

Distributed Gaussian Processes

Marc Peter Deisenroth

Department of Computing, Imperial College London, United Kingdom

M.DEISENROTH@IMPERIAL.AC.UK

Jun Wei Ng

Department of Computing, Imperial College London, United Kingdom

JUNWEI.NG10@ALUMNI.IMPERIAL.AC.UK

Abstract

We propose the generalised Bayesian Committee Machine (gBCM), a practical and scalable hierarchical Gaussian process model for large-scale distributed non-parametric regression. The gBCM is a family of product-of-experts models that hierarchically recombines independent computations to form an approximation of a full Gaussian process. The gBCM includes classical product-of-experts models and the Bayesian Committee Machine as special cases, while it addresses their respective shortcomings. Closed-form computations allow for efficient and straightforward parallelisation and distributed computing with a small memory footprint, but without an explicit sparse approximation. Since training and predicting is independent of the computational graph our model can be used on heterogeneous computing infrastructures, ranging from laptops to large clusters. We provide strong experimental evidence that the gBCM works well on large data sets.

1. Introduction

Gaussian processes (GPs) (Rasmussen & Williams, 2006) are the method of choice for probabilistic nonlinear regression: Their non-parametric nature allows for flexible modelling without specifying low-level assumptions (e.g., the degree of a polynomial) in advance. Inference can be performed in a principled way simply by applying Bayes' theorem. GPs have had substantial impact in various research areas, including geostatistics (Cressie, 1993), optimisation (Jones et al., 1998; Brochu et al., 2009), data visualisation (Lawrence, 2005), robotics and reinforcement learning (Deisenroth et al., 2015), spatio-temporal mod-

elling (Luttinen & Ilin, 2012), and active learning (Krause et al., 2008). A strength of the GP is that it is a fairly reliable black-box function approximator, i.e., it produces reasonable predictions without manual parameter tuning. A practical limitation of the GP is its computational demand: Training and predicting scale in $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$, respectively, where N is the size of the training data set.

For large N (e.g., $N > 10,000$) sparse approximations are often used (Williams & Seeger, 2001; Quiñero-Candela & Rasmussen, 2005; Hensman et al., 2013; Titsias, 2009; Lázaro-Gredilla et al., 2010; Shen et al., 2006). Typically, they lower the computational burden by implicitly (or explicitly) using a subset of the data. This enables sparse methods to scale GPs to training set sizes of $\mathcal{O}(10^5)$. However, even with sparse approximations it is inconceivable to apply GPs to training set sizes of $\geq \mathcal{O}(10^7)$. Recently, Gal et al. (2014) proposed an approach that scales variational sparse GPs (Titsias, 2009) further by exploiting distributed computations. In particular, they derive an exact re-parameterisation of the variational sparse GP model by Titsias (2009) to update the variational parameters independently on different computing nodes. This is implemented within a Map-Reduce framework, and the corresponding computational graph consists of a central node and many local nodes, i.e., a single-layer tree.

An alternative to sparse approximations is to distribute the computations by using independent local models. These local models typically require stationary kernels for a notion of “distance” and “locality”. Shen et al. (2006) used KD-trees to recursively partition the data space into a multi-resolution tree data structure, which scale GPs to $\mathcal{O}(10^4)$ training points. However, no solutions for variance predictions are provided, and the approach is limited to stationary kernels. Along the lines of exploiting locality, mixture-of-experts (MoE) models (Jacobs et al., 1991) have been applied to GP regression (Rasmussen & Ghahramani, 2002; Meeds & Osindero, 2006; Yuan & Neubauer, 2009). However, these models have not primarily been used to speed up GP regression, but rather to increase the expressiveness of the model, i.e., allowing for heteroscedasticity and

non-stationarity. Each local model possesses its own set of hyper-parameters to be optimised. Predictions are made by collecting the predictions of all local expert models, and weighting them using the responsibilities assigned by the gating network. In these MoE models, a Dirichlet process prior is placed on the multinomial responsibility vector of each local expert, which allows for data-driven partitioning on the fly. Inference in these models requires MCMC or variational approximations to assign data points to each GP expert. [Nguyen & Bonilla \(2014\)](#) sidestep this computationally demanding process and speed the GP-MoE model up by (i) fixing the number of GP experts, (ii) combining it with the pseudo-input sparse approximation by [Snelson & Ghahramani \(2006\)](#). This approach assigns data points to expert probabilistically using proximity information provided by stationary kernels and scales GPs to $\mathcal{O}(10^5)$ data points.

Product-of-GP-experts models (PoEs) sidestep the weight assignment problem of MoE models: Since PoEs multiply predictions made by independent GP experts, the overall prediction naturally weights the contribution of each expert. However, the model tends to be overconfident ([Ng & Deisenroth, 2014](#)). [Cao & Fleet \(2014\)](#) recently proposed a generalised PoE-GP model in which the contribution of an expert in the overall prediction can be increased/decreased individually. This model is often too conservative, i.e., it over-estimates variances. Furthermore, the model can be inconsistent in the sense that it does not necessarily fall back to the prior when leaving the range of the training data. The Bayesian Committee Machine (BCM) by [Tresp \(2000\)](#) can be considered a PoE-GP model and provides a consistent framework for combining independent estimators but suffers for weak experts.

In this paper, we exploit the fact that the computations of PoE models can be distributed amongst individual computing units and propose the generalised BCM (gBCM), a new family of hierarchical PoE-GP models that (i) includes the BCM ([Tresp, 2000](#)) and to some degree the generalised PoE-GP ([Cao & Fleet, 2014](#)) as special cases, (ii) provides consistent approximations of a full GP, (iii) scales to arbitrarily large data sets by parallelisation. Unlike sparse GPs our gBCM operates on the full data set but distributes the computational and memory load amongst a large set of independent computational units. The gBCM recursively recombines these independent computations to form an efficient distributed GP inference/training framework.

A key advantage of the gBCM is that all computations can be performed analytically, i.e., no sampling is required. With sufficient computing power our model can handle arbitrarily large data sets. We demonstrate that the gBCM can be applied to data sets of size $\mathcal{O}(10^7)$, which exceeds the typical data set sizes sparse GPs deal with by orders

of magnitude. However, even with limited resources, our model is practical: A GP with a million data points can be trained in less than half an hour on a laptop.

2. Problem Set-up and Objective

We consider a regression problem $y = f(\mathbf{x}) + \epsilon$, where $\mathbf{x} \in \mathbb{R}^D$. The Gaussian likelihood $p(y|f(\mathbf{x})) = \mathcal{N}(f(\mathbf{x}), \sigma_\epsilon^2)$ accounts for the i.i.d. measurement noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. The objective is to infer the latent function f from a training data set $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N, \mathbf{y} = \{y_i\}_{i=1}^N$. For small data set sizes N , a Gaussian process (GP) is a method of choice for probabilistic non-parametric regression. A GP is defined as a collection of random variables, any finite number of which is Gaussian distributed. A GP is fully specified by a mean function m and a covariance function k (kernel) with hyper-parameters ψ . Without loss of generality, we assume that the prior mean function is 0.

