A Tutorial on Sparse Variational Gaussian Processes

P.L.Green
Institute for Risk and Uncertainty
School of Engineering
University of Liverpool
Liverpool, L69 7ZF
United Kingdom

p.l.green@liverpool.ac.uk
https://www.liverpool.ac.uk/engineering/staff/peter-green/

May 22, 2019

Acknowledgments

Thank you to Ryan Jackson (University of Liverpool, UK) and Kaikai Zhao (Indiana University, Bloomington, USA) for their feedback!

1 Introduction

This document is supposed to provide a tutorial-style introduction to Sparse Variational Gaussian Processes which are typically able to analyse larger data sets than standard Gaussian Processes. As with our other tutorials, the current document is accompanied by some example Python code. The code is heavily commented and is designed to help those new to the topic. It is recommended that this tutorial should only be studied once a basic understanding of standard Gaussian Processes has been established. We offer such a tutorial here: https://github.com/plgreenLIRU/Gaussian-Process-regression-tutorial.

2 Tutorial Structure

The tutorial is structured as follows. A general overview of the approach behind Sparse Variational Gaussian Processes (herein simply referred to as 'Sparse GPs') as well as a summary of key results is given in Section 3. Example code is then described immediately afterwards, in Section 4. At this point the reader may wish to stop, however, for those wanting more information, more detailed derivations are then given in Section 5 before a summary of some interesting references is given in Section 6.

3 Overview

In this tutorial, we specifically consider regression problems where we define $\boldsymbol{y} = (y_1, ..., y_N)^T$ as noisy observations of a function, f, at inputs $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, such that our full set of training data is $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N)$. We assume that our observations are made according to the noise model

$$y_i = f(\boldsymbol{x}_i) + \epsilon, \qquad \epsilon \sim \mathcal{N}(\epsilon; 0, \sigma^2) \tag{1}$$

Using the notation $f_i \equiv f(\boldsymbol{x}_i)$, the vector of true function values is written as $\boldsymbol{f} = (f_1, ..., f_N)^T$. If N is large, computational cost often prevents us from using 'standard' GPs. With the Variational Sparse GP described in [1] and [2], we train our GP on a smaller set of data, consisting of observations at the set of inputs $\boldsymbol{X}_m = \{\boldsymbol{x}_{m,1}, ..., \boldsymbol{x}_{m,M}\}$ (where, for this tutorial, we will assume that $\boldsymbol{X}_m$ is a subset of $\boldsymbol{X}$). The aim is to make the Sparse GP 'look like' a standard GP that had been trained on the full set of training data as closely as possible. This approach therefore assumes that there is some redundancy in our data - it builds on the idea that we only really need a small subset of 'important' points to get a reasonable estimate of the true function, f.

Stated briefly, we proceed in 3 key steps. These are described in the boxes below:

Step 1: Define the optimum inducing points.

In the following, we define the set of inputs $\boldsymbol{X}_m$ as 'inducing points'. As a first step, we have to define what we mean by the *optimum* set of inducing points - in other words, we ask the question: what do we use to help us choose the 'best' $\boldsymbol{X}_m$?

Say $\boldsymbol{f}_m$ is the vector of true function values at the points in $\boldsymbol{X}_m$ (such that $f_{m,i} \equiv f(\boldsymbol{x}_{m,i})$). We begin by stating that the optimum choice of $\boldsymbol{X}_m$ would lead to $\boldsymbol{f}_m$ such that

$$p(\boldsymbol{y} \,|\, \boldsymbol{f}_m) = p(\boldsymbol{y} \,|\, \boldsymbol{f}, \boldsymbol{f}_m) \tag{2}$$

(where dependence on $\boldsymbol{X}$ and $\boldsymbol{X}_m$ isn't written for notational simplicity). Essentially, if we pick the best possible $\boldsymbol{f}_m$, then knowing $\boldsymbol{f}$ wouldn't change the probability of witnessing the observations $\boldsymbol{y}$. It turns out that enforcing this condition implies the following factorisation of our posterior probability distribution over $\boldsymbol{f}$ and $\boldsymbol{f}_m$:

$$p(\boldsymbol{f}, \boldsymbol{f}_m \,|\, \boldsymbol{y}) = p(\boldsymbol{f}_m \,|\, \boldsymbol{y}) p(\boldsymbol{f} \,|\, \boldsymbol{f}_m) \tag{3}$$

From this we conclude that a good choice of $\boldsymbol{X}_m$ is one that leads to equation (3) being true.

