *subset of regressors* [12].

Observe that the mean of Equation (2) can be viewed as a linear combination of the elements of $\boldsymbol{k}_*$. Indeed, the posterior distribution of GPR is equivalent to solving the following Bayesian linear regression with random variables weights $\boldsymbol{w} = (w_1, \ldots, w_m)$:

$$f(\boldsymbol{x}) = w_1\, \kappa(\boldsymbol{x}, \boldsymbol{x}_i) + \cdots + w_m\, \kappa(\boldsymbol{x}, \boldsymbol{x}_m) \tag{3}$$

where $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, K_{m,m}^{-1})$ [14]. That is, the posterior of $\boldsymbol{w}$ is $\mathcal{N}(A^{-1}\, K_{n,m}^\top\, \boldsymbol{y},\ A^{-1})$.

3 Gaussian process regression in latent space

In this section, we first construct LATENTGP's model, which consists of orthogonal non-linear regressors. Those orthogonal regressors are called *latent regressors*. Since these latent regressors are associated with varying smoothness levels, their linear combinations accordingly produce the models with different smoothness levels. Those individual models are called *submodels* to distinguish them from the LATENTGP itself. Then, we describe how to train those submodels and how to make inference using them.

3.1 Latent regressors from standard Gaussian process regression

We describe how to construct an alternative function f^u containing orthogonal regressors (i.e., latent regressors) while retaining the same expressiveness as the one constructed using the kernels with m inducing variables (i.e., Equation (2)). That is,

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} w_i\, \kappa(\boldsymbol{x}, \boldsymbol{x}_i) \qquad \overset{\text{expressivness}}{\Longleftrightarrow} \qquad f^u(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i\, u_i(\boldsymbol{x}) \tag{4}$$

where $u_i(\boldsymbol{x})$ for $i = 1, \ldots, m$ are the latent regressors, and $\alpha_1, \ldots, \alpha_m$ are their weights; $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$ jointly follows a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \Sigma_\alpha)$. We will discuss how to set Σ_α, shortly. The orthogonality of $u_i(\boldsymbol{x})$, for $i = 1, \ldots, m$, means

$$\int u_i(\boldsymbol{x})\, u_j(\boldsymbol{x})\, p(\boldsymbol{x})\, d\boldsymbol{x} = 0 \qquad \text{for } i \neq j.$$

where $p(\boldsymbol{x})$ is the probability density function of data points.

Like Nyström approximation for kernel matrices [24], we approximately obtain those m latent regressors using q ($\geq m$) number of randomly chosen data points, $\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_q$ from D. That is, let $K_{q,m}$ be the covariance matrix between those q data points and m inducing variables, and let $\boldsymbol{k}(\boldsymbol{x})$

be $(\kappa(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}, \boldsymbol{x}_m))^\top$. Then, the following operation produces the values of those m latent regressors:

$$\boldsymbol{u}(\boldsymbol{x}) = (u_1(\boldsymbol{x}), \ldots, u_m(\boldsymbol{x})) = \boldsymbol{k}(\boldsymbol{x})^\top V \Lambda^{-1/2} = \boldsymbol{k}(\boldsymbol{x})^\top L \tag{5}$$

where $L \coloneqq V\,\Lambda^{-1/2}$, V is a q-by-m matrix containing the right singular vectors of $K_{q,m}$ in its columns, and Λ is a m-by-m diagonal matrix containing corresponding singular values in as its diagonal entires. Since $\boldsymbol{k}(\boldsymbol{x})^\top L$ is equivalent to $U\,\Lambda^{1/2}$, where U is a q-by-m matrix containing left singular vectors of $K_{q,m}$ in its columns, $u_i(\boldsymbol{x})$ and $u_j(\boldsymbol{x})$ in Equation (5) are orthogonal one another for $i \neq j$ with respect to the q data points. That is, since Λ is a diagonal matrix,

$$\sum_{k=1}^{q} u_i(\boldsymbol{x}'_k)\, u_j(\boldsymbol{x}'_k) = (U\,\Lambda^{1/2})^\top (U\,\Lambda^{1/2}) = \Lambda_{i,j} = 0 \quad \text{for } i \neq j. \tag{6}$$

There are no parametric expressions for $u_1(\boldsymbol{x}), \ldots, u_m(\boldsymbol{x})$; thus, their values must be obtained by first computing $\boldsymbol{k}(\boldsymbol{x})$, and then applying L. Note that in the traditional setting where all kernels in $\kappa(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}, \boldsymbol{x}_m)$ are associated with the same hyperparameters, $U = V$ when $q = m$. However, we explicitly distinguish U and V because we will introduce kernels with different hyperparameters and will extract latent regressors from those heterogeneous kernels in Section 3.2.

The following theorem shows how to set Σ_α so that the posterior distribution of $f(\boldsymbol{x}_*)$ and $f_u(\boldsymbol{x}_*)$ in Equation (4) are identical.