A GP is typically trained by finding hyper-parameters $\theta = \{\psi, \sigma_\epsilon\}$ that maximise the log-marginal likelihood

$$\begin{aligned} \log p(\mathbf{y}|\mathbf{X}, \theta) \\ \doteq -\frac{1}{2}(\mathbf{y}^T(\mathbf{K} + \sigma_\epsilon^2 \mathbf{I})^{-1} \mathbf{y} + \log |\mathbf{K} + \sigma_\epsilon^2 \mathbf{I}|), \end{aligned} \quad (1)$$

where $\mathbf{K} = k(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N}$ is the kernel matrix.

For a given set of hyper-parameters θ , a training set $\mathbf{X}, \mathbf{y}$ and a test input $\mathbf{x}_* \in \mathbb{R}^D$, the GP posterior predictive distribution of the corresponding function value $f_* = f(\mathbf{x}_*)$ is Gaussian with mean and variance given by

$$\mathbb{E}[f_*] = m(\mathbf{x}_*) = \mathbf{k}_*^T (\mathbf{K} + \sigma_\epsilon^2 \mathbf{I})^{-1} \mathbf{y}, \quad (2)$$

$$\text{var}[f_*] = \sigma^2(\mathbf{x}_*) = k_{**} - \mathbf{k}_*^T (\mathbf{K} + \sigma_\epsilon^2 \mathbf{I})^{-1} \mathbf{k}_*, \quad (3)$$

respectively, where $\mathbf{k}_* = k(\mathbf{X}, \mathbf{x}_*)$ and $k_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$.

Training requires the inversion and the determinant of $\mathbf{K} + \sigma_\epsilon^2 \mathbf{I}$ in (1), both of which scale in $\mathcal{O}(N^3)$ with a standard implementation. For predictions, we cache $(\mathbf{K} + \sigma_\epsilon^2 \mathbf{I})^{-1}$, such that the mean and variance in (2) and (3) require $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ computations, respectively. For $N > 10,000$ training and predicting become time-consuming procedures, which additionally require $\mathcal{O}(N^2 + ND)$ memory.

Throughout this paper, we assume that a vanilla GP is a good model for the latent function f . However, due to the data set size N the full GP is not applicable.

To scale GPs to large data sets with $N \gg \mathcal{O}(10^3)$, we address both the computational and the memory issues of full GPs by distributing the computational and memory loads to many individual (independent) computational units that only operate on subsets of the data. For this purpose, we devise a scalable hierarchical product-of-GP-experts model.

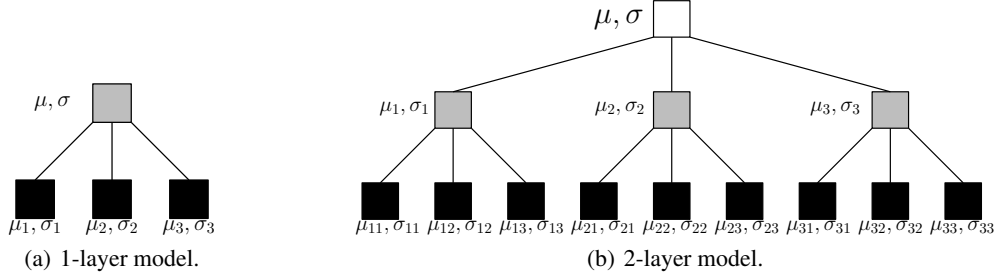


Figure 1. Hierarchical PoE model. Main computations are at the leaf nodes (black). All other nodes (linearly) recombine computations from their direct children, allowing for an arbitrarily deep computational graph.

3. Distributed Product-of-GP-Experts Models

Product-of-experts models (PoEs) are generally promising for parallelisation and distributed computing. In a PoE model, an overall computation is the product of many independent (smaller) computations, performed by “experts”. In our case, every expert is a GP that accesses only a part of the training data. In this paper, we consider a GP with a training data set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$. We partition the training data into M sets $\mathcal{D}^{(k)} = \{\mathbf{X}^{(k)}, \mathbf{y}^{(k)}\}$, $k = 1, \dots, M$, and use a GP on each of them as a (local) expert¹. Each GP expert performs computations (e.g., mean/variance predictions) conditioned on their respective training data $\mathcal{D}^{(k)}$. These (local) predictions are recombined by a parent node (see Fig. 1(a)), which subsequently may play the role of an expert at the next level of the model architecture. Recursive application of these recombinations results in a tree-structured computational graph with arbitrarily many layers, see Fig. 1(b).²

The PoE’s independence assumption leads to an approximation of the kernel matrix by a block-diagonal, which allows for efficient training and predicting and can be computed efficiently (time and memory) by parallelisation.

3.1. Training

Due to the independence assumption, the marginal likelihood $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$ in a PoE model factorises into the product of M individual terms, such that

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \approx \prod_{k=1}^M p_k(\mathbf{y}^{(k)}|\mathbf{X}^{(k)}, \boldsymbol{\theta}), \quad (4)$$

where each factor p_k is determined by the k th GP expert. For training the PoE model, we seek GP hyper-parameters $\boldsymbol{\theta}$ that maximise the corresponding log-marginal likelihood

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \approx \sum_{k=1}^M \log p_k(\mathbf{y}^{(k)}|\mathbf{X}^{(k)}, \boldsymbol{\theta}) \quad (5)$$

¹The notion of “locality” is misleading as our model does not require similarity measures induced by stationary kernels.

²We discuss different architecture choices in Section 4.

where M is the number of GP experts. The terms in (5) are independently computed and given by

$$\begin{aligned} \log p(\mathbf{y}^{(k)}|\mathbf{X}^{(k)}, \boldsymbol{\theta}) = & -\frac{1}{2} \mathbf{y}^{(k)T} (\mathbf{K}_{\boldsymbol{\theta}}^{(k)} + \sigma_{\epsilon}^2 \mathbf{I})^{-1} \mathbf{y}^{(k)} \\ & - \frac{1}{2} \log |\mathbf{K}_{\boldsymbol{\theta}}^{(k)} + \sigma_{\epsilon}^2 \mathbf{I}| + \text{const}, \quad (6) \end{aligned}$$

i.e., training can be easily parallelised. Since we assume that a vanilla GP is sufficient to model the latent function, all GP experts at the leaves of the tree-structured model are trained jointly and share a single set of hyper-parameters $\boldsymbol{\theta}$.