Step 2: Variational Bayes framework

Briefly stated, variational Bayesian approaches involve the construction of a *variational distribution*, q, that closely approximates some distribution of interest. A good choice of variational distribution is usually defined as that which minimises the *KL divergence* between q and the distribution of interest (see [3][4] for more information).

In our case, we choose to approximate the posterior with the PDF, $p(\boldsymbol{f}, \boldsymbol{f}_m \,|\, \boldsymbol{y})$, with

$$q(\boldsymbol{f}, \boldsymbol{f}_m) = \phi(\boldsymbol{f}_m) p(\boldsymbol{f} \,|\, \boldsymbol{f}_m) \tag{4}$$

where $\phi(\boldsymbol{f}_m)$ is a PDF that is currently unknown. We hypothesise that, if we pick the optimum $\boldsymbol{f}_m$, then the posterior will factorise as shown in equation (3) and it would

then be possible to find the distribution, $\phi(\boldsymbol{f}_m)$, that sets the KL divergence between $p(\boldsymbol{f}, \boldsymbol{f}_m \,|\, \boldsymbol{y})$ and $q(\boldsymbol{f}, \boldsymbol{f}_m)$ equal to zero. Conveniently, we know that minimising the KL divergence is equivalent to maximising a lower bound on the log marginal likelihood, which we denote as L (again, see [3][4] for more details).

Step 3: Maximise a lower bound

Rather than maximising L directly it turns out that, by instead maximising a lower bound on L, we can still select some 'good' inducing points while avoiding having to find the variational distribution $\phi(\boldsymbol{f}_m)$. This lower bound on L is denoted L_{lb}, and is given by

$$L_{lb} = \log\left[\mathcal{N}(\boldsymbol{y}; \boldsymbol{0}, \boldsymbol{I}_n\sigma^2 + \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn})\right] - \frac{1}{2\sigma^2}\operatorname{Tr}\left(\operatorname{Cov}[\boldsymbol{f} \,|\, \boldsymbol{f}_m]\right) \tag{5}$$

where

$$\operatorname{Cov}[\boldsymbol{f} \,|\, \boldsymbol{f}_m] = \boldsymbol{K}_{nn} - \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn} \tag{6}$$

$$\boldsymbol{K}_{nn}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j), \quad \boldsymbol{K}_{nm}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_{m,j}), \quad \boldsymbol{K}_{mm}(i,j) = k(\boldsymbol{x}_{m,i}, \boldsymbol{x}_{m,j}) \tag{7}$$

and $k(\cdot,\cdot)$ represents the GP kernel function. The approach is effective as, if M is small, we can evaluate this lower bound relatively quickly, particularly as we only really need the diagonal terms of $\boldsymbol{K}_{nn}$. Note that the same objective function is also used to optimise the hyperparameters. The selection of inducing points and optimisation of the hyperparameters can be conducted sequentially, in a similar manner to the EM algorithm. Importantly, to evaluate our objective function, the only matrix we need to invert is $\boldsymbol{K}_{mm}$. When N is large and M is relatively small, the approach leads to significant computational savings compared to standard GPs.

4 Example Python Code

Code accompanying this tutorial is written in Python (version 2), is heavily commented and is designed to be readable. The module 'LIRU_SparseGP' contains the functions needed to run a simple Sparse GP, while the script '1D_example' will run the Sparse GP on a test case that has univariate inputs. Specifically, '1D_example' will generate a full set of 500 training points, before attempting to fit a Sparse GP using 10 points only. To do so, it randomly selects 100 candidate sets of $\boldsymbol{X}_m$, before choosing the set that maximises the lower bound, L_{lb}. Once $\boldsymbol{X}_m$ has been chosen, scipy's 'optimize.minimize' module is used to minimise the (negative) lower bound with respect to the GP hyperparameters. Figure 1 shows an example of the results you should expect to see.

