MRes in AI and Machine Learning

Imperial College London

Thesis

Insights for Sparse Gaussian Process Regression

Author:
Jacob James Hughes-Hallett

CID:
01342416

Supervisor
Mark van der Wilk
Computing

September 14, 2022

Abstract

Sparse Gaussan Process Regression (SGPR), proposed by Titsias [20], is a scalable Gaussian process method that shows excellent performance at regression tasks. It uses M *inducing variables* to summarise a dataset of N points (where $M \leq N$) and makes an improvement to the computational cost of full Gaussian processes, which is impractically high when modelling large datasets: the time for computation scales as $\mathcal{O}(N^3)$ and thus, to learn from as few as 1000 datapoints, exact GPs are an unaffordable solution [13, 18]. For SGPR however, complexity scales as $\mathcal{O}(NM^2)$. This allows us to make use of the perks of GPs (uncertainty estimates, automatic regularisation, flexible priors, etc.) for a much broader range of problems.

Despite having all these theoretical benefits, SGPR has its practical problems. As well as training the model hyperparameters $\boldsymbol{\theta}$, as we do in the full Gaussian process model, we are additionally required to train the M inducing variables. A joint optimisation over this large number of hyperparameters (as is the currently accepted method) is cumbersome. Furthermore, for many datasets, SGPR is unable to sufficiently summarise the dataset with the popular squared exponential (SE) kernel until $M = N$, thereby rendering the sparse method obsolete for that regression task.

This paper discusses experiments that aim to addresses these problems. Specifically, we verify that harnessing *greedy variance* selection, a method for selecting the inducing points proposed by Burt et al. in [3], allows us to obtain an optimal solution for the SGPR model much faster and more consistently than a joint optimisation over all the model parameters. We additionally explore sparsity in SGPR. It is found that a good choice of kernel can allow us to use fewer inducing variables to ideally summarise the dataset than may be required for other kernels. Experiments are demonstrated on both toy and real-world datasets.

Acknowledgements

I am filled with gratitude for an amazing life experience it was to do this course in Machine Learning and AI at Imperial College London. It has been a challenging, and somewhat spiritual journey of learning and self-development.

I want to thank the module leaders for the MRes. course, Aldo Faisal and Ahmed Fetit for this awesome opportunity, and putting together a great course.

Thanks, Yingzhen Li for taking the time and effort to help mark my stuff!

I owe a great deal of thanks to Britta Ross, who caringly supported me with an open mind and heart thoughout my degree,

and finally, my wise and steady supervisor Mark van der Wilk, who patiently held my hand and kept me from stumbling (too much).

Contents

Chapter 1

Introduction

Imagine you are sat in a self-driving car and it assigns 5% probability to another car crossing your path at an intersection. The car's software, being rational, decides that we should not proceed, as doing so would result in a crash 5% of the time. The probability of 5% is an important number, since the car seems to use this number when deciding how to proceed. If this probability were different, the subsequent decisions might also change: if this probability is 0.000001% instead, we might justify that proceeding would be a sensible thing to do, as only one in a million times would a collision occur. We as humans already automatically carry out this probabilistic inference to some extent on a daily basis, weighing up the consequences of an action and how likely an outcome is to occur based on how we understand the world, and then, subsequently, deciding our actions.

Non-probabilistic methods, such as multi-layer perceptrons (MLPs), do not offer us this kind of flexibility. They do not allow us to feed the predictions they make into our own decision framework, as they effectively make deterministic decisions on their own: MLPs, instead of giving us a probability, would return the most probable class (that the other car does not cross the cars path) and thus perhaps decide that our car should proceed – clearly the unsafe option [15].

This simple example highlights the fundamental importance of the use of probabilities in data modelling. As AI pervades society, and as we find ourselves handing over the responsibility of increasingly important decisions to AI, it is indispensable that we thoroughly research these probabilistic models, constantly trying to improve their accuracy and efficiency, since these models form the foundation for building rational artificial agents [2].

1.1 The Problem

Bayesian Neural Networks (BNNs) are a hot topic in the machine learning community. They place probability distributions over the network parameters rather than treating the biases and weights as single values as is done in traditional neural networks. Yet, they remain difficult to use; they are computationally expensive not only due to the large numbers of parameters, but to infer the output distribution, we are required to sample from the network multiple times, further increasing complexity.

The Gaussian process (GP) machine learning method comes at probabilistic modelling from a different angle. It uses very few trainable parameters, and is thus considered a *non-parametric* model. GPs show very promising performance at uncertainty-aware supervised learning tasks, and are applied widely from control systems [12, 16] to reinforcement learning [10]. Unlike neural networks, which perform well on large noiseless datasets, GPs excel at regression tasks that suffer from a lack of data that are noisy, and that require uncertainty estimates on predictions [21, 18].

The fact that GPs are non-parametric should prove to be a very attractive option for non-experts. For the most basic GP model, the user only needs to make a decision on one component of the model called the *kernel* and even then, recent promising methods have been proposed that allow automatic inference of the optimal kernel from the data [19, 9]. However, again there is a catch. GPs suffer from two major practical drawbacks.

Firstly, in the case that data is indeed abundant, but where the benefit of uncertainty estimates is still desired, computation of new predictions using exact Gaussian processes becomes expensive. With a dataset of N training points, algorithmic complexity scales as $\mathcal{O}(N^3)$, rendering the intermediate matrix calculations infeasible for even moderately sized training datasets of a few thousand points [17].

Secondly, software errors often arise when carrying out intermediate matrix calculations in GPs. An implementation that can guarantee that a good solution will be found without running into errors is essential for that machine learning model to be trusted in society. While there exist strategies that can circumvent these failures, unfortunately, this problem has been unduly overlooked for many years by the machine learning community, and good reliable implementations of GPs have yet to be made and thoroughly tested. So, even though these methods provide us with a richer insight into our data, their uptake to solve real-world problems still remains rare.

1.2 Main Contributions

Much research has gone into developing scalable versions of GPs that are able to learn from large datasets. *Sparse* Gaussian process methods aim to solve the problem of poor GP scalability by making approximations in order to reduce computational cost. Sparse Gaussian process regression (SGPR) [20], a state-of-the-art sparse Gaussian process method, will be discussed in this review.

Current implementations of SGPR and many other sparse GP methods require copious background knowledge to use optimally – there are often many "moving parts" to modern Gaussian process models that need to be considered for which literature does not give clear rule-of-thumbs.

This thesis provides some practical insights on training SGPR to make progress towards resolving these matters. In particular, the following is discussed:

- how employing a training strategy during optimisation of the parameters of the SGPR model can improve performance and learning time,
- and how different kernels affect the sparsity of the model – i.e. how much we can simplify the model so that it can be applied to larger datasets.

To do this, we will use a combination of employing methods and findings from recent literature, and developing novel strategies based on experiments and theory.

1.3 Impact of GP Research

Currently neural networks reign supreme in the world of AI, but more research into GPs to fully harness their useful properties might see a significant increase in their use in society. Where neural networks or other models were once employed, Gaussian process might take over if proven to exhibit better performance. Making machine learning models easier to employ such that they are less exclusive to those who understand the full inner-workings, will expedite the useful application of AI.

Clarifying the questions that still overshadow GPs will not only make Gaussian processes easier to employ by everyone, but also provide a firm footing for the development of implementations of more exciting and complex models that use Gaussian processes as just a part of the whole model architecture. For example, deep kernel learning (DKL), a model that harnesses both the representational power of neural networks, but also can provide accurate uncertainty estimates on predictions by using a Gaussian process, works by feeding the outputs of a neural network as inputs into the kernel of a Gaussian process [23, 4]. Such a model has huge potential for the machine learning community, since it can benefit from the best of both worlds of GPs and NNs, and so could prove to show excellent flexibility. Practically useful and scalable versions of DKL [24] have yet to be implemented into easy-to-use software that performs reliably without human intervention as a result of there not being clear general consensus on how to use GPs. In many respects we are building these ideas on an unstable foundational understanding on how exactly to implement GPs for optimal performance. A clearer insight on how to use sparse GPs would pave the way for developing better DKL models, and other similar models such as Gaussian process regression networks [22] and deep Gaussian processes [7].

Chapter 2

SGPR in Theory

This chapter explores the theory behind Gaussian processes regression [1], and how they can be approximated to be made more scalable to larger datasets.

2.1 Gaussian Processes Regression

2.1.1 The regression task

Suppose we have a dataset that contains a vector of N noisy observations $\mathbf{y} = \{y_n\}_{n=1}^{N}$ and a corresponding matrix of N concatenated input vectors (of length D for the number of input dimensions) $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^{N}$ [2]. The aim of an arbitrary regression task is to infer the noise-free *latent* function $\mathbf{f} \triangleq f(\mathbf{X})$ that underpins the data, and hence to determine latent function values $\mathbf{f}_* \triangleq f(\mathbf{X}_*)$ (and new target values $\mathbf{y}_*$) at new test locations $\mathbf{X}_*$.

In order to be able to carry out this inference, an important assumption must be made: There must be a dependent relationship between the input and output value pairs. In other words, for any two inputs $\mathbf{x}$ and $\mathbf{x}'$ that are very similar to each other, we can expect that their corresponding function values, $f(\mathbf{x})$ and $f(\mathbf{x}')$, will be similar too. Without this assumption, inference is impossible, since we are unable to learn how the model behaves at new test points from knowing the training data.

2.1.2 A Bayesian approach

Before we have observed any data from which to draw conclusions, we might have preconceptions about how we expect the data to behave. For example, if we are asked to model some phenomenon that we know should be periodic, like the month-average temperature in London which we expect to have a period of one year, we should be able to factor this insightful knowledge into our model, such that when predicting future temperature values, we see that same periodicity. This allows us to make predictions based not only on the data, but also on a *hypothesis*, $\mathcal{H}$. Any insider information on how we expect the underlying function to behave is valuable, and should be contained in $\mathcal{H}$.

Bayesian machine learning methods use a prior distribution to model the data $\{\mathbf{y}, \mathbf{X}\}$ *prior* to any data being observed. The prior is chosen only based on our preconceptions about the data, $\mathcal{H}$. With each observed datapoint, we then can update the model using

[1]For a more comprehensive introduction to GPs, see Mackay's [13] or Rasmussen's [18] textbooks.
[2]$\mathbf{X}$ is therefore an $N \times D$ matrix.

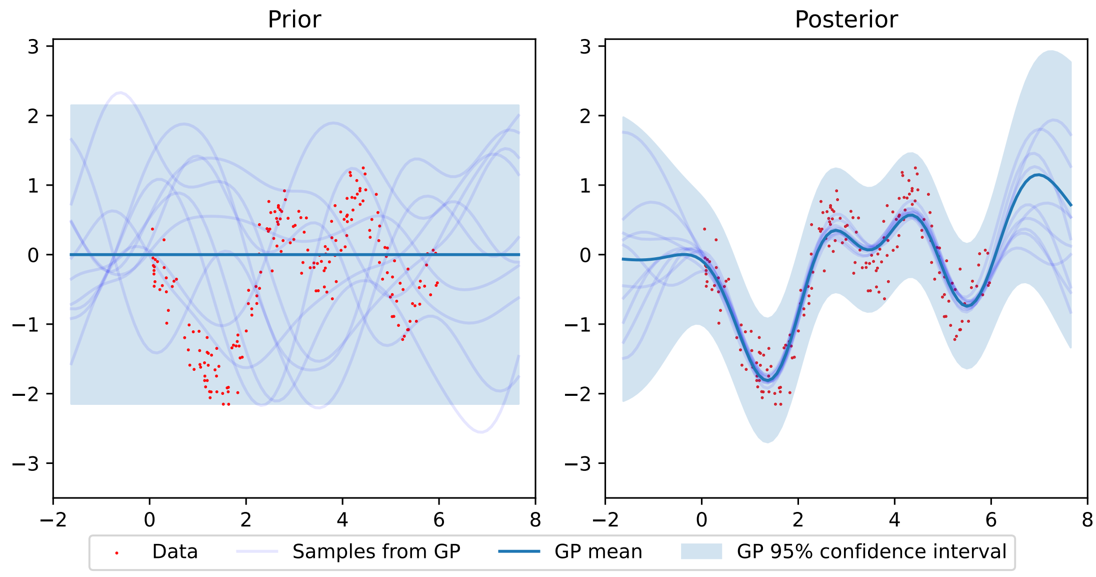


Figure 2.1: GP demonstration, comparing the prior with the posterior on the Snelson 1D dataset. Our posterior contains a more restricted set of functions than the prior.