Computing the log-marginal likelihood terms in (6) requires the inversion and determinant of $\mathbf{K}_{\boldsymbol{\theta}}^{(k)} + \sigma_{\epsilon}^2 \mathbf{I}$, where $\mathbf{K}_{\boldsymbol{\theta}}^{(k)} = k(\mathbf{X}^{(k)}, \mathbf{X}^{(k)})$ is an $n_k \times n_k$ matrix, and n_k is the size of the data set associated with the k th GP expert. These computations can be performed in $\mathcal{O}(n_k^3)$ time with a standard implementation. Since $\sum_k n_k = N$, it holds that $n_k \ll N$ i.e., the size of the full data set. The memory consumption is $\mathcal{O}(n_k^2 + n_k D)$ for each individual model.

In (5), the number of parameters $\boldsymbol{\theta}$ to be optimised is relatively small since we do not consider additional variational parameters or inducing inputs that we optimise. The gradients of (6) with respect to $\boldsymbol{\theta}$ can be computed independently at all k nodes, which (i) allows for straightforward parallelisation, (ii) provides a significant speed-up of training compared to a full GP, (iii) leads to a low-dimensional $\mathcal{O}(D)$ optimisation problem compared to sparse GPs, which optimise inducing inputs or variational parameters.

3.2. Predictions

In the following, we assume that a set of M GP experts has been trained according to Section 3.1 and detail how the PoE (Ng & Deisenroth, 2014), the generalised PoE (Cao & Fleet, 2014) and the Bayesian Committee Machine (Tresp, 2000) combine predictions of the GP experts to form an overall prediction. Furthermore, we highlight strengths and weaknesses of these models, which motivates our generalised Bayesian Committee Machine (gBCM). The gBCM unifies many other models while providing additional flexibility, which can address the shortcomings of the PoE,

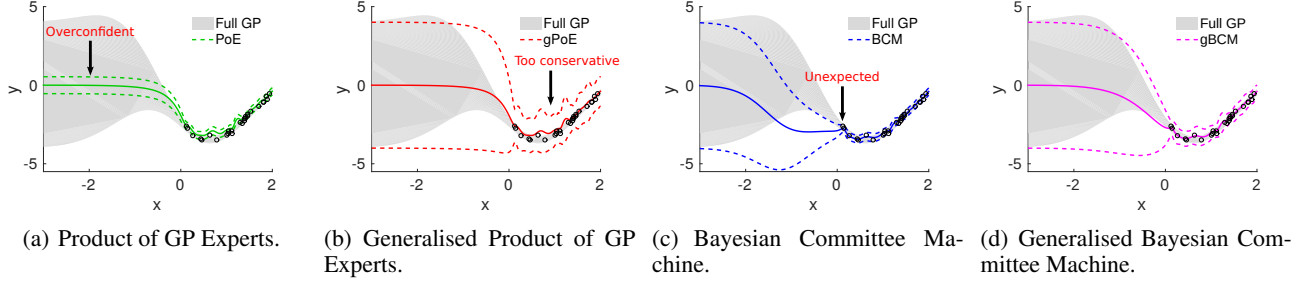


Figure 2. Predictions with four different product-of-experts models. The true GP to be approximated is shown by the shaded area, representing 95% of the GP confidence intervals. Training data is shown as black circles. Each GP expert was assigned two data points. (a): The standard PoE model does not fall back to the prior when leaving the training data. (b): The generalised PoE model falls back to the prior, but over-estimates the variances in the regime of the training data. (c): The BCM is close to the true GP in the range of the training data, but the predictive mean can suffer. (d): The generalised BCM is more robust than the (g)PoE and the BCM and produces reasonable predictions.

gPoE and the BCM. For illustration purposes, we focus on the model in Fig. 1(a), but many models generalise to an arbitrarily deep computational graph (see Section 4).

3.2.1. PRODUCT OF GP EXPERTS

The product-of-GP-experts model predicts a function value f_* at a corresponding test input $\mathbf{x}_*$ according to

$$p(f_*|\mathbf{x}_*, \mathcal{D}) = \prod_{k=1}^M p_k(f_*|\mathbf{x}_*, \mathcal{D}^{(k)}), \quad (7)$$

where M GP experts operate on different training data subsets $\mathcal{D}^{(k)}$. The M GP experts predict means $\mu_k(\mathbf{x}_*)$ and variances $\sigma_k^2(\mathbf{x}_*)$, $k = 1, \dots, M$, independently. The joint prediction $p(f_*|\mathbf{x}_*, \mathcal{D})$ is obtained by the product of all experts' predictions. The product of these Gaussian predictions is proportional to a Gaussian with mean and precision

$$\mu_*^{\text{poe}} = (\sigma_*^{\text{poe}})^2 \sum_k \sigma_k^{-2}(\mathbf{x}_*) \mu_k(\mathbf{x}_*), \quad (8)$$

$$(\sigma_*^{\text{poe}})^{-2} = \sum_k \sigma_k^{-2}(\mathbf{x}_*), \quad (9)$$

respectively. For $k = 1$, this model corresponds to the full GP we wish to approximate.

A strength of the PoE is that (a) the overall prediction $p(f_*|\mathbf{x}_*, \mathcal{D})$ is straightforward to compute, (b) there are no free weight parameters to be assigned to each prediction (unlike in MoE models). A shortcoming of this model is that with an increasing number of GP experts the predictive variances vanish (the precisions add up, see (9)), which leads to overconfident predictions, especially in regions without data. Thus, the PoE model is inconsistent in the sense that it does not fall back to the prior, see Fig. 2(a).

3.2.2. GENERALISED PRODUCT OF GP EXPERTS

The generalised product-of-experts model (gPoE) by Cao & Fleet (2014) adds the flexibility of increasing/reducing

the importance of experts. The predictive distribution is

$$p(f_*|\mathbf{x}_*, \mathcal{D}) = \prod_{k=1}^M p_k^{\beta_k}(f_*|\mathbf{x}_*, \mathcal{D}^{(k)}), \quad (10)$$

where the β_k weight the contributions of the experts. The predictive mean and precision are, therefore,

$$\mu_*^{\text{gpoe}} = (\sigma_*^{\text{gpoe}})^2 \sum_k \beta_k \sigma_k^{-2}(\mathbf{x}_*) \mu_k(\mathbf{x}_*), \quad (11)$$

$$(\sigma_*^{\text{gpoe}})^{-2} = \sum_k \beta_k \sigma_k^{-2}(\mathbf{x}_*), \quad (12)$$

respectively. An strength of the gPoE is that with $\sum_k \beta_k = 1$ the model falls back to the prior outside the range of the data. A weakness of this model is that in the range of the data, it over-estimates the variances, i.e., the predictions are generally too conservative, especially with an increasing number of GP experts. Fig. 2(b) illustrates these two properties. Additionally, the constraint $\sum_k \beta_k = 1$ requires us to know the β_k -values from all GP experts. In a single-layer computational graph, such as Fig. 1(a), this is possible since the central node possesses this global knowledge. However, for the computational graph in Fig. 1(b), this is no longer possible and would require sending information between (leaf) nodes to ensure $\sum_k \beta_k = 1$ —a communication overhead that is not desirable.