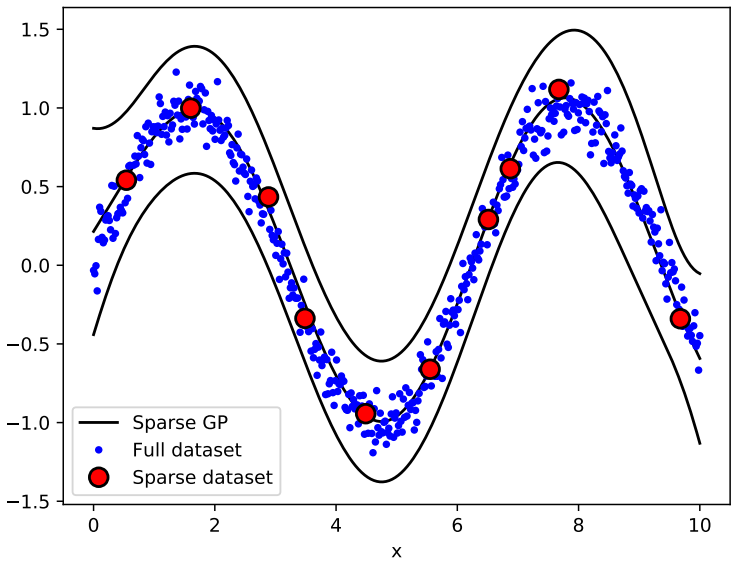

Figure 1: Example output of '1D_example' Python script.

Note that a more thorough approach to this problem would be to recursively add additional training points and update the GP hyperparameters. In [1][2] it is shown that the lower bound, L_{lb}, must increase every time an additional training point is included. This is convenient, as it allows the user to analyse convergence - new training points and hyperparameter updates can be run until L_{lb} is judged to have converged. For now though, for the sake of simplicity, our example code simply tries to identify a fixed number of training points (and optimum hyperparameters) for the Sparse GP implementation.

5 Detailed Methodology

Defining a prior $p(\boldsymbol{f}, \boldsymbol{f}_m)$ and likelihood $p(\boldsymbol{y} \,|\, \boldsymbol{f}, \boldsymbol{f}_m)$ gives us the posterior

$$p(\boldsymbol{f}, \boldsymbol{f}_m \,|\, \boldsymbol{y}) = \frac{p(\boldsymbol{y} \,|\, \boldsymbol{f}, \boldsymbol{f}_m) p(\boldsymbol{f}, \boldsymbol{f}_m)}{p(\boldsymbol{y})} \tag{8}$$

We decide that the optimum choice of $\boldsymbol{f}_m$ would lead to

$$p(\boldsymbol{y} \,|\, \boldsymbol{f}_m) = p(\boldsymbol{y} \,|\, \boldsymbol{f}, \boldsymbol{f}_m) \tag{9}$$

In other words, if we knew the optimum $\boldsymbol{f}_m$ then, regarding the probability of witnessing $\boldsymbol{y}$, $\boldsymbol{f}$ doesn't add any additional information[1]. It turns out that the optimum $\boldsymbol{f}_m$ will factorise the posterior, so that we can write $p(\boldsymbol{f}, \boldsymbol{f}_m \,|\, \boldsymbol{y}) = p(\boldsymbol{f} \,|\, \boldsymbol{y}) p(\boldsymbol{f}_m \,|\, \boldsymbol{y})$. Proof:

$$p(\boldsymbol{f}, \boldsymbol{f}_m \,|\, \boldsymbol{y}) = \frac{p(\boldsymbol{y} \,|\, \boldsymbol{f}_m) p(\boldsymbol{f}, \boldsymbol{f}_m)}{p(\boldsymbol{y})} \tag{10}$$

$$= \frac{p(\boldsymbol{y} \,|\, \boldsymbol{f}_m) p(\boldsymbol{f}_m \,|\, \boldsymbol{f}) p(\boldsymbol{f})}{p(\boldsymbol{y})} \times \frac{p(\boldsymbol{f}_m)}{p(\boldsymbol{f}_m)} \tag{11}$$