Bayes' rule:

$$\underbrace{p(\text{model} \mid \text{data}, \mathcal{H})}_{\text{posterior}} = \frac{\overbrace{p(\text{data} \mid \text{model}, \mathcal{H})}^{\text{likelihood}}\,\overbrace{p(\text{model}, \mathcal{H})}^{\text{prior}}}{\underbrace{p(\text{data}, \mathcal{H})}_{\text{marginal likelihood}}}. \tag{2.1}$$

An advantage of using a Bayesian approach is that we obtain a distribution over possible models rather than a single single model. For example, the posterior in Bayesian linear regression tells us the distribution of weights, $p(\mathbf{w})$, on the basis functions, whereas a non-Bayesian approach would just reveal to us a single value for $p(\mathbf{w})$. This means that when we make predictions with the model, we have a measure of how confident we feel about our predictions, as well as the predictions themselves.

2.1.3 Gaussian Processes

In Bayesian linear regression (BLR), a distribution is placed on the weight of certain basis functions that are predefined. For example, for a problem with 1D data, we might use a linear model with basis functions $[1, x]$ to model our data, and place a 2D Gaussian distribution on weights $[w_0, w_1]$ (a dimension for each weight) for these basis functions. We can then subsequently update this prior distribution to a posterior using Bayes rule.

Gaussian processes are a generalisation of BLR: Gaussian processes instead use an infinite number of basis functions. A Gaussian process can be considered to be a Gaussian distribution that has infinite dimensions. The mean and covariance in this case cannot be represented by a vector and matrix, and is instead represented by functions: $\mathcal{GP}(\mu(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}'|\boldsymbol{\tau}))$. The infinite basis functions, however, are not completely arbitrary; they are restricted to a domain of functions that are defined by the kernel κ. It takes two input values, $\mathbf{x}$ and $\mathbf{x}'$, and hyperparameters $\boldsymbol{\tau}$, and returns the correlation between the two inputs – a measure of similarity between the output function values, $f(\mathbf{x})$ and $f(\mathbf{x}')$. This kernel effectively becomes a generalisation of the covariance matrix of a Gaussian distribution. The GP is also defined by its mean, $\mu(\mathbf{x})$, about which all sampled functions are centred.

Because GPs are distributions, Bayesian inference can be employed to tackle the regression task at hand: a prior GP is hypothesised, and using Bayes' rule with an appropriate likelihood, we can obtain a posterior distribution that takes into account the training data. With this updating of the GP from a prior to a posterior, the data essentially narrows down the distribution on functions, making us more certain about what the true latent function f is (see figure 2.1).

To find the posterior distribution $p(\mathbf{f}, \mathbf{f}_* \mid \mathbf{y}, \mathbf{X}, \mathbf{X}_*, \boldsymbol{\theta})$[3] from which we can obtain our prediction values and their uncertainty estimates, we can write down a Bayesian formulation of the model:

$$p(\mathbf{f}, \mathbf{f}_* \mid \mathbf{y}) = \frac{p(\mathbf{f}, \mathbf{f}_*, \mathbf{y})}{p(\mathbf{y})} = \frac{p(\mathbf{y} \mid \mathbf{f}, \overset{\mathbf{y}\text{ independent of }\mathbf{f}_*}{\cancel{\mathbf{f}_*}})p(\mathbf{f}, \mathbf{f}_*)}{\int p(\mathbf{y} \mid \mathbf{f})p(\mathbf{f}, \mathbf{f}_*)d\mathbf{f}d\mathbf{f}_*} \tag{2.2}$$

where the first factor in the numerator on the right-hand-side is the likelihood and the second is the prior, and the denominator is the marginal likelihood. The exact calculation of the posterior and marginal likelihood is possible so long as the prior and likelihood are Gaussian distributions.

Prior

Without loss of generality, our GP prior is usually chosen with $\mu = 0$, as we generally have no preconceived notions on how we expect the model to behave. Either way, any prior knowledge can generally be factored into our prior by picking the appropriate kernel function. As we are considering only the relevant input locations $\mathbf{X}$ and $\mathbf{X}_*$ we can write the GP prior as a marginal distribution over the infinite-dimensional GP:

$$p(\mathbf{f}, \mathbf{f}^*) = \mathcal{N}\left(\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix}; \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right) \tag{2.3}$$

where $\mathbf{K} = \kappa(\mathbf{X}, \mathbf{X})$, $\mathbf{K}_* = \kappa(\mathbf{X}, \mathbf{X}_*)$ and $\mathbf{K}_{**} = \kappa(\mathbf{X}_*, \mathbf{X}_*)$.

Likelihood

Exact inference is possible in the case of *homoskedastic* noise i.e. the case where the noise across the samples is uniform and Gaussian such that the likelihood is given by: $p(\mathbf{y} \mid \mathbf{f}) = \mathcal{N}(y; \mathbf{f}, \mathbb{I}_N\sigma_y^2)$, where $\mathbf{f} \triangleq f(\mathbf{X})$. The noise variance, σ_y^2, is an unknown model hyperparameter that we will be required to determine.

[3]We henceforth omit the implicit dependence on the training and test input values $\mathbf{X}$ and $\mathbf{X}_*$, and kernel hyperparameters $\boldsymbol{\theta}$.

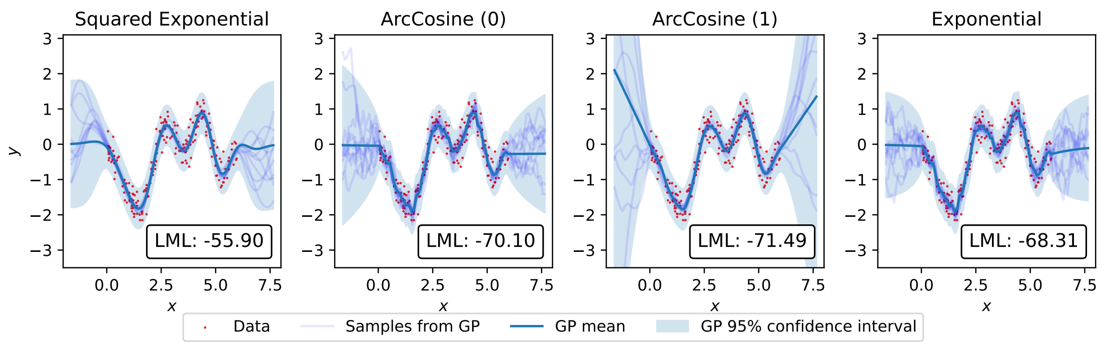


Figure 2.2: The resulting posterior GP distributions and final LML scores when a GP is trained on the 1D Snelson dataset using different kernels.

Posterior

We can marginalise out the latent variables $\mathbf{f}$ from the posterior in 2.2 to obtain the following *predictive* posterior distribution:

$$p(\mathbf{f}_* \mid \mathbf{y}) = \mathcal{N}(\mathbf{f}_*; \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \tag{2.4a}$$

$$\boldsymbol{\mu}_* \triangleq \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{y} \tag{2.4b}$$

$$\boldsymbol{\Sigma}_* \triangleq \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{K}_* \tag{2.4c}$$

where $\mathbf{K}_y = \mathbf{K} + \mathbb{I}_N \sigma_y^2$. This gives us a new mean, $\boldsymbol{\mu}_*$ about which the our samples are centred. This is our prediction, and the diagonals of $\boldsymbol{\Sigma}_*$ are our uncertainties.

Marginal Likelihood

The marginal likelihood, $p(\mathbf{y})$, which can also be shown to be Gaussian, plays an important role in GP training: the log marginal likelihood (LML), given by

$$\log p(\mathbf{y}) = -\frac{1}{2}\mathbf{y}\mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}_y| - \frac{N}{2}\log(2\pi) \tag{2.5}$$

is the objective function used during optimisation to find the optimal kernel and model hyperparameters $\boldsymbol{\theta} = \{\boldsymbol{\tau}, \sigma_y\}$. This optimisation process automatically regularises our probabilistic model by maximising the likelihood of the data given $\boldsymbol{\theta}$. The first and second terms of the RHS of 2.5 are called the *data-fit* and *complexity-penalty* terms respectively, the latter of which plays a role to suppress the model complexity and hence to help to prevent overfitting [18]. With our optimised hyperparameters, we can use 2.4 to obtain predictions, as well as the corresponding errors, at new test locations.

Choice of Kernel

The process of computing the predictions is almost entirely automatic. The only component of the GP that the user must influence is the kernel. It is the kernel function that determines what kinds of functions we expect to obtain when sampling from the GP distribution. One example of a popular kernel is the squared exponential (SE) kernel, which takes two hyperparameters as arguments:

$$k(\mathbf{x}, \mathbf{x}' \mid \boldsymbol{\tau}) = \sigma_f^2 \exp\left[-\frac{1}{2}\frac{(\mathbf{x}-\mathbf{x}')^T(\mathbf{x}-\mathbf{x}')}{l^2}\right]. \tag{2.6}$$

The *lengthscale* of this kernel, l defines over what characteristic distance the functions vary. The larger the lengthscale, the the higher the correlation between two inputs over the same distance will be, and so our functions will vary less over the same distance. The variance of the kernel, σ_f^2, defines the amplitudes of the functions that we expect, how much the functions vary in the output domain.

Figure 2.2 shows the posterior distribution of four models, each with a different kernel to model the data. We can see that our choice of kernel affects the final LML – the measure of quality of our predictions; these kernels have different properties that affect the goodness of fit to the data.

Rough and smooth kernels The SE kernel tends to give us functions that are smooth, and so it is a suitable kernel for modelling smooth functions. On the other hand, there are kernels, such as the exponential kernel that gives us functions that are more jagged and rough. We can see that the data in 2.2 is better modelled by a smooth function, hence why we get a better LML for the SE kernel than the exponential kernel.

Stationary and non-stationary kernels Stationary kernels give predictions that tend to the mean, μ of the GP when far from the data. The order 1 arccosine kernel is a non-stationary kernel, which explains why the predictions deviate away from the mean when far from the data in 2.2. If we know that our predictions should follow an upwards trend, for example, it would be more appropriately modelled by a non-stationary kernel.

ARD kernels In this thesis, we will only consider *automatic relevance determination* (ARD) kernels – kernels that have a separate lengthscale hyperparameter for each dimension of the input data. A dimension that has a large lengthscale will be less *relevant*, since there is little variation across the input space [9].

Combinations of kernels Kernels can be combined together by addition or multiplication. The resulting kernel gives sample functions that are multiplicative or additive combinations of the samples of its component kernels. This is a good way of adding flexibility into the model, as we are effectively giving the model a broader GP prior. However, as we have more hyperparameters to train, this increased flexibility can also cause problems during optimisation.