Cao & Fleet (2014) suggest to set β_k to the difference in the differential entropy between the prior and the posterior to determine the importance. In this paper, we do not consider this setting for the gPoE for two reasons: (i) $\sum_k \beta_k \neq 1$ leads to unreasonable error bars; (ii) Even with a normalisation, this setting does not allow for deep computational graphs. In fact, it only allows for a single-layer computational graph, such as Fig. 1(a). To apply the gPoE to a general computational graph we focus on the case that $\beta_k = 1/M$, where M is the number of GP experts (leaves in the trees in Fig. 1(b)). For $\beta_k = 1/M$ the predicted means in (8) and (11) are identical, but the precisions differ, see also Fig. 2(a) and 2(b) for a comparison.

3.2.3. BAYESIAN COMMITTEE MACHINE

A third model that falls in the category of PoE models is the Bayesian Committee Machine (BCM) proposed by [Tresp \(2000\)](#). Unlike the (g)PoE, the BCM explicitly incorporates the GP prior $p(f)$ when combining predictions (and not only at the leaves).

For two experts j, k and corresponding training data sets $\mathcal{D}^{(j)}, \mathcal{D}^{(k)}$, the predictive distribution is generally given by

$$p(f_* | \mathcal{D}^{(j)}, \mathcal{D}^{(k)}) \propto p(\mathcal{D}^{(j)}, \mathcal{D}^{(k)} | f_*) p(f_*), \quad (13)$$

where $p(f_*)$ is the GP prior over functions. The BCM makes the conditional independence assumption that $\mathcal{D}^{(j)} \perp\!\!\!\perp \mathcal{D}^{(k)} | f_*$. With (13) this yields

$$p(f_* | \mathcal{D}^{(j)}, \mathcal{D}^{(k)}) \stackrel{\text{BCM}}{\propto} p(\mathcal{D}^{(k)} | f_*) p(\mathcal{D}^{(j)} | f_*) p(f_*) \quad (14)$$

$$= \frac{p(\mathcal{D}^{(k)}, f_*) p(\mathcal{D}^{(j)}, f_*)}{p(f_*)} \quad (15)$$

$$\propto \frac{p_k(f_* | \mathcal{D}^{(k)}) p_j(f_* | \mathcal{D}^{(j)})}{p(f_*)}, \quad (16)$$

which is the PoE model in (7) divided by the GP prior.

In general, for M training data sets $\mathcal{D}^{(k)}$, $k = 1, \dots, M$, the BCM applies the above approximation repeatedly, leading to the BCM's posterior predictive distribution

$$p(f_* | \mathbf{x}_*, \mathcal{D}) = \frac{\prod_{k=1}^M p_k(f_* | \mathbf{x}_*, \mathcal{D}^{(k)})}{p^{M-1}(f_* | \mathbf{x}_*)} \quad (17)$$

The $(M-1)$ -fold division by the prior plays the role of a correction and is the decisive difference between the BCM and the PoE model in (7) and leads to the BCM's predictive mean and precision are

$$\mu_*^{\text{bcm}} = (\sigma_*^{\text{bcm}})^2 \sum_{k=1}^M \sigma_k^{-2}(\mathbf{x}_*) \mu_k(\mathbf{x}_*) \quad (18)$$

$$(\sigma_*^{\text{bcm}})^{-2} = \sum_{k=1}^M \sigma_k^{-2}(\mathbf{x}_*) + (1-M)\sigma_{**}^{-2} \quad (19)$$

respectively, where σ_{**}^{-2} is the prior precision of $p(f_*)$.

The repeated application of Bayes's theorem and the corresponding $(M-1)$ -fold division by the prior in (17) leads to a "correction" term in (19) that ensures a consistent model that falls back to the prior. The error bars of the BCM within the range of the data are usually good, but it is possible to "break" the BCM when only few data points are assigned to each GP expert. In Fig. 2(c), we see that the posterior mean suffers from weak experts when leaving the data (around $x = 0$).³

3.2.4. GENERALISED BAYESIAN COMMITTEE MACHINE

In this section, we propose the generalised Bayesian Committee Machine (gBCM), a unified model that (a) includes

the gPoE and BCM as special cases, (b) yields consistent predictions, (c) can be implemented on a distributed computing architecture.

Inspired by the gPoE in (10), our gBCM is a BCM with the added flexibility of increasing/decreasing an expert's importance. In particular, the gBCM's predictive distribution is given by

$$p(f_* | \mathbf{x}_*, \mathcal{D}) = \frac{\prod_{k=1}^M p_k^{\beta_k}(f_* | \mathbf{x}_*, \mathcal{D}^{(k)})}{p^{-1+\sum_k \beta_k}(f_* | \mathbf{x}_*)}, \quad (20)$$

where the predictive mean and precision are given as

$$\mu_*^{\text{gbcm}} = (\sigma_*^{\text{gbcm}})^2 \sum_k \beta_k \sigma_k^{-2}(\mathbf{x}_*) \mu_k(\mathbf{x}_*) \quad (21)$$

$$(\sigma_*^{\text{gbcm}})^{-2} = \sum_{k=1}^M \beta_k \sigma_k^{-2}(\mathbf{x}_*) + (1 - \sum_{k=1}^M \beta_k) \sigma_{**}^{-2}, \quad (22)$$

respectively. The derivation of the gBCM in (20) is analogous to the BCM's derivation in (14)–(16).