[1]Sometimes in the literature optimality is defined as $p(\boldsymbol{f}, \boldsymbol{f}_m \,|\, \boldsymbol{y}) = p(\boldsymbol{f} \,|\, \boldsymbol{f}_m)$. In fact, expanding both sides using Bayes' theorem you can show that $p(\boldsymbol{f}, \boldsymbol{f}_m \,|\, \boldsymbol{y}) = p(\boldsymbol{f} \,|\, \boldsymbol{f}_m) \implies p(\boldsymbol{y} \,|\, \boldsymbol{f}_m) = p(\boldsymbol{y} \,|\, \boldsymbol{f}, \boldsymbol{f}_m)$ so it doesn't make any difference. I prefer stating that $p(\boldsymbol{y} \,|\, \boldsymbol{f}_m) = p(\boldsymbol{y} \,|\, \boldsymbol{f}, \boldsymbol{f}_m)$ is the optimal condition as it seems a bit more intuitive.

$$= \frac{p(\boldsymbol{f}_m \mid \boldsymbol{f})p(\boldsymbol{f})}{p(\boldsymbol{f}_m)} \frac{p(\boldsymbol{y} \mid \boldsymbol{f}_m)p(\boldsymbol{f}_m)}{p(\boldsymbol{y})} \tag{12}$$

$$= p(\boldsymbol{f} \mid \boldsymbol{f}_m)p(\boldsymbol{f}_m \mid \boldsymbol{y}) \tag{13}$$

So in the optimal case we can factorise the posterior pdf as

$$p(\boldsymbol{f}, \boldsymbol{f}_m \mid \boldsymbol{y}) = p(\boldsymbol{f}_m \mid \boldsymbol{y})p(\boldsymbol{f} \mid \boldsymbol{f}_m) \tag{14}$$

For this reason, we choose a variational distribution *which can be factorised in the same way* by defining

$$q(\boldsymbol{f}, \boldsymbol{f}_m) = \phi(\boldsymbol{f}_m)p(\boldsymbol{f} \mid \boldsymbol{f}_m) \tag{15}$$

and we try to minimise the KL divergence between $q(\boldsymbol{f}, \boldsymbol{f}_m)$ and the posterior $p(\boldsymbol{f}, \boldsymbol{f}_m \mid \boldsymbol{y})$. This is based on the hypothesis that, if we find inputs $\boldsymbol{X}_m$ that make $q(\boldsymbol{f}, \boldsymbol{f}_m) = \phi(\boldsymbol{f}_m)p(\boldsymbol{f} \mid \boldsymbol{f}_m)$ and $p(\boldsymbol{f}, \boldsymbol{f}_m \mid \boldsymbol{y})$ exactly the same, then it must be possible to factorise the posterior distribution into $p(\boldsymbol{f}_m \mid \boldsymbol{y})p(\boldsymbol{f} \mid \boldsymbol{f}_m)$, thus implying that our choice of $\boldsymbol{X}_m$ is optimal. Noting that minimising

$$\mathrm{KL}(q(\boldsymbol{f}, \boldsymbol{f}_m) || p(\boldsymbol{f}, \boldsymbol{f}_m \mid \boldsymbol{y})) \tag{16}$$

is equivalent to maximising

$$L = \int\int q(\boldsymbol{f}, \boldsymbol{f}_m) \log \left[\frac{p(\boldsymbol{f}, \boldsymbol{f}_m, \boldsymbol{y})}{q(\boldsymbol{f}, \boldsymbol{f}_m)} \right] d\,\boldsymbol{f}\, d\,\boldsymbol{f}_m \tag{17}$$

then, by enforcing the factorisation property of the variational distribution ($q(\boldsymbol{f}, \boldsymbol{f}_m) = \phi(\boldsymbol{f}_m)p(\boldsymbol{f} \mid \boldsymbol{f}_m)$) and writing $p(\boldsymbol{f}, \boldsymbol{f}_m, \boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{f})p(\boldsymbol{f} \mid \boldsymbol{f}_m)p(\boldsymbol{f}_m))$ we have

$$L = \int\int p(\boldsymbol{f} \mid \boldsymbol{f}_m)\phi(\boldsymbol{f}_m) \log \left\{ \frac{p(\boldsymbol{y} \mid \boldsymbol{f})p(\boldsymbol{f}_m)}{\phi(\boldsymbol{f}_m)} \right\} d\,\boldsymbol{f}\, d\,\boldsymbol{f}_m \tag{18}$$

$$= \int \phi(\boldsymbol{f}_m) \underbrace{\left[\int p(\boldsymbol{f} \mid \boldsymbol{f}_m) \log p(\boldsymbol{y} \mid \boldsymbol{f}) d\,\boldsymbol{f} \right]}_{\log G(\boldsymbol{f}_m, \boldsymbol{y})} + \log \left(\frac{p(\boldsymbol{f}_m)}{\phi(\boldsymbol{f}_m)} \right) d\,\boldsymbol{f}_m \tag{19}$$

