Tudor Paraschivescu

Multi-output Gaussian Process Regression at Scale

MPhil in Machine Learning and Machine Intelligence

Churchill College

August 20, 2020

Abstract

Multi-output regression problems are a subset of supervised learning problems where we try to infer multiple scalar outputs from a given input. Usually, the outputs in such problems exhibit inter-dependencies, so a performant model will take those dependencies into account during training and inference. Gaussian Process models perform well on single-output problems, but their extension to multi-output problems comes at the cost of significant computational expenses and limited expressivity. The Gaussian Process Autoregressive Regression model allows us to cleanly extend the power of GPs to multi-output problems through the chaining of Gaussian Processes. The goal of this thesis is to scale the GPAR model so that it can be applied to large datasets in a reasonable timeframe.

Declaration

I, Tudor Paraschivescu of Churchill College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: Tudor Paraschivescu

Date: 16.05.2019

Acknowledgements

I would first like to thank my supervisors Dr Richard E. Turner with Will Tebbutt and Wessel Bruinsma of the Department of Engineering at the University of Cambridge. The door to their (virtual) office was always open whenever I ran into a trouble spot or had a question about my project or writing. They consistently allowed this paper to be my own work, but steered me in the right direction whenever the need asked for it.

I would also like to express my very profound gratitude to my parents and my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Proforma

Candidate Number:	**G714**
College:	**Churchill College**
Project Title:	**Multi-output Gaussian Process Regression at Scale**
Examination:	**MPhil in Machine Learning and Machine Intelligence**
Word Count:	**13935**
Project Originator:	Dr Richard E. Turner, Will Tebbutt, and Wessel Bruinsma
Project Supervisor:	Dr Richard E. Turner, Will Tebbutt and Wessel Bruinsma

Original aims of the project

To scale the Gaussian Process Autoregressive Regression to larger time-series datasets using approximate inference through methods such as pseudo-points[Seeger et al., 2003] and state space[Sarkka et al., 2013] approximations. This would allow us to achieve more accurate predictions in areas such as climate science.

Work Completed

A library that allows the user to perform GPAR inference on time-series datasets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multi-output regression problems are a subset of supervised learning problems where we try to infer multiple scalar outputs from a given input. Usually, the outputs in such problems exhibit inter-dependencies, so a performant model will take those dependencies into account during training and inference.

Two examples of multi-output regression problems can be seen in Figure 1.1. The left figure [Requeima et al., 2018] shows the classic example where we attempt to infer CO2 emissions, temperature, and ice cap levels over time. The problem presented by the right figure is trying to extrapolate EEG sensorial data for three sensors given the first half of the readings and the full readings of the other sensors.

A Gaussian Process (GP) is a stochastic process such that any finite subset of the random variables it models is distributed according to a multivariate Gaussian distribution. Gaussian Process models are powerful and popular for doing single-output regression [Williams and Rasmussen, 2006] as they directly capture model uncertainty and allow the user to input prior knowledge into the model through the selection of kernel functions. However, naively extending GP models to multi-output regression problems leads to a blow-up in complexity and limited representation.

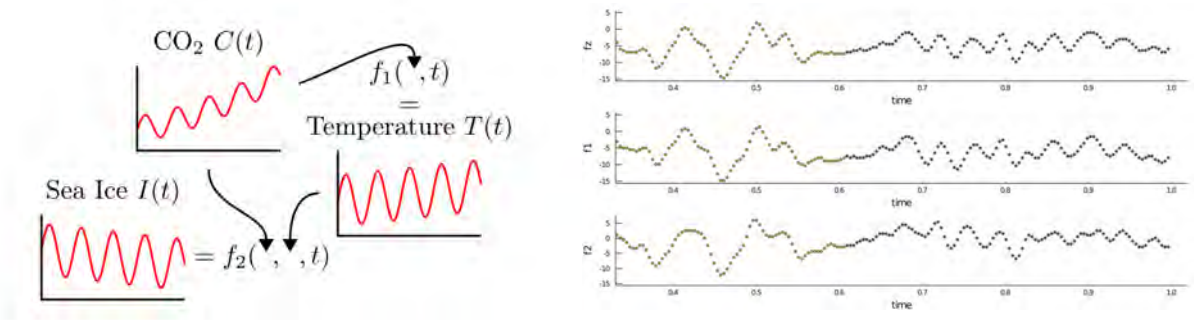

Figure 1.1: Examples of multi-output regression problems.

The Gaussian Process Autoregressive Regression (GPAR) model addresses this limitation by chaining GP models in a predetermined order based on conditional probabilities. Evaluations [Requeima et al., 2018] of GPAR on small datasets show that it can more accurately capture dependencies between features than usual GPs. However, one of the limitations of GPAR is that it has $O(MN^3)$ [1], which is unreasonably expensive for large datasets where N can take values in the tens of thousands.

This project will scale GPAR through a modified popular pseudo-point [Seeger et al., 2003] technique called the Deterministic Training Conditional (DTC) [Quiñonero-Candela and Rasmussen, 2005], combined with state space [Sarkka et al., 2013] approximations in order to speed up computation. The state space approximation is required because our DTC modification leads to the need of doing a Cholesky decomposition of an $N \times N$ matrix, which is a cubic operation. Using state space methods allow us to get around the need for this decomposition.

[1] N being the number of observations and M being the number of output dimensions

Chapter 2

Literature review

This chapter begins by providing a brief overview of Gaussian Processes and the Gaussian Process Autoregressive Regression model. Afterwards, we will study established ways of scaling usual GP models which we'll later use for scaling GPAR.

2.1 Gaussian Processes

The Gaussian Process (GP) model attempts to estimate the underlying function of a supervised learning task by assuming Gaussian distribution at each random variable location. GP models are powerful tools known for their modularity, tractability, and interpretability [Williams and Rasmussen, 2006]. Gaussian Processes can model nonlinear dependencies between inputs and outputs by letting the covariance between two points take the form of nonlinear functions of these points.

2.1.1 The Gaussian Process prior

The Gaussian Process model is the extension of a multivariate Gaussian to infinitely many variables. Wikipedia defines the Gaussian Process as "a stochastic process, such that every finite collection of those random variables has a multivariate normal distribution".

A D-dimensional multi-variate Gaussian distribution is fully specified by a mean vector μ and a covariance matrix Σ as in Equation 2.1. A Gaussian Process can be thought of as the extension of the multivariate Gaussian distribution to infinitely long vectors.

$$\mathcal{N}(\mathbf{x}; \mu, \Sigma) = (2\pi)^{-D/2}|\Sigma|^{-1/2}\exp(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)) \tag{2.1}$$

The GP is fully defined by a mean function $m(x)$ and a covariance function $k(x, x')$ (also known as the kernel). Any finite collection of random variables from a Gaussian Process form a multivariate Gaussian distribution with a mean and a covariance matrix

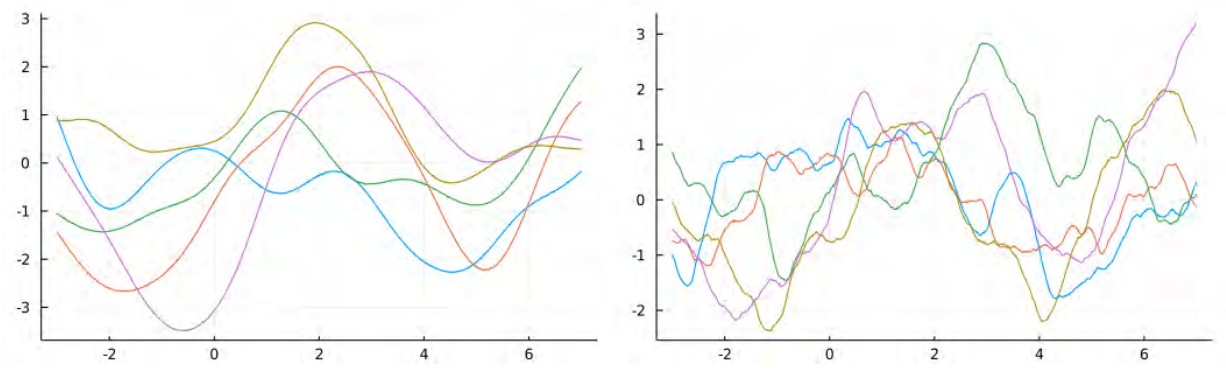

Figure 2.1: Functions drawn from two GPs with sinusoidal mean function and squared exponential and, respectively, Matern-3/2 kernels. The squared exponential covariance function leads to infinitely differentiable sample functions, whereas its Matern-3/2 outputs once differentiable covariance functions. Therefore, we expect smoother function in the left figure, which is what we observe.

obtained by evaluating the mean and covariance functions at the specific points. This is only possible if the kernel is a positive definite function, such that it outputs positive definite matrices when evaluated.