Theorem 1. *Let $u(\boldsymbol{x}) = k(\boldsymbol{x})^\top L$ be an alternative feature vector using an invertible linear mapping L, i.e., $L^{-1}\,L = L\,L^{-1} = I$. Also, let $\Sigma_\alpha = L^{-1}\,\Sigma_w\,(L^\top)^{-1}$. Then, the following two models produce the same posterior distribution for an unseen data point $\boldsymbol{x}_*$:*

$$f_1(\boldsymbol{x}) = k(\boldsymbol{x})^\top \boldsymbol{w}, \qquad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_w) \tag{7a}$$

$$f_2(\boldsymbol{x}) = u(\boldsymbol{x})^\top \boldsymbol{\alpha}, \qquad \boldsymbol{\alpha} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_\alpha) \tag{7b}$$

The proof to the above theorem is in our supplementary material (Appendix A). If $q = m$, $L^{-1}\,K_{m,m}^{-1}\,(L^\top)^{-1} = I$. Thus, setting Σ_α to the identity matrix makes $f(\boldsymbol{x}_*)$ and $f^u(\boldsymbol{x}_*)$ produce the same posterior distribution for $\boldsymbol{x}_*$. Using an identity matrix for Σ_α, which is the covariance matrix of the prior distribution of $\boldsymbol{\alpha}$, is intuitive because we have ensured that those latent regressors to be orthogonal one another.

3.2 Latent regressors from overloaded regressors

Applying the approach described in Section 3, LATENTGP extracts latent regressors from the linear model with heterogeneous kernels. We call such a model *overloaded regressors*. Let $\kappa(\,\cdot\,, \cdot\,; \boldsymbol{h})$ indicate the kernel with hyperparameters $\boldsymbol{h}$. Then, the overloaded regressors is defined, using the m inducing variables, as follows:

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} \sum_{j=1}^{\ell} w_{i,j}\, \kappa(\boldsymbol{x}, \boldsymbol{x}_i; \boldsymbol{h}_j) \tag{8}$$

where ℓ determines how many kernels are placed for each inducing variable. The above model construction permits any combinations of $\boldsymbol{h}_j$ for $j = 1, \ldots, \ell$. However, LATENTGP uses one specific approach described as follows. Let $\boldsymbol{h}_0$ denote the optimal (length-scale) hyperparameters obtained for `SparseGP` or `FullGP`. Then, LATENTGP sets $\boldsymbol{h}_j = \boldsymbol{h}_0 \times 2^{j-1}$. The kernels with larger j include larger hyperparameters; thus, they can capture smoother patterns in a dataset. Also, although it is non-trivial to directly find the posterior distribution of $\boldsymbol{w} = (w_{1,1}, \ldots, w_{m,\ell})$ due to the kernels with different hyperparameters,[3] we can still extract latent regressors and alternatively train the weights of those latent regressors.

Extracting latent regressors from overloaded regressors follows the similar procedure as in Section 3. That is, a covariance matrix $K_{q,m\ell}$ is constructed by computing the kernels between q randomly-chosen data points and m inducing variables. The number of columns is now $m\ell$ since ℓ different

[3]The kernels no longer serve as covariance functions between pairs of data points.

kernels are evaluated for each inducing variable. The orders of columns do not matter. Then, the values of latent regressors, $u_1(\boldsymbol{x}), \ldots, u_m(\boldsymbol{x})$, are computed as follows:

$$\boldsymbol{u}(\boldsymbol{x}) = (u_1(\boldsymbol{x}), \ldots, u_m(\boldsymbol{x})) = \boldsymbol{k}_\ell(\boldsymbol{x})^\top V_\ell \, \Lambda_\ell^{-1/2} = \boldsymbol{k}_\ell(\boldsymbol{x})^\top L_\ell \tag{9}$$

where $L_\ell \coloneqq V_\ell \, \Lambda_\ell^{-1/2}$, V_ℓ and Λ_ℓ are matrices for the right-singular vectors and the singular values of $K_{q,m\ell}$, respectively, and $\boldsymbol{k}_\ell(\boldsymbol{x})$ is the output of the overloaded regressors for $\boldsymbol{x}$. Those latent regressors are again orthogonal one another with respect to those q randomly-chosen data points (as in Equation (6)) due to the property of the singular value decomposition.