To evaluate $\log G = \int p(\boldsymbol{f} \mid \boldsymbol{f}_m) \log p(\boldsymbol{y} \mid \boldsymbol{f}) d\,\boldsymbol{f}$, we first need to find an expression for $p(\boldsymbol{f} \mid \boldsymbol{f}_m)$. Noting that

$$p(\boldsymbol{f}, \boldsymbol{f}_m) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{f}_m \end{pmatrix}; \boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}_{nn} & \boldsymbol{K}_{nm} \\ \boldsymbol{K}_{mn} & \boldsymbol{K}_{mm} \end{bmatrix} \right) \tag{20}$$

where $\boldsymbol{K}_{nn}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{K}_{nm}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_{m,j})$ and $\boldsymbol{K}_{mm}(i,j) = k(\boldsymbol{x}_{m,i}, \boldsymbol{x}_{m,j})$. Using standard properties of Gaussian distributions we can then show that

$$p(\boldsymbol{f} \mid \boldsymbol{f}_m) = \mathcal{N}(\boldsymbol{f}; \boldsymbol{0}, \mathrm{E}[\boldsymbol{f} \mid \boldsymbol{f}_m], \mathrm{Cov}[\boldsymbol{f} \mid \boldsymbol{f}_m]) \tag{21}$$

where

$$\mathrm{E}[\boldsymbol{f} \mid \boldsymbol{f}_m] = \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\,\boldsymbol{f}_m \tag{22}$$

and

$$\mathrm{Cov}[\boldsymbol{f} \mid \boldsymbol{f}_m] = \boldsymbol{K}_{nn} - \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn} \tag{23}$$

We can now try to evaluate $\log G$. To simplify notation, we first define $\boldsymbol{\mu} = \mathrm{E}[\boldsymbol{f} \mid \boldsymbol{f}_m]$ and $\boldsymbol{\Sigma} = \mathrm{Cov}[\boldsymbol{f} \mid \boldsymbol{f}_m]$ such that the aim is to evaluate

$$\log G = \int \mathcal{N}(\boldsymbol{f}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \log p(\boldsymbol{y} \mid \boldsymbol{f}) d\,\boldsymbol{f} \tag{24}$$

where

$$\log p(\boldsymbol{y} \mid \boldsymbol{f}) = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f_i)^2 \tag{25}$$

therefore

$$\log G = \int \mathcal{N}(\boldsymbol{f}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \left[-\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f_i)^2\right] d\,\boldsymbol{f} \tag{26}$$

$$= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}\underbrace{\left[\int \mathcal{N}(f_i; \mu_i, \Sigma_{ii})(y_i - f_i)^2 df_i\right]}_{I} \tag{27}$$

Noting that we can write

$$(y_i - f_i)^2 = \left(y_i^2 - 2f_i y_i + (f_i - \mu_i^2)\right) + 2f_i\mu_i - \mu_i^2 \tag{28}$$

then the intergral I becomes

$$y_i^2 - 2\mu_i y_i + \mu_i^2 + \Sigma_{ii} = (y_i - \mu_i)^2 + \Sigma_{ii} \tag{29}$$

therefore

$$\log G = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left[(y_i - \mu_i)^2 + \Sigma_{ii}\right] \tag{30}$$

$$= \log(\mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}, \boldsymbol{I}_N\sigma^2)) - \frac{1}{2\sigma^2}\mathrm{Tr}(\boldsymbol{\Sigma}) \tag{31}$$