2.1.4 GP scalability

In the case that we have a large dataset, computation of new predictions using exact Gaussian processes becomes expensive. Notice that the computation of both the posterior of the GP in equation 2.4 and the log marginal likelihood in equation 2.5 require the computation of the inverse of an $N \times N$ matrix. This has computational cost of $\mathcal{O}(N^3)$, which becomes computationally unaffordable for even moderately sized training datasets of a few thousand points [11, 17].

2.2 Sparse GP Regression

The objective of modern *sparse* Gaussian process methods is to mimic the power of full GPs, while being significantly less computationally expensive in order to make use of large datasets.

The most elementary of the sparse GP methods is the Subset of Data (SoD) method. Using only a subset of M datapoints of the full dataset (with $M \ll N$) to tackle the scalability problem is not an ideal solution however; we are discarding valuable information in the data on which the GP could be trained, thereby compromising on the quality of the model [17]. Many sparse methods hypothesise that we are able to summarise the N datapoints $\{\mathbf{X}, \mathbf{y}\}$ with M learnable *inducing variables* $\mathbf{u}$ (in the same space as $\mathbf{f}$) at specific input locations $\mathbf{Z}$, where $M \ll N$. The idea is that with a smaller set of M points that retain as much information as possible from the data, computation times should be reduced with little impact to the quality of the solution.

In this review, in particular, Sparse Gaussian Process Regression[4] (SGPR) proposed by Titsias in 2009 [20], a sparse GP method that uses *variational inference* (VI) to directly approximate the exact posterior (as given on the left-hand-side of 2.2) will be discussed. The main assumption with SGPR is that $\mathbf{u}$ is a *sufficient statistic* for $\mathbf{f}$ and $\mathbf{f}_*$ – the inducing points $\{\mathbf{Z}, \mathbf{u}\}$ hold all the information required to infer either $\mathbf{f}$ or $\mathbf{f}_*$ i.e. $p(\mathbf{f} \mid \mathbf{u}, \mathbf{y}) = p(\mathbf{f} \mid \mathbf{u})$.

2.2.1 Solving the regression task with VI

Let us restate the problem that we are trying to solve: to find the distribution $p(\mathbf{f}_* \mid \mathbf{y})$ in order to obtain predictions $\mathbf{f}_*$ and their corresponding errors at test locations $\mathbf{X}_*$ [5]. Starting from the joint posterior over all of $\mathbf{f}$, $\mathbf{f}_*$ and $\mathbf{u}$, we can obtain the predictive posterior by simply marginalising out $\mathbf{f}$ and $\mathbf{u}$ if we only desire to find $\mathbf{f}_*$:

$$p(\mathbf{f}_* \mid \mathbf{y}) = \int p(\mathbf{f}, \mathbf{f}_*, \mathbf{u} \mid \mathbf{y}) d\mathbf{f} d\mathbf{u}. \tag{2.7}$$

Finding this joint posterior becomes our intermediate objective. This posterior, however, is clearly intractable because of the high computational cost. If we rewrite this posterior as

$$p(\mathbf{f}, \mathbf{f}_*, \mathbf{u} \mid \mathbf{y}) = p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{f}_* \mid \mathbf{u}) p(\mathbf{u} \mid \mathbf{y}), \tag{2.8}$$

we observe that the distribution $p(\mathbf{u} \mid \mathbf{y})$ is the bottleneck. The other two factors on the right-hand-side of 2.8 have significantly lower computational cost.

We introduce an approximate posterior distribution:

$$p(\mathbf{f}, \mathbf{f}_*, \mathbf{u} \mid \mathbf{y}) \approx q(\mathbf{f}, \mathbf{f}_*, \mathbf{u} \mid \mathbf{y}) = p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{f}_* \mid \mathbf{u}) \phi(\mathbf{u} \mid \mathbf{y}) \tag{2.9}$$

where we have replaced the intractable $p(\mathbf{u} \mid \mathbf{y})$ with a free Gaussian posterior $\phi(\mathbf{u} \mid \mathbf{y}) \triangleq \mathcal{N}\left(\mathbf{u} \mid \boldsymbol{\mu}_\phi, \boldsymbol{\Sigma}_\phi\right)$. Varying the variational parameters $\{\boldsymbol{\mu}_\phi, \boldsymbol{\Sigma}_\phi, \mathbf{Z}\}$ will change the quality of the approximation q. Therefore, to get as good an approximation as possible, our aim is to select the optimal variational parameters such that the KL-divergence between q

[4]Also known as the Variational Free-Energy (VFE) approximation.

[5]We can additionally find the prediction observations $\mathbf{y}_*$ just by adding Gaussian noise to $\mathbf{f}_*$.

and the true joint posterior, a measure of difference between the two distributions, is minimised. The KL-divergence is given by:

$$\mathcal{KL}\left[q(\mathbf{f},\mathbf{f}_*,\mathbf{u}\mid\mathbf{y})\|p(\mathbf{f},\mathbf{f}_*,\mathbf{u}\mid\mathbf{y})\right]=\log p(\mathbf{y})-\mathcal{L}_{\text{lower}}(\boldsymbol{\mu}_\phi,\boldsymbol{\Sigma}_\phi,\mathbf{Z}) \tag{2.10}$$

where $\mathcal{L}_{\text{lower}}$ is the evidence lower bound (ELBO), and $\mathcal{KL}\geq 0$ with equality when the two distributions are identical. Notice that equation 2.10 contains a log marginal likelihood term which is intractable for large datasets. However, this term is independent of the variational parameters and hence remains constant during optimisation. Maximisation of the ELBO is therefore equivalent to a minimisation of the KL-divergence. To obtain predictions, we can insert the optimal values for the variational parameters into the approximate predictive posterior given by:

$$\int q(\mathbf{f},\mathbf{f}_*,\mathbf{u}\mid\mathbf{y})d\mathbf{f}d\mathbf{u}=q\left(\mathbf{f}_*\mid\mathbf{y}\right)=\mathcal{N}\left(\mathbf{f}_*\mid\boldsymbol{\mu}_*^q,\boldsymbol{\Sigma}_*^q\right) \tag{2.11a}$$

$$\boldsymbol{\mu}_*^q\triangleq\mathbf{K}_{*m}\mathbf{K}_{mm}^{-1}\boldsymbol{\mu}_\phi \tag{2.11b}$$

$$\boldsymbol{\Sigma}_*^q\triangleq\mathbf{K}_{**}-\mathbf{K}_{*m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{m*}+\mathbf{K}_{*m}\mathbf{K}_{mm}^{-1}\boldsymbol{\Sigma}_\phi\mathbf{K}_{mm}^{-1}\mathbf{K}_{m*} \tag{2.11c}$$

where $\mathbf{K}_{mm}=\kappa(\mathbf{Z},\mathbf{Z})$, $\mathbf{K}_{*m}=\kappa(\mathbf{X}_*,\mathbf{Z})$ and $\mathbf{K}_{m*}=\mathbf{K}_{*m}^T$.

In sparse Gaussian process regression (SGPR), we analytically solve for the optimal values for $\boldsymbol{\mu}_\phi$ and $\boldsymbol{\Sigma}_\phi$ in closed form, which results in a bound only dependent on the inducing inputs $\mathbf{Z}$:

$$\mathcal{L}_{\text{lower}}(\mathbf{Z})=\log\mathcal{N}\left(\mathbf{y}\mid\mathbf{0},\mathbb{I}_M\sigma_y^2+\mathbf{Q}_{nn}\right)-\frac{1}{2\sigma_y^2}\operatorname{Tr}\left(\mathbf{K}-\mathbf{Q}_{nn}\right) \tag{2.12}$$

where $\mathbf{Q}_{nn}=\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^T$ and $\mathbf{K}_{nm}=\kappa(\mathbf{X},\mathbf{Z})$ [20].

The optimal variational parameters $\{\boldsymbol{\mu}_\phi,\boldsymbol{\Sigma}_\phi,\mathbf{Z}\}$ can then be inserted into the predictive posterior (equation 2.11) to obtain the predictions. This whole procedure is carried out in $\mathcal{O}(NM^2)$ time, also a significant improvement to the full GP. Training the SGPR model then becomes a matter of optimising the model parameters, $\{\boldsymbol{\theta},\mathbf{Z}\}$ to maximise the ELBO.

2.2.2 Why SGPR as opposed to another sparse method?

SGPR is excellent at carrying out regression tasks. Other models like Fully Independent Training Conditional (FITC) model (look this up) shows pathalogical behaviour that leads to inconsistency at finding the optimal solution. Additionally, SGPR is guaranteed to recover the quality of the solution of the exact GP when we have $M=N$ [1]. This is not guaranteed with other models.

SVGP

Sparse Variational Gaussian Processes (SVGP) is very similar to SGPR, except that a joint optimisation over all the parameters, including the variational distribution parameters is run [11], instead of analytically computing the variational parameters as is done in SGPR. The bound used for this optimisation is built directly from 2.12 to give:

$$\mathcal{L}_{\text{lower}}=\sum_{n=1}^{N}\int\phi\left(\mathbf{u}\mid\mathbf{y}\right)p\left(f_n\mid\mathbf{u}\right)\log p\left(y_n\mid f_n\right)df_nd\mathbf{u}-\mathcal{KL}\left[\phi\left(\mathbf{u}\mid\mathbf{y}\right)\|p\left(\mathbf{u}\right)\right], \tag{2.13}$$

which is used as the objective function during training.

SVGP also exhibits cutting-edge performance, and is in fact more flexible than the SGPR model: since the variational parameters can be trained freely and thus can lend it self to modelling data with non-Gaussian likelihoods, as is the case in classification tasks. Furthermore, with the bound given in 2.13, the stochastic gradient is an unbiased estimator of the full gradient (unlike in SGPR) and so can make use of stochastic minibatching for gradient descent, reducing computational cost to $\mathcal{O}(M^3)$ time.

In spite of these benefits, preliminary tests show that SGPR in general still outperforms SVGP in speed of optimisation and model accuracy in the regression setting, with SGPR overestimating the noise variance even more than SGPR does. Along with increased flexibility in the model comes so many parameters to train in SVGP, which makes optimisation cumbersome.

Chapter 3

SGPR in Practice

While we might have solved the scalability problem, as a result of introducing a more complicated model, we require more decisions to be made by the user: the number of inducing points to be chosen and whether or not train the inducing points, which could be potentially large in number. Remember that in order to obtain an ideal solution in SGPR, we require that the M inducing variables can sufficiently summarise the full dataset and allow us to recover the latent function f. M needs to be large enough that this condition is satisfied, but also as small as possible so that we are not wasting computational resources (the smaller M is, the lower the computational complexity which scales as $\mathcal{O}(NM^2)$). Furthermore, the choice of kernel is likely to have an impact on these decisions.

One of the main benefits of using GPs is their non-parametric nature, meaning that the models have (almost) no parameters that we are required to tune; introducing an extra moving part into the model takes away from this. The fact that these questions on how to use the model have yet to be concretely answered in the machine learning community is a major reason why GPs are not more ubiquitous today.

This section focuses on the practical task of deploying SGPR models for real-life application, in order that we consistently obtain an optimal solution. An optimal solution exists – it's now just a matter of finding it.