The reason why we can work with GPs without having to store infinite information is thanks to the marginalization property. This property is given for multivariate Gaussian distributions in Equation 2.2, but it extends cleanly to Gaussian Processes; *i.e* in the case of GPs, one could think of $\mathbf{y}$ as an infinitely long vector and the marginalization property still holds. This is an amazing property of Gaussians that allows us to ignore the random variables in which we are not interested. Therefore, we can work with the finite set of random variables which interest us whilst still having an underlying model defined on an infinite domain.

$$p(\mathbf{x}, \mathbf{y}) = \mathcal{N}\left(\mathbf{x}, \mathbf{y}; \begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{B^T} & \mathbf{C}\end{bmatrix}\right) \rightarrow p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{a}, \mathbf{A}) \tag{2.2}$$

Figure 2.1 shows sample functions drawn from two GPs with similar sinusoidal mean but different covariance matrices. These kind of plots are useful for getting an understanding of how the covariance function impacts the shape of the output. To generate a sample function, we first evaluate the GP mean and kernel functions at 1000 regularly spaced points $\mathbf{x}$ and retrieve the multi-dimensional Gaussian distribution from which we generate samples.

2.1.2 Posterior Gaussian Process

In the previous section, we discussed the Gaussian Process prior and showed some example functions generated from it. Now we'll go into how a Gaussian Process can be updated to take into account observations. This will be called a posterior Gaussian Process. The posterior GP is another GP with mean and kernel functions obtained from linear combinations between the prior mean and kernel functions and the mean vector and covariance matrix of the multi-dimensional Gaussian distribution obtained by evaluating the prior GP at the observation points. This implies that:

- In particular, a posterior GP is still a Gaussian Process, so it does have the same properties as a prior GP.
- A posterior GP can be thought of as an updated/new prior in light of observed data. This means that GP models fit in the Bayesian inference framework in that it can be trained "online" as we receive more data.

Expanding on Equation 2.2, the marginalization property also allows us to get an analytic solution for the conditional probability as in Equation 2.3. Again, we only care about the random variables at the inference locations and the observed locations, the other random variables being marginalised out.

$$p(\mathbf{x}, \mathbf{y}) = \mathcal{N}\left(\mathbf{x}, \mathbf{y}; \begin{bmatrix}\mathbf{a} \\ \mathbf{b}\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{B} \\ \mathbf{B^T} & \mathbf{C}\end{bmatrix}\right) \rightarrow p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}; \mathbf{a} + \mathbf{B}\mathbf{C}^{-1}(\mathbf{y} - \mathbf{b}), \mathbf{A} - \mathbf{B}\mathbf{C}^{-1}\mathbf{B}^T) \tag{2.3}$$

Equation 2.3 also describes how to naively perform inference in a Gaussian Process. Given observations $\mathbf{y}$, we perform inference at the points $\mathbf{x}$ by performing linear combinations on the parameters of the multi-dimensional Gaussian distributions retrieved by evaluating the GP at $\mathbf{x}$ and $\mathbf{y}$. This is another example of where the marginalisation property saves us from having to work with infinitely many random variables.

The bottleneck with the previously described approach is that it requires inverting the matrix $\mathbf{C}$, which is a $N \times N$ matrix, where N is the number of observations. This means that computing the posterior is a $O(N^3)$ operation, thus leading to infeasible runtimes when the number of observations exceeds a certain (low) threshold. We will discuss more about improving the complexity when we discuss pseudo-point approximations and the Deterministic Training Conditional in Section 2.3.

An example of how conditioning impacts the GP can be seen in Figure 2.2. We plot function drawn from the same GPs as in Figure 2.1, but we condition them on two observations. We can see how the functions get clamped around the observation points, and then exhibit more variance the further we get away from them. This simulates the expected

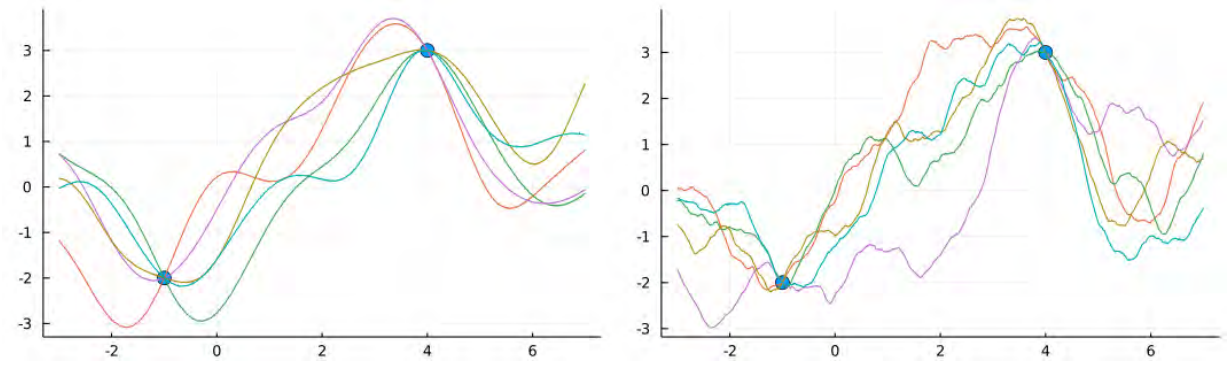

Figure 2.2: Functions drawn from two posterior GPs with sinusoidal mean function and squared exponential and, respectively, Matern-3/2 kernels. We condition each GPs on the two blue observations. The observed points are the same in both cases.

behaviour, we have more information about an inference location the closer it is to an observation. We also notice that the kernel of the posterior process retains the properties of the prior's kernel; the functions sampled from the GP with a squared exponential kernel are smoother than the ones sampled from its counterpart.

2.1.3 Covariance functions

The covariance function is the most important element to consider when creating a Gaussian Process. As we previously explored, the kernel directly impacts the shape of the functions sampled from the GP. There are multiple classes of covariance functions; so far we've used the squared exponential (Equation 2.4) and Matern-3/2 (Equation 2.5) covariance functions for generating the previous figures. There isn't a one size fits all covariance function, an experienced engineer will change the prior covariance function based on the data they're modelling.

$$k(x, x') = \sigma^2 exp\left(-\frac{(x - x')^2}{2l^2}\right) + \sigma^2_{noise}\delta_{xx'} \tag{2.4}$$

$$k(x, x') = \sigma^2\left(1 + \frac{\sqrt{3}|x - x'|}{l}\right) exp\left(-\frac{\sqrt{3}|x - x'|}{l}\right) \tag{2.5}$$

These covariance functions introduce hyperparameters that require tuning. The specific hyperparameters introduced can change for different classes of kernels, but they usually include the characteristic lengthscale l and the magnitude scale parameter σ^2. Informally, the lengthscale specifies the distance from which two points will be uncorrelated, and the magnitude scale parameter controls the overall variance of the process. The optimization of

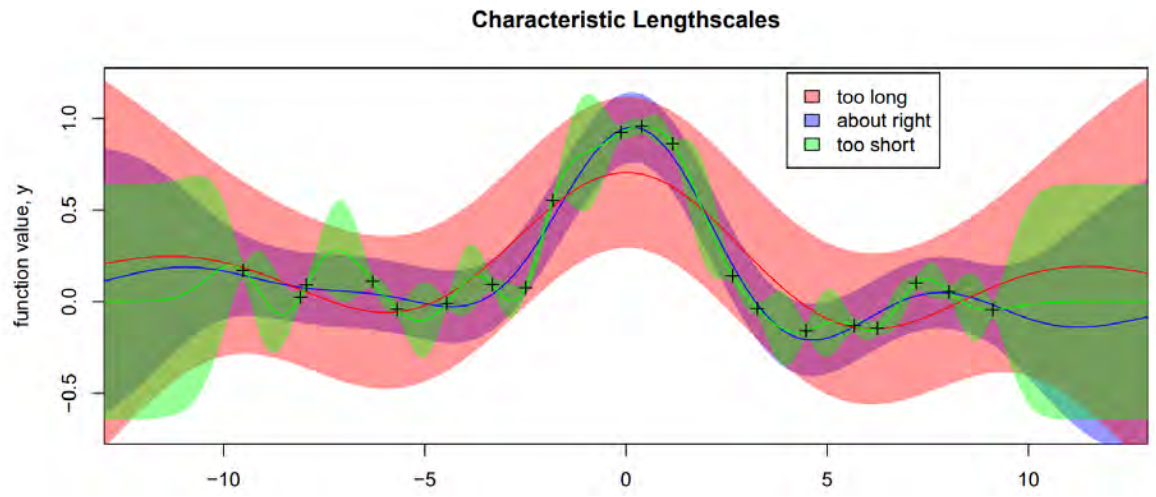

Figure 2.3: GP predictions when using different lengthscales. Credit for graphic goes to Carl Rasmussen.

these parameters is usually accomplished through Maximum Marginal Likelihood Estimation since we can analytically compute the marginal likelihood of a set of hyperparameters given the observations because we work with Gaussian distributions.

