Sparse Information Filter for Fast Gaussian Process Regression

Lucas Kania[1(✉)], Manuel Schürch[1,2], Dario Azzimonti[2], and Alessio Benavoli[3]

[1] Università della Svizzera italiana (USI), Via Buffi 13, Lugano, Switzerland
`lucas.kania@usi.ch`

[2] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Via la Santa 1, Lugano, Switzerland
`{manuel.schuerch,dario.azzimonti}@idsia.ch`

[3] School of Computer Science and Statistics, Trinity College, Dublin, Ireland
`alessio.benavoli@tcd.ie`

Abstract. Gaussian processes (GPs) are an important tool in machine learning and applied mathematics with applications ranging from Bayesian optimization to calibration of computer experiments. They constitute a powerful kernelized non-parametric method with well-calibrated uncertainty estimates, however, off-the-shelf GP inference procedures are limited to datasets with a few thousand data points because of their cubic computational complexity. For this reason, many sparse GPs techniques were developed over the past years. In this paper, we focus on GP regression tasks and propose a new algorithm to train variational sparse GP models. An analytical posterior update expression based on the Information Filter is derived for the variational sparse GP model. We benchmark our method on several real datasets with millions of data points against the state-of-the-art Stochastic Variational GP (SVGP) and sparse orthogonal variational inference for Gaussian Processes (SOLVEGP). Our method achieves comparable performances to SVGP and SOLVEGP while providing considerable speed-ups. Specifically, it is consistently four times faster than SVGP and on average 2.5 times faster than SOLVEGP.

Keywords: Gaussian process regression · Sparse variational method · Information filter variational bound

1 Introduction

Gaussian processes (GPs) are an important machine learning tool [14] widely used for regression and classification tasks. The well-calibrated uncertainty quantification provided by GPs is important in a wide range of applications such as

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-86523-8_32) contains supplementary material, which is available to authorized users.

Bayesian optimization [20], visualization [10], and analysis of computer experiments [16]. The main drawback of GPs is that they scale poorly with the training size.

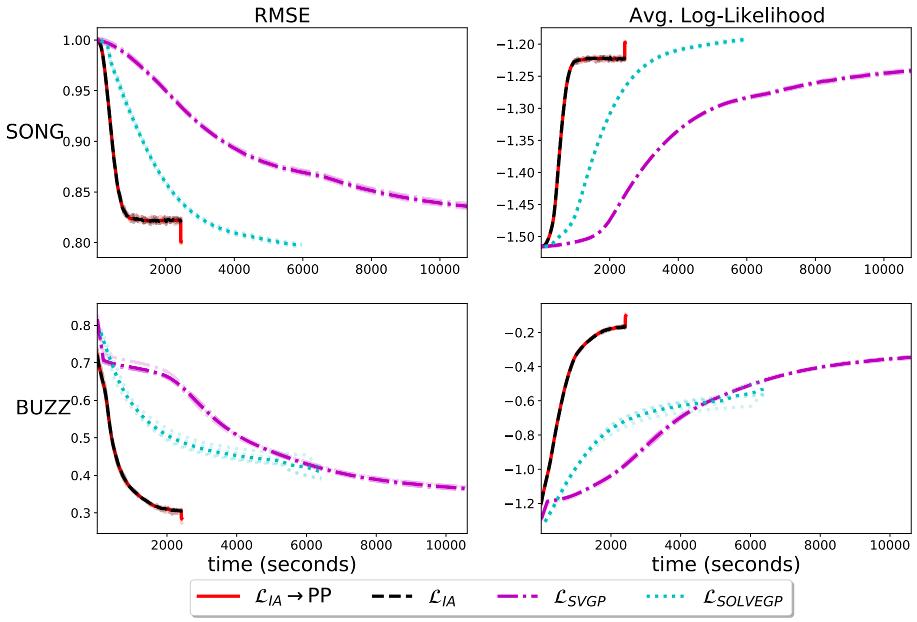

Fig. 1. The main method presented in this paper, $\mathcal{L}_{\text{IA}} \rightarrow$ PP, compared against $\mathcal{L}_{\text{SVGP}}$ [7] and $\mathcal{L}_{\text{SOLVEGP}}$ [21] on the high dimensional UCI datasets `SONG` (89 dimensions) and `BUZZ` (77 dimensions). All methods run for the same number of iteration, plotted against wall-clock time. This behaviour is also observed in the other experiments, see Sect. 4.3.

Given N data points, GP training has a computational complexity of $O(N^3)$ due to the inversion of a $N \times N$ covariance matrix. In the last few decades, different families of methods were introduced to address this issue. Aggregation or statistical consensus methods [5,8,11,24] solve the issue by training smaller GP models and then aggregating them. Other techniques exploit numerical linear algebra approximations to efficiently solve the inversion problem [26] or state-space representations [17,18].

In this work, we focus on the family of sparse inducing points approximations [13]. Such methods employ $M \ll N$ inducing points to summarize the whole training data and reduce the computational complexity to $O(NM^2)$. In such models, the key parameters are the positions of the inducing points which need to be optimized. Initial attempts did not guarantee a convergence to full GP [4,22], however, ref. [23] introduced a variational lower bound which links the sparse GP approximation to a full GP. This method (Variational Free Energy, VFE) approximates the posterior with a variational distribution chosen by minimizing the Kullback-Leibler divergence between the variational distribution and the exact posterior.

The method proposed by [23] allows for large training data sizes, however, it is still not appropriate for big data since the optimization of the lower bound, required to choose the positions of the inducing points, cannot be split into mini-batches. In order to address this issue, [7] proposed a stochastic gradient variational method (SVGP) employing an uncollapsed version of the lower bound which splits into a sum over mini-batches and allows for stochastic gradient descent. In a regression setting, VFE [23] provides analytical updates for the variational distribution while SVGP [7] requires the optimization of all the variational distribution's parameters. This increases the size of the parameter space and can lead to unstable results. Natural gradients [15] are often used to speed up the convergence of this optimization problem. Nonetheless, analytical updates for regression tasks could lead to higher speed-ups.

Reference [19] proposed a recursive collapsed lower bound for regression that exploits analytical updates for the posterior distribution and splits into a sum over mini-batches. Consequently, the method can be scaled to millions of data points by stochastic gradient descent. This recursive approach provides a performance competitive with SVGP both in terms of accuracy and computational time, however, it requires storing past gradients in memory. When the input space dimension or the number of inducing points is very large, this method becomes problematic memory-wise as the past Jacobian matrices are cumbersome to store.