3.1 Training

Let us recap our objective for obtaining an optimal solution for the regression task: we are to optimise the parameters such that a maximum ELBO is obtained. This in turn minimises the KL divergence between the approximate posterior $q(\mathbf{f}, \mathbf{f}_*, \mathbf{u} \mid \mathbf{y})$ and the true posterior $p(\mathbf{f}, \mathbf{f}_*, \mathbf{u} \mid \mathbf{y})$, therefore getting us as close to the quality of the full GP solution as possible. As mentioned in the previous chapter, since the optimal parameters of the variational distribution are calculated analytically for the case of a Gaussian likelihood, the only parameters that are left to train to maximise the ELBO are the model hyperparameters $\boldsymbol{\theta}$ and the inducing inputs $\mathbf{Z}$. The bound in equation 2.12 allows us to jointly optimise all these parameters simultaneously.

However, now with a large number of trainable parameters (both $\mathbf{Z}$ and $\boldsymbol{\theta}$), optimisation becomes a more challenging task as the chance of getting trapped in a local optimum increases. A joint optimisation over all the parameters could prove to be problematic. In this section, we aim to shed light on the matter of training SGPR models, and make use of strategies to bypass the full joint optimisation of all the parameters.

Initialisation	Energy	Concrete	Wine	Airfoil	Solar	SML
Random	1011.59	-426.36	-995.84	-776.20	-1291.25	3790.11
Greedy var.	1011.86	-388.62	-938.94	-599.32	-1277.72	3835.87

Table 3.1: A comparison of final ELBOs between a random initialisation of $\mathbf{Z}$ and a greedy variance initialisation of $\mathbf{Z}$ for different UCI datasets. The 250 inducing points were used for all the SGPR models. $\mathbf{Z}$ were kept fixed for the duration of the training; only the model hyperparameters were trained. A greedy variance initialisation makes an improvement to the final ELBO for all the tested datasets.

3.1.1 *Greedy Variance* Selection

Burt et al. suggest that with a good initial placement of the inducing inputs, the benefits of then optimising over the inputs is not worth the additional cost [3]. M can be large when N is large, so excluding $\mathbf{Z}$ from the optimisation of the ELBO can be beneficial. In their paper, the *greedy variance* inducing point selection technique is introduced, an alternative method of finding optimal locations of the inducing inputs $\mathbf{Z}$. Inducing points are strategically selected, rather than being randomly initialised as is the currently accepted procedure in the GP community. This method of selecting $\mathbf{Z}$ is claimed to "provide consistent fast convergence to the exact model".

Even when inducing variables are kept fixed, greedy variance initialisation is advantageous: table 3.1 compares the final ELBO after training the hyperparameters but keeping $\mathbf{Z}$ fixed, for a randomised and greedy variance initialisation of $\mathbf{Z}$ for different UCI datasets.

For greedy variance selection, as the name suggests, inducing points are chosen at locations where there is the largest variance of $p(f|\mathbf{u})$ (distribution of the latent function given the inducing variables) which is given by [1]:

$$\mathbb{V}[p(\mathbf{f}|\mathbf{u})] = \kappa(\mathbf{X},\mathbf{X}) - \kappa(\mathbf{Z},\mathbf{X})\kappa(\mathbf{Z},\mathbf{Z})\kappa(\mathbf{X},\mathbf{Z}). \tag{3.1}$$

With this selection strategy, you are, so to speak, "selecting the uncertainty away". Figure 3.1 demonstrates how the first three inducing point locations are selected with a SE kernel for the Snelson 1D dataset.

Since $\mathbb{V}[p(\mathbf{f}|\mathbf{u})]$ is dependent on the kernel and its hyperparameters, $\boldsymbol{\tau}$, the inducing points selected with greedy variance will be different for different $\boldsymbol{\tau}$. As the model trains and $\boldsymbol{\tau}$ change, our initially selected $\mathbf{Z}$ will no longer be optimal. It is therefore suggested in [3] to reinitialise the inducing inputs every few iterations of training the model.

The paper experiments with reinitialising every 25 iterations of training; reinitialisation of $\mathbf{Z}$ should not to be too frequent, as this hinders progress and training becomes slow. Nor should it be too infrequent, otherwise $\boldsymbol{\theta}$ and $\mathbf{Z}$ are not getting trained simultaneously, and we risk compromising on performance. After considering these criteria, within a certain range, the choice of frequency of reinitialisation was shown in our experiments to be rather independent of the final quality of the model. For all datasets, every 25 iterations was a frequency that performed ideally [2].

[1]While the inducing points can technically be selected to be anywhere in the input domain, in practice, it is easier to let this to be the training input locations, as this already covers the input space well.

[2]For smaller datasets, training was so fast that the reinitialisation frequency could be reduced to every 50 iterations, since reinitialisation time was a significant proportion of the total training time. But either way, in these cases, a full GP might probably be more appropriate.

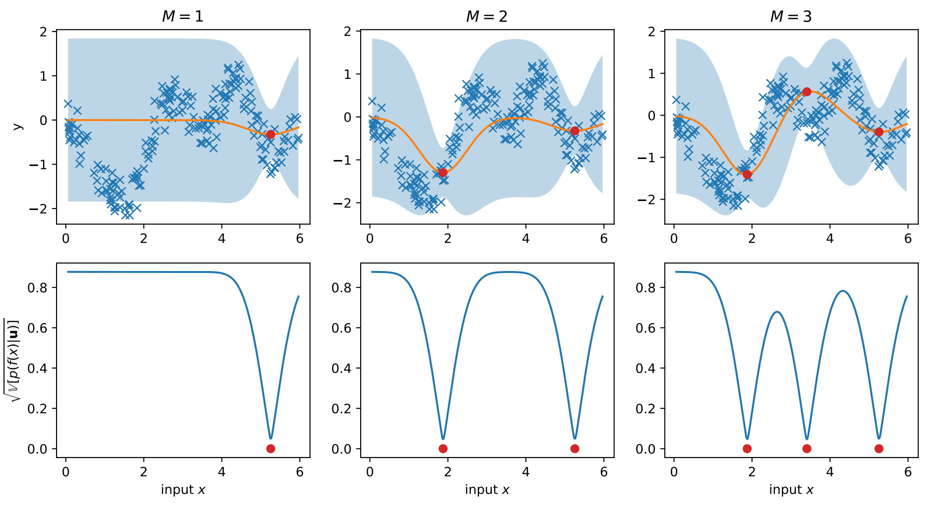


Figure 3.1: Demonstration of the greedy variance inducing point selection method with the Snelson 1D dataset. The first inducing point location is selected at a random location. Each subsequent inducing point is selected at a location where the variance of $p(f|\mathbf{u})$ is largest.

3.1.2 Alternative Training Strategies

One can develop alternative training strategies that attempt to bypass the full joint optimisation of all the parameters including $\mathbf{Z}$. All the methods that are explored in this thesis are:

1. *joint* method
2. *fix* method
3. *reinitialise* method
4. *fix-retrain* method
5. *reinitialise-retrain* method

All of these methods begin with an initialisation of $\mathbf{Z}$ with greedy variance, but the ensuing training strategy varies.

Joint method All parameters are jointly optimised.

Fix method Only the hyperparameters $\boldsymbol{\theta}$ are trained. The $\mathbf{Z}$ are fixed for the remainder of the optimisation after the initial greedy variance selection.

Reinitialise method As previously mentioned, only the hyperparameters $\boldsymbol{\theta}$ are trained, and $\mathbf{Z}$ are frequently reselected with greedy variance (every 25 iterations) [3]. Once the reinitialisation returns a model with a worse ELBO ten times, a further 100 iterations are carried out to ensure convergence.

Fix-Retrain method This method builds on the *fix* method: we carry out an optimisation of $\boldsymbol{\theta}$ only, then, once the optimisation has converged, the model is reinitialised with greedy variance (for a total of two times) and we proceed to re-train the model jointly, this time including all parameters, $\boldsymbol{\theta}$ and $\mathbf{Z}$. The motivation behind this method is the model can quickly reach a near-optimal ELBO as it is not hindered by the training or reinitialisation of $\mathbf{Z}$ but then the subsequent steps allow us to reach an even higher ELBO. This method should always score better than the *fix* method.

Reinitialise-Retrain method This procedure begins in the same way as the *reinitialise* method, but when a worse model ELBO comes from the reinitialisation, a joint optimisation is carried out. This method builds on the method proposed in [3], but focuses on joint optimisation once a good ELBO is reached, rather than using up resources continuing reinitialising with greedy variance.

3.1.3 Experiments on UCI datasets

The aim of the following experiments is to test the claims made by Burt et al, who state that a joint training over all the parameters, both $\boldsymbol{\theta}$ and $\mathbf{Z}$, "in some cases is not worth the additional effort". Further still, we test additional training strategies to the reinitialise method proposed in the paper, notably *reinitialise-retrain*, and *fix-retrain*, to see whether a further improvement to the ELBO and consistency of training can be made.

Figure 3.2 shows how the different training strategies compare to each other when applied to training SGPR models with an SE kernel on several UCI datasets. The mean of the training curves of several different initilasations of the model hyperparameters $\boldsymbol{\theta}$ is plotted for the different training strategies [3].

While the *fix* method proved to train very fast, the final solution was sub-optimal. The inducing points being fixed proved to be the limiting factor in the quality of the model. As the *fix-retrain* method is identical to the *fix* method except that optimisation is continued, it always makes an improvement to the ELBO, and therefore renders the *fix* method obsolete. For the remainder of this thesis, the *fix* method shall not be considered.

Our results verify the claim made in [3]. Training the inducing variables from the beginning is indeed not worthwhile. On average, the *joint* method performed the most poorly out of all the training methods. This method resulted in an optimal ELBO for some hyperparameter initialisations and a suboptimal ELBO for others, as with so many parameters to train, the model often got stuck in a local optimum. Worse still, in the cases of larger datasets, the model was far off from converging to an optimal ELBO before an impractically large training time. For example, for the *keggundirected* dataset, extrapolating from the *joint* (blue) training curve, it appears that an optimal solution would not be reached before the order of 10^8 seconds. Employing one of the other training strategies always resulted in more desirable behaviour: an optimal solution was found much quicker and more consistently.

While the *fix-retrain* method was in most cases very consistent at producing an excellent ELBO, sometimes it underperformed. For the *keggundirected* dataset for example,

[3]An L-BFGS optimiser was used to optimised the SGPR models. Experiments in this thesis were carried out using GPflow [14], a Python library for employing Gaussian processes, on an Intel, Core i7-6700 3.40GHz CPU.