3.3 Submodels

Let the diagonal elements of Λ_ℓ and the columns of V_ℓ be sorted in the descending order of singular values. Then, due to the property of the singular value decomposition, the latent regressors associated with lower index numbers (e.g., $u_1(\boldsymbol{x})$) tend to capture smoother *orthogonal* patterns in a dataset in comparison to the latent regressors with higher index numbers (e.g., $u_{1000}(\boldsymbol{x})$). This is also the basic idea behind latent semantic analysis in natural language processing [4]. Exploiting this property, LATENTGP defines m submodels, $f_1^u(\boldsymbol{x}), \ldots, f_m^u(\boldsymbol{x})$ as follows:

$$f_k^u(\boldsymbol{x}) = \sum_{j=1}^{k} \alpha_j \, u_j(\boldsymbol{x}) \quad \text{for } k = 1, \ldots, m \tag{10}$$

The submodels with lower index numbers (e.g., $f_1^u(\boldsymbol{x})$) reveal smoother patterns in comparison to the submodels with higher index numbers (e.g., $f_{1000}^u(\boldsymbol{x})$). For convenience, $(m-k)/m \times 100\%$ is called the *smoothness level* of $f_k^u(\boldsymbol{x})$, which will be used when experiment results are reported.

3.4 Training and inference

Given the values of latent regressors (in Equation (9)), computing the posterior distribution of $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$ and the posterior distribution of $f_k^u(\boldsymbol{x}_*)$ is straightforward. That is, the posterior distribution of $\boldsymbol{\alpha}$ corresponds to the posterior distribution of Bayesian linear regression with feature matrix $K_{n,m\ell} L_\ell$ and $\boldsymbol{\alpha}$'s prior I:

$$\boldsymbol{\alpha} \mid D \sim \mathcal{N}\left(A^{-1} \left(K_{n,m\ell} \, L_\ell\right)^\top \boldsymbol{y}, \, A^{-1}\right), \quad A = \left(K_{n,m\ell} \, L_\ell\right)^\top \left(K_{n,m\ell} \, L_\ell\right) + \sigma_n^2 I \tag{11}$$

Also, the posterior distribution of $f_k(\boldsymbol{x}_*)$ is expressed as:

$$f_k^u(\boldsymbol{x}_*) \mid D \sim \mathcal{N}\left(\boldsymbol{k}_*^\top L_\ell \, I_{k/m} \, A^{-1} \left(K_{n,m\ell} \, L_\ell\right)^\top \boldsymbol{y}, \quad \boldsymbol{k}_*^\top L_\ell \, I_{k/m} \, A^{-1} \, I_{k/m} \left(\boldsymbol{k}_*^\top L_\ell\right)^\top\right) \tag{12}$$

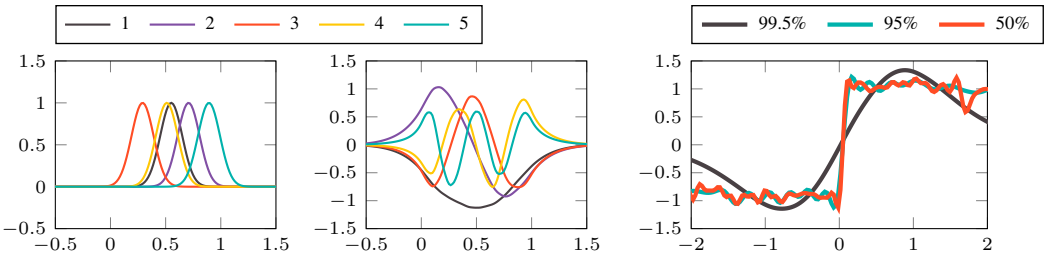
where $I_{k/m}$ is an $m \times m$ matrix such that the first k rows and columns are from an identity matrix and all the other elements are zeros.

Connection to GRP LATENTGP from overloaded regressors could be regarded as GPR with redefined covariance matrix $\hat{K}_{n,n}$ between training data points as follows. First, suppose $m = n$, and let the singular value decomposition of $K_{n,n\ell}$ be $\hat{U}\hat{\Lambda}\hat{V}^\top$. Then, $\hat{K}_{n,n} \coloneqq \hat{U}\hat{\Lambda}\hat{U}^\top$. The k-th submodel of LATENTGP uses the first k columns of $\hat{U}$ and corresponding diagonal elements in $\hat{\Lambda}$. If $m < n$, LATENTGP corresponds to the sparse approximation of $\hat{K}_{n,n}$.

3.5 Time complexity analysis

We analyze the computational costs of LATENTGP. For training, LATENTGP performs the singular value decomposition of overloaded regressors. This process takes $O(m^2 \, \ell)$ for computing the values of overloaded regressors, and $O(m \, (m \, \ell)^2) = O(m^3 \, \ell^2)$ for the singular value decomposition. Next, LATENTGP computes the kernel values of overloaded regressors for all n training data points. This takes $O(n \, m \, \ell)$ for computing kernel values and $O(n \, m^2 \, \ell)$ for computing the values of latent regressors. Finally, computing A^{-1} takes $O(m^2 \, n)$ for matrix multiplications and $O(m^3)$ for matrix inversion. Since $m \leq n$, the total time complexity for training is $O(m^3 \, \ell^2 + n \, m^2 \, \ell)$.