We should also note that we use the term "marginal likelihood" since we only care about the likelihood at the locations (random variables) of interest; *i.e.* the probability of the observed random variables given the hyperparameters. Marginal likelihood gets its name from the process of marginalising out the latent function values. In the literature, people prefer the term "marginal likelihood", so we'll stick to using this term.

An example of how hyperparameters can affect the GP can be seen in Figure 2.3, where we study how the value of the lengthscale l can affect predictions. Large lengthscales mean that observations have a "larger reach". However, this might lead to underfitting. In contrast, short lengthscales can lead to overfitting since we only look at a limited horizon.

It is known that Maximum Likelihood Estimation is prone to overfitting. However, Maximum Marginal Likelihood Estimation (as in our case) circumvents MLE's predisposition to overfitting since we are marginalising out the latent function values, thus doing Bayesian inference over these function values; for every set of hyperparameters we consider every function value. This leads to to a kind of constrained optimization since we are optimizing a small number of hyperparameters given a much larger set of observations. One can draw the parallel between this and Bayesian Occam's razor which says that a more complex model can accommodate more data, but complex models are automatically penalised through Bayesian inference, thus automatically keeping the balance.

Some other useful covariance functions would include the periodic covari-

ance functions [MacKay, 1998], neural-network covariance functions [Lee et al., 2017, Matthews et al., 2018, Wilson et al., 2016, Calandra et al., 2016], and composite versions of covariance functions [Duvenaud et al., 2013].

2.2 Gaussian Process Autoregressive Regression

Gaussian processes offer a powerful approach for tackling single-output regression problems case since they offer modularity, tractability, and interpretability [Williams and Rasmussen, 2006] coupled with a probabilistic approach to regression. However, they fall short when doing multi-output regression since they produce models that are computationally demanding and are limited in capturing output inter-dependencies; they can only capture linear relationships between outputs. During this section, we'll look at the Gaussian Process Autoregressive Regression (GPAR) model [Requeima et al., 2018] and how it can address this issue.

2.2.1 The GPAR model

The main problem behind extending GPs to multi-output regression is that it fails to capture the complex inter-dependencies between the outputs. There are multiple ways to capture these correlations if they are linear and fixed across the input space [Nguyen et al., 2014, Dai et al., 2017]. However, they fail when the dependencies between the outputs are non-linear, which is a big downside since non-linear interdependency capture is one of the strengths of GPs.

The idea behind the GPAR model is to use the product rule to decompose the joint output distribution into a set of conditionals, each of which is modelled by a standard GP.

If, like in [Requeima et al., 2018], we consider modelling M outputs $p(y_{1:M}(x)) = (y_1(x), \dots y_M(x))$, then applying the product rule yields Equation 2.6. This states that $y_1(x)$ is first generated from x according to an unknown function f_1; that $y_2(x)$ is generated from x and y_1 according to an unknown function f_2; *et cetera*.

$$p(y_{1:M}(x)) = \underbrace{p(y_1(x))}_{f_1} \underbrace{p(y_2(x)|y_1(x))}_{f_2} \dots \underbrace{p(y_M(x)|y_{1:M-1})}_{f_M} \tag{2.6}$$

GPAR uses Gaussian processes to model these unknown functions f_i. Although this leads to individual conditionals from Equation 2.6 being Gaussian, the overall joint distribution is non-Gaussian and usually intractable. However, we can sequentially generate samples from the conditionals in order to generate joint samples.

Learning is accomplished by sequentially conditioning the GPs on the previous outputs (and x) in the pre-specified order. More formally, let $y_m^{(n)} = y_m(x^{(n)})$ denote the n'th

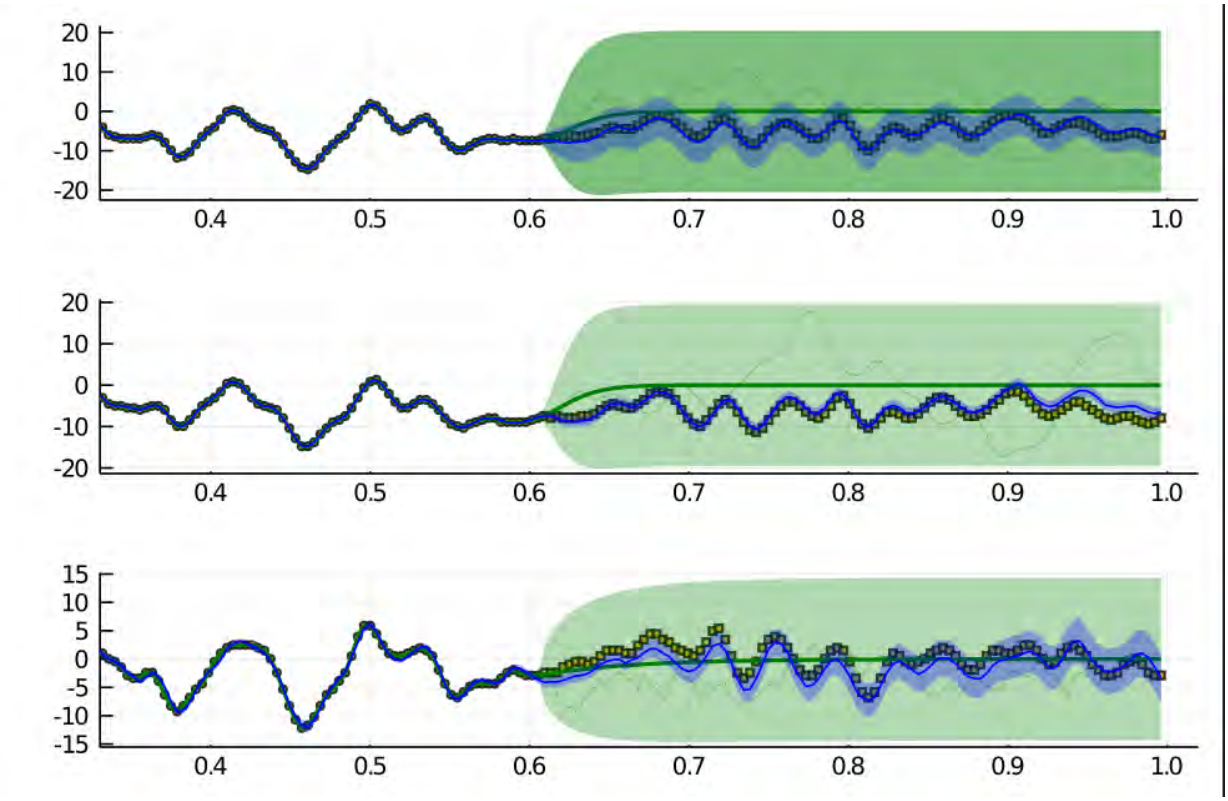

Figure 2.4: GPAR vs independent GPs on the EEG dataset. GPAR manages to accuratelly extrapolate the last 100 readings, whereas independent GPs are unable to model them.

observation for output m ([Requeima et al., 2018]). Then, if we assume that all outputs are observed at each input, we find that the posterior GPAR cleanly decomposes into a product of posterior GPs as in Equation 2.7. This means that the posterior of f_m can be computed by conditioning upon the observations for output m at the locations generated by grouping the previous outputs and the input x. The evidence also decomposes in a similar manner, thus allowing us to sequentially optimize the kernel hyperparameters.

$$p(f_{1:M})|(y_{1:M}^{(n)}, x^{(n)})_{n=1}^{N}) = \prod_{m=1}^{M} p(f_m|(y_m^{(n)})_{n=1}^{N}, (y_{1:m-1}^{(n)}, x^{(n)})_{n=1}^{N}) \tag{2.7}$$

2.2.2 Example

An example of how GPAR performs when compared to independent GPs can be seen in Figure 2.4. The EEG dataset[1] consists of 256 measurements from 7 electrodes placed

[1]The EEG data set can be downloaded at https:// archive.ics.uci.edu/ml/datasets/eeg+database.

on the patient's scalp. Our task is to predicts the last 100 measurements for three of these electrodes given the first 156 samples and full samples for the reading of the other electrodes. We can see that whereas GPs fail to capture any meaningful information, GPAR can predict with low variance the voltage for the unseen electrodes.

2.2.3 Limitations

A limitation arising from the GPAR model is that one needs to decide upon an ordering of the outputs. Fortunately, most datasets have a natural conditional ordering of the outputs [Requeima et al., 2018]. Alternatively, if no natural ordering can be found, one can greedily optimize the ordering by looking at the evidence. Therefore, we can greedily choose y_1, then y_2, and so on. Taking this approach leads to analysing $M(M+1)/2$ possible configurations instead of the brute force approach of $M!$ configurations. We won't delve deeper into order choice for the purposes of this thesis.