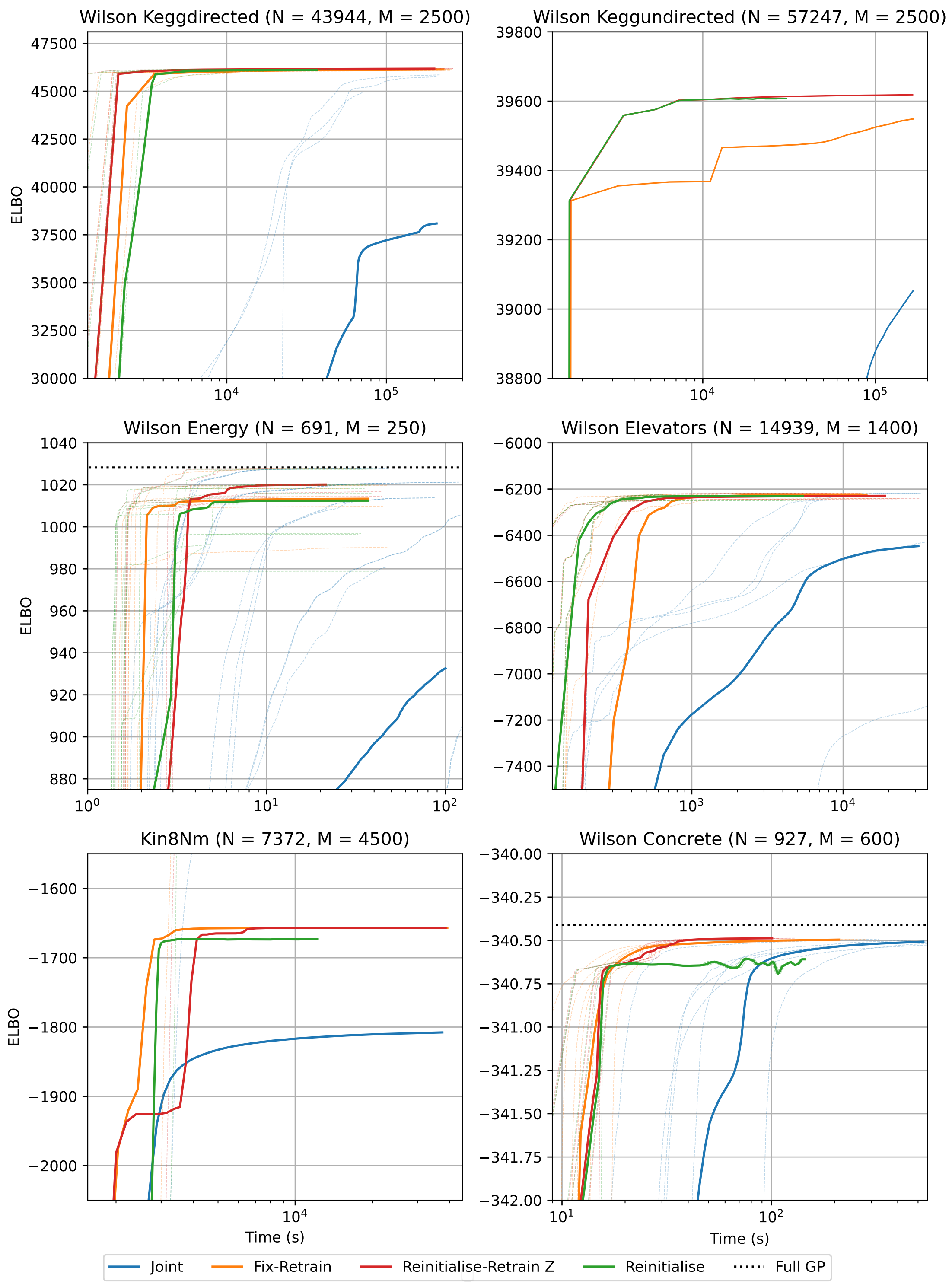


Figure 3.2: A time-wise comparison of the mean of the training curves for different training strategies (all except the *fix* method) on UCI datasets. The faint dotted lines are the individual training curves for different initialisations and training methods. For the *concrete* and *energy* datasets, a full GP LML is shown for comparison. See a comparison on negative log predictive density loss (NLPD) in figure A.2.

fixing the inducing variables for many iterations at the beginning of training was found to restrict the model from reaching an optimal value. This highlights the importance of the simultaneous adjustment of the $\mathbf{Z}$ and $\boldsymbol{\theta}$: a good inducing point selection begets well-optimised kernel hyperparameters and vice versa. Reinitialising the inducing variables regularly during training is essential for great consistency.

The *reinitialise* method posed by Burt et al. quickly and consistently reached a very high ELBO score. It proved to be much better method for inducing point training than the joint method. The *reinitialise-retrain* method, however, performed the best out of the training strategies. Giving the model flexibility to fine-tune its parameters when close to a global optimum almost always allowed a better optimum to be reached than the *reinitialise* method. *Reinitialise* rarely outperforms *reinitialise-retrain* for that matter.

3.2 Sparsity of SGPR

When selecting the number of inducing variables $\mathbf{u}$, we must select enough that they can fully summarise the full dataset such that we can infer the latent function from them, i.e. $p(\mathbf{f} \mid \mathbf{u}, \mathbf{y}) = p(\mathbf{f} \mid \mathbf{u})$. As this condition is a requirement for an ideal approximation to be made, we should expect that we will obtain a suboptimal solution when $\mathbf{u}$ are too few. For the datasets used for the experiments in figure 3.2, the M were chosen such that the optimal ELBO were reached [4].

For many datasets however, we only reach an optimal ELBO, and model performance, when the number of inducing points equals the full dataset size i.e. when $M = N$ when using the SE kernel. Unfortunately, there are not many datasets that exhibit this desired *sparsity effect* (where we get an optimal ELBO for $M < N$). In these cases, choosing $M < N$ makes a compromise to the optimal solution, and thus the key advantageous feature of SGPR is lost. The rarity of this sparsity effect renders SGPR less useful than is suggested in literature.

As this effect is apparently dataset dependent, we hypothesise that the choice of kernel plays an important role, and that a more appropriate kernel for the dataset should be able to model the dataset better with fewer inducing points, thus re-wielding SGPR with its useful trademark. In this section, we run experiments on both toy and real-world datasets to explore to what extent this is true – whether different kernels allow us to use fewer inducing points in our model but still to reach an optimal ELBO.

In these experiments, zeroth and first order arccosine kernels are used. These kernels are non-stationary, unlike SE or exponential kernels. The computation carried out by this family of kernels is very similar to that of a wide neural network [6], and thus could prove to be good at modelling step functions.

3.2.1 Toy dataset: Step function

The performance of arccosine kernels of orders 0 and 1 were compared to that of SE and exponential kernels on simple toy dataset, made from just a Heaviside step function. The dataset comprises of 3000 datapoints, has added noise, and is normalised. Figure 3.3 shows the prediction of an SGPR model using 50 inducing points for all the models.

[4]For the *concrete* dataset, there is in fact a noticeable discrepancy between the full GP solution LML and the best SGPR solutions, which implies that not enough inducing inputs were used for an optimal solution.

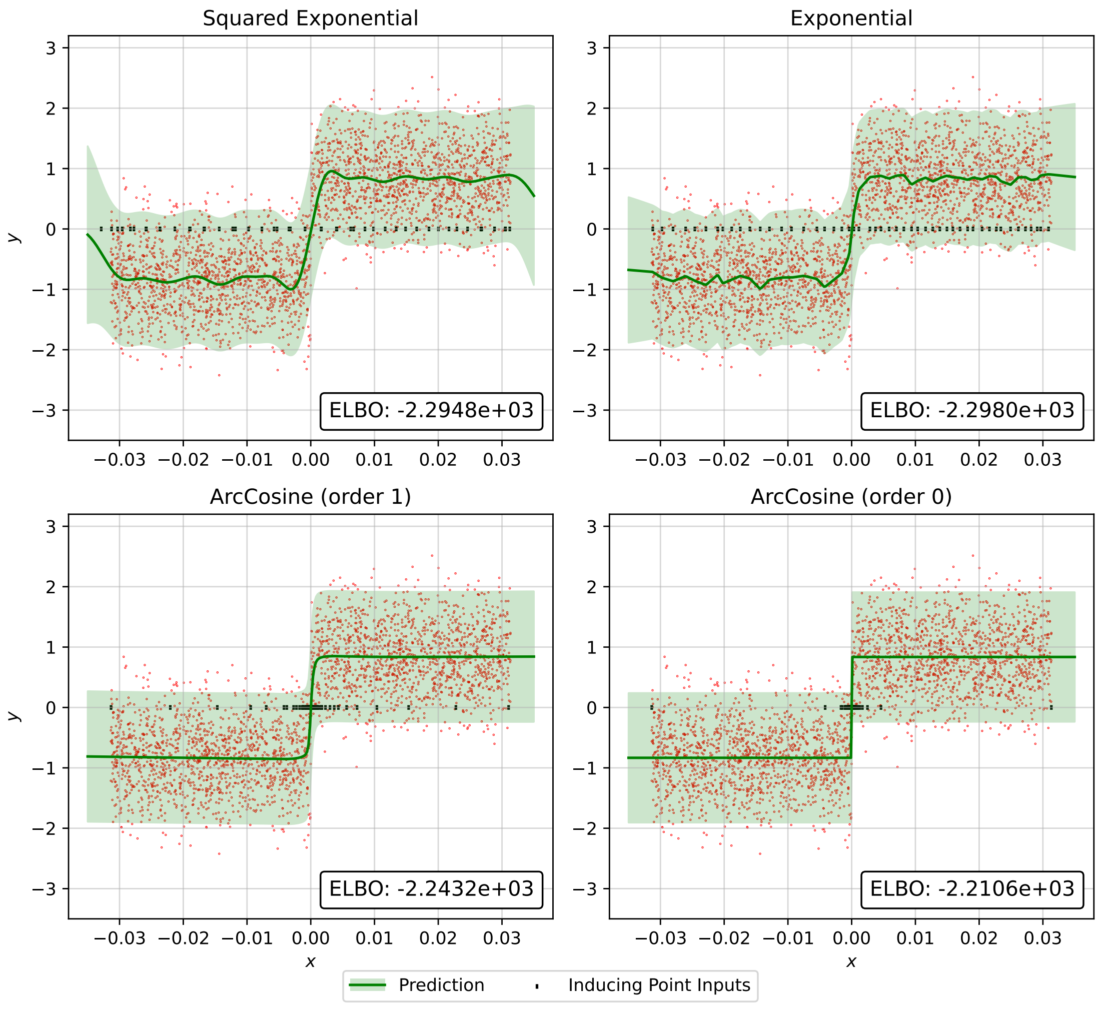
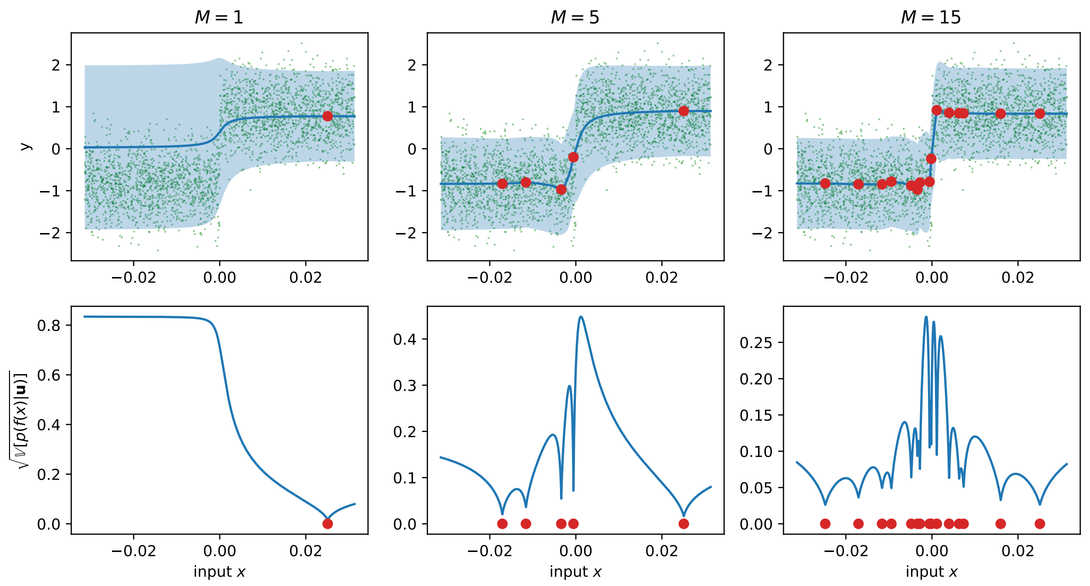