For inference, suppose there are t data points for which we need to compute the posterior distributions. Computing the values of overloaded regressors takes $O(t \, m \, \ell)$, which is converted to the values of latent regressions at the cost of $O(t \, m^2 \, \ell)$. Since A^{-1} has been computed in the training stage, computing the mean of the posterior distribution takes $O(t \, m)$. Computing the variance takes $O(t \, m^3)$. Thus, the total cost for inference is $O(t \, m^2 \, \ell + t \, m^3)$.



(a) Original Regressors (b) Latent Regressors (c) Submodels

Figure 2: Numerical analysis of latent regressors and submodels. (a) Five randomly-chosen original regressors, and (b) the first five latent regressors. For (a) and (b), the numbers in the legend indicate the orders of the regressors. A latent regressor with a lower order number is associated with a higher smoothness level. In (c), three submodels fitted to the unit step function are shown. Their smoothness levels are in the legend.

4 Numerical evaluations

In this section, we numerically analyze LATENTGP's characteristics and evaluate its performance against the full Gaussian process regression (`FullGP`) and the sparse Gaussian process regression (`SparseGP`). First, the characteristics of LATENTGP's latent regressors are studied in Section 4.1. Second, Section 4.2 compares the latencies of LATENTGP against `SparseGP` for each of three data analysts stages: (1) model training, (2) initial inference, and (3) smoothness adjustment. Third, Section 4.3 studies the errors of LATENTGP for various datasets and for various smoothness levels in comparison to `FullGP` and `SparseGP`. All methods were implemented and run in Matlab 2017b. All experiments were conducted on an 112-core (each 2.2 GHz) machine with 1,000 GB main memory.

4.1 Characteristics of latent regressors

To understand the characteristics of latent regressors, we visually analyze both original regressors and latent regressors. For this, we generated 10,000 uniform-random data points between 0 and 1. Each original regressor was defined using the squared exponential kernel with the same length-scale parameter. Figure 2a shows five randomly chosen original regressors. Since their length-scale parameters were identical, their shapes are basically identical. In Figure 2b, we visualize the first five latent regressors extracted from those original regressors. The first latent regressor (in black) is the smoothest function. As the order number associated with those latent regressors increases (from 1 to 5), they vary more quickly, indicating their higher data-sensitivity. Naturally, LATENTGP's submodel with a few low-ordered latent regressors only captures a smooth pattern, while its data sensitivity increases as more latent regressors are incorporated for training and inference.

Figure 2c visualizes the submodels associated with different smoothness levels. In this figure, LATENTGP's model was fit to the unit step function, and three submodels are depicted. Observe that the submodel with 99.5% smoothes out the abrupt change at 0, which implies its resilience against potential noisy fluctuations in raw data. As submodels' smoothness levels decrease (e.g., 95%, 50%), they now capture the abrupt change at 0. Since LATENTGP's smoothness level adjustments are almost instant (Section 4.2), the user can interactively generate various submodels and choose a desirable one according to his/her needs.

4.2 Runtime analysis

In this section, we compare the latencies of LATENTGP against `SparseGP`. `FullGP` was also tested; however, its results are not reported since its training failed for all real-life datasets tested (either training did not finish within fours hours or out-of-memory errors occurred). The latencies are measured for each of the three data exploration steps: (1) model training, (2) initial inference, and (3) smoothness adjustment. The model training includes a hyperparameter optimization process. For finding the optimal hyperparameters, we used the state-of-the-art sparse GP optimization algorithm [2]. LATENTGP used the same hyperparameters. Our experiments used the following five datasets (two synthetic and three real-life):

Table 1: Runtime comparison for each of the three data exploration steps: (1) training with hyperparameter tuning, (2) initial inference, and (3) smoothness adjustment.

	Training (sec)		Initial inference (sec)		Smoothness adjustment (sec)	
Dataset	SparseGP	LatentGP	SparseGP	LatentGP	SparseGP	LatentGP
step	1.68	2.04	0.0205	0.1372	0.0308	0.0019
gaussian	18.3	19.4	0.0124	0.2831	0.0650	0.0034
kin40K	2527.3	2539.5	0.0306	2.2730	1.0017	0.0300
flight	193.4	195.1	0.0083	0.1934	0.1069	0.0026
combustion	29587.6	29836.0	0.0267	2.9733	17.879	0.0207

1. step: This is a one-dimensional unit-step function. This dataset is typically used to examine how well a regression model behaves with the occurrence of abrupt changes. 600 points were used for training, and 1,800 points were used for testing.
2. gaussian: This is a mixture of five two-dimensional normal distributions (with different values of covariance matrices). We used this as an example of a dataset that is comprised of multiple subgroups with different characteristics. 1,000 points were used for training, and 3,000 points were used for testing.
3. kin40K: This is a medium-sized (40K points) multivariate dataset commonly used for testing GPR (e.g., [2]). The dataset contains forward kinematics of a robot arm. 30K points were used for training and 10K points were used for testing.
4. flight: This is another medium-sized (10K points) multivariate dataset commonly used for testing GPR (e.g., [9]). The dataset contains US flight delays. 7,900 points were used for training, and 1,100 points were used for testing.
5. combustion: This is a large-scale dataset (about 200K data points) obtained from a turbulent combustion experiment [25]. Input variables are physical properties and the output variable is the speed of air flux. 200K points were used for training, and 3,000 points were used for testing.