The gBCM combines the flexibility of the generalised PoE with the appropriate Bayesian treatment of the BCM, which leads to the correction term $(1 - \sum_k \beta_k) \sigma_{**}^{-2}$ in the precision in (22). This correction term ensures that the predictive variance falls back to the prior when leaving the data. Note that we no longer require $\sum_k \beta_k = 1$ to ensure this, which will facilitate computational graphs with multiple layers. The gPoE from Section 3.2.2 and the BCM from Section 3.2.3 are recovered for $\beta_k = 1/M$ and $\beta_k = 1$, respectively. For $\beta_k = 0$, the gBCM is identical to the GP prior, and for $\beta_k = 1$ but without the correction, the gBCM recovers the PoE from Section 3.2.1

The parameters β_k control not only the importance of the individual experts, but they also control how strong the influence of the prior is. Assuming each GP expert is a good predictive model, we would set $\beta_k = 1$ for all k , such that we retain the BCM. If the quality of the GP experts is weak, e.g., data is noisy and the experts' data sets $\mathcal{D}^{(k)}$ are small, β_k allows us to weaken the experts' votes and to robustify the predictive model by putting more weight on the prior. Therefore, we follow the suggestion of [Cao & Fleet \(2014\)](#) and choose the β_k according to the predictive power of each expert at $\mathbf{x}_*$. Specifically, we use the difference in differential entropy between the prior $p(f_* | \mathbf{x}_*)$ and the posterior $p(f_* | \mathbf{x}_*, \mathcal{D}^{(k)})$. This quantity can be computed efficiently and is given as $\beta_k = \frac{1}{2}(\log \sigma_{**}^2 - \log \sigma_k^2(\mathbf{x}_*))$, where σ_{**}^2 is the prior variance and $\sigma_k^2(\mathbf{x}_*)$ is the predictive variance of the k th expert. Fig. 2(d) illustrates that for this choice of β_k , the gBCM expresses more uncertainty about the learned model than the BCM: Due to the adaptive influence of the prior in (21)–(22), the variances within the range of the data (black circles) are on the conservative side, but the predictive mean no longer suffers from the dominant "kink" at

³Here, each expert was assigned only two data points.

around $x = 0$ compared to the BCM in Fig. 2(c). Overall, this model provides more reasonable predictions than any other model in Fig. 2.

4. Distributed Computations

In the following, we show that for a given number M of GP experts, the gBCM can be implemented in different computational graphs while providing identical predictions. For instance, with 32 experts, we show that a single-layer computational graph with 32 experts and one central node, see Fig. 1(a), is equivalent to a two-layer computational graph with 32 experts, 8 parent nodes (each of which is responsible for 4 GP experts), and one central node. This property can be exploited in large distributed system or to balance the communication between computing units.

In a single-layer model as shown in Fig. 1(a) the gBCM predictions in (20) can be constructed by a gPoE (numerator) combining predictions of GP experts, followed by a correction via the prior (denominator). Let us consider a two-layer model as illustrated in Fig. 1(b) with L grey nodes. Each grey node k is responsible for L_k GP experts (black nodes). Each GP expert k_i , $k_i = 1, \dots, M$, computes the (weighted) predictive distribution $p_{k_i}^{\beta_{k_i}}(f_* | \mathcal{D}^{(k_i)})$. These GP expert predictions are then combined by the corresponding L grey nodes to $p_k^{\beta_k}(f_* | \mathcal{D}^{(k)})$, where $\beta_k = \sum_i \beta_{k_i}$ is the overall weight of the subtree following the k th node and $\mathcal{D}^{(k)} = \bigcup_i \mathcal{D}^{(k_i)}$. The overall prediction at the top node is

$$\frac{\prod_{k=1}^M p_k^{\beta_k}(f_* | \mathcal{D}^{(k)})}{p^{\sum_k \beta_k - 1}(f_*)} = \frac{\prod_{k=1}^L \prod_{i=1}^{L_k} p_{k_i}^{\beta_{k_i}}(f_* | \mathcal{D}^{(k_i)})}{p^{\sum_k \beta_k - 1}(f_*)} \quad (23)$$

where we accounted for the $(\sum_k \beta_k - 1)$ -fold correction through the prior. This computation can be obtained by a gPoE model for the grey nodes, followed by a PoE (red) and a correction (blue) in (23). Hence, for a given number of GP experts, the gBCM predictions can be equivalently realised in a single and two-layer computational graph.

This can be generalised further to an arbitrarily deep computational graph, whose general implementation structure is shown in Fig. 3. The GP experts at the leaves compute their individual means, variances and confidence values β_i . The next layer consists of gPoE models, which compute the weighted means and variances according to (8) and (12), respectively (plus their overall weights $\beta_k = \sum_{i \in \text{children}} \beta_{k_i}$, which are passed on to the next-higher level). The gPoE is followed by an arbitrary number of PoE models, which compute means and precisions accord-

ing to (8) and (9), respectively (plus their overall weights $\beta_k = \sum_{j \in \text{children}} \beta_{k_j}$, which are passed on to the next-higher level). The top layer accounts for the prior (blue term in (23)), which uses all the β_k from the subtrees starting at its children to compute the overall mean and precision according to (21) and (22), respectively.

Hence, for a given number of GP experts, there are many equivalent computational graphs for the gBCM. This allows us to choose the gBCM implementation, which works best with the computing infrastructure available: Shallow graphs do not cause much overall traffic. However, they are more vulnerable to communication bottlenecks at the central node since it has a large number of connections. Deeper computational graphs cause more overall communication, but the gBCM tree has a smaller branching factor. In practice, for a set of computational graphs we computed the time it requires to compute the gradient of the marginal likelihood and chose the “fastest” gBCM architecture.

5. Experiments

We empirically assess three aspects: (1) The required training time of our model, (2) the approximation error induced by the block-diagonal approximation of the kernel matrix (due to the independence assumption of the experts), (3) a comparison with state-of-the-art large-scale GP methods.

In all experiments, we chose the standard squared exponential kernel with automatic relevance determination and a Gaussian likelihood. Moreover, we assigned training data to experts *randomly* for two reasons: First, we demonstrate that our models do not need locality information; second, random assignment is very fast compared to clustering methods, e.g., KD-trees.

5.1. Training Time for Large Data Sets

To evaluate the training time for the gBCM, we measured the amount of time required to compute the log-marginal likelihood and its gradient with respect to the kernel hyper-parameters. Since the model is trained using LBFGS, the overall training time is proportional to the time it takes to compute the log-marginal likelihood and its gradient. For this evaluation, we chose a computer architecture of 64 nodes with four cores each. Furthermore, we chose a three-layer computational graph with varying branching factors. For data sets of $\leq 2^{20}$ data points the GP experts possessed 512 data points each, for data set sizes of $> 2^{20}$, we chose the number of data points per node to be 128.

Fig. 4 shows the time required for computing the log-marginal likelihood and its gradient with respect to the hyper-parameters. The horizontal axis shows the size of the training set (logarithmic scale), the left vertical axis shows the computation time in seconds (logarithmic scale) for our

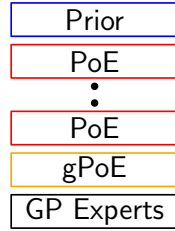


Figure 3. General architecture for the gBCM with arbitrarily many layers.