In this paper, we address this issue and propose a simple and cheap training method that efficiently achieves state-of-the-art performance in practice. Figure 1 shows an example of the behaviour of our method, denoted $\mathcal{L}_{\mathrm{IA}} \rightarrow \mathrm{PP}$, compared against SVGP [7] and SOLVEGP [21] on two high dimensional large UCI [6] datasets. The plots show root mean squared error (RMSE) and average log-likelihood as a function of computational time. All methods are run for a fixed number of iterations; note the computational speed-ups achieved by $\mathcal{L}_{\mathrm{IA}} \rightarrow \mathrm{PP}$.

We develop our method with a straightforward approach: first, we stochastically train the model parameters on independent mini-batches and then we compute the full approximate posterior on the whole dataset with analytical updates. In particular we

1. formulate the VFE model with an Information Filter (IF) approach allowing for analytical posterior updates in natural parameters. We would like to stress that IF is used to reformulate posterior updates and not to obtain alternative state-space representations;
2. describe a training algorithm, $\mathcal{L}_{\mathrm{IA}} \rightarrow \mathrm{PP}$, that employs the IF formulation on independent mini-batches as a warm-up phase and recovers the previously ignored data dependencies by employing analytic posterior updates in the final phase of the optimization;
3. show on real datasets that our training method achieves comparable performances with respect to state-of-the-art techniques (SVGP [7], SOLVEGP [21]) in a fraction of their runtime as shown in Fig. 1 and Sect. 4.3.

2 Gaussian Process Regression

2.1 Full Gaussian Processes

Consider a dataset $\mathcal{D} = (x_i, y_i)_{i=1}^N$ of input points $X = (x_i)_{i=1}^N$, $x_i \in \mathbb{R}^D$ and observations $y_i \in \mathbb{R}$, where the ith observation y_i is the sum of an unknown function $f : \mathbb{R}^D \to \mathbb{R}$ evaluated at x_i and independent Gaussian noise, i.e.

$$y_i = f(x_i) + \epsilon_i \overset{iid}{\sim} \mathcal{N}\left(f(x_i), \sigma_n^2\right). \tag{1}$$

We model f by using a Gaussian Process [14] with mean function m and a covariance function k. A GP is a stochastic process such that the joint distribution of any finite collection of evaluations of f is distributed as a multivariate Gaussian

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{bmatrix} \right),$$

where $m : \mathbb{R}^D \to \mathbb{R}$ is an arbitrary function and $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is a positive definite kernel. Both m and k could depend on a vector of parameters $\theta \in \Theta$; in this paper we assume that $m \equiv 0$ and k is a kernel from a parametric family such as the radial basis functions (RBF) or the Matérn family, see [14, Chapter 4].

Let $\mathbf{f} = \left[f(x_1), \ldots, f(x_N)\right]$, $\mathbf{y} = \left[y_1, \ldots, y_N\right]$ and consider test points $X_* = (x_j^*)_{j=1}^A$ with the respective function values $\mathbf{f}_* = f(X_*)$. The joint distribution of $(\mathbf{y}, \mathbf{f}_*)$ is normally distributed due to Eq (1). Thus, the predictive distribution can be obtained by conditioning $\mathbf{f}_*|\mathbf{y} \sim \mathcal{N}\left(\mathbf{f}_*|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}\right)$ [14, Chapter 2] where

$$\begin{aligned} \mu_{\mathbf{y}} &= \mathrm{K}_{\mathrm{X}_*\mathrm{X}}(\mathrm{K}_{\mathrm{XX}} + \sigma_n^2 \mathbb{I})^{-1}\mathbf{y}; \\ \Sigma_{\mathbf{y}} &= \mathrm{K}_{\mathrm{X}_*\mathrm{X}_*} - \mathrm{K}_{\mathrm{X}_*\mathrm{X}}(\mathrm{K}_{\mathrm{XX}} + \sigma_n^2 \mathbb{I})^{-1}\mathrm{K}_{\mathrm{XX}_*}. \end{aligned}$$

where $\mathbb{I}$ is the identity matrix and $K_{VW} := [k(v_i, w_j)]_{i,j}$ is the kernel matrix obtained for two sets of points: $V = (v_i)_{i=1}^{N_V} \subseteq \mathbb{R}^D$ and $W = (w_j)_{j=1}^{N_W} \subseteq \mathbb{R}^D$. Given the prior $\mathbf{f} \sim \mathcal{N}(0, \mathrm{K}_{\mathrm{XX}})$, we can compute the log marginal likelihood $\log p(\mathbf{y}|\theta) = \mathcal{N}\left(\mathbf{y}|0, \mathrm{K}_{\mathrm{XX}} + \sigma_n^2 \mathbb{I}\right)$ which is a function of θ, the parameters associated to the kernel k, and of the Gaussian noise variance σ_n^2. We can estimate those parameters by maximizing $\log p(\mathbf{y}|\theta)$. Note that both training and marginal likelihood computation require the inversion of a $N \times N$ matrix which makes this method infeasible for large datasets due to the $O(N^3)$ time complexity of the operation.

2.2 Sparse Inducing Points Gaussian Processes

The cubic time complexity required by GP regression motivated the development of sparse methods for GP regression. The key idea is to find a set of M so-called inducing points, $R = (r_j)_{j=1}^M$ where $r_j \in \mathbb{R}^D$, such that $\mathbf{f}_{\mathrm{R}} := (f(r_j))_{j=1}^M \in \mathbb{R}^M$

are an approximate sufficient statistic of the whole dataset [13]. Thereby, the key challenge is to find the location of these inducing points. Early attempts [4,22] used the marginal likelihood to select the inducing points, however, they are prone to overfitting. Reference [23] instead proposed to perform approximate variational inference, leading to the Variational Free Energy (VFE) method which converges to full GP as M increases. The inducing points and the parameters are selected by maximizing the following VFE lower bound

$$\mathcal{L}_{\mathrm{VFE}} = \log \mathcal{N}\left(\mathbf{y}|\mathbb{0}, \mathrm{Q}_{\mathrm{XX}} + \sigma_n^2 \mathbb{I}\right) - \frac{Tr(\mathrm{K}_{\mathrm{XX}} - \mathrm{Q}_{\mathrm{XX}})}{2\sigma_n^2}, \tag{2}$$

where $\mathrm{Q}_{\mathrm{XX}} = \mathrm{H}_{\mathrm{X}}\mathrm{K}_{\mathrm{RR}}\mathrm{H}_{\mathrm{X}}^{\mathrm{T}}$ and $\mathrm{H}_{\mathrm{X}} = \mathrm{K}_{\mathrm{XR}}\mathrm{K}_{\mathrm{RR}}^{-1}$, see [23] for details. In this method, only K_{RR} needs to be inverted, therefore training with $\mathcal{L}_{\mathrm{VFE}}$ has time complexity $O(M^3 + NM)$. This allows training (sparse) GPs with tens of thousands of points; the method, however, becomes infeasible for larger N because the lower bound in Eq. (2) cannot be split into mini-batches and optimized stochastically.