All datasets were normalized to set their means equal to zero and their standard deviations equal to one. For both SparseGP and LatentGP, the number of inducing variables was set to $\min(n, 1000)$, where n is the number of data points in the training set. For LatentGP, the value of q (i.e., the number of random data points used for extracting latent regressors) was set equal to m, and ℓ (i.e., the number of kernels placed on each inducing variable) was set to 10.

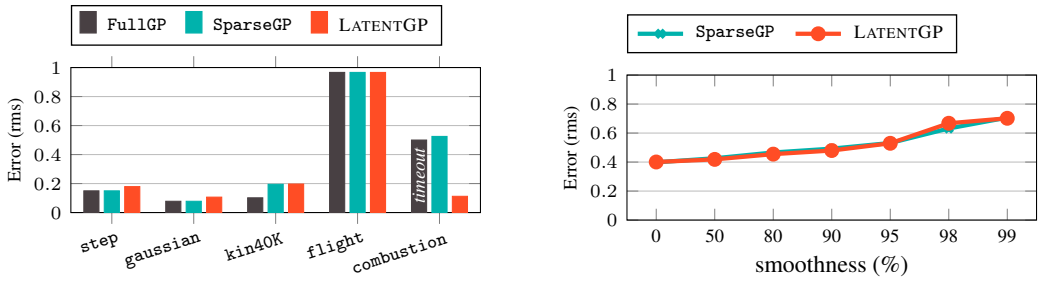
Table 1 summarizes the results. In the training step, the hyperparameter optimization made up a significant portion of the runtime for both LatentGP and SparseGP. When the size of the datasets were larger than 10K (i.e., kin40K and combustion), the training took more than 40 mins and 8 hours, respectively. However, the hyperparameter optimization is performed using some predetermined criteria (e.g., marginal likelihood, cross validation, etc.); thus, immediate user responses are less needed.

For the other two steps (i.e., initial inference and smoothness adjustment), more interactive responses are needed. The reported latencies are the inference times for all test data points (e.g., 10K points for kin40K). LatentGP's initial inference was slower than SparseGP due to its extra kernel evaluations (i.e., $m\ell$ instead of m). However, once the initial inference was finished, LatentGP could generate subsequent regression curves almost immediately for all datasets. For a large dataset (i.e., combustion), SparseGP took about 17 seconds for every smoothness adjustment, while LatentGP took only 0.02 seconds (i.e., 860$\times$ faster). In the following section, we study the quality of those regression curves with different smoothness levels.

4.3 Error analysis

In this section, we study the quality of LatentGP's inference. First, we evaluate LatentGP's quality when all its latent regressors are used. Second, we evaluate the quality of LatentGP's submodels (associated with different smoothness levels) against SparseGP (also associated with corresponding smoothness levels).

First, we analyze LatentGP's inference quality when all of its latent regressors are used. We used the same five datasets described in the previous section. All the same hyperparameter values were



(a) Different datasets (b) Different smoothness

Figure 3: The quality of LatentGP's inference for different datasets and smoothness levels. The quality was measured using the root-mean-square error against the groudtruth.

used for FullGP, SparseGP, and LatentGP. The root-mean-square errors were used for evaluating the quality of inferences. Figure 3a shows the results. In general, FullGP's errors were slightly lower than others. In all cases, the errors of SparseGP and LatentGP were comparable.

Second, we analyze the quality of LatentGP's submodels associated with different smoothness levels. Note that as the smoothness level increases, the error is expected to increase as well (since the regression curve is less data-sensitive). However, it would be desirable to minimize those error increments for a certain smoothness level. Note that LatentGP's smoothness level does not directly correspond to a SparseGP's model associated with certain hyperparameters. However, for this evaluation, we could still generate the SparseGP's model associated with a similar smoothness level by manually inspecting many different SparseGP models. Find those curves in our supplementary material (Appendix B). Figure 3b reports the errors of two models (by LatentGP and SparseGP) associated with similar smoothness levels. Their errors were almost identical when the associated smoothness levels were similar. This indicates that the quality of LatentGP's submodels are quantitatively (almost) equivalent to the individually trained SparseGP's models.