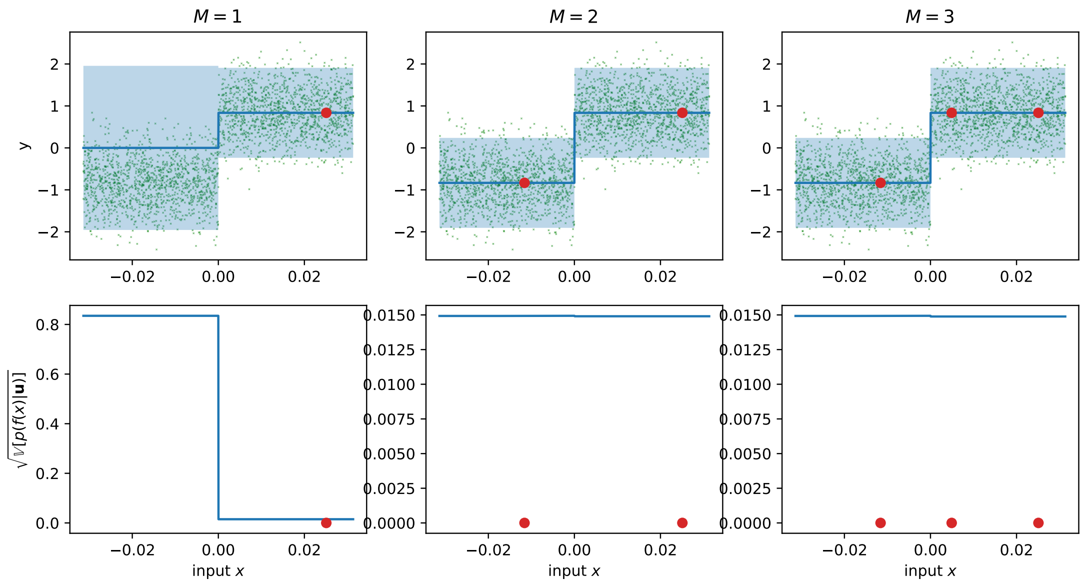
Figure 3.3: Comparison of posterior predictions, as well as inducing point locations, for SGPR models with different kernels. The dataset being modelled is made from a step function and has added noise. The non-stationary arccosine kernels model the step better than the stationary SE and exponential kernels.

The arccosine kernels model the step function excellently. They very precisely model the discontinuity, with the order 0 kernel giving the best ELBO and resembling the latent function almost exactly. The SE and Exponential kernels, on the other hand, struggle to recognise such structure in the data.

We can see that for the arccosine kernels, especially for order 0, the inducing points clump at the discontinuity. The model sees fit that it is necessary to focus computational resources at that location, and less elsewhere. This phenomenon occurs with the *reinitialise-retrain* method as well as the *reinitialise* method, which perform equally well – inducing points are selected in this way with greedy variance. Since the greedy variance method is only dependent on kernel and its hyperparameters, the kernel must be learning this feature very accurately. The choice of kernel has both an effect on where the inducing points are placed, and the final quality of prediction. It is evident here that the choice of kernel is a key decision in deploying SGPR models: a good kernel choice begets good inducing point locations and thus good model performance.



(a) Kernel with near-optimal hyperparameter settings (bias variance $= 10^{-5}$). In this case, inducing points are encouraged by the kernel to be selected near the discontinuity.



(b) Kernel with optimal hyperparameter settings (bias variance $> 10^{-20}$). In this case, the kernel is in its optimal setting, and the model only requires two inducing points. Additional inducing points will not reduce variance further.

Figure 3.4: Demonstration on how greedy kernel hyperparameters affect how inducing points are selected and how many are required for an good ELBO score and prediction. A dataset made from a Heaviside step function is modelled with an order 0 arccosine kernel with two different hyperparameter settings. A hyperparameter of the arccosine kernel, the bias variance, becomes large at the optimal solution.

A closer inspection into how inducing points are selected by greedy variance with an order 0 arccosine kernel for this dataset is given in figure 3.4. Inducing points are placed at the discontinuity when hyperparameter settings are not ideal. However, as hyperparameters improve to the optimal settings, the placement of the inducing points becomes rather arbitrary, so long as there is an inducing point either side of the step. A SGPR model with an arccosine kernel was trained on this dataset using just 2 inducing points. The model successfully and very quickly trained to an ELBO of $-2.2106 \cdot 10^{-3}$, scoring as well as the model with 50 inducing points. A good kernel choice allows us to not only model our data better, but also allows us to use fewer inducing points.

As the arccosine kernels are non stationary, selecting an inducing point at one location will affect the variance of the far from that location. The non-stationarity contributes towards the arccosine kernels modelling step functions very well with few inducing points.

On the left of figure 3.5, we see that different kernels give rise to different model sparsities. When enough inducing points are used, the exponential kernel has a better performance than the widely-used SE kernel, but when not enough are used, it performs worse, thanks to the better sparsity of the SE kernel.

The step function was also modelled with a summed kernel of a cubic and order 0 arccosine kernels. Adding extra flexibility of the cubic to the already ideal arccosine kernel did not improve the maximum ELBO, and actually, in some instances, caused problems in training — extra parameters to train, resulted in sub-optimal solution being found. Kernel flexibility is not always an advantage.

3.2.2 Toy dataset: Step + Cubic function

A similar experiment was carried out with a dataset composed of 3000 points and made from a latent function described by

$$f(x) = 50x^3 - 8x^2 - 1.5x - 0.2e^x + H(x) \tag{3.2}$$

where H is the Heaviside step function, and with added noise variance of 5. Again, the dataset had added noise and was normalised.

Six different kernels were tested on this dataset: two additional kernels made from a sum of an order 3 polynomial kernel and an arccosine kernel were introduced. It could be hypothesised that since the dataset itself comprises of a cubic and step function [5], these two kernels should model the data the most accurately. Figure 3.6 shows the predictions for the six models, all of which use 50 inducing points.

The 1st order arccosine performs the best as a standalone kernel, and is able to model the step function and cubic curve well, with a maximum ELBO of $4.2051 \cdot 10^{-2}$. It is the only standalone kernel that selects more inducing points closer to the discontinuity.

Despite the order 0 arccosine kernel not being an optimal kernel choice on its own, evident from the jagged predictions and poorer ELBO, it outperforms all the other kernels when summed with an order 3 polynomial kernel. The placement of inducing points is quite evenly spread. Once the model has found optimal hyperparameters for the kernel, it only needs one inducing point either side of the step to model it. Additional inducing points would then be determined by the cubic kernel component only; similar

[5]The motivation behind adding $-0.2e^x$ was to introduce a small perturbation to the dataset that was not cubic to make the problem more realistic.

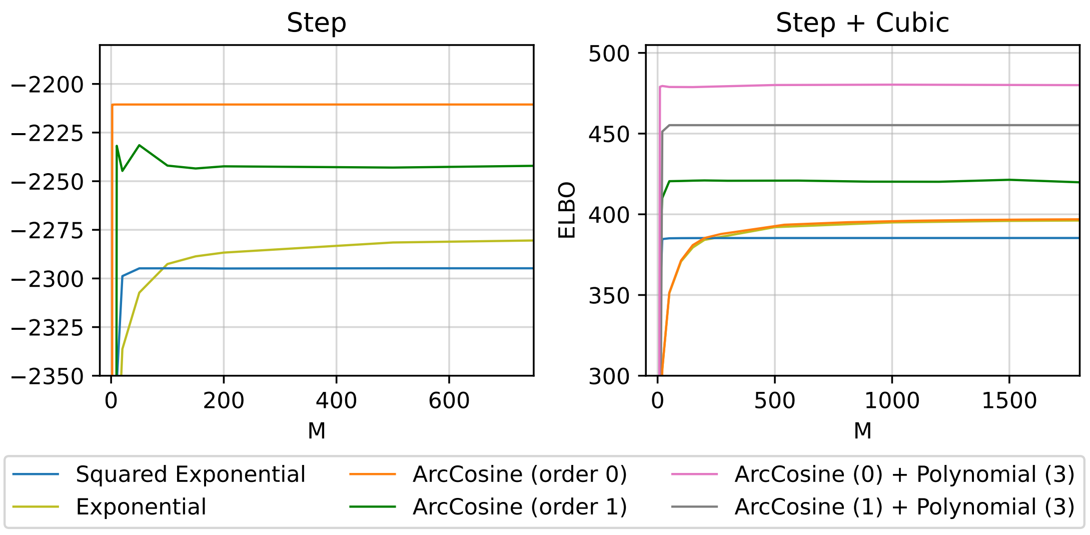


Figure 3.5: Maximum ELBO reached using the reinitialise-retrain training method on the dataset described by equation 3.2 of 3000 datapoints against the number of inducing points used, M. Not only do different kernels give different qualities of prediction, as indicated by the maximum ELBO reached, but they also exhibit different sparsity; for some kernels, like the arccosine (order 1) kernel and SE kernel, we can use fewer $\mathbf{Z}$ to reach a maximum ELBO.

to what happens in figure 3.4, more inducing points won't reduce the variance further for the optimised arccosine kernel, and thus this component of the kernel can no longer contribute to where the next inducing point is chosen.

In figure 3.5 (right), the number of inducing points is varied, and we plot the final ELBO reached for the M for the different models. The order 0 arccosine kernel needs a large M to reach an optimal solution. Adding a polynomial order 3 kernel to the order 0 arccosine kernel, however, not only improves the quality of the solutions, but allows a smaller M to be used; there is more of a sparsity effect. In this case, adding flexibility to the model *does* improve the model.

3.2.3 UCI datasets: *Airfoil* and *Kin40k*

To observe further the effect of kernel on sparsity, two real-world datasets, *Airfoil* and *Kin40k* (which was subsampled to 5000 datapoints), were tested, and their results were plotted in figure 3.7. As can be observed with the blue line, these datasets only reach an optimal solution with the SE kernel when $M = N$, hence why they were chosen for these experiments. The objective was to determine whether different kernels might allow fewer inducing points to be used in the model, as experiments in the toy dataset hinted at.