2.3 Stochastic Variational Gaussian Processes

Stochastic Variational Gaussian Processes (SVGP) [7] avoid the above-mentioned problem by splitting the data $\mathcal{D}$ into K mini-batches of B points, i.e. $(\mathbf{y}_k, X_k) \in \mathbb{R}^B \times \mathbb{R}^{B \times D}$ for $k = 1, \ldots, K$, and by stochastically optimizing the following uncollapsed lower bound

$$\mathcal{L}_{\mathrm{SVGP}} = \left(\sum_{k=1}^{K} \mathbb{E}_{\mathbf{f}_k \sim q(\cdot)}[\log p(\mathbf{y}_k|\mathbf{f}_k, \theta)]\right) - \mathbb{KL}[q(\mathbf{f}_{\mathrm{R}})||p(\mathbf{f}_{\mathrm{R}}|\theta)]. \tag{3}$$

This lower bound uses a normal variational distribution $q(\mathbf{f}_{\mathrm{R}})$ where the variational parameters, i.e. mean and covariance, need to be optimized numerically in addition to the inducing points and the kernel parameters. Natural gradients [15] for the variational parameters speed up the task. Nonetheless, all entries in the inducing point's posterior mean vector and posterior covariance matrix have to be estimated numerically. The lack of analytical posterior updates introduces $O(M^2)$ additional variational parameters, whose optimization becomes cumbersome when using a large number of inducing points.

2.4 Recursively Estimated Sparse Gaussian Processes

Reference [19] obtained analytical updates of the inducing points' posterior by recursively applying the technique introduced by [23]. That is, the distribution $p(\mathbf{f}_R|\mathbf{y}_{1:k})$ is approximated by a moment parameterized distribution $q_k(\mathbf{f}_{\mathrm{R}}) = \mathcal{N}(\mathbf{f}_{\mathrm{R}}|\mu_k, \Sigma_k)$. Recursively performing variational inference leads to the bound

$$\mathcal{L}_{\mathrm{REC}} = \sum_{k=1}^{K} \log \mathcal{N}(\mathbf{y}_k|\mathrm{H}_{\mathrm{X_k}}\mu_{k-1}, S_k) - \frac{Tr(\mathrm{K}_{\mathrm{X_K X_K}} - \mathrm{Q}_{\mathrm{X_K X_K}})}{2\sigma_n^2} \tag{4}$$

where $S_k = \mathrm{H}_{\mathrm{X_k}} \Sigma_{k-1} \mathrm{H}_{\mathrm{X_k}}^{\mathrm{T}} + \sigma_n^2 \mathbb{I}$ and $\mathrm{H}_{\mathrm{X_k}} = \mathrm{K}_{\mathrm{X_k R}} \mathrm{K}_{\mathrm{RR}}^{-1}$. The posterior can be analytically computed by the moment parameterized posterior propagation

$$\mu_k = \Sigma_k \left(\frac{\mathrm{H}_{\mathrm{X_k}}^{\mathrm{T}} \mathbf{y}_k}{\sigma_n^2} + \Sigma_{k-1}^{-1} \mu_{k-1} \right), \tag{5}$$

$$\Sigma_k = \left(\Sigma_{k-1}^{-1} + \frac{\mathrm{H}_{\mathrm{X_k}}^{\mathrm{T}} \mathrm{H}_{\mathrm{X_k}}}{\sigma_n^2} \right)^{-1}. \tag{6}$$

By keeping the same parameters for the whole epoch, the posterior equals that of its batch counterpart VFE. Similarly, the sum of $\mathcal{L}_{\mathrm{REC}}$ over all mini-batches is equal to the batch lower bound $\mathcal{L}_{\mathrm{VFE}}$. However, the authors proposed to stochastically approximate its gradient by computing the gradient w.r.t. one mini-batch and plugging-in the derivative of the variational distribution parameters from the previous iteration in order to speed up the optimization. This method has two main drawbacks. Firstly, it is constrained to use small learning rates for the approximation to be valid and stable in practice. Second, the gradients of the last iteration must be stored. Such storage becomes problematic when the input space dimension or the number of inducing points is very large.

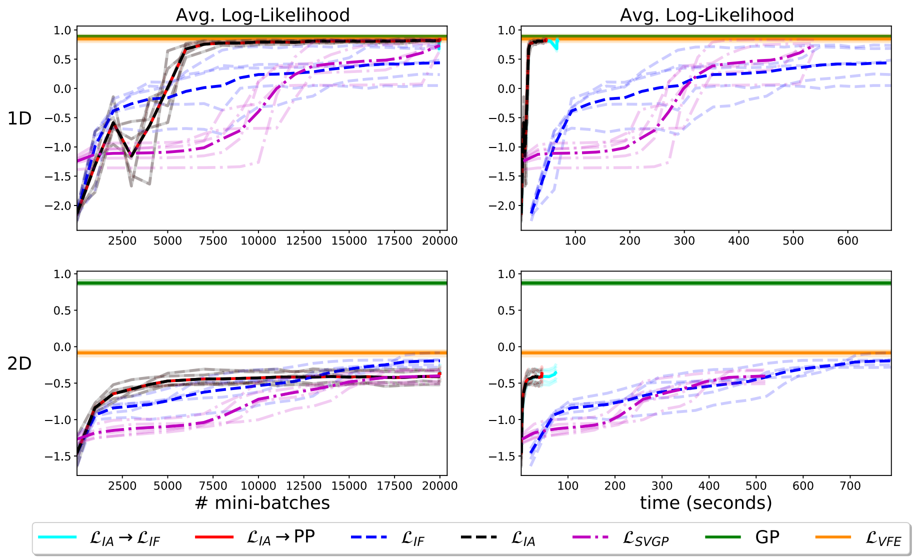

Fig. 2. Toy experiments: 1D and 2D generated GP data. Performances of GP and $\mathcal{L}_{\mathrm{VFE}}$ in horizontal lines as their optimization does not use mini-batches. On the left comparison in number of mini-batches, on the right runtime comparison.

3 Information Filter for Sparse Gaussian Processes

In order to overcome the previous issues but keep the advantage of analytical posterior updates, we propose an Information Filter update for the posterior, a very efficient and easy to interpret method for estimation in sparse GP methods.