5 Related work

Due to the prohibitive nature of GPR, there has been significant research on sparse approximations of GPR [1], including well known methods such as Subset of Data, Subset of Regressors [17, 18, 22], Deterministic Training Conditional [3, 16], Fully Independent Training Conditional [19], and Nyström method [10]. The relationship between these different approaches has been studied in [12]. LatentGP is different from these methods in its model construction and inference mechanism (which uses latent regressors). LatentGP's approach enables interactive-speed smoothness adjustments for GPR, which has not been pursued by previous work.

LatentGP's model construction involves regressors (or equivalently, kernels) with different hyperparameters. This idea has been used for a different purpose to better represent data whose properties differ in its sub-groups. For example, non-stationary kernels [11] assume that there exists a smooth function that returns a hyperparameter at every location. Walder et al. also study GPR in a similar setting [23]. These methods aim at a better modeling of heterogeneous datasets, a goal pursued by many others as well [5, 7, 13, 20]. Lastly, enabling efficient inference for new Gaussian process models is also a popular topic in the literature [6, 8, 21].

6 Conclusion

This work has proposed Gaussian process regression in latent space (LatentGP), which enables real-time smoothness adjustments of regression curves. Rather than requiring a hyperparameter update and model retraining, LatentGP adjusts smoothness levels by combining a different set of *latent regressors*. Our experiments showed that the quality of LatentGP's regression curves is comparable to the regression curves obtained by manually adjusting the hyperparameters of standard Gaussian process regression. Despite LatentGP's slightly longer initial training, its ability to adjust regression curves almost instantly could make it an appealing choice for modern machine learning-enabled data analytics systems.

References

[1] T. D. Bui, C. Nguyen, and R. E. Turner. Streaming sparse gaussian process approximations. In *Advances in Neural Information Processing Systems*, pages 3301–3309, 2017.

[2] Y. Cao, M. A. Brubaker, D. J. Fleet, and A. Hertzmann. Efficient optimization for sparse gaussian process regression. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2013.

[3] L. Csató and M. Opper. Sparse on-line gaussian processes. *Neural computation*, 2002.

[4] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 1990.

[5] D. Durante, B. Scarpa, and D. B. Dunson. Locally adaptive bayesian multivariate time series. In *Advances in Neural Information Processing Systems*, pages 1664–1672, 2013.

[6] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen. Additive gaussian processes. In *Advances in neural information processing systems*, pages 226–234, 2011.

[7] E. Fox and D. B. Dunson. Multiresolution gaussian processes. In *Advances in Neural Information Processing Systems*, pages 737–745, 2012.

[8] R. Frigola, Y. Chen, and C. E. Rasmussen. Variational gaussian process state-space models. In *Advances in Neural Information Processing Systems*, pages 3680–3688, 2014.

[9] J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, page 282, 2013.

[10] S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the nyström method. *Journal of Machine Learning Research*, 13:981–1006, 2012.

[11] C. J. Paciorek and M. J. Schervish. Nonstationary covariance functions for gaussian process regression. In *Advances in neural information processing systems*, pages 273–280, 2004.

[12] J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

[13] C. E. Rasmussen. The infinite gaussian mixture model. In *Advances in neural information processing systems*, pages 554–560, 2000.

[14] C. E. Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*. 2004.

[15] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

[16] M. Seeger, C. Williams, and N. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *Artificial Intelligence and Statistics 9*, 2003.

[17] B. W. Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1985.

[18] A. J. Smola and P. L. Bartlett. Sparse greedy gaussian process regression. In *Advances in neural information processing systems*, pages 619–625, 2001.

[19] E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

[20] S. Sun and X. Xu. Variational inference for infinite mixtures of gaussian processes with applications to traffic flow prediction. *IEEE Transactions on Intelligent Transportation Systems*, 2011.

[21] F. Tobar, T. D. Bui, and R. E. Turner. Learning stationary time series using gaussian processes with nonparametric kernels. In *Advances in Neural Information Processing Systems*, pages 3501–3509, 2015.

[22] G. Wahba, X. Lin, F. Gao, D. Xiang, R. Klein, and B. Klein. The bias-variance tradeoff and the randomized gacv. In *Advances in Neural Information Processing Systems*, pages 620–626, 1999.

[23] C. Walder, K. I. Kim, and B. Schölkopf. Sparse multiscale gaussian process regression. In *Proceedings of the 25th international conference on Machine learning*, pages 1112–1119. ACM, 2008.

[24] C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682–688, 2001.

[25] Z. Zhang, K. Duraisamy, and N. A. Gumerov. Efficient multiscale gaussian process regression using hierarchical clustering. *arXiv preprint arXiv:1511.02258*, 2015.