Another limitation arising from the GPAR model is that noise gets propagated through the outputs; from the earlier dimensions to the later dimensions. This is a limitation when the noise is simply aleatoric uncertainty. However, noise propagation can also be a benefit in the case where the outputs have correlated noise, and thus noise might be an important signal. In any case, noise propagation can be mitigated by using a denoising input transformation for the kernels as in [Requeima et al., 2018]. Denoising the inputs involves using the posterior predictive mean for intermediate outputs before they are fed as inputs into the next covariance function.

2.3 Deterministic training conditional

As previously mentioned, Gaussian processes don't scale well to large datasets due to the cubic complexity. During this section, we'll look into how GPs can be scaled by using pseudo-point approximation techniques, namely the Deterministic training conditional (DTC) [Quiñonero-Candela and Rasmussen, 2005].

The main idea behind pseudo-point approximation techniques is to introduce a new set of inducing variables $\mathbf{u}$ which will act as latent function values, thus "accumulating" the training information and pass it on to the training process. If we use M pseudo-points, where $M <<< N$, then DTC reduces the GP complexity from $O(N^3)$ to $O(M^2N + M^3)$. The deterministic training conditional is a poor approximation to Titsias's Variational Free Energy (VFE) method [Titsias, 2009] since it is prone to overfitting and under-estimating posterior uncertainties when provided with too few pseudo-points. However, DTC has been chosen for scaling GPAR because VFE also includes a trace term $trace\left(\mathbf{C}_{ff} - \mathbf{C}_{fu}\mathbf{C}_{uu}^{-1}\mathbf{C}_{uf}\right)$ which is more challenging to scale than DTC, thus leading to quadratic complexity in N.

DTC is used to approximate models of the form as in Equation 2.8. The assumption is that we generate the function from a Gaussian Process with zero-mean, and then we add i.i.d. noise. We use zero mean processes since it slightly simplifies the mathematics; the mean can be added without significant changes.

$$\begin{aligned} f &\sim \mathcal{GP}(0, \mathcal{K}(v, v')) \\ \epsilon &\sim \mathcal{GP}(0, \sigma^2 \delta(v - v')) \\ y(v) &= f(v) + \epsilon(v) \end{aligned} \tag{2.8}$$

2.3.1 Notation

We wish to scale a Gaussian Process f using DTC.

Let $\mathcal{X}$ be the input domain, *e.g.* $\mathbb{R}^D$. From this domain, we draw the locations of the observations $\mathbf{v} \in \mathcal{X}^N$, and their associated values $\mathbf{y} \in \mathbb{R}^N$. We also draw the locations of the pseudo-points $\mathbf{z} \in \mathbb{R}^M$. One could think of these pseudo-points as regularly spaced points in the input domain. Their location can also be trained to improve the performance of the model, but we won't delve into these details for the purposes of this thesis.

Let $\mathbf{u} \sim \mathcal{N}(0, \mathbf{C_{uu}})$ be the multi-variate Gaussian distribution obtained by evaluating the GP f at locations $\mathbf{z}$. These are called the pseudo-points. Similarly, let $\mathbf{f} \sim \mathcal{N}(0, \mathbf{C_{ff}})$ be the multi-variate Gaussian obtained by evaluating the GP at locations $\mathbf{v}$. We call $\mathbf{C_{uu}}$ and $\mathbf{C_{ff}}$ the covariance matrices and they are obtained as in Equation 2.9.

$$\begin{aligned} [\mathbf{C_{ff}}]_{i,j} &= \mathcal{K}(\mathbf{v}_i, \mathbf{v}_j) \\ [\mathbf{C_{uu}}]_{i,j} &= \mathcal{K}(\mathbf{z}_i, \mathbf{z}_j) \\ [\mathbf{C_{fu}}]_{i,j} &= \mathcal{K}(\mathbf{v}_i, \mathbf{z}_j) \\ \mathbf{C_{uf}} &= \mathbf{C_{uf}}^T \end{aligned} \tag{2.9}$$

Finally, let $\Sigma \in \mathbb{R}^{N \times N}$ be the covariance matrix of the noise process. This is typically a scaled identity matrix.

2.3.2 Log Marginal Likelihood approximation

The log marginal likelihood of our dataset is $\log \mathcal{N}(\mathbf{y}|\mathbf{C_{ff}} + \Sigma)$. This has $O(N^3)$ computational cost when computed directly since it requires the inversion of the covariance matrix $\mathbf{C_{ff}}$. By using M pseudo-points, we can drop this complexity to $O(NM^2)$. The new approximated log marginal likelihood is thus $\log \mathcal{N}(\mathbf{y}|0, \mathbf{C_{fu}}\mathbf{C_{uu}}^{-1}\mathbf{C_{uf}} + \Sigma)$. The approximated covariance matrix is similar to the covariance matrix retrieved after applying the Nystrom approximation to Gaussian Processes [Williams and Seeger, 2001].

The new marginal likelihood can be rewritten as in Equation 2.10 using the matrix inversions and the determinant lemmas. If inverting Σ can be done in linear time in the number of observations N (*e.g.* if Σ is diagonal, which it usually is), then the complexity of computing the log marginal likelihood is $O(NM^2)$

$$\begin{aligned}
&\log\mathcal{N}(\mathbf{y}|0, \mathbf{C_{fu}}\mathbf{C_{uu}^{-1}}\mathbf{C_{uf}} + \Sigma) = \\
&\quad = -\frac{1}{2}\left[N\log 2\pi + \log\det(\mathbf{C_{fu}}\mathbf{C_{uu}^{-1}}\mathbf{C_{uf}} + \Sigma) + \mathbf{y}^T(\mathbf{C_{fu}}\mathbf{C_{uu}^{-1}}\mathbf{C_{uf}} + \Sigma)^{-1}\mathbf{y}\right] \\
&\quad = -\frac{1}{2}[N\log 2\pi + \log\det(\mathbf{C_{uf}}\Sigma^{-1}\mathbf{C_{fu}} + \mathbf{C_{uu}}) - \log\det\mathbf{C_{uu}} + \log\det\Sigma + \\
&\qquad + \mathbf{y}^T\Sigma^{-1}\mathbf{y} - \mathbf{y}^T\Sigma^{-1}\mathbf{C_{fu}}\left(\mathbf{C_{uf}}\Sigma^{-1}\mathbf{C_{fu}} + \mathbf{C_{uu}}\right)^{-1}\mathbf{C_{uf}}\Sigma^{-1}]
\end{aligned} \tag{2.10}$$

The log marginal likelihood is used for optimizing the hyperparameters of the GP. Given enough pseudo-points (*e.g.* $\mathbf{z} = \mathbf{v}$), the DTC approximation is almost perfect and it will lead to identical optimised hyperparameters.

2.3.3 Inference

For doing inference we approximate the posterior process with another GP whose mean and kernel are computed using the pseudo-points as in Equation 2.11. Here, $[\mathcal{K}(x, \mathbf{z})]_i = \mathcal{K}(x, \mathbf{z}_i)$ and $[\mathbf{m_u}]_i = mean(\mathbf{z}_i)$. The newly introduced parameters $\hat{\mathbf{m}}_\mathbf{u}$ and $\hat{\mathbf{\Lambda}}_\mathbf{u}$ are the mean and the precision matrix of the approximate posterior process for the pseudo-points GP, *i.e.* for a process $q(\mathbf{u}) \approx p(\mathbf{u}|\mathbf{y})$.

$$\begin{aligned}
\hat{m}(x) &= m(x) + \mathcal{K}(x, \mathbf{z})\mathbf{C_{uu}^{-1}}(\hat{\mathbf{m}}_\mathbf{u} - \mathbf{m_u}) \\
\hat{\mathcal{K}}(x, x') &= \mathcal{K}(x, x') - \mathcal{K}(x, \mathbf{z})\mathbf{C_{uu}^{-1}}\mathcal{K}(\mathbf{z}, x') + \mathcal{K}(x, \mathbf{z})\mathbf{C_{uu}^{-1}}\hat{\mathbf{\Lambda}}_\mathbf{u}^{-1}\mathbf{C_{uu}}\mathcal{K}(\mathbf{z}, x')
\end{aligned} \tag{2.11}$$

By parametrising over $\epsilon = \mathbf{U}^{-T}(\mathbf{u} - \mathbf{m}_u)$ instead of simply $\mathbf{u}$, we can obtain a form for the mean and kernel which involve fewer operations and are thus less computationally demanding. This new diagonalised version is described in Equation 2.12, where $\mathbf{U}$ is the upper-triangular Cholesky factorisation of the matrix $\mathbf{C_{uu}}$.

$$\begin{aligned}
\hat{m}(x) &= m(x) + \mathcal{K}(x, \mathbf{z})\mathbf{U^{-1}}\hat{\mathbf{m}}_\epsilon \\
\hat{\mathcal{K}}(x, x') &= \mathcal{K}(x, x') - \mathcal{K}(x, \mathbf{z})\mathbf{U^{-1}}\mathbf{U^{-T}}\mathcal{K}(\mathbf{z}, x') + \mathcal{K}(x, \mathbf{z})\mathbf{U^{-1}}\hat{\mathbf{\Lambda}}_\epsilon^{-1}\mathbf{U^{-T}}\mathcal{K}(\mathbf{z}, x')
\end{aligned} \tag{2.12}$$

2.3.4 Optimizing the posterior process

When using pseudo-point approximation techniques we have to find good locations for the pseudo-inputs $\mathbf{z}$, and to find the optimal values for $\hat{\mathbf{m}}_\epsilon$ and $\hat{\mathbf{\Lambda}}_\epsilon$ once $\mathbf{z}$ is fixed. Whereas the

former is intractable and typically addressed with iterative algorithms, the latter challenge is tractable as shown in [Matthews et al., 2016].