Alternatively to the moment parameterized propagation in Eqs. (5) and (6), the posterior over the inducing points can be more efficiently propagated using a natural parameterization $\mathcal{N}^{-1}(\mathbf{f}_R|\eta_k, \Lambda_k)$ with $\eta_k = \Sigma_k^{-1}\mu_k$ and $\Lambda_k = \Sigma_k^{-1}$. The advantage of this Information Filter formulation are the compact updates

$$\eta_k = \eta_{k-1} + \frac{1}{\sigma_n^2}\mathrm{H}_{\mathrm{X_k}}^{\mathrm{T}}\mathbf{y}_k \qquad \text{and} \qquad \eta_0 = \mathbb{0}; \tag{7}$$

$$\Lambda_k = \Lambda_{k-1} + \frac{1}{\sigma_n^2}\mathrm{H}_{\mathrm{X_k}}^{\mathrm{T}}\mathrm{H}_{\mathrm{X_k}} \qquad \text{and} \qquad \Lambda_0 = \mathrm{K}_{\mathrm{RR}}^{-1}, \tag{8}$$

where $\mathrm{H}_{\mathrm{X_k}} = \mathrm{K}_{\mathrm{X_kR}}\mathrm{K}_{\mathrm{RR}}^{-1}$. Further computational efficiency is gained by using the equivalent rotated parameterization: $\eta_{\mathrm{k}}^{\mathrm{R}} = \mathrm{K}_{\mathrm{RR}}\eta_k$ and $\Lambda_{\mathrm{k}}^{\mathrm{R}} = \mathrm{K}_{\mathrm{RR}}\Lambda_k\mathrm{K}_{\mathrm{RR}}$ with updates

$$\eta_{\mathrm{k}}^{\mathrm{R}} = \eta_{k-1}^{R} + \frac{1}{\sigma_n^2}\mathrm{K}_{\mathrm{RX_k}}\mathbf{y}_k \qquad \text{and} \qquad \eta_0^R = \mathbb{0}; \tag{9}$$

$$\Lambda_{\mathrm{k}}^{\mathrm{R}} = \Lambda_{k-1}^{R} + \frac{1}{\sigma_n^2}\mathrm{K}_{\mathrm{RX_k}}\mathrm{K}_{\mathrm{X_kR}} \qquad \text{and} \qquad \Lambda_0^R = \mathrm{K}_{\mathrm{RR}}. \tag{10}$$

This parameterization constitutes a computational shortcut since several matrix computations in each step can be avoided. The overall computational complexity of the method remains the same as in $\mathcal{L}_{\mathrm{REC}}$, however, it allows for smaller constant terms, thus reducing computational time in practice. Particularly, this Information Filter propagation is more efficient than the Kalman Filter formulation used by [19] when the mini-batch size is greater than the number of inducing points, which is usually the case in practice. The corresponding lower bound is

$$\mathcal{L}_{\mathrm{IF}} = \sum_{k=1}^{K}\log\mathcal{N}^{-1}\left(r_k|0, S_k^{-1}\right) - \frac{Tr(\mathrm{K_{X_KX_K}} - \mathrm{Q_{X_KX_K}})}{2\sigma_n^2}, \tag{11}$$

where $r_k = \mathbf{y}_k - \mathrm{Cholesky}(\Lambda_{\mathrm{k}}^{\mathrm{R}})^{-1}\mathrm{K}_{\mathrm{RX_k}}$ and $S_k^{-1} = \frac{\mathbb{I}}{\sigma_n^2} - \frac{1}{\sigma_n^4}\mathrm{K}_{\mathrm{X_kR}}(\Lambda_{\mathrm{k}}^{\mathrm{R}})^{-1}\mathrm{K}_{\mathrm{RX_k}}$.

The bound consist of two parts: the first term corresponds to the log marginal likelihood of each mini-batch, and the second term is the correction term resulting from the variational optimization. We refer the reader to the supplementary material for the derivation and efficient computation of the posterior propagation and the lower bound in terms of the rotated posterior mean and covariance.

Computing $\mathcal{L}_{\mathrm{IF}}$ for an epoch using the rotated parameterization in Eqs. (9) and (10) while keeping the same parameters for all mini-batches produces a posterior that equals its batch counterpart VFE.

3.1 Stochastic Hyperparameter Optimization

The additive structure of the $\mathcal{L}_{\mathrm{IF}}$, together with the efficient IF posterior propagation allows for more frequent parameter updates. Analogously to the approach in [19], $\mathcal{L}_{\mathrm{IF}}$ can be stochastically optimized w.r.t. to all the kernel parameters and the inducing points' locations. Let $\mathcal{L}_{\mathrm{IF}k}$ be the k-term of the $\mathcal{L}_{\mathrm{IF}}$ sum

$$\mathcal{L}_{\mathrm{IF}k}(\theta_k, \Lambda^R_{\mathrm{k}-1}, \eta^R_{k-1}) = l_k(\theta_k, \Lambda^R_{k-1}, \eta^R_{k-1}) - a_k(\theta_k),$$

where $l_k = \log \mathcal{N}^{-1}\left(r_k|0, S_k^{-1}\right)$ and $a_k = \frac{Tr\left(\mathrm{K}_{\mathrm{X_K X_K}} - \mathrm{Q}_{\mathrm{X_K X_K}}\right)}{2\sigma_n^2}$. The gradient of $\mathcal{L}_{IF}k$ w.r.t. the parameters at iteration t, denoted θ_t, can be approximated using Jacobian matrices of the previous iteration

$$\frac{\partial \mathcal{L}_{\mathrm{IF}k}}{\partial \theta_t} \approx \frac{\partial l_k}{\partial \theta_t} + \frac{\partial l_k}{\partial \Lambda^R_{k-1}} \frac{\partial \Lambda^R_{k-1}}{\partial \theta_{t-1}} + \frac{\partial l_k}{\partial \eta^R_{k-1}} \frac{\partial \eta^R_{k-1}}{\partial \theta_{t-1}} - \frac{\partial a_k}{\partial \theta_t}. \tag{12}$$

This recursive propagation of the gradients of the rotated posterior mean and covariance indirectly takes into account all the past gradients via $\frac{\partial \Lambda^R_{k-1}}{\partial \theta_{t-1}}$ and $\frac{\partial \eta^R_{k-1}}{\partial \theta_{t-1}}$. It would be exact and optimal if the derivatives with respect to the current parameters were available. However, when changing the parameters too fast between iterations, e.g. due to a large learning rate, this approximation is too rough and leads to unstable optimization results in practice. Furthermore, the performance gain provided by the recursive gradient propagation does not compensate for the additional storage requirements on the order of $O(M^2U)$ where U is the number of hyperparameters. For instance, if 500 inducing points are used in a 10-dimensional problem, only storing the Jacobian matrices requires 10 GB, under double float precision, i.e. around the memory limit of the GPUs used for this work. Additionally, using the approximated posterior mean and covariance in the initial stages of the optimization slows down the convergence due to the lasting effect of the parameters' random initialization. This effect is noticeable in the test cases presented in Fig. 2: compare $\mathcal{L}_{\mathrm{IF}}$ with the more stable method introduce below called $\mathcal{L}_{\mathrm{IA}}$.