Supplementary Materials to "Gaussian Process Regression in Latent Space"

A Proof to a theorem

Theorem 1. *Let $u(\boldsymbol{x}) = L^\top k(\boldsymbol{x})$ be an alternative feature vector using an invertible linear mapping L, i.e., $L^{-1}\,L = L\,L^{-1} = I$. Also, let $\Sigma_\alpha = L^{-1}\,\Sigma_w\,(L^\top)^{-1}$. Then, the following two GP models produce the same posterior distribution for an unseen data point $\boldsymbol{x}_*$:*

$$f_1(\boldsymbol{x}) = k(\boldsymbol{x})^\top \boldsymbol{w}, \qquad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_w) \tag{13a}$$

$$f_2(\boldsymbol{x}) = u(\boldsymbol{x})^\top \boldsymbol{\alpha}, \qquad \boldsymbol{\alpha} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_\alpha) \tag{13b}$$

Observe that the above theorem does not assume any properties of feature-generating functions. Thus, its result is applicable even if a model includes multiple hyperparameters when defining its regressors.

Proof of Theorem 1. We derive the mean and the variance of the posterior distribution of the model in Equation (13b), and show that the derived mean and variance are equivalent to the mean and variance for Equation (13a), which is given in Equation (2). Since a normal distribution is sufficiently described by its mean and variance, the equivalence of those mean and variance shows the identity of two posterior distributions.

Let $\hat{U}$ be a $n \times m$ matrix such that $\hat{U} = KL$. Recall that K contains $k(\boldsymbol{x}_i)^\top$ in its i-th row. First, we show the equivalence of means.

$$\begin{aligned}
E[f_2(\boldsymbol{x})] &= \frac{1}{\sigma_n^2} u(\boldsymbol{x}_*)^\top \left(\frac{1}{\sigma_n^2} \hat{U}^\top \hat{U} + \Sigma_\alpha^{-1} \right)^{-1} \hat{U}^\top y \\
&= \frac{1}{\sigma_n^2} k(\boldsymbol{x}_*)^\top L \left(\frac{1}{\sigma_n^2} L^\top K^\top K L + L^\top \Sigma_w^{-1} L \right)^{-1} L^\top K^\top y \\
&= \frac{1}{\sigma_n^2} k(\boldsymbol{x}_*)^\top L \left[L^\top \left(\frac{1}{\sigma_n^2} K^\top K + \Sigma_w^{-1} \right) L \right]^{-1} L^\top K^\top y \\
&= \frac{1}{\sigma_n^2} k(\boldsymbol{x}_*)^\top L\, L^{-1} \left(\frac{1}{\sigma_n^2} K^\top K + \Sigma_w^{-1} \right)^{-1} (L^\top)^{-1} L^\top K^\top y \\
&= \frac{1}{\sigma_n^2} k(\boldsymbol{x}_*)^\top \left(\frac{1}{\sigma_n^2} K^\top K + \Sigma_w^{-1} \right)^{-1} K^\top y = E[f_1(\boldsymbol{x})]
\end{aligned}$$

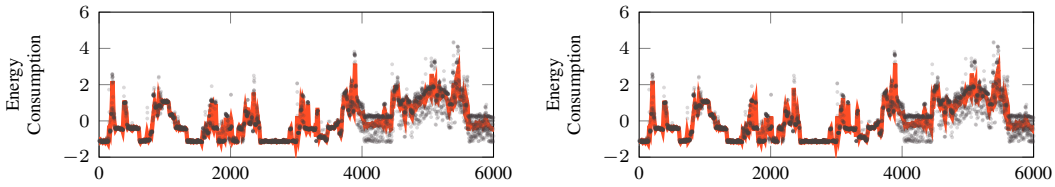
Next, we show the equivalence of variances.

$$\begin{aligned}
Var[f_2(\boldsymbol{x})] &= u(\boldsymbol{x}_*)^\top \left(\frac{1}{\sigma_n^2} \hat{U}^\top \hat{U} + \Sigma_\alpha^{-1} \right)^{-1} u(\boldsymbol{x}_*) \\
&= k(\boldsymbol{x}_*)^\top L \left(\frac{1}{\sigma_n^2} L^\top K^\top K L + L^\top \Sigma_w^{-1} L \right)^{-1} L^\top k(\boldsymbol{x}_*) \\
&= k(\boldsymbol{x}_*)^\top \left(\frac{1}{\sigma_n^2} K^\top K + \Sigma_w^{-1} \right)^{-1} k(\boldsymbol{x}_*) = Var[f_1(\boldsymbol{x})]
\end{aligned}$$

for which we used the same property of matrix inversion as used for showing the equivalence of the means. □

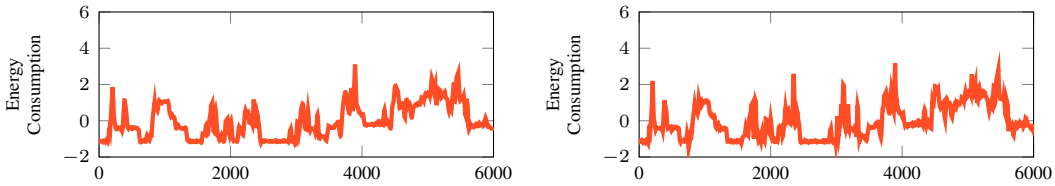
B Regression curves used in Figure 3b

We show the regression curves of `SparseGP` and LATENTGP used in our error analysis experiment (Figure 3b). For comparison, we juxtapose their regression curves side by side, with associated hyperparameter settings. The raw data used for fitting those curves are only displayed on the first figure, for visual clearance. For this experiment, we used an energy-consumption time-series data.[4]