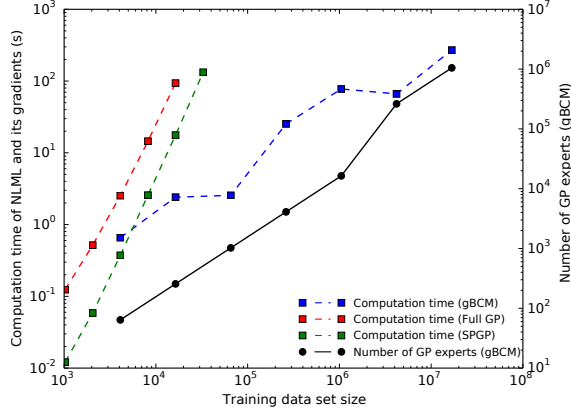


Figure 4. Computing time for the log-marginal likelihood and its gradient with respect to the kernel hyper-parameters as a function of the size of the training data. The gBCM (and equally the PoE, gPoE and the BCM) scales favourably to large-scale data sets. With an increasing number of GP experts (but fixed computational resources), the gBCM scales to more than 10^7 data points.

model (gBCM, blue-dashed) and a full GP (red-dashed) and a sparse GP with inducing inputs (Snelson & Ghahramani, 2006) (green-dashed). For the sparse GP model, we chose the number M of inducing inputs to be 10% of the size of the training set, i.e., the computation time is of the order of $\mathcal{O}(NM^2) = \mathcal{O}(N^3/100)$, which offsets the curve of the full GP. Taking even fewer inducing inputs (e.g., 1% or 0.1% of the data) would push the sparse approximation towards $\mathcal{O}(10^5)$ data points. However, this can only be done if the data set possesses a high degree of redundancy. The right vertical axis shows the number of GP experts (black-solid) amongst which we distribute the computation. While the training time of the full GP becomes impractical at data set sizes of about 10,000, the sparse GP model can be reasonably trained up to 50,000 data points.⁴ The computational time required for the gBCM to compute the marginal likelihood and gradients is significantly lower than that of the full GP, and we scaled it up to $2^{24} \approx 1.7 \times 10^7$ training data points, which required about the same amount of time (≈ 230 s) for training a full GP with $2^{14} \approx 1.6 \times 10^4$ and a sparse GP with $2^{15} \approx 3.2 \times 10^4$ data points. The figure shows that for any problem size we can find a computational graph that allows us to train the model within a reasonable amount of time.

Even if a big computing infrastructure is not available our model is useful in practice: We performed a full training cycle of the gBCM with 10^6 data points on a standard laptop in about 20 minutes. This is also a clear indicator that the memory consumption of the gBCM is relatively small.

⁴In this comparison we did not include any computational overhead for learning the inducing inputs, which is often time consuming.

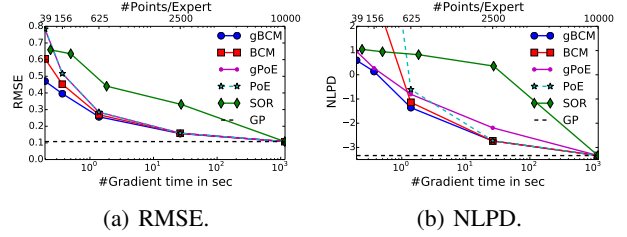


Figure 5. Performance as a function of training time (lower horizontal axes) and the number of data points per expert (upper horizontal axes). (a) RMSE, (b) NLPD. Standard errors (not displayed) are smaller than 10^{-2} .

5.2. Empirical Approximation Errors

To evaluate the approximation quality of the models introduced in Section 3, we compare them with a full GP on the Kin40K data set. The data set represents the forward dynamics of an 8-link all-revolute robot arm. The goal is to predict the distance of the end-effector from a target, given the joint angles of the eight links as features. The Kin40K data set consists of 10,000 training points and 30,000 test points. We use the same split into training and test data as (Seeger et al., 2003), (Lázaro-Gredilla et al., 2010), and (Nguyen & Bonilla, 2014).

We considered two baselines: a full GP and the subset-of-regressors (SOR) approximation, which uses a random subset of the full training data set to train a sparse GP. Taking training time into account, Chalupka et al. (2013) identified the SOR method as a method of choice when it comes to efficient sparse approximations. All models (full GP, PoE, gPoE, BCM, gBCM, SOR) used the hyper-parameters from the full GP, such that we could assess the approximation quality appropriately. For every model, we took the time for computing the gradient of the marginal likelihood (training is proportional to this amount of time). We selected the number of regressors for SOR, such that the gradient computations take approximately the same time as the gradient computation with the distributed models.

Experiments were repeated 10 times to average out the effect of the random assignment of data points to experts and the selection of the subset of training data for the SOR approximation. For this experiment, we used a Virtual Machine with 16 3 GHz cores and 8 GB RAM.

Fig. 5 shows the average performance (RMSE and negative log predictive density NLPD per data point) of all models as a function of (a) the time it takes to compute the gradient of the marginal likelihood (b) the number of training data point per GP expert. The full GP (black, dashed) shows the desired performance, but requires 1162 seconds for the gradient computation. The performances of the (g)BCM and the (g)POE have been evaluated using

Table 1. US Flight Data Set. (Dist)-SVIGP results are reported from the respective papers. Results with $^+$ are the best solution found.

	700K/100K		2M/100K		5M/100K	
	RMSE	NLPD	RMSE	NLPD	RMSE	NLPD
SVIGP	33.0	—	—	—	—	—
Dist-SVIGP	33.0⁺	—	35.3 ⁺	—	—	—
gBCM	27.8	8.7	34.6	8.5	37.1	9.1
BCM	35.3	13.0	48.8	15.5	57.6	18.1
gPoE	29.3	8.1	35.3	8.5	38.4	8.6
PoE	29.3	11.4	35.3	18.5	38.4	14.3

256, 64, 16, 4, and 1 expert with 39, 156, 625, 2500 and 10000 data points per expert, respectively. In Fig. 5(a), the gBCM consistently outperforms all other methods, where SOR is substantially worse than all hierarchical models. The NLPD in Fig. 5(b) allows us to make some more conclusive statements: While the gBCM again outperforms all other methods, the BCM and the PoE’s performances suffer when only a small number of data points is assigned to the GP experts. The PoE suffers from variance underestimation (see Fig. 2(a)) whereas the BCM cannot appropriately combine “weak” experts (see Fig. 2(c)). The SOR method does not work well, even with 2500 data points. Overall, the gBCM provides an enormous training speed-up compared to a full GP, with a significantly better predictive performance than SOR.⁵

5.2.1. AIRLINE DELAYS (US FLIGHT DATA)

In the following, we assess the performance of the distributed algorithms PoE, gPoE, BCM and gBCM on a large-scale non-stationary data set reporting flight arrival and departure times for every commercial flight in the US from January to April 2008⁶. This data set contains extensive information about almost 6 million flights. We followed the procedure described by Hensman et al. (2013)⁷ to predict the flight delay (in minutes) at arrival: We selected the first P data points to train the model and the following 100,000 to test it. We chose the same eight input variables $\mathbf{x}$ as Hensman et al. (2013): age of the aircraft, distance that needs to be covered, airtime, departure and arrival times, day of the week and month, month. This data set has been evaluated by Hensman et al. (2013) and Gal et al. (2014), both of which use sparse variational GP (SVIGP) methods to deal with this training set size.