Independence Assumption for Optimization. In order to circumvent these instabilities in the optimization part, we propose a fast and efficient method that ignores the (approximated) correlations between the mini-batches in the stochastic gradient computation in the beginning. Specifically, the mini-batches are assumed to be mutually independent, which simplifies the lower-bound of the marginal likelihood to

$$\mathcal{L}_{\mathrm{IA}} = \sum_{k=1}^{K} \log \mathcal{N}^{-1}\left(\mathbf{y}_k|0, \frac{\mathbb{I}}{\sigma_n^2} - \frac{1}{\sigma_n^4}\mathrm{Q}_{\mathrm{X_K X_K}}\right) - \frac{Tr(\mathrm{K}_{\mathrm{X_K X_K}} - \mathrm{Q}_{\mathrm{X_K X_K}})}{2\sigma_n^2}. \tag{13}$$

Algorithm 1: $\mathcal{L}_{\mathrm{IA}} \rightarrow$ PP Algorithm

```
1 Choose number of epochs $E_{IA}$ for $\mathcal{L}_{\mathrm{IA}}$ ;
2 Split $\mathcal{D}$ into $K$ mini-batches of $B$ points ;
3 for $k = 1 \rightarrow K \cdot E_{IA}$ do
4     compute gradients on the $k$th mini-batch, eq. (13);
5     update the hyperparameters ;
6 end
7 for $k = 1 \rightarrow K$ do                                  // i.e. for one epoch
8     compute posterior propagation with eqs. (9),(10) ;
9 end
```

Recovery of the Full Posterior. In order to incorporate all the dependencies once reasonable hyperparameters are achieved, we could switch to optimize the bound $\mathcal{L}_{\mathrm{IF}}$ during the last few epochs. Eventually, all the ignored data dependencies are taken into account with this method, denoted $\mathcal{L}_{\mathrm{IA}} \rightarrow \mathcal{L}_{\mathrm{IF}}$.

A further computational shortcut is to propagate the inducing points posterior according to Eqs. (9), (10), and not perform any optimization in the last epoch. This method, denoted $\mathcal{L}_{\mathrm{IA}} \rightarrow$ PP, constitutes a practical alternative when the computation and storage of the Jacobian matrices is costly, for instance in high dimensional problems where a large number of inducing points are needed. For clarity, a pseudo-code of the method is provided in Algorithm 1. Note that the full sparse posterior distribution including all data is achieved. The independence assumption influences only the optimization of the hyperparameters. No approximation is used when computing the posterior distribution.

Distributed Posterior Propagation. An advantage of the Information Filter formulation is that it allows computing the posterior in parallel. Note that Eqs. (9), (10) can be rewritten as

$$\eta_K^R = \sum_{k=1}^{K} \frac{1}{\sigma_n^2} \mathrm{K}_{\mathrm{RX_k}} \mathbf{y}_k \text{ and } \Lambda_K^R = \Lambda_0^R + \sum_{k=1}^{K} \frac{1}{\sigma_n^2} \mathrm{K}_{\mathrm{RX_k}} \mathrm{K}_{\mathrm{X_kR}}.$$

Since the sums terms are functionally independent, the computation of the posterior distribution can be easily distributed via a map-reduce scheme.

Prediction. Given a new $X_* \in \mathbb{R}^{A \times D}$, the predictive distribution after seeing $\mathbf{y}_{1:k}$ of the sparse GP methods can be computed by

$$p\left(\mathbf{f}_* \mid \mathbf{y}_{1:k}\right) = \int p\left(\mathbf{f}_* \mid \mathbf{f}_{\mathrm{R}}\right) p\left(\mathbf{f}_{\mathrm{R}} \mid \mathbf{y}_{1:k}\right) \mathrm{d}\mathbf{f}_{\mathrm{R}} = \mathcal{N}\left(\mathbf{f}_* | \mu_*, \Sigma_*\right)$$
$$\text{where } \mu_* = \mathrm{K}_{\mathrm{X_*R}} (\Lambda_{\mathrm{k}}^{\mathrm{R}})^{-1} \eta_{\mathrm{k}}^{\mathrm{R}};$$
$$\Sigma_* = \mathrm{K}_{\mathrm{X_*X_*}} - \mathrm{Q}_{\mathrm{X_*X_*}} + \mathrm{K}_{\mathrm{X_*R}} (\Lambda_{\mathrm{k}}^{\mathrm{R}})^{-1} \mathrm{K}_{\mathrm{RX_*}}.$$

The predictions for $\mathbf{y}_*$ are obtained by adding $\sigma_n^2 \mathbb{I}$ to the covariance of $\mathbf{f}_* | \mathbf{y}_{1:k}$. A detailed derivation is provided in the supplementary material.

4 Experiments

We repeated every experiment presented in this Sect. 5 times, using different random splits for each dataset. We always use the same hardware equipped with a GeForce RTX 2080 Ti and an Intel(R) Xeon(R) Gold 5217 (3 GHz).

Furthermore, we switch from optimizing $\mathcal{L}_{\text{IA}}$ to the posterior propagation, denoted $\mathcal{L}_{\text{IA}} \rightarrow \text{PP}$, in the last 5 epochs of training for all the synthetic and real datasets. Although only one epoch is needed to incorporate all the information of a dataset, we do 5 epochs so the effect is noticeable in the figures. Alternatively, an adaptive switch could be used. For simplicity, we briefly discuss it in the supplementary material. Moreover, we did not exploit the parallelization of the posterior propagation in order to be able to compare all training algorithms on the number of iterations.

For full GP, VFE and SVGP, we used the GPflow 2.0 library [12,25], which is based on Tensorflow 2.0 [1]. We further compare with the recently introduced SOLVEGP [21], implemented in GPflow 1.5.1 and Tensorflow 1.15.4. Our algorithms, namely the training of $\mathcal{L}_{\text{IF}}$, $\mathcal{L}_{\text{IA}} \rightarrow \mathcal{L}_{\text{IF}}$ and $\mathcal{L}_{\text{IA}} \rightarrow \text{PP}$ were implemented in Tensorflow 2.0[1]. All stochastic methods were optimized with the ADAM optimizer [9] using the default settings ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e^{-7}$); we used the recommended settings ($\gamma = 0.1$) for the stochastic optimization of natural gradients in SVGP [15].