Let $\mathbf{V}$ be the upper-triangular the upper-triangular Cholesky factorisation of the noise matrix Σ. The matrix $\mathbf{V}$ is diagonal if Σ is diagonal. Furthermore, let $\mathbf{B}_{\epsilon\mathbf{f}} = \mathbf{U}^{-T}\mathbf{C}_{\mathbf{uf}}\mathbf{V}^{-1}$. The approximate posterior process $q(\mathbf{u})$ is defined by the mean and kernel as in Equation 2.13. By plugging in these values into Equation 2.12, we obtain the mean and kernel of the approximate posterior process at points $\mathbf{f}$.

$$\begin{aligned} \hat{\mathbf{\Lambda}}_\epsilon &= \mathbf{B}_{\epsilon\mathbf{f}}\mathbf{B}_{\mathbf{f}\epsilon} + I \\ \hat{\mathbf{m}}_\epsilon &= \hat{\mathbf{\Lambda}}_\epsilon^{-1}\mathbf{B}_{\epsilon\mathbf{f}}\mathbf{V}^{-T}(\mathbf{y} - \mathbf{m}_\mathbf{f}) \end{aligned} \tag{2.13}$$

2.4 Relation between GP and linear SDEs / Linear-Gaussian SSMs

All the previous complexities of DTC were derived for a diagonal noise matrix Σ. However, we might also be interested in applying DTC to a GP which does not have a diagonal noise matrix, thus leading to a cubic complexity again for the naive approach. However, we can accelerate this computation through the use of Linear Gaussian State Space models (LGSSMs). During this section, we'll explore how a Gaussian Process acting on time series (temporal GP) can be mapped to an LGSSM and how this allows us to compute the log marginal likelihood and how to accelerate the bottleneck computations [Eubank and Wang, 2002].

2.4.1 State space models

In general, a state space model is used to describe a system for which we can attribute a latent variable called state x_t to each observation y_t at time t such that the Markov property from Equation 2.14 is respected. The Markov property refers to the model's memorylessness; the previous state x_{t-1} contains all the required information to compute the distribution of x_t, and the current state x_t contains all the required information to compute the observation y_t. The initial state is sampled from the prior $x_0 \sim p(x_0)$.

$$\begin{aligned} p(x_t|[x_i]_{i=0}^{i=t-1}, [y_i]_{i=0}^{i=t-1}) &= p(x_t|x_{t-1}) \\ p(y_t|[x_i]_{i=0}^{i=t}, [y_i]_{i=0}^{i=t-1}) &= p(y_t|x_t) \end{aligned} \tag{2.14}$$

The most common type of state space models is the Hidden Markov Model (HMM) shown in Figure 2.5. Here, the transition and output probabilities are governed by underlying transition and output matrices which are learned from the data.

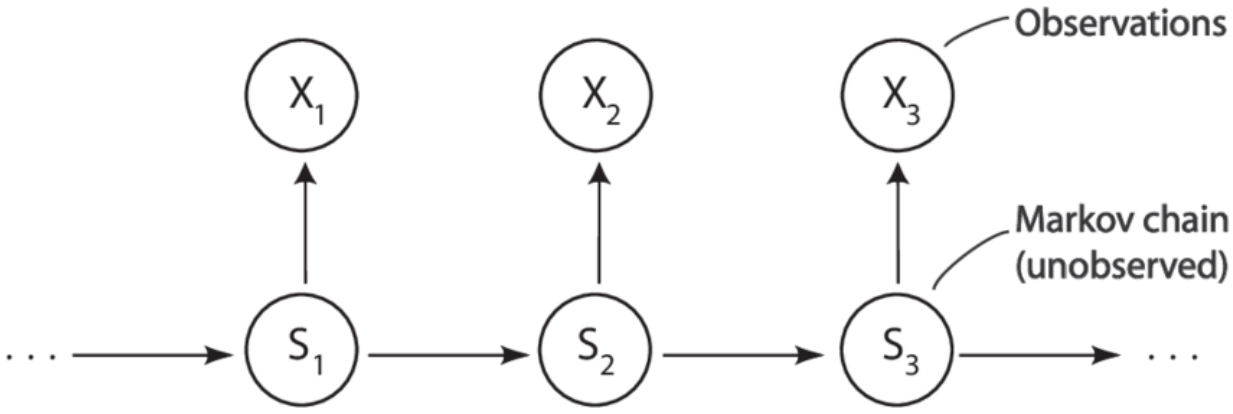

Figure 2.5: Basic structure of a Hidden Markov Model from [Bulla, 2006]

However, state space models are not limited to just the discrete case. We wish to define models which can be used to represent one-dimensional Gaussian Processes. These will be called linear time-invariant stochastic differential equation (LTISDE) models [Solin et al., 2016], and their mapping to a discrete state space model is called a linear gaussian state space model (LGSSM). We are interested in LGSSMs because there is a vast literature around them that defines filtering and smoothing operations; operations which are equivalent to matrix inversion operations we wish to perform in order to scale a Gaussian Process.

2.4.2 Temporal GPs as Stoachastic Differential Equations

Certain temporal Gaussian Process regression problems can be rewritten in terms of a solution to an m'th order stochastic differential equation [Øksendal, 2003] as in Equation 2.15. Here, $f(t)$ is the value of our temporal GP at time t; *i.e.* the state value at time t for the state space model. The white noise process $w(t)$ is also a Gaussian Process, and since Gaussianity is preserved under linear operations, the solution trajectory of $f(t)$ will also be a Gaussian Process.

$$a_0 f(t) + a_1 \frac{df(t)}{dt} + a_2 \frac{d^2 f(t)}{dt^2} + \dots a_m \frac{d^m f(t)}{dt^m} = w(t) \tag{2.15}$$

We can rewrite Equation 2.15 in terms of its *companion form* [Andrews, 2001] by collecting the derivative terms in one vector-valued function as in Equation 2.16. By rearranging

and groping the SDE terms such that we use the newly introduced vector-valued function, we get Equation 2.17 [Solin et al., 2016].

$$\mathbf{f}(t) = (f(t), df(t)/dt, \dots d^{m-1}t/dt^{m-1}) \tag{2.16}$$

$$\frac{d\mathbf{f}(t)}{dt} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \dots & \\ \vdots & \ddots & \ddots & \\ & & 0 & 1 \\ -a_0 & -a_1 & \dots & -a_{m-1} \end{bmatrix} \mathbf{f}(t) + \begin{bmatrix} \mathbf{0} \\ \vdots \\ 0 \\ 1 \end{bmatrix} w(t) \tag{2.17}$$

We are only interested in $f(t)$, i.e. the first component of our state $\mathbf{f}(t)$. Extracting this is a linear operation; we define the extraction operator $\mathbf{H} = (1 \;\; 0 \;\; \dots \;\; 0)$ such that $f(t) = \mathbf{H}\mathbf{f}(t)$.

There are certain classes of covariance functions for our GP of interest such that the GP can be represented in terms of a dynamical model and a measurement model as in Equation 2.18. This equates to rewriting the Gaussian Process as an infinite state space model.

$$\begin{aligned} \frac{d\mathbf{f}(t)}{dt} &= \mathbf{F}\mathbf{f}(t) + \mathbf{L}\mathbf{w}(t) \\ y_k &= \mathbf{H}\mathbf{f}(t_k) + \epsilon_k \end{aligned} \tag{2.18}$$

We have introduced the following terms in Equation 2.18:

- $\mathbf{f}(t) = (f_1(t), f_2(t), \dots, f_m(t))$ is a vector valued function containing the m stochastic processes.
- $\mathbf{w}(t)$ is a multi-dimensional noise process with spectral density matrix $Q \in \mathbb{R}^{s \times s}$. A spectral density matrix represents the covariance of the extra noise injected into the SDE over a unit time increment.
- $\mathbf{F} \in \mathbb{R}^{m \times m}$ is called the feedback matrix.
- $\mathbf{L} \in \mathbb{R}^{m \times s}$ is called the noise effect matrix.
- The measurements are corrupted by i.i.d. noise $\epsilon_k \sim \mathcal{N}(0, \sigma^2_{noise})$.
- The initial state is sampled from a process with covariance matrix $\mathbf{P}_0$.