Using a VM with 16 3 GHz cores and 16 GB RAM, we conducted experiments with $P = 7 \times 10^5$, $P = 2 \times 10^6$ and $P = 5 \times 10^6$ where we chose 4096, 8192 and 32768 experts corresponding to 170, 244 and 152 training data points per expert, respectively. The computation of the marginal like-

lihoods required 19, 149, and 171 seconds, respectively.⁸ The computational graphs were (16-16-16), (16-16-16-2), and (8-8-8-8-8), respectively, where each number denotes the branching factor at the corresponding level in the tree. Every single experiment consisted of a full training and testing cycle. Training the distributed GP models normally required 30–100 line searches.

Table 1 reports the performance (RMSE and NLPD) of various large-scale GP methods for the flight data set. The results for (Dist)SVIGP are taken from Hensman et al. (2013) and Gal et al. (2014). Since (Dist)-SVIGP is difficult to optimise, Gal et al. (2014) report only their best results (indicated by $^+$, whereas we report an average of *all* experiments conducted (potential local optima included).

The data set exhibits the property that the 700K/100K data set is more stationary than the 2M/100K and 5M/100K data sets. Therefore, we observe a decreasing performance although we include more training data. This effect has already been reported before by Gal et al. (2014).

The standard errors (not shown in Table 1) of the gBCM and the gPoE are consistently below 0.3, whereas the BCM and the PoE suffered from a few outliers, which is also indicated by the relatively large NLPD values. Compared to the (Dist)-SVIGP on the 700K data set, the gBCM, gPoE and PoE perform significantly better in RMSE. The table highlights the weaknesses of the PoE (under-estimation of the variance) and the BCM (problems with weak experts) very clearly. The property of the gPoE (too conservative) is a bit hidden: Although the RMSE of the gPoE is consistently worse than that of the gBCM, its NLPD tends to be a bit lower. The NLPD values of the gBCM and the gPoE are relatively consistent across all three experiments.

6. Conclusion

We presented the generalised Bayesian Committee Machine (gBCM), a conceptually straightforward, but effective, hierarchical product-of-GP experts model that scales Gaussian processes to (in principle) arbitrarily large data sets. The gBCM addresses shortcomings of other hierarchical models by appropriately incorporating the GP prior when combining predictions. The gBCM parallelises computations by distributing them amongst independent computational units. A recursive and closed-form recombination of these independent computations results in a practical model that is both computationally and memory efficient. Training and predicting is independent of the computational graph. Thus, the gBCM can be used on heterogeneous computing infrastructures, ranging from laptops to large clusters: Training a gBCM with a million data points takes less than 30 minutes on a laptop. With more com-

⁵More details are provided in Appendix A.

⁶<http://stat-computing.org/dataexpo/2009/>

⁷Thanks to J Hensman for the pre-processing script.

⁸All experiments can also run on a MacBook Air (2012).

puting power training the gBCM with more than 10^7 data points can be done in a few hours. Compared to the most recent sparse GP approximations, our model performs very well, learns fast, requires little memory, and does not suffer from high-dimensional optimisation.

Future work will focus on approximation bounds, relaxation of the independence assumption of individual computing nodes, and large-scale GP classification.

References

- Brochu, Eric, Cora, Vlad M., and de Freitas, Nando. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. Technical Report TR-2009-023, Department of Computer Science, University of British Columbia, 2009.
- Cao, Yanshuai and Fleet, David J. Generalized Product of Experts for Automatic and Principled Fusion of Gaussian Process Predictions. <http://arxiv.org/abs/1410.7827>, October 2014.
- Chalupka, Krzysztof, Williams, Christopher K. I., and Murray, Iain. A Framework for Evaluating Approximate Methods for Gaussian Process Regression. *Journal of Machine Learning Research*, 14:333–350, February 2013.
- Cressie, Noel A. C. *Statistics for Spatial Data*. Wiley-Interscience, 1993.
- Deisenroth, Marc P., Fox, Dieter, and Rasmussen, Carl E. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:408–423, February 2015.
- Gal, Yarin, van der Wilk, Mark, and Rasmussen, Carl E. Distributed Variational Inference in Sparse Gaussian Process Regression and Latent Variable Models. In *Advances in Neural Information Processing Systems*. 2014.
- Hensman, James, Fusi, Nicolò, and Lawrence, Neil D. Gaussian Processes for Big Data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2013.
- Jacobs, Robert A., Jordan, Michael I., Nowlan, Steven J., and Hinton, Geoffrey E. Adaptive Mixtures of Local Experts. *Neural Computation*, 3:79–87, 1991.
- Jones, Donald R., Schonlau, Matthias, and Welch, William J. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, December 1998. ISSN 0925-5001.
- Krause, Andreas, Singh, Ajit, and Guestrin, Carlos. Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *Journal of Machine Learning Research*, 9:235–284, February 2008.
- Lawrence, Neil. Probabilistic Non-linear Principal Component Analysis with Gaussian Process Latent Variable Models. *Journal of Machine Learning Research*, 6: 1783–1816, November 2005.
- Lázaro-Gredilla, Miguel, Quionero-Candela, Joaquin, Rasmussen, Carl E., and Figueiras-Vidal, Aníbal R. Sparse Spectrum Gaussian Process Regression. *Journal of Machine Learning Research*, 11:1865–1881, June 2010.
- Luttinen, Jaakko and Ilin, Alexander. Efficient Gaussian Process Inference for Short-Scale Spatio-Temporal Modeling. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 22 of *JMLR W&CP*, pp. 741–750, 2012.
- Meeds, Edward and Osindero, Simon. An Alternative Infinite Mixture of Gaussian Process Experts. In *Advances in Neural Information Processing Systems*, volume 18, pp. 883. The MIT Press, 2006.
- Ng, Jun Wei and Deisenroth, Marc P. Hierarchical Mixture-of-Experts Model for Large-Scale Gaussian Process Regression. <http://arxiv.org/abs/1412.3078>, December 2014.
- Nguyen, Trung V. and Bonilla, Edwin V. Fast Allocation of Gaussian Process Experts. In *Proceedings of the International Conference on Machine Learning*, 2014.
- Quiñero-Candela, Joaquin and Rasmussen, Carl E. A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6 (2):1939–1960, 2005.
- Rasmussen, Carl E. and Ghahramani, Zoubin. Infinite Mixtures of Gaussian Process Experts. In *Advances in Neural Information Processing Systems*, pp. 881–888. MIT Press, 2002.
- Rasmussen, Carl E. and Williams, Christopher K. I. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA, 2006.
- Seeger, Matthias, Williams, Christopher K. I., and Lawrence, Neil D. Fast Forward Selection to Speed up Sparse Gaussian Process Regression. In *Ninth International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2003.