This is a good point to summarise what we've done so far! We want to find the inputs, $\boldsymbol{X}_m$, that maximise

$$L = \int \phi(\boldsymbol{f}_m)\left[\log G(\boldsymbol{f}_m, \boldsymbol{y}) + \log\left(\frac{p(\boldsymbol{f}_m)}{\phi(\boldsymbol{f}_m)}\right) d\,\boldsymbol{f}_m\right] \tag{32}$$

where

$$\log G = \log\left(\mathcal{N}(\boldsymbol{y}; \mathrm{E}[\boldsymbol{f} \mid \boldsymbol{f}_m], \boldsymbol{I}_N\sigma^2)\right) - \frac{1}{2\sigma^2}\mathrm{Tr}\left(\mathrm{Cov}[\boldsymbol{f} \mid \boldsymbol{f}_m]\right) \tag{33}$$

$$\mathrm{E}[\boldsymbol{f} \mid \boldsymbol{f}_m] = \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{f}_m \tag{34}$$

and

$$\mathrm{Cov}[\boldsymbol{f} \mid \boldsymbol{f}_m] = \boldsymbol{K}_{nn} - \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn} \tag{35}$$

Noting that we can write

$$L = \int \phi(\boldsymbol{f}_m)\left[\log\left(G(\boldsymbol{f}_m, \boldsymbol{y})\frac{p(\boldsymbol{f}_m)}{\phi(\boldsymbol{f}_m)}\right)\right] d\,\boldsymbol{f}_m \tag{36}$$

then, using Jensen's inequality, we can now calculate a lower bound on L:

$$L \geq L_{lb} = \log \left[\int \phi(\boldsymbol{f}_m) \frac{G(\boldsymbol{f}_m, \boldsymbol{y}) p(\boldsymbol{f}_m)}{\phi(\boldsymbol{f}_m)} d\,\boldsymbol{f}_m \right] \tag{37}$$

$$= \log \int G(\boldsymbol{f}_m, \boldsymbol{y}) p(\boldsymbol{f}_m) d\,\boldsymbol{f}_m \tag{38}$$

Note that, to maximise L_{lb}, we don't actually have to find the variational distribution $\phi(\boldsymbol{f}_m)$ (a nice trick!). To evaluate the lower bound:

$$L_{lb} = \log \underbrace{\left[\int \mathcal{N}(\boldsymbol{y}; \mathrm{E}[\boldsymbol{f} \mid \boldsymbol{f}_m], \boldsymbol{I}_N \sigma^2) p(\boldsymbol{f}_m) d\,\boldsymbol{f}_m \right]}_{I_2} - \frac{1}{2\sigma^2} \mathrm{Tr}\left(\mathrm{Cov}[\boldsymbol{f} \mid \boldsymbol{f}_m]\right) \tag{39}$$

The integral I_2 can be evaluated using standard expressions for Gaussian distributions (see Appendix A) such that, finally, we have our objective function:

$$L_{lb} = \log \left[\mathcal{N}(\boldsymbol{y}; \boldsymbol{0}, \boldsymbol{I}_n \sigma^2 + \boldsymbol{K}_{nm} \boldsymbol{K}_{mm}^{-1} \boldsymbol{K}_{mn}) \right] - \frac{1}{2\sigma^2} \mathrm{Tr}\left(\mathrm{Cov}[\boldsymbol{f} \mid \boldsymbol{f}_m]\right) \tag{40}$$

where $\mathrm{Cov}[\boldsymbol{f} \mid \boldsymbol{f}_m] = \boldsymbol{K}_{nn} - \boldsymbol{K}_{nm} \boldsymbol{K}_{mm}^{-1} \boldsymbol{K}_{mn}$.

6 Some Interesting References

[5] produced a Variational Sparse GP that could also treat heterscedastic problems. [6] specifically considered classification but, in the process, looked into how a Variational Sparse GP could be applied when there are non-conjugate likelihoods. [7] introduced 'Deep Gaussian Processes' (A deep belief network based on GP mappings), where variational GPs was central to the approach. They showed that Deep GPs are applicable to large data sets (like other deep methods but also small data sets (unlike other deep methods). This was illustrated on a case study with more dimensions than data. [8] produced a variant of Variational Sparse GPs that is suitable for stockastic variational inference. This was applied to datasets with around 700,000 points. [9] looked at Deep GPs again and proposed a 'nested' approach to variational inference that makes them more suitable for parallel computer architectures. Notes the danger of overfitting when using deep methods. [10] proposed a Variational Sparse GP that is suitable for a Map-Reduce setting and applied to datasets with around 2 million data points (implemented on open-source MapReduce software in Python).