For the *Airfoil* dataset, while the order 1 arccosine kernel gave a better model performance than the SE kernel when using $M = N$, it suffered from poor sparsity. It was hypothesised that a sum of an arccosine and SE kernel might give the best sparsity and performance, as it has more flexibility than other kernels. Athough this summed kernel did outperform the SE kernel, its sparsity behavior is not ideal — still many inducing points are required to get close to modelling the dataset well. Furthermore, the kernel

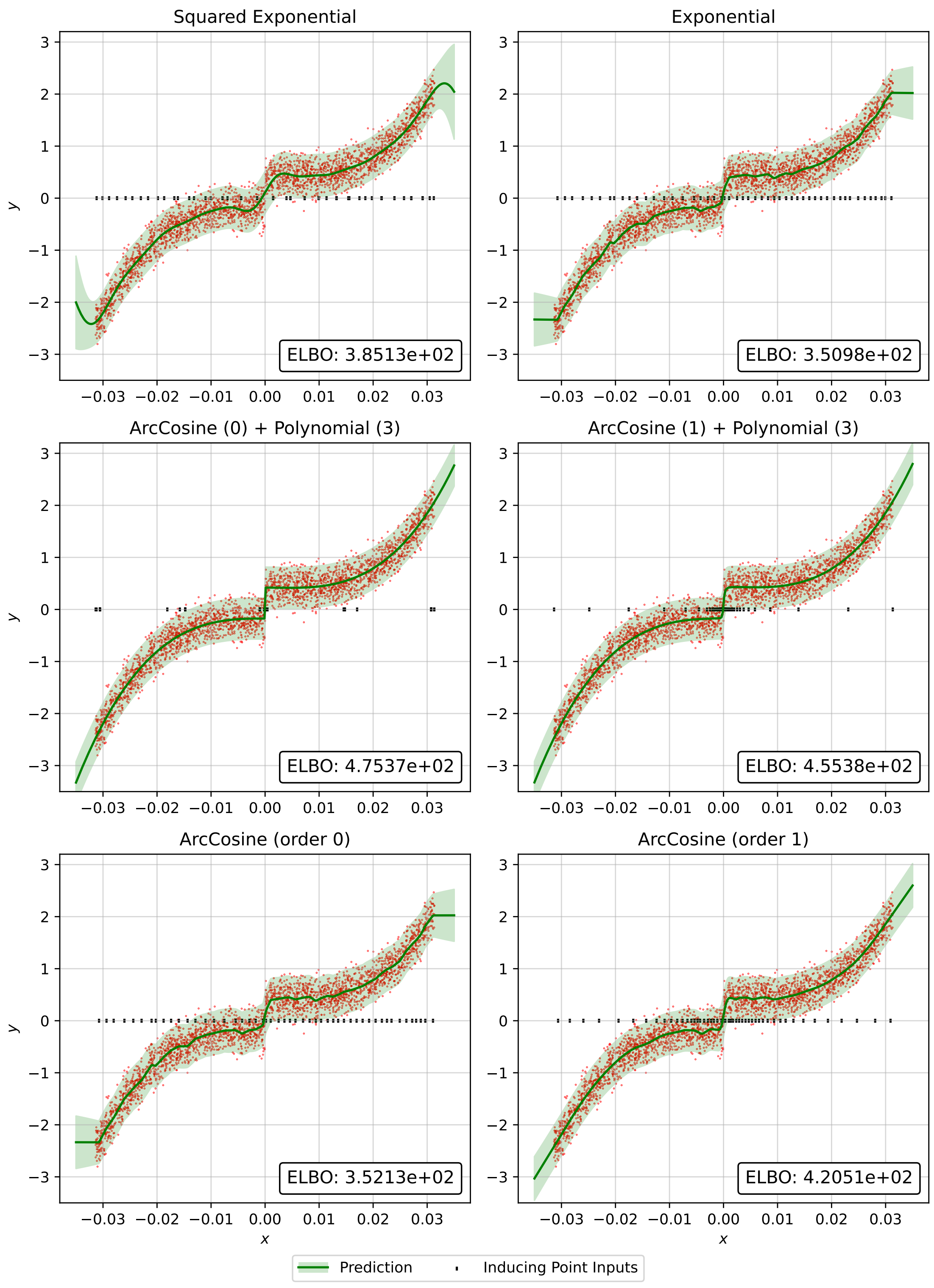


Figure 3.6: Comparison of posterior predictions, as well as inducing point locations, for SGPR models with different kernels on a dataset modelled using equation 3.2.

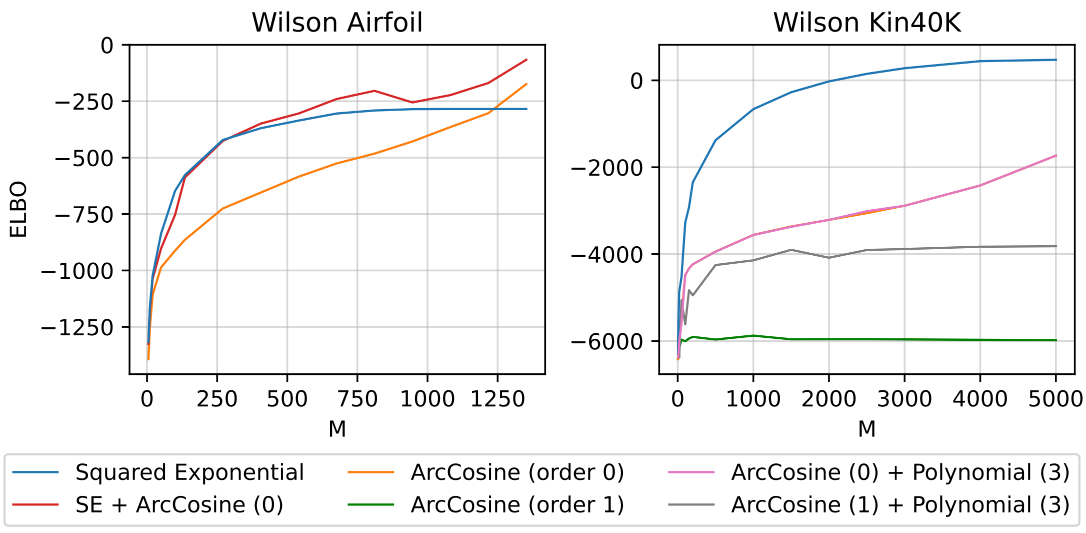


Figure 3.7: Maximum ELBO reached using the reinitialise-retrain training method on the *Airfoil* and (subsampled) *Kin40k* datasets against the number of inducing points used, M.

being more flexible resulted in slightly inconsistent performance, as was noticed in previous experiments. For example, we see the ELBO decrease when the number of inducing point increases from 811 to 946. This shows that at in this case, the model with 946 inducing points was not training optimally.

Similarly, for the *Kin40k* dataset, combinations of kernels generally resulted in erratic baviour, sometimes getting trapped in a local optimum, and sometimes performing well. The SE kernel outperforms the other kernels for this dataset: a better ELBO is scored for every M.

Summing the order 1 arccosine kernel with a cubic kernel did improve performance, while showing signs of sparse behaviour. However, adding a cubic to the order 0 arccosine kernel, on the other hand, did not improve performance. Either the latter model might have been getting stuck in a local optimum or simply the extra flexibility made no improvement to the model.

The kernels chosen in this experiment were performing better on these datasets than many other kernels. However, as there exists a huge vocabulary of available kernels, consisting of sums and products of kernels, and kernel families with multiple orders, it may be that not enough kernels were tested for us to observe the sparsity effect we desire. Employing a kernel search method, such as those described in [19, 9], might supply our model with a kernel that is very specifically suitable for the dataset, allowing us to sparsify our model better. Those methods may rescue us from having to venture into composing complicated kernels that could make the model perform inconsistently.

When the luxury of finding the perfect kernel is far-fetched, the SE kernel is an obvious choice. From the experiments, it is clear to see why the SE kernel is such a popular choice of kernel. On average it achieves an excellent ELBO score, gives the best sparse behaviour, and trains the most consistently.

3.3 Other Practical Considerations

Our choice of kernel can improve the quality of our solutions, and employing a training strategy can help us to find those solutions faster, and more consistently. However, the chance of the optimising not succeeding ideally is still non-negligible. The failures we can face are in one of two categories:

- Optimisation failures – we fail to find the optimal solution due to getting trapped in a local optimum.
- Numerical failures – the computer fails at carrying out an intermediate matrix calculation.

This section briefly discusses methods to mitigate against encountering these failures based on observations during the experiments in this thesis.

3.3.1 Dataset Normalisation

While GPs do not require dataset normalisation to function well, normalising the dataset, both the inputs and the outputs, did in fact improve optimisation for SGPR. We normalised the inputs such that the standard deviation and mean of the inputs were 1 and 0 respectively. By representing the different dimensions similarly, they are likely to have similar lengthscales. This can improve the speed and reliability during gradient based optimisation, since each step perpendicular to the gradient is more likely to be pointing towards the optimum. Normalising the inputs was a significant help in avoiding local optima.

As our GP prior was chosen to have a mean of zero, it also makes sense to normalise the outputs, $\mathbf{y}$. We can avoid having to sum our kernel with a constant kernel to account for the offset which would add an extra hyperparameter to our model [8].

3.3.2 Hyperparameter Initialisation and Constraints

It was observed in the case of the *kin8nm* dataset, for example, with a particular initialisation of $\boldsymbol{\theta}$, a maximum ELBO was reached at ≈ -2232 nats, far from the best solution found at ≈ -1370 nats which is obtained with a different hyperparameter initialisation (see figure 3.2). It was found that when a hyperparameter initialisation that was tailored to the dataset was used, the global optimum was reached consistently.

The procedure for determining these initial hyperparameters is described in [8]. For example, we can expect that the signal variance of the SE kernel, σ_f^2 would be on the same order of magnitude as the variance of the data in the output domain, i.e. $\sigma_f^2 \approx \mathbb{V}[\mathbf{y}]$. It is also wise to constrain the hyperperameters within a certain reasonable range; if the dataset is normalised, we can expect that the optimal lengthscales would be extremely small or large.

3.3.3 Cholesky Errors

Rather than calculating inverses and log determinants of relevant matrices directly in GP models, for improved numerical stability, Cholesky decompositions of the matrices are used instead [18]. However, despite being a more stable option, computational errors still often arise due to the finite precision to which numbers can be represented on a digital computer [3]; the Cholesky decomposition becomes *ill-conditioned*, whereby the model becomes very sensitive to small changes to the latent function values. Encountering these errors can prove to be a major inconvenience and means that we cannot proceed.

To improve stability and to help avoid these errors, a *jitter* matrix, a diagonal matrix with a small value ϵ, can be added to $\mathbf{K}_{mm}$ such that $\mathbf{K}_{mm} \to \mathbf{K}_{mm} + \epsilon\mathbb{I}_M$ in 2.12. Although this additional matrix prevents the optimal solution from being reached, the effect is usually negligible.

A high jitter value (10^{-5}) was used for the experiments in this thesis and yet, errors were still encountered frequently. A tactic to resolve the error was to reinitialise the inducing variables when the error occurred. This allowed the optimisation to continue for many further iterations before again encountering another error. If the errors could not be avoided in a particular training run, then the experiment had to be restarted with either a different hyperparameter initialisation or initial first inducing point location (as described in §3.1.1).

Another strategy that was not tested in this thesis, suggested by a collaborator, Sebastian Ober, was to use a *dynamic* jitter that increased when an error was encountered, providing a larger buffer against the ill-conditioning when it is needed. Ober reported to have faced fewer Cholesky errors in his experiments thanks to the dynamic jitter. Avoiding cholesky errors is vital for the success of the model, so this method could be an important tool for improving the robustness of training in GPs.

3.4 Summary of Findings

We summarise the main findings of this thesis below.

- Making use of greedy variance during training can greatly improve the speed of optimisation. A reinitialisation procedure, as described by Burt et al. in [3] is a strategy that evidently outperforms the joint training procedure, at both consistency and speed of finding an optimum solution. In this thesis, we build upon that method to propose the *reinitialise-retrain* method, which in general reaches an optimal solution the fastest and most consistently, especially for real world datasets.

- Kernel choice can indeed have an effect on sparsity: in some cases where the kernel fits the data very well, very few inducing points are required. Despite this being observed for toy datasets, many real-world datasets still suffer from not being sparsifiable (where we can obtain an optimal ELBO for $M \leq N$). Research in testing a wider vocabulary of kernel on more datasets could reveal kernels that fit better to certain datasets and show better sparsity than the SE kernel, which currently boasts best all-round applicability for SGPR. Using a kernel search method before training may improve sparsity in SGPR models.

- Its best to have a kernel that suits the data the best, not necessarily the most flexible kernel. In the case that an ideal kernel cannot be found, a flexible kernel may improve sparsity or model accuracy, but it will be at the cost of troublesome optimisation.