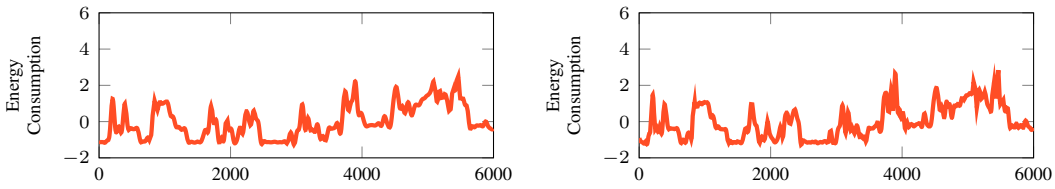
(a) `SparseGP`, $h = 13.388$ (b) LATENTGP, $k = 1000$

Figure 4: Curves with a similar smoothness level (0%).



(a) `SparseGP`, $h = 20.082$ (b) LATENTGP, $k = 500$

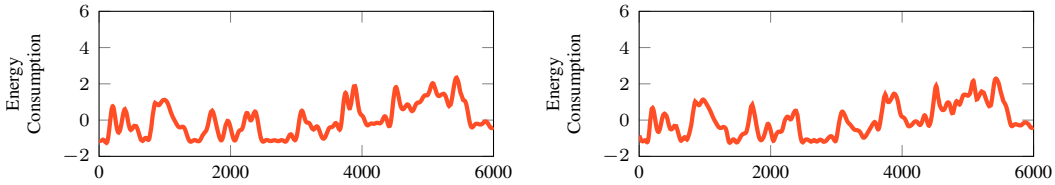
Figure 5: Curves with a similar smoothness level (50%).



(a) `SparseGP`, $h = 40.164$ (b) LATENTGP, $k = 200$

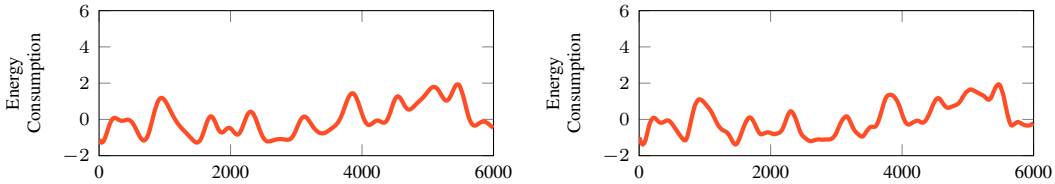
Figure 6: Curves with a similar smoothness level (80%).

[4] `https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption`



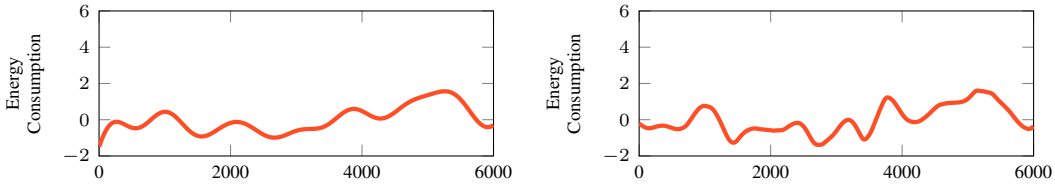
(a) `SparseGP`, $h = 99.940$ (b) LATENTGP, $k = 100$

Figure 7: Curves with a similar smoothness level (90%).



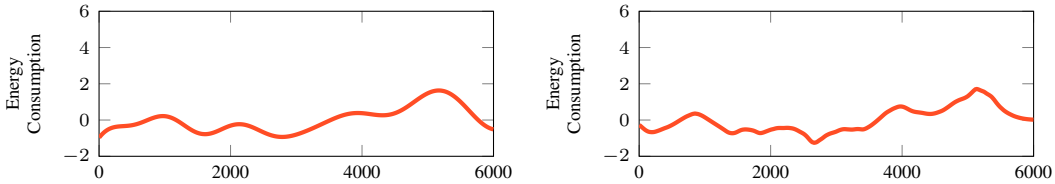
(a) `SparseGP`, $h = 160.656$ (b) LATENTGP, $k = 50$

Figure 8: Curves with a similar smoothness level (95%).



(a) `SparseGP`, $h = 535.520$ (b) LATENTGP, $k = 20$

Figure 9: Curves with a similar smoothness level (98%).



(a) `SparseGP`, $h = 669.400$ (b) LATENTGP, $k = 10$

Figure 10: Curves with a similar smoothness level (99%).