2.4.3 From GP to LTISDE

We are interested in obtaining the parameters of the LTISDE, namely $\mathbf{F}$, $\mathbf{L}$, $\mathbf{Q}$, $\mathbf{P}_0$, and $\mathbf{H}$. These "model matrices" are defined for individual covariance function classes. In the following subsections, we'll explore how they can be derived for covariance functions in the exponential, squared exponential and Matern classes.

Exponential

The exponential covariance function is defined as in Equation 2.19 [Williams and Rasmussen, 2006]. The model matrices of the SDE representation of covariance functions in this class have analytic solutions; $\mathbf{F} = -1/l$, $\mathbf{L} = 1$, $\mathbf{Q} = 2v^2/l$, and $\mathbf{H} = 1$.

$$\mathcal{K}(t,t') = \sigma^2 exp\left(-\frac{|t-t'|}{l}\right) \tag{2.19}$$

Matern

The general form of the covariance functions in the Matern class [Matérn, 1960] is given in Equation 2.20, where σ^2 and l are the magnitude and the scale parameters, and the newly introduced v is the smoothness parameter. $B_v(.)$ is the modified Bessel function of the second kind. An advantage of the Matern class is that we have control over how many times a function is differentiable through the value of v. If $v > k$ then the function is k times differentiable.

$$\mathcal{K}(t,t') = \sigma^2 \frac{2^{1-v}}{\Gamma(v)} \left(\frac{\sqrt{2v}|t-t'|}{l}\right)^v B_v \left(\frac{\sqrt{2v}|t-t'|}{l}\right) \tag{2.20}$$

By setting different values for v we generate other classes of covariance functions. Usually, we work with v taking half-integer values; *e.g.* setting $v = 1/2$ recovers the exponential class of functions. Let us focus on $v = 3/2$ which gives us the Matern-3/2 covariance function class defined in Equation 2.5. This class has the SDE representation as in Equation 2.21.

$$\mathbf{F} = \begin{bmatrix} 0 & 1 \\ -\frac{3}{l^2} & -\frac{2\sqrt{3}}{l} \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad Q = 4\sigma^2 \frac{3\sqrt{3}}{l^3} \qquad \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.21}$$

Squared exponential

The squared exponential function is given as in Equation 2.4. The squared exponential function can be recovered from the Matern class by taking the limit $v \to \infty$. This implies

that the model is infinitely differentiable, thus requiring an infinitely long function-valued vector $\mathbf{f}$. Therefore, the only way we can recover the model matrices for its LTISDE representation is through approximation methods such as approximated Taylor expansion [Hartikainen and Särkkä, 2010].

2.4.4 Converting LTISDE into LGSSM

In order to use state space methods for acceleration [Sarkka et al., 2013] we need to convert our linear time-invariant stochastic differential equation model (LTISDE) from Equation 2.18 into a Linear Gaussian State Space model (LGSSM) by solving for discrete time instances corresponding to the input data and prediction points [Solin et al., 2016].

By discretising $\mathbf{f}$ such that $\mathbf{f}_k = \mathbf{f}(t_k)$ for each input location t_k, we can rewrite the LTISDE as in Equation 2.22, where $\mathbf{A}_k$ is the transition matrix between states $\mathbf{f}_{k-1}$ and $\mathbf{f}_k$, and $\mathbf{Q}_k$ is the covariance matrix of the process generating the noise.

$$\begin{aligned} \mathbf{f}_k &= \mathbf{A}_{k-1}\mathbf{f}_{k-1} + \mathbf{q}_{k-1} \quad \text{where } q_{k-1} \sim \mathcal{N}(0, \mathbf{Q}_{k-1}) \\ y_k &= \mathbf{H}\mathbf{f}_k + \epsilon_k \qquad\qquad\quad \text{where } \epsilon_k \sim \mathcal{N}(0, \sigma^2_{noise}) \end{aligned} \tag{2.22}$$

The transition and noise matrices are computed as in Equation 2.23. We define $\Delta t_k = t_{k+1} - t_k$ and $\Phi(\tau) = exp(\mathbf{F}\tau)$. The integral can be resolved using matrix fraction decomposition [Särkkä et al., 2006].

$$\begin{aligned} \mathbf{A}_k &= \Phi(\Delta t_k) \\ \mathbf{Q}_k &= \int_0^{\Delta t_k} \Phi(\Delta t_k - \tau)\mathbf{L}\mathbf{Q}\mathbf{L}^T\Phi(\Delta t_k - \tau)^T d\tau \end{aligned} \tag{2.23}$$

2.4.5 Relationship between Kalman filtering and Cholesky factorisations

This whole section discussed the conversion between Temporal GPs and linear Gaussian state space models (LGSSMs) so that we can perform avoid performing expensive Cholesky factorisations on the GPs covariance matrix, and instead perform efficient filtering and smoothing on the LGSSM. This subsection discusses the equivalence between these two operations.

Let there be a Temporal GP with timeseries input locations $[t_k]_{k=1}^N$, constant mean function $\mathbf{0}$, and covariance matrix Σ. For the purposes of this dissertation, we are interested in accelerating two types of computation:

- Retrieve the marginal distributions of the outputs at each input location given a complete measurement $\mathbf{y}$. This is accelerated by performing smoothing on the LGSSM.

- Let $\mathbf{L} = chol(\Sigma)$ be the lower-triangular Cholesky factorisation of the covariance matrix, and let $\mathbf{a}$ be an arbitrary vector. We are interested in accelerating the computation of $\mathbf{L}^{-1}\mathbf{a}$. This is accelerated through applying decorrelation on the LGSSM.

Smooth

Below we will go through the necessary equations to perform smoothing on the LGSSM equivalent of a Temporal GP as in [Särkkä, 2013]. The method we apply is Kalman filtering [Kalman, 1960] followed by the Rauch-Tung-Striebel smoother, fully derived in [Rauch et al., 1965].

Let the LGSSM be as previously defined in Equation 2.22, $\mathbf{f}_k$ be the state at time t_k, and $\mathcal{D}_k = [(t_i, y_i)]_{i=1}^k$ be the data up to time instance t_k. We are interested in computing the posterior marginals of the states given the whole dataset $\mathbf{f_k}|\mathcal{D}_N \sim \mathcal{N}(\mathbf{m}_{k|n}, \mathbf{P}_{k|N})$.

We start the process from the first state $\mathbf{f}_0 \sim \mathcal{N}(\mathbf{m}_{0|0}, \mathbf{P}_{0|0})$ where $\mathbf{m}_{0|0} = \mathbf{0}$ and $\mathbf{P}_{0|0} = \mathbf{P}_0$ from the LGSSM definition. The Kalman prediction step from Equation 2.24 is iteratively applied to obtain the marginal for state $\mathbf{f}_k$ given the outcome of the previous step. We remind the reader that $\mathbf{Q}_k$ is the covariance matrix of the noise generating process.

$$\begin{aligned} \mathbf{m}_{k|k-1} &= \mathbf{A}_{k|k-1}\mathbf{m}_{k-1|k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{A}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1} \end{aligned} \tag{2.24}$$

We further apply the Kalman update step in order to retrieve the state posterior given the data up to that point. The equations governing this update step are described in Equation 2.25. We refer to the terms $\mathbf{v}_k$ and $\mathbf{S}_k$ as the innovation mean and covariance and to $\mathbf{K}_k$ as the Kalman gain. σ^2_{noise} is the measurement noise variance and we assume broadcasting of terms.

$$\begin{aligned} \mathbf{v}_k &= \mathbf{y}_k - \mathbf{H}\mathbf{m}_{k|k-1} \\ \mathbf{S}_k &= \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \sigma^2_{noise} \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}\mathbf{S}_k^{-1} \\ \mathbf{m}_{k|k} &= \mathbf{m}_{k|k-1} + \mathbf{K}_k\mathbf{v}_k \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T \end{aligned} \tag{2.25}$$

Finally, we obtain the smoothing solution by applying a backwards recursion step starting from the last state at time t_n. The equations governing the backwards step are described

in Equation 2.26. $\mathbf{G}_k$ is called the smoother gain.

$$\begin{aligned}
\mathbf{m}_{k+1|k} &= \mathbf{A}_k \mathbf{m}_{k|k} \\
\mathbf{P}_{k+1|k} &= \mathbf{A}_k \mathbf{P}_{k|k} \mathbf{A}_k^T + \mathbf{Q}_k \\
\mathbf{G}_k &= \mathbf{P}_{k|k} \mathbf{A}_k^T \mathbf{P}_{k+1|k}^{-1} \\
\mathbf{m}_{k|N} &= \mathbf{m}_{k|k} + \mathbf{G}_k (\mathbf{m}_{k+1|N} - \mathbf{m}_{k+1|k}) \\
\mathbf{P}_{k|N} &= \mathbf{P}_{k|k} + \mathbf{G}_k (\mathbf{P}_{k+1|N} - \mathbf{P}_{k+1|k}) \mathbf{G}_k^T
\end{aligned} \tag{2.26}$$

The smoothing operation is also applied at the test locations $f(t*)$. At these locations, we assume arbitrary observations and infinite measurement noise, therefore not allowing our arbitrary observation to impact the outcome. The marginal distribution at the inference locations is retrieved by applying the measurement matrix $\mathbf{H}$ to the smoothed mean and variance.