It is worth noting that, while these datasets are quite large, [11] presented the 'robust Bayesian Committee Machine' (extension of a product-of-experts model) that was applicable to data sets of the order 10^7. They also have some interesting criticisms of Sparse GPs, particularly with regard to the number of parameters that need to be trained.

Finally, Sparse GP code is available for free through the University of Sheffield's Python module GPy. Other interesting variants of GPs for large data sets can also be found in [12][13].

A Evaluating equation (39)

Generally speaking, for the situation where

$$p(\boldsymbol{x}_a) = \mathcal{N}(\boldsymbol{x}_a; \boldsymbol{\mu}, \boldsymbol{A}^{-1}), \qquad p(\boldsymbol{x}_b \,|\, \boldsymbol{x}_a) = \mathcal{N}(\boldsymbol{x}_b; \boldsymbol{H}\,\boldsymbol{x}_a + \boldsymbol{b}, \boldsymbol{L}^{-1}) \tag{41}$$

it can be shown that

$$p(\boldsymbol{x}_b) = \int p(\boldsymbol{x}_b \,|\, \boldsymbol{x}_a) p(\boldsymbol{x}_a) d\,\boldsymbol{x}_a = \mathcal{N}(\boldsymbol{x}_b; \boldsymbol{H}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{L}^{-1} + \boldsymbol{H}\boldsymbol{A}^{-1}\boldsymbol{H}^T) \tag{42}$$

This is proved, for example, in Bishop's book [14] and also forms part of the derivation of the well-known Kalman filter equations. Applying this generalised result to the integral in equation (39), we find the following:

$$\int \mathcal{N}\left(\boldsymbol{y}; \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{f}_m, \boldsymbol{I}_n\sigma^2\right) \mathcal{N}\left(\boldsymbol{f}_m; \boldsymbol{0}, \boldsymbol{K}_{mm}\right) d\boldsymbol{f}_m \tag{43}$$

$$= \mathcal{N}\left(\boldsymbol{y}; \boldsymbol{0}, \boldsymbol{I}_n\sigma^2 + \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mm}\boldsymbol{K}_{mm}^{-T}\boldsymbol{K}_{mn}\right) \tag{44}$$

$$= \mathcal{N}\left(\boldsymbol{y}; \boldsymbol{0}, \boldsymbol{I}_n\sigma^2 + \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn}\right) \tag{45}$$

References

[1] Michalis Titsias. Variational Learning of Inducing Variables in Sparse Gaussian Processes. *Artificial Intelligence and Statistics*, 5:567–574, 2009.

[2] Michalis K Titsias. Variational Model Selection for Sparse Gaussian Process Regression. (2006):1–20, 2009.

[3] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[4] Christopher M Bishop. *Pattern Recognition and Machine Learning*, volume 53. 2013.

[5] Miguel Lazaro-Gredilla and Michalis Titsias. Variational Heteroscedastic Gaussian Process Regression. *Proceedings of International Conference on Machine Learning (ICML 2011), Bellevue, Washington, USA*, page 8, 2011.

[6] Kian Ming A. Chai. Variational Multinomial Logit Gaussian Process. *J. Mach. Learn. Res.*, 98888:1745–1808, 2012.

[7] Andreas C. Damianou and Neil D. Lawrence. Deep Gaussian Processes. *Artificial Intelligence and Statistics*, 31, 2012.

[8] James Hensman, N Fusi, and Neil D. Lawrence. Gaussian Processes for Big Data. *Uncertainty in Artificial Intelligence*, pages 282–290, 2013.

[9] James Hensman and Neil D. Lawrence. Nested Variational Compression in Deep Gaussian Processes. pages 1–21, 2014.

[10] Yarin Gal, Mark van der Wilk, and Carl Rasmussen. Distributed Variational Inference in Sparse Gaussian Process Regression and Latent Variable Models. *Proc. Neural Information Processing Systems (NIPS)*, pages 1–9, 2014.

[11] Marc Peter Deisenroth and Jun Wei Ng. Distributed Gaussian Processes. *International Conference on Machine Learning*, 37, 2015.

[12] Andrew Gordon Wilson, Christoph Dann, and Hannes Nickisch. Thoughts on Massively Scalable Gaussian Processes. pages 1–25, 2015.

[13] Andrew Gordon Wilson and Hannes Nickisch. Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). 37, 2015.

[14] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.