We do not compare the methods against optimizing $\mathcal{L}_{\text{REC}}$ [19] because their code does not run on GPU, making the computational times not comparable. Another candidate method that is not considered is Exact Gaussian Processes (ExactGP) [26]. The aim of approximated sparse GPs (SGPs, which includes our method, SVGP and SOLVEGP) and ExactGPs are different, the latter propose a computationally expensive procedure to compute a full GP while the former propose a computationally cheap approximation of a SGP. Consequently, an ExactGP would yield higher accuracy than an approximated SGP. The accuracy of approximated SGP is upper-bounded by VFE's accuracy which is itself upper-bounded by full GP's accuracy. Moreover, for high dimensional datasets such as BUZZ or SONG used later, ExactGP requires an infrastructure with many GPUs in parallel, e.g. 8 in their experiments, which makes the method hard to replicate in practice.

We compare each method in terms of average log-likelihood, root mean squared error (RMSE) and computational time. The comparison in computational time is fair because all methods were run on the same hardware which ran exclusively the training procedure.

4.1 Toy Data

We start by showcasing the methods on a simple dataset generated by a SGP with 2000 random inducing points, and a RBF kernel with variance 1, Gaussian noise 0.01 and lengthscales 0.1 and 0.2 for 1 and 2 dimensions correspondingly.

[1] The code is available at https://github.com/lkania/Sparse-IF-for-Fast-GP.

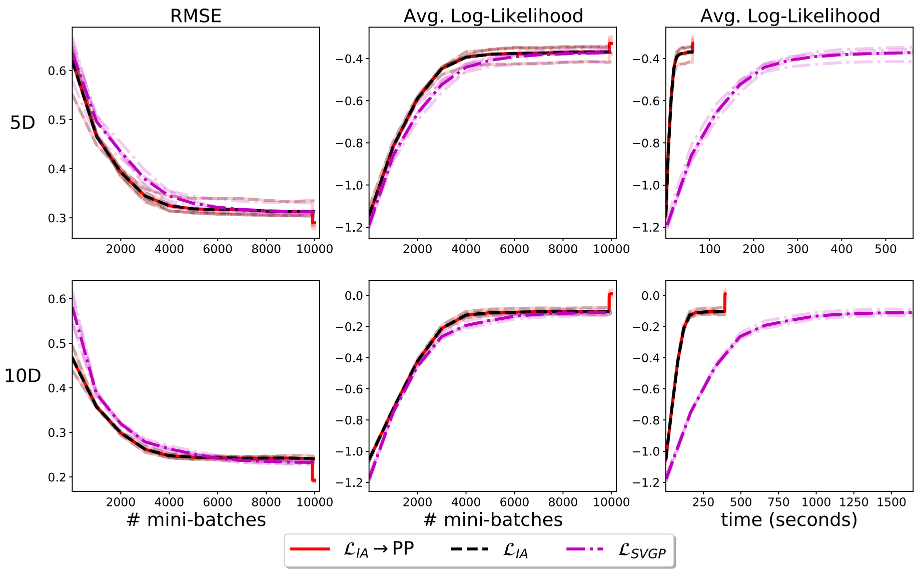

Fig. 3. Synthetic experiments: 5D and 10D data generated from GPs. RMSE vs number of mini-batches (left), average log-likelihood vs number of mini-batches (center), and average log-likelihood vs runtime (right).

We consider $N = 5000$ training points and 500 test points. In all experiments, the initial parameters for the RBF kernel were 1 for the lengthscales and Gaussian noise, and 2 for the variance. Moreover, 20 inducing points were randomly selected from the training data. In this example, the switch for $\mathcal{L}_{\rm IA} \rightarrow \mathcal{L}_{\rm IF}$ and $\mathcal{L}_{\rm IA} \rightarrow$ PP was done in the last 200 epochs to make the difference between them visually noticeable. All methods were run for 20K iterations with a learning rate of 0.001 and mini-batches of 500 data points.

Figure 2 shows that in 1D all sparse training methods converge to the $\mathcal{L}_{\rm VFE}$ solution, which itself converges to the GP solution. However, this is not the case in the 2D example. $\mathcal{L}_{\rm IF}$ comes closer to the VFE solution, followed by $\mathcal{L}_{\rm IA} \rightarrow \mathcal{L}_{\rm IF}$. Note that the convergence of $\mathcal{L}_{\rm IA} \rightarrow \mathcal{L}_{\rm IF}$ is expected since using $\mathcal{L}_{\rm IA}$ at the beginning is just a way of *warm-starting* the parameters for $\mathcal{L}_{\rm IF}$.

4.2 Synthetic Data

Using the same generating process used in the previous experiments, we produced two 100K points datasets for 5 and 10 dimensions using an RBF kernel with variance 1, Gaussian noise 0.01, and lengthscales 0.5 and 1 respectively. The SGPs models, with a RBF kernel, were initialized with Gaussian noise 1, kernel variance equal to 2 and the lengthscales all equal to 1 in the 5-dimensional case, and 2 in the 10-dimensional problem. All methods run for 10K iterations in all the datasets using 100 and 500 inducing points for 5 and 10 dimensions

Table 1. Average log-likelihood of last epoch (higher is better), average RMSE of the last epoch (lower is better), and average runtime in minutes (lower is better). The algorithm with the best metric is highlighted in each case.

D	Metric	$\mathcal{L}_{\text{IA}} \to \text{PP}$	$\mathcal{L}_{\text{IA}}$	$\mathcal{L}_{\text{SVGP}}$
5	`avg log-lik`	$\mathbf{-0.33 \pm 0.03}$	-0.45 ± 0.14	-0.48 ± 0.15
10	`avg log-lik`	$\mathbf{0.01 \pm 0.01}$	-0.21 ± 0.20	-0.24 ± 0.20
5	`RMSE`	$\mathbf{0.29 \pm 0.01}$	0.34 ± 0.05	0.36 ± 0.06
10	`RMSE`	$\mathbf{0.19 \pm 0.003}$	0.26 ± 0.04	0.27 ± 0.05
5	`runtime`	$\mathbf{1.03 \pm 0.11}$	1.05 ± 0.11	9.37 ± 1.16
10	`runtime`	$\mathbf{6.59 \pm 0.06}$	6.65 ± 0.09	27.40 ± 0.79

respectively. In all cases, we used learning rates equal to 0.001 and mini-batches of 5000 points. Note, that $\mathcal{L}_{\text{IF}}$ and $\mathcal{L}_{\text{IA}} \to \mathcal{L}_{\text{IF}}$ were not run due to the memory constraints of our equipment.