Chapter 4

Final Remarks

This thesis discusses practical insights into SGPR, a scalable Gaussian process method that shows excellent performance at regression tasks. Despite having many theoretical benefits, SGPR has practical problems that have not been thoroughly addressed in the GP community.

Being able to use fewer inducing variables than the full dataset size is a key requirement for SGPR to be practically useful. For many datasets tested, SGPR struggled to summarise the dataset with $M < N$.

In our experiments, the SE kernel proved why it is so popular. On average it achieved the highest model accuracy as well as being able to sparsify the SGPR model the best. That being said, there exist instances where other kernels show better sparsity and performance.

For example in the case of the arccosine kernel being used to model a dataset made from a step function, very few inducing points are required to get model the step function excellently. The kernel itself seems to be able to learn features in the data. As the kernel was optimised, it was able to recognise that more inducing points should be placed around the discontinuity, without needing to see the output data. And as it gets optimised further, fewer inducing variables are required to summarise the data. This begs the question as to whether the number of inducing points can be reduced as the hypeparameters of the model get closer to the optimal solution.

While SVGP (mentioned in §2.2.2) is infererior at carrying out regression tasks for the same number of inducing points as SGPR, its flexibility might allow it to be more sparsifiable than SGPR. It would be interesting to observe how sparsifiable SVGP is for datasets where the likelihood is non-Gaussian, like for classification tasks.

We observe, though experements on UCI datasets, that the greedy variance method for selecting inducing variables is very useful for optimising SGPR models. We can drastically cut down training time while improving consistency at reaching an optimal solution. However, even when methods from this thesis are employed to speed up training, SGPR still remains expensive for datasets that have on the order of 10^5 datapónts.

A noteworthy area of research that could greatly benefit GP is understanding how stochastic gradient descent (SGD) works in correleted settings. Chen et al. take a step closer towards being able to apply SGD in GPs [5]. Being able to apply stochastic gradient descent to SGPR would open up a whole new world usefulness for the model. In the meantime, until more attention has been brought to the matter, and more progress has been made, SVGP may be a better option for regression tasks for very large datasets.

For some cases, since a full GP has M fewer parameters to train (we are only required to train $\boldsymbol{\theta}$), we have lower dimensional space for optimisation, and the full GP is advantageous to use. In fact, for most cases where $N < 1000$, training of the full GP results in a faster and more consistent optimal solution than SGPR, even when the number of inducing points is small. There is, however, threshold at around $N = 5000$ where the full GP is practically infeasible for most computers, and a sparse approximation must be used.

A repository for carrying out the experiments is available to view here.

Bibliography

[1] Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen. Understanding probabilistic sparse gaussian process approximations. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29 (NIPS)*, pages 1533–1541. Curran Associates, Inc., 2016.

[2] James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[3] David Burt, Carl Edward Rasmussen, and Mark van der Wilk. Rates of convergence for sparse variational Gaussian process regression. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 862–871, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[4] Roberto Calandra, Jan Peters, Carl Edward Rasmussen, and Marc Peter Deisenroth. Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345, 2016.

[5] Hao Chen, Lili Zheng, Raed AL Kontar, and Garvesh Raskutti. Stochastic gradient descent in correlated settings: A study on gaussian processes. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2722–2733. Curran Associates, Inc., 2020.

[6] Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. *Advances in neural information processing systems*, 22, 2009.

[7] Andreas Damianou and Neil D. Lawrence. Deep Gaussian processes. In Carlos M. Carvalho and Pradeep Ravikumar, editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 207–215, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR.

[8] Marc Peter Deisenroth, Yicheng Luo, and Mark van der Wilk. A practical guide to gaussian processes, 2020. https://infallible-thompson-49de36.netlify.app/#section-1.

[9] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[10] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208, 2005.

[11] James Hensman, Nicolo Fusi, and Neil Lawrence. Gaussian processes for big data. *Uncertainty in Artificial Intelligence - Proceedings of the 29th Conference, UAI 2013*, 09 2013.

[12] Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Bojan Likar. Predictive control with gaussian process models. In *The IEEE Region 8 EUROCON 2003. Computer as a Tool.*, volume 1, pages 352–356. IEEE, 2003.

[13] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003. Available from http://www.inference.phy.cam.ac.uk/mackay/itila/.

[14] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

[15] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. International Joint Conferences on Artificial Intelligence, Inc., 2017.

[16] Duy Nguyen-Tuong and Jan Peters. Learning robot dynamics for computed torque control using local gaussian processes regression. In *2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*, pages 59–64. IEEE, 2008.

[17] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(65):1939–1959, 2005.

[18] Carl Edward Rasmussen and Christopher K I Williams. *Gaussian processes for machine learning.* MIT Press, London, England, 2005.

[19] Fergus Simpson, Ian Davies, Vidhi Lalchand, Alessandro Vullo, Nicolas Durrande, and Carl Edward Rasmussen. Kernel identification through transformers. *Advances in Neural Information Processing Systems*, 34:10483–10495, 2021.

[20] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 567–574, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

[21] Mark Van der Wilk. *Sparse Gaussian process approximations and applications.* PhD thesis, University of Cambridge, 2019.

[22] Andrew Wilson, David Knowles, and Zoubin Ghahramani. Gaussian process regression networks. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 1, 10 2011.

[23] Andrew Gordon Wilson, Zhiting Hu, Ruslan R. Salakhutdinov, and Eric P. Xing. Deep kernel learning. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 370–378, Cadiz, Spain, 09–11 May 2016. PMLR.

[24] Andrew Gordon Wilson, Zhiting Hu, Russ R. Salakhutdinov, and Eric P. Xing. Stochastic variational deep kernel learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Appendix A

Extra plots

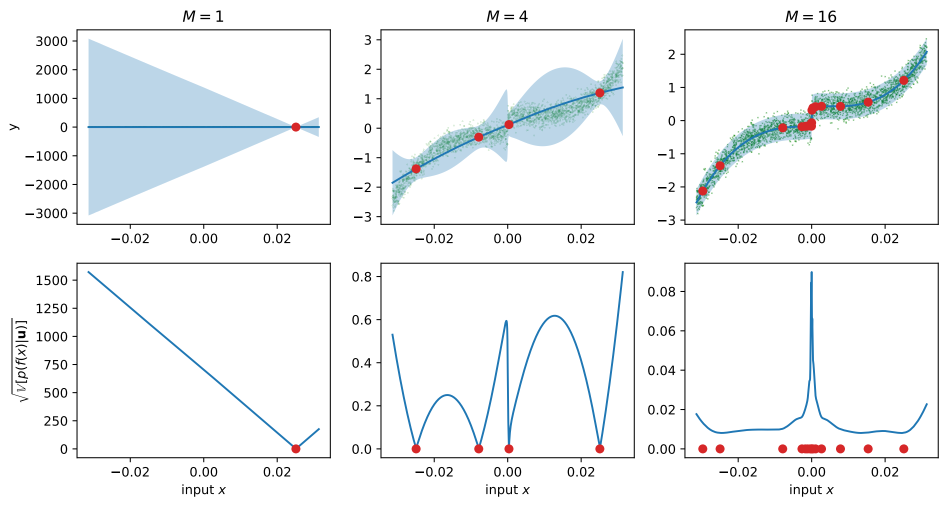


Figure A.1: How inducing points are chosen one at a time for the toy dataset, with a kernel composed of the sum of a cubic and order 1 arccosine kernels. With optimal hyperparameters, the kernel appears to have learned important features in the dataset. The kernel encourages inducing points to be selected at the location of the discontinuity, explaining the clumping seen in figure 3.6.

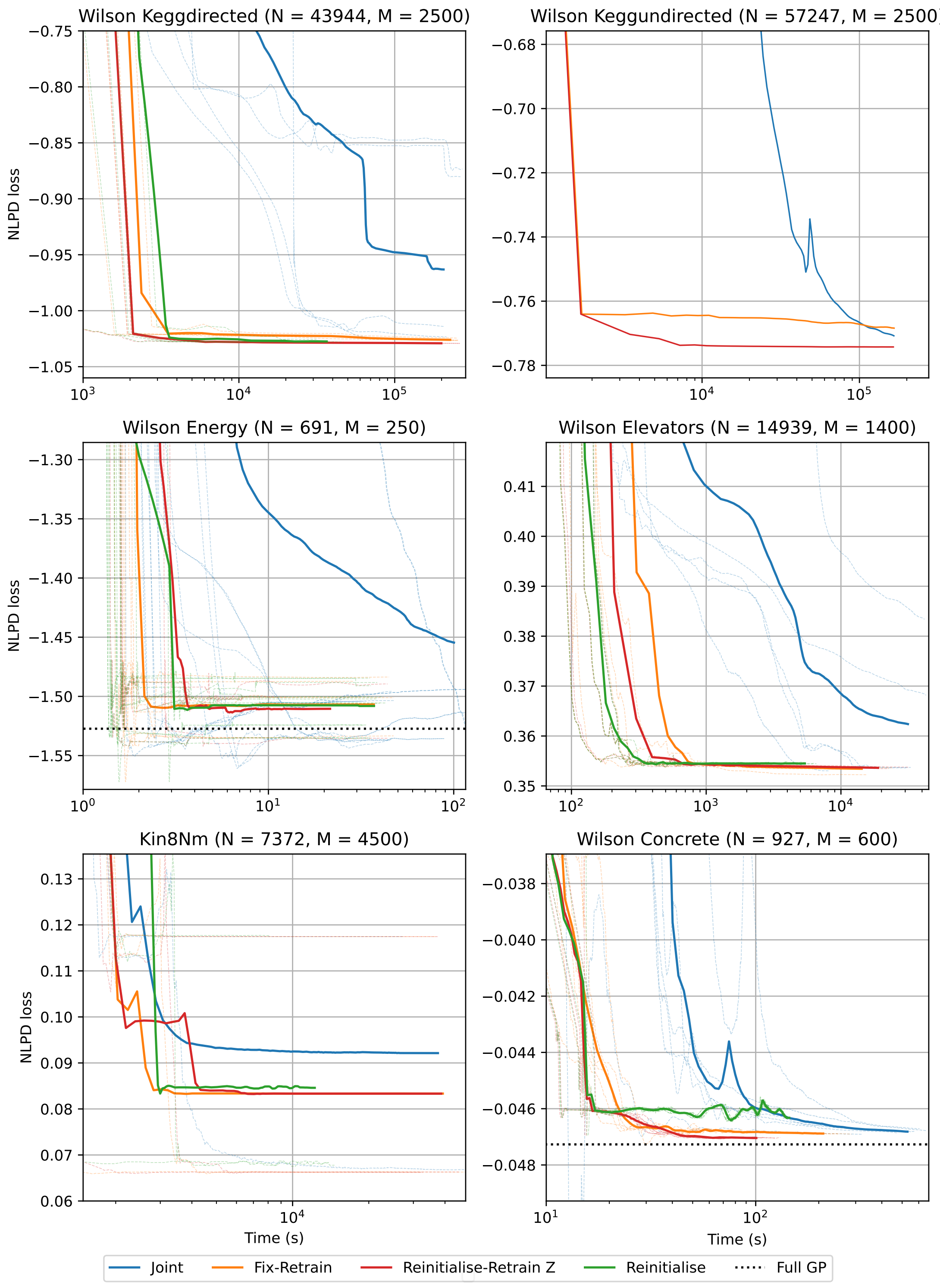


Figure A.2: A time-wise comparison of the mean of the NLPD loss training curves for different training strategies (all except the *fix* method) on UCI datasets. The faint dotted lines are the individual training curves for different initialisations and training methods. For the *concrete* and *energy* datasets, a full GP NLPD loss is shown for comparison.