Decorrelate

Below we'll rewrite a part of [Eubank and Wang, 2002] in order to fit our use case and assumptions. We will skip the derivations since they warrant a longer talk; the reader is invited to read the paper. We're interested in computing $\epsilon = \mathbf{L}^{-1}\mathbf{y}$. Again, our LGSSM is governed by the equations in 2.22.

The vector of interest ϵ can be computed component by component as in Equation 2.27, where $\mathbf{v}_k$ is the k'th element of vector $\mathbf{v}$. The vector $\mathbf{m}_{k|k-1}$ can be computed as in Equation 2.24.

$$\epsilon_k = \mathbf{y}_k - \mathbf{H}\mathbf{m}_{k|k-1} \tag{2.27}$$

We call this method 'decorrelate' since multiplying a sample from a multivariate Gaussian distribution by the inverse cholesky decomposition of the covariance matrix retrieves a vector of i.i.d. samples from a Normal distribution $\mathcal{N}(0, 1)$. This is the generalisation of data standardisation.

2.5 Summary

In summary, we went through a brief overview of Gaussian Process models and their limitations when applying to multi-output supervised learning problems. In order to address this issue, we introduced the Gaussian Process Autoregressive Regression model [Requeima et al., 2018] that uses chained standard GP models for multi-output problems. We went through the time complexity of computing a GP posterior and saw that it is cubic in the number of observations, an impractical complexity if we'd like to work with datasets

containing tens of thousands of observations (which is a medium-sized dataset). In order to reduce this time complexity, we introduced pseudo-point approximation techniques in terms of the Deterministic Training Conditional. This approximation technique allows us to compute an approximate GP posterior in linear time in the number of observations. Finally, we introduced state space models and how they can be used to reduce the complexity of inverting a covariance matrix from cubic to linear. We explored ways to convert from temporal GPs to time-invariant stochastic differential equations and then further to linear Gaussian state space models. Given a state space model, we explored how we can efficiently perform filtering and smoothing.

Chapter 3

Methodology

This chapter contains a review of the methods used to implement the algorithm used to scale GPAR. We will combine the Deterministic Training Conditional [Quiñonero-Candela and Rasmussen, 2005] with the state space acceleration methods [Solin et al., 2016] in order to scale the Gaussian Process Autoregressive Regression model [Requeima et al., 2018].

3.1 Starting Point

We use Julia as the implementation language. It was chosen because it is a high-level, dynamic programming language similar to Python, but it was preferred over Python for its faster execution speed.

The choice of Julia was also motivated by the existing libraries, namely Stheno [1], dealing with GP creation and inference, and TemporalGPs[2], which deals with mapping Temporal GPs to LTISDE and LGSSM models and performing filtering and smoothing on them.

3.2 GPAR in terms of DTC

Let us consider the n'th output of a GPAR model from Equation 3.1.

$$y_n(x,t) \sim \mathcal{GP}(0, \mathcal{K}_n((x,t),(x',t'))) \tag{3.1}$$

Here, $t \in \mathbb{R}$ is the time, $x(t) \in \mathbb{R}^{n-1}$ are the previous outputs at timesteps t, and the kernel $\mathcal{K}_n$ is defined as in Equation 3.2. The kernel we use is composite; it is the sum of two distinct kernels $\mathcal{K}_n^t$ and $\mathcal{K}_n^x$ that act on the time and the previous outputs

[1] https://github.com/willtebbutt/Stheno.jl/

[2] https://github.com/willtebbutt/TemporalGPs.jl

respectively. Finally, $\sigma^2\delta((x-x')(t-t'))$ is the measurement noise term. We split the temporal kernel from the kernel acting on previous outputs since we would like to avoid using DTC to approximate the temporal part of the GP. This is because the use of pseudo-point approximations on the temporal channel would involve scaling the number of pseudo-points with the number of units of time. We'd like to avoid this since it tangles the number of observation with the number of pseudo-points required for a good approximation, thus leading again to cubic complexity.

$$\mathcal{K}_n((x,t),(x',t')) = \mathcal{K}_n^x(x,x') + \mathcal{K}_n^t(t,t') + \sigma^2\delta((x-x')(t-t')) \tag{3.2}$$

By combining Equations 3.1 and 3.2, we can rewrite the GPAR output in terms of a summation over GPs as in Equation 3.3.

$$\begin{aligned} f_n^x &\sim \mathcal{GP}(0, \mathcal{K}_n^x) \\ f_n^t &\sim \mathcal{GP}(0, \mathcal{K}_n^t) \\ \epsilon_n &\sim \mathcal{GP}(0, \sigma^2\delta((x-x')(t-t')) \\ y_n(x,t) &= f_n^x(x) + f_n^t(t) + \epsilon_n(x,t) \end{aligned} \tag{3.3}$$

As previously mentioned, the Deterministic Training Conditional (DTC) [Quiñonero-Candela and Rasmussen, 2005] is used to scale GP models of the form described in Equation 2.8. The GPAR model does not conform to this as it contains the sum of two GPs. In order to use DTC, we propose rewriting GPAR in terms of DTC as in Equation 3.4. This means that we treat the temporal part of GPAR as part of the noise process.

$$\begin{aligned} v &= (x,t) \\ f(v) &= f_n^x(x) \\ \epsilon(v) &= f_n^t(t) + \epsilon_n(x,t) \end{aligned} \tag{3.4}$$

Treating the temporal GP as part of the noise process means that the noise process is now correlated, its covariance matrix no longer being a scaled identity matrix. Therefore, all the complexities we discussed in Section 2.3 become $O(N^3)$ since there is no fast way to invert Σ. In order to accelerate this computation, we use the state space acceleration discussed in Section 2.4.

3.3 Approximating the marginal likelihood using DTC

We are interested in optimizing our process's hyperparameters through Maximum Marginal Likelihood Estimation. Simply plugging our terms in Equation 2.10 leads to cubic complexity, which is intractable. However, we can use state space approximations in order to avoid performing costly Cholesky decompositions. The first step towards this is rewriting the marginal likelihood equation such that we can take advantage of the 'decorrelate' method. The new form is given in Equation 3.6, where the newly included terms are given in Equation 3.5. The goal of grouping the terms is to show that we can avoid having to perform the Cholesky decomposition if we can quickly compute α and $\mathbf{B}$.

$$\begin{aligned} \mathbf{L} &= chol(\Sigma) \\ \alpha &= \mathbf{L}^{-1}\mathbf{y} \\ \mathbf{B} &= \mathbf{L}^{-1}\mathbf{C}_{fu} \end{aligned} \tag{3.5}$$

$$\begin{aligned} &\log \mathcal{N}(\mathbf{y}|0, \mathbf{C}_{fu}\mathbf{C}_{uu}^{-1}\mathbf{C}_{uf} + \Sigma \\ &\quad = \log \mathcal{N}(\mathbf{y}|0, \Sigma) - \frac{1}{2}\left[\log\det\left(\mathbf{B}^T\mathbf{B} + \mathbf{C}_{uu}\right) - \log\det\mathbf{C}_{uu} - \alpha^T\mathbf{B}\left(\mathbf{B}^T\mathbf{B} + \mathbf{C}_{uu}\right)^{-1}\mathbf{B}^T\alpha\right] \end{aligned} \tag{3.6}$$

As shown in [Eubank and Wang, 2002] and in Section 2.4.5, we can compute $\mathbf{L}^{-1}\alpha$ in linear time given that Σ is the covariance matrix of a temporal GP. This is the case for us. Therefore, we can quickly evaluate Equation 3.6 in the following way:

- Transform the temporal GP f_n^t into its equivalent LTISDE.
- Transform the LTISDE into an LGSSM by indexing at the training points.
- Compute α and $\mathbf{B}$ using accelerated state space techniques.
- Plug α and $\mathbf{B}$ into Equation 3.6. All the other computations are at most $O(M^2N)$.

Now that we have a way to quickly compute an approximation to the marginal log-likelihood, we can optimize the kernel hyperparameters by calling the function that computes this approximation in our optimization loop; similar to how we'd call the function computing the marginal likelihood. Tests have confirmed that in the degenerate case where the pseudo-points are the same as the inputs, the DTC approximation is perfect and we retrieve the same optimum hyperparameters.

3.4 Making predictions using scaled GPAR

Up until now, we learned how to optimize the hyperparameters of GPAR. We will now explore how to make predictions for new inference points. More formally, let $\mathbf{t}_*$ be the timesteps at which we wish to infer the values $\mathbf{f}_*$ for the n'th output of the GPAR. We assume that the previous $n-1$ outputs are observed at timesteps $\mathbf{t}_*$. This assumption does not violate generality thanks to the sequential nature of GPAR; even if the other outputs are not part of the training data, we can infer them during the previous steps.