Figure 3 displays the log-likelihood and RMSE for all the methods. In 5 and 10 dimensions, we can clearly see the effect of a few posterior updates, for $\mathcal{L}_{\text{IA}} \to \text{PP}$, after fixing the parameters with $\mathcal{L}_{\text{IA}}$. Additionally, Table 1 displays the average runtimes for each method. Note that $\mathcal{L}_{\text{IA}} \to \text{PP}$ offers a speed-up of $9\times$ and $4\times$ over $\mathcal{L}_{\text{SVGP}}$ for the 5- and 10-dimensional datasets respectively.

4.3 Real Data

Table 2. Training dataset size, test dataset size and dimension of each one of the benchmarked datasets. The last two columns show the learning rate and the number of iterations for all training methods run in each dataset.

Name	Train	Test	D	Learning rate	Iterations
`AIRLINES`	1.6M	100K	8	0.0001	60000
`GAS`	1.4M	100K	17	0.0001	40000
`BUZZ`	0.46M	100K	77	0.0001	60000
`SONG`	0.41M	100K	89	0.0001	60000
`SGEMM`	0.19M	48K	14	0.001	15000
`PROTEIN`	0.036M	9K	9	0.001	5000
`BIKE`	0.013M	3.4K	12	0.0001	40000

We trained SGPs, whose RBF kernels had all their hyperparameters initialized to 1, using $\mathcal{L}_{\text{SVGP}}$, $\mathcal{L}_{\text{SOLVEGP}}$, $\mathcal{L}_{\text{IA}}$ and $\mathcal{L}_{\text{IA}} \to \text{PP}$ in several real-world datasets, shown in Table 2, using mini-batches of 5000 points. All datasets were downloaded from the UCI repository [6] except for `AIRLINES`, where we follow the original example in [7].

For each training algorithm, we present average log-likelihood, RMSE and runtime for the same number of iterations. Figure 1 shows the results corresponding to the datasets SONG and BUZZ. Tables 3, 4 and 5 report the average log-likelihood, RMSE and runtime of all the algorithms. Note that training with $\mathcal{L}_{\text{IA}} \rightarrow$ PP always provides comparable results to the state-of-the-art. However, such results are achieved, on average, approximately 4 times faster than $\mathcal{L}_{\text{SVGP}}$ and 2.5 times faster than $\mathcal{L}_{\text{SOLVEGP}}$, see the last two columns of Table 5. While $\mathcal{L}_{\text{IA}} \rightarrow$ PP is uniformly around 4 times faster than $\mathcal{L}_{\text{SVGP}}$, its performance with respect to $\mathcal{L}_{\text{SOLVEGP}}$ strongly depends on the dataset. In general, $\mathcal{L}_{\text{SOLVEGP}}$ is faster on smaller and simpler datasets, while it becomes computationally costly on noisy datasets like `AIRLINE` or `BUZZ`.

The results show that in high dimensional, large problems a simple method such as $\mathcal{L}_{\text{IA}} \rightarrow$ PP guarantees fast results and should be used in practice. Note that, compared to $\mathcal{L}_{\text{IA}}$, the inducing point posterior propagation step added in $\mathcal{L}_{\text{IA}} \rightarrow$ PP always increases performances (lower RMSE, higher log-likelihood) and it is not noticeable in terms of computational time.

Table 3. Average Log-Likelihood of the last epoch, higher is better. The algorithm with the highest average mean log-likelihood is highlighted for each dataset.

Dataset	$\mathcal{L}_{\text{IA}} \rightarrow$ PP	$\mathcal{L}_{\text{IA}}$	$\mathcal{L}_{\text{SVGP}}$	$\mathcal{L}_{\text{SOLVEGP}}$
`AIRLINES`	$\mathbf{-1.31 \pm 0.01}$	-1.33 ± 0.01	-1.33 ± 0.01	-1.32 ± 0.01
`GAS`	-0.13 ± 0.02	-0.24 ± 0.02	-0.25 ± 0.01	$\mathbf{-0.05 \pm 0.05}$
`BUZZ`	$\mathbf{-0.10 \pm 0.01}$	-0.17 ± 0.01	-0.36 ± 0.01	-0.55 ± 0.04
`SONG`	$\mathbf{-1.20 \pm 0.001}$	-1.22 ± 0.002	-1.25 ± 0.004	$\mathbf{-1.20 \pm 0.002}$
`SGEMM`	0.47 ± 0.002	0.33 ± 0.02	0.28 ± 0.05	$\mathbf{0.63 \pm 0.17}$
`PROTEIN`	$\mathbf{-1.04 \pm 0.01}$	-1.13 ± 0.05	-1.14 ± 0.08	-1.06 ± 0.02
`BIKE`	$\mathbf{0.08 \pm 0.02}$	0.03 ± 0.02	0.04 ± 0.02	0.01 ± 0.04

Table 4. Average RMSE of the last epoch, lower is better. The algorithm with the lowest average mean RMSE is highlighted for each dataset.

Dataset	$\mathcal{L}_{\text{IA}} \rightarrow$ PP	$\mathcal{L}_{\text{IA}}$	$\mathcal{L}_{\text{SVGP}}$	$\mathcal{L}_{\text{SOLVEGP}}$
`AIRLINES`	$\mathbf{0.89 \pm 0.01}$	0.92 ± 0.01	0.92 ± 0.01	0.90 ± 0.01
`GAS`	0.28 ± 0.01	0.32 ± 0.01	0.32 ± 0.01	$\mathbf{0.25 \pm 0.01}$
`BUZZ`	$\mathbf{0.28 \pm 0.01}$	0.31 ± 0.004	0.37 ± 0.01	0.42 ± 0.02
`SONG`	$\mathbf{0.80 \pm 0.001}$	0.82 ± 0.002	0.84 ± 0.003	$\mathbf{0.80 \pm 0.002}$
`SGEMM`	0.14 ± 0.001	0.17 ± 0.004	0.17 ± 0.01	$\mathbf{0.13 \pm 0.03}$
`PROTEIN`	$\mathbf{0.68 \pm 0.005}$	0.74 ± 0.01	0.74 ± 0.04	0.69 ± 0.01
`BIKE`	$\mathbf{0.22 \pm 0.004}$	0.24 ± 0.005	0.23 ± 0.004	0.23 ± 0.01

Table 5. Average runtimes in hours, lower is better. The algorithm with the lowest average mean runtime is highlighted for each dataset. Δ_{SV} is the ratio between the average runtimes of $\mathcal{L}_{\mathrm{SVGP}}$ and $\mathcal{L}_{\mathrm{IA}} \to \mathrm{PP}$. Δ_{SOLVE} is the ratio between the average runtimes of $\mathcal{L}_{\mathrm{SOLVEGP}}$ and $\mathcal{L}_{\mathrm{IA}} \to \mathrm{PP}$.