- Shen, Yirong, Ng, Andrew Y., and Seeger, Matthias. Fast Gaussian Process Regression Using KD-Trees. In *Advances in Neural Information Processing Systems*, 2006.
- Snelson, Edward and Ghahramani, Zoubin. Sparse Gaussian Processes using Pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pp. 1257–1264. The MIT Press, Cambridge, MA, USA, 2006.
- Titsias, Michalis K. Variational Learning of Inducing Variables in Sparse Gaussian Processes. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, 2009.
- Tresp, Volker. A Bayesian Committee Machine. *Neural Computation*, 12(11):2719–2741, 2000.
- Williams, Christopher K.I. and Seeger, Matthias. Using the Nyström Method to Speed up Kernel Machines. In *Advances in Neural Information Processing Systems*, pp. 682–688, 2001.
- Yuan, Chao and Neubauer, Claus. Variational Mixture of Gaussian Process Experts. In *Advances in Neural Information Processing Systems*, pp. 1897–1904, 2009.

A. Detailed Kin40K Results

Table 2 details the average performances of the gBCM, BCM, gPoE, PoE and SOR including the time required to compute the gradient of the marginal likelihood with respect to the kernel hyper-parameters. Note the substantial speedup of the distributed models compared to the full GP and the performance improvement compared with the SOR approximation. When the number of data points per expert becomes small the BCM and the PoE run into problems. Although the RMSE of the gPoE is equally bad as the PoE’s, the NLPD is substantially better since the gPoE “protects itself” by predictions that are generally too conservative (inflated predictive variances).

B. Implementation Details

In the following, we provide a few implementation details that are important for our python implementation.

B.1. True Concurrency in Python

A known issue of the CPython interpreter, which we use in our implementation, is the lack of true concurrency using the in-built threading library. Due to the *Global Interpreter Lock* (GIL, which is implemented in the interpreter because Python’s memory management is not thread safe), only a single thread of Python code can be executed at any point in time. Therefore, the use of threads in the Python context

Table 2. Performance on the Kin40K data set and the time required to compute the gradient of the log-marginal likelihood, averaged over 10 experiments. All standard errors (not shown) are below 10^{-2} .

	RMSE	NLPD	Grad. Time
GP	0.11	−3.11	1161.0 s
gBCM (4 experts)	0.16	−2.73	26.3 s
BCM (4 experts)	0.16	−2.73	26.3 s
gPoE (4 experts)	0.16	−2.19	26.3 s
PoE (4 experts)	0.16	−2.71	26.3 s
SOR (2500 regressors)	0.35	0.36	27.2 s
gBCM (16 experts)	0.26	−1.36	1.39 s
BCM (16 experts)	0.27	−1.14	1.39 s
gPoE (16 experts)	0.28	−0.80	1.39 s
PoE (16 experts)	0.28	−0.61	1.39 s
SOR (900 regressors)	0.55	0.83	1.83 s
gBCM (64 experts)	0.40	0.14	0.37 s
BCM (64 experts)	0.45	3.37	0.37 s
gPoE (64 experts)	0.52	0.27	0.37 s
PoE (64 experts)	0.52	10.97	0.37 s
SOR (500 regressors)	0.65	0.95	0.50 s
gBCM (256 experts)	0.47	0.60	0.20 s
BCM (256 experts)	0.61	11.24	0.20 s
gPoE (256 experts)	0.78	0.97	0.20 s
PoE (256 experts)	0.78	58.0	0.20 s
SOR (300 regressors)	0.70	1.05	0.24 s

only provides logical concurrency in terms of the flow of programs, but not true simultaneous computations.

There exists a workaround for the true concurrency problem in Python, via the use of processes instead of threads to perform simultaneous computations. In the POSIX model, threads are lightweight units of computations belonging to the same process, thus, sharing the same memory space. Processes have their own memory space and come with increased system overheads compared to threads. However, on Linux (which we use for this implementation), the creation of duplicate processes (forking) does not incur large memory costs since Linux implements a copy-on-write model. This means that when a process forks into two, the memory is not copied, unless the new process attempts to modify it. In the context of our implementation, we make no modification to the training data, which is shared amongst all child-GPs. In terms of the memory usage, each child-GP only needs to compute its own kernel matrix and the corresponding Jacobian matrix per hyperparameter, which have no interaction with any other child-GP. Therefore, computing each child-GP using a separate process does not incur any large, redundant memory costs that would not be present in a true concurrency model implemented by native threads.

B.2. Memory Management

This duplication of memory can be prevented, since the main training data set is shared and does not get modified. Therefore, each GP expert can have access to its training data subset without copying any of it. To do this, we implement a `DataSet` class, which manages the data. There will be a single instance of this class, which holds all the data in the full training set. We can create additional `DataSet` instances by invoking the `subset` method on the `DataSet` object. We specify a set of integers corresponding to the indices of the data points (which we require in the subset) in the main data set. A new `DataSet` instance is then created with no actual data, but a list of indices, and a reference to the main `DataSet` object as its superset. The only exception to this occurs when distributed computing is used, in which case, the subset of data that is required at a different machine on the network is copied from one memory space to another, and a new ‘main’ `DataSet` object is created.

B.3. Remote Object Management

Managing network communications for a distributed system poses a large challenge since there are many details one has to manage (e.g., retrying failed message transmissions, timeouts). This adds much complexity to the system and may cause unnecessary failures in the system if not properly implemented. The remote procedure call (RPC) protocol enables all of the issues at the network layer to be abstracted, and allows us to use objects on a remote host with the same interface as local objects. This is done by having a dummy object on the client (‘client stub’), which provides the interface of the object. However, instead of executing methods on the client, it invokes the required computation on a remote host (‘server’), and (upon completion of the remote computation) receives the results from the remote host and returns. With this interface, the code generally remains simple and readable. In our implementation, we use the Pyro4 library, which implements remote procedural calls in Python. RPC is used for communication between nodes in the gBCM tree. Furthermore, the classes in our object-oriented implementation can be set up as remote objects (as required). This allows the exact functionality of all classes for both local and distributed computation.