Dataset	$\mathcal{L}_{\mathrm{IA}} \to \mathrm{PP}$	$\mathcal{L}_{\mathrm{IA}}$	$\mathcal{L}_{\mathrm{SVGP}}$	$\mathcal{L}_{\mathrm{SOLVEGP}}$	Δ_{SV}	Δ_{SOLVE}
AIRLINES	**0.65 ± 0.004**	0.66 ± 0.004	2.78 ± 0.07	3.72 ± 0.16	4.29	5.75
GAS	**0.43 ± 0.001**	0.44 ± 0.004	1.84 ± 0.04	2.20 ± 0.11	4.28	5.12
BUZZ	**0.67 ± 0.004**	0.68 ± 0.005	2.95 ± 0.04	1.79 ± 0.08	4.38	2.66
SONG	**0.68 ± 0.01**	0.69 ± 0.01	3.00 ± 0.11	1.65 ± 0.02	4.41	2.43
SGEMM	**0.16 ± 0.001**	0.16 ± 0.001	0.69 ± 0.02	0.30 ± 0.01	4.22	1.84
PROTEIN	**0.05 ± 0.0002**	0.05 ± 0.0001	0.23 ± 0.004	0.07 ± 0.001	4.20	1.33
BIKE	**0.44 ± 0.005**	0.44 ± 0.003	1.88 ± 0.07	0.56 ± 0.02	4.27	1.28

5 Conclusion

In this paper, we presented $\mathcal{L}_{\mathrm{IA}} \to \mathrm{PP}$, a fast method to train sparse inducing points GP models. Our method is based on

(A) optimizing a simple lower bound under the assumption of independence between mini-batches, until the convergence of hyperparameters;
(B) recovering a posteriori the dependencies between mini-batches by exactly computing the posterior distribution.

We focused on VFE-like models and provided a method that achieves a performance comparable to state-of-the-art techniques with considerable speed-ups.

The method could be adapted to the power EP model [3]. Additionally, our lower bound exploits an Information Filter formulation that could potentially be distributed. We outlined how to distribute the posterior propagation, which could be exploited by future work to distribute the computation of gradients.

Our technique is based on analytical updates of the posterior that are in general not available for classic GP classification models. Recently, SkewGP for classification [2] proposed a full classification model with analytical updates for the posterior. Our work could be exploited to provide fast sparse training for such models.

Acknowledgments. Lucas Kania thankfully acknowledges the support of the Swiss National Science Foundation (grant number 200021_188534). Manuel Schürch and Dario Azzimonti gratefully acknowledge the support of the Swiss National Research Programme 75 “Big Data” (grant number 407540_167199/1). All authors would like to thank the IDSIA Robotics Lab for granting access to their computational facilities.

References

1. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). https://www.tensorflow.org/. Software available from tensorflow.org

2. Benavoli, A., Azzimonti, D., Piga, D.: Skew gaussian processes for classification. In: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020. LNCS. Springer, Heidelberg (2020)
3. Bui, T.D., Yan, J., Turner, R.E.: A unifying framework for sparse Gaussian process approximation using power expectation propagation. J. Mach. Learn. Res. **18**, 1–72 (2017)
4. Csató, L., Opper, M.: Sparse on-line Gaussian processes. Neural Comput. **14**(3), 641–668 (2002)
5. Deisenroth, M.P., Ng, J.W.: Distributed Gaussian processes. In: 32nd International Conference on Machine Learning, ICML 2015.,vol. 37, pp. 1481–1490 (2015)
6. Dua, D., Graff, C.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
7. Hensman, J., Fusi, N., Lawrence, N.D.: Gaussian processes for big data. In: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (2013)
8. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Comput. **14**(8), 1771–1800 (2002)
9. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
10. Lawrence, N.: Probabilistic non-linear principal component analysis with Gaussian process latent variable models. J. Mach. Learn. Res. **6**, 1783–1816 (2005)
11. Liu, H., Cai, J., Wang, Y., Ong, Y.S.: Generalized robust Bayesian committee machine for large-scale Gaussian process regression. In: 35th International Conference on Machine Learning, Stockholm, Sweden. ICML 2018, vol. 7, pp. 4898–4910 (2018)
12. Matthews, A.G.D.G., et al.: GPflow: a Gaussian process library using TensorFlow. J. Mach. Learn. Res. **18**(40), 1–6 (2017)
13. Quiñonero-Candela, J., Rasmussen, C.E.: A unifying view of sparse approximate Gaussian process regression. J. Mach. Learn. Res. **6**, 1939–1959 (2005)
14. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
15. Salimbeni, H., Eleftheriadis, S., Hensman, J.: Natural gradients in practice: non-conjugate variational inference in Gaussian process models. In: International Conference on Artificial Intelligence and Statistics, AISTATS, vol. 2018, pp. 689–697 (2018)
16. Santner, T.J., Williams, B.J., Notz, W.I.: The Design and Analysis of Computer Experiments. SSS, Springer, New York (2018). https://doi.org/10.1007/978-1-4757-3799-8
17. Särkkä, S., Hartikainen, J.: Infinite-dimensional Kalman filtering approach to spatio-temporal Gaussian process regression. J. Mach. Learn. Res. **22**, 993–1001 (2012)
18. Särkkä, S., Solin, A.: Applied Stochastic Differential Equations. Cambridge University Press, Cambridge (2019)
19. Schürch, M., Azzimonti, D., Benavoli, A., Zaffalon, M.: Recursive estimation for sparse Gaussian process regression. Automatica **120**, 109127 (2020)
20. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: a review of Bayesian optimization. Proc. IEEE **104**(1), 148–175 (2016)
21. Shi, J., Titsias, M.K., Mnih, A.: Sparse orthogonal variational inference for Gaussian processes. In: Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS), Palermo, Italy, vol. 108 (2020)

22. Snelson, E., Ghahramani, Z.: Sparse Gaussian processes using pseudo-inputs Edward. In: Weiss, Y., Schölkopf, B., Platt, C., J. (eds.) Advances in Neural Information Processing Systems, vol. 18. pp. 1257–1264. MIT Press (2006)
23. Titsias, M.K.: Variational learning of inducing variables in sparse Gaussian processes. In: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 5, pp. 567–574 (2009)
24. Tresp, V.: A Bayesian committee machine. Neural Computation **12**, 2719–2741 (2000)
25. van der Wilk, M., Dutordoir, V., John, S., Artemev, A., Adam, V., Hensman, J.: A framework for interdomain and multioutput Gaussian processes. arXiv:2003.01115 (2020)
26. Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K.Q., Wilson, A.G.: Exact Gaussian processes on a million data points. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc. (2019)