We are interested in retrieving the posterior distributions for the test data-points $\mathbf{f}_*$ given the observed data $\mathbf{y}$. Through the use of the sum and product rules, we rewrite this quantity as in Equation 3.7. This reformulation is useful because, together with the equality from Equation 3.3 means that we can take a divide and conquer approach to estimating the posterior distribution; we address the temporal part when computing $p(\mathbf{f}_*|\mathbf{f}_*^x, \mathbf{y})$ and the previous outputs part when computing $p(\mathbf{f}_*^x|\mathbf{y})$.

$$p(\mathbf{f}_*|\mathbf{y}) = \int p(\mathbf{f}_*|\mathbf{f}_*^x, \mathbf{y}) p(\mathbf{f}_*^x|\mathbf{y}) d\mathbf{f}_*^x \tag{3.7}$$

The first challenge we tackle is the computation of $p(\mathbf{f}_*^x|\mathbf{y})$. As one might expect, this is $O(N^3)$ in the naive case. However, we can use pseudo-point approximations in order to generate samples from this distribution. Let $p(\mathbf{f}_*^x|\mathbf{y}) \approx q(\mathbf{f}_*^x)$, where our approximation is given as in Equation 3.8.

$$q(\mathbf{f}_*^x) = \mathcal{N}\left(\mathbf{f}|\mathbf{m}_\mathbf{f} + \mathbf{C}_{fu}\mathbf{C}_{uu}^{-1}(\hat{\mathbf{m}}_u - \mathbf{m}_u), \hat{\mathbf{C}}_{ff}\right) \tag{3.8}$$

Similarly to Section 2.3, $\mathbf{m}_u$ is the mean vector of the approximate pseudo-point posterior distribution $q(\mathbf{u}) \sim \mathcal{N}(\hat{\mathbf{m}}_\mathbf{u}, \hat{\mathbf{\Lambda}}_\mathbf{u}^{-1})$. The distribution parameters have an analytic solution, same as in Equation 2.13. Unfortunately, naively going through the computations from Equation 2.13 leads to cubic complexity, but this can be accelerated in the same manner as discussed in the previous section; we use LGSSMs to perform the operations of the form $\mathbf{L}^{-1}\alpha$. Afterwards, we can plug in these parameters and obtain a solution for our approximate posterior distribution $q(\mathbf{f}_*^x)$, from which we can sample.

We now have a method of generating samples from $q(\mathbf{f}_*^x)$, and we are interested in computed the rewritten integral from Equation 3.9.

$$p(\mathbf{f}_*|\mathbf{y}) = \int p(\mathbf{f}_*|\mathbf{f}_*^x, \mathbf{y}) q(\mathbf{f}_*^x) d\mathbf{f}_*^x \tag{3.9}$$

The integral is intractable. However, we can sample from $p(\mathbf{f}_*|\mathbf{f}_*^x, \mathbf{y})$ by using ancestral sampling as follows:

1. Generate a sample $\mathbf{f}_*^x \sim q(\mathbf{f}_*^x)$.

2. Let $\mathbf{y}' = \mathbf{y} - \mathbf{f}_*^x$.

3. Generate samples from the LGSSM representation of the Temporal GP $\mathbf{f}_*^t$ with outputs $\mathbf{y}'$. The process of generating samples is similar to smoothing algorithm; more details can be found in [Doucet, 2010]. This will retrieve the temporal prediction $\mathbf{f}_*^t$.

4. Now compute the value $\mathbf{f}_* = \mathbf{f}_*^x + \mathbf{f}_*^t$. This is akin to having sampled from $p(\mathbf{f}_*|\mathbf{f}_*^x, \mathbf{y})$.

Finally, we use Monte Carlo approximations to compute the mean and variance of the distribution as in Equation 3.10.

$$\mathbb{E}[p(\mathbf{f}_*|\mathbf{y})] \approx \frac{1}{Z}\sum_{i=1}^{Z} \mathbf{f}_* \quad \text{where } \mathbf{f}_* \sim p(\mathbf{f}_*|\mathbf{f}_*^x, \mathbf{y}) \quad \text{and } \mathbf{f}_*^x \sim q(\mathbf{f}_*^x) \tag{3.10}$$

After the Monte Carlo process is finished we get an accurate approximation of the mean and variance of $p(\mathbf{f}_*|\mathbf{y})$, which is what we were interested in.

3.5 Implementation details

There are implementation details which we glossed over in order to avoid encumbering the previous sections. They are minor changes meant to either speed up the code or improve the accuracy. We will enumerate them as follows:

- We did not fully write out the derivation of $\hat{\mathbf{C}}_{ff}$ in Equation 3.8 because we do not need to compute it in order to generate samples from $q(\mathbf{f}_*^x)$. Instead, we simply generate a sample $\mathbf{u}_s \sim q(\mathbf{u})$ and set $\mathbf{m}_u = \mathbf{u}_s$ when computing the mean of $q(\mathbf{f}_*^x)$. This achieves the desired effect since $\hat{\mathbf{C}}_{ff}$ is a low-rank matrix, having all eigenvalues bar one equal to zero.

  $\hat{\mathbf{C}}_{ff}$ is a low-rank matrix because the second term in Equation 2.12 $\mathcal{K}(x, \mathbf{z})\mathbf{U}^{-\mathbf{1}}\mathbf{U}^{-\mathbf{T}}\mathcal{K}(\mathbf{z}, x')$ is our DTC approximation to $\mathbf{C}_{ff}$. If our approximation is good, then these two terms cancel out and the covariance matrix is given by Equation 3.11.

$$\hat{\mathcal{K}}(x, x') = \mathcal{K}(x, \mathbf{z})\mathbf{U}^{-\mathbf{1}}\hat{\mathbf{\Lambda}}_{\epsilon}^{-\mathbf{1}}\mathbf{U}^{-\mathbf{T}}\mathcal{K}(\mathbf{z}, x') \tag{3.11}$$

  In order to see why the form of the matrix allows us to skip the computation of $\hat{\mathbf{C}}_{ff}$, we have to look at how one can generate samples from a high-dimensional Gaussian distribution with covariance matrix $\mathbf{C}_{big}$ using a low-dimensional Gaussian distribution with covariance matrix $\mathbf{C}_{small}$. Assume that there exists a matrix $\mathbf{A}$ such that $\mathbf{A}\mathbf{C}_{small}\mathbf{A}^T = \mathbf{C}_{big}$, and let $\mathbf{s}_{small}$ be a sample from our low-dimensional Gaussian

distribution. We can generate samples from our high-dimensional distribution by applying A to our low-dimensional sample as in Equation 3.12.

$$\mathbf{s}_{big} = \mathbf{A}\mathbf{s}_{small} \tag{3.12}$$

The aforementioned points imply that we can generate samples from $q(\mathbf{f}_*^x)$ by generating a sample from $q(\mathbf{u})$ and multiplying it by $\mathcal{K}(x, \mathbf{z})\mathbf{U}^{-\mathbf{1}}$. This avoids the need to actually compute the covariance matrix of the distribution $q(\mathbf{f}_*^x)$.

- When doing inference for timesteps $\mathbf{t}_*$ we also include the observation data at locations $\mathbf{t}$, therefore doing inference at timesteps $(\mathbf{t}_*, \mathbf{t})$. This change includes the observed data in our LGSSM in order to allow it to make more accurate predictions and lower the sample variance.

3.6 Summary

To summarise, we introduced a way of rewriting GPAR so that we can take advantage of pseudo-point approximations whilst avoiding the need to use pseudo-points for approximating the temporal channel. This lead to the GPAR models no longer taking the required form for scaling using DTC. This was addressed by including the temporal process as part of the noise, thus leading to an uncorrelated noise matrix. All the speed-up DTC approximations provide are based on the noise matrix being easily invertible; for example if it is a diagonal matrix. Rewriting GPAR means that the noise matrix is no longer a scaled identity, thus leading to cubic complexity. State space approximations were used in order to accelerate/avoid these costly computations by mapping the temporal GP to an LGSSM model and applying filtering and smoothing to the LGSSM model.

Chapter 4

Experimental evaluation

We are interested in evaluating two metrics, the accuracy of the approximations introduced to scale GPAR and the speed-accuracy tradeoff when using scaling. To that end, we will perform three types of evaluations:

1. Sanity check evaluation; we compare standard GPAR against the scaled GPAR version on two small datasets: a synthetic dataset and the EEG dataset. We set the pseudo-points to be the same as the inputs so that we expect the same results for both models. The purpose of this evaluation is to sanity check that our implementation behaves exactly as standard GPAR when fed enough information.

2. Speed-accuracy tradeoff; we will compare multiple GPAR models using variable numbers of pseudo-points against each other and the standard GPAR model. The comparison will be performed on small datasets such that the standard model runs in a reasonable time. All the models will be timed, and we plot the mean squared error versus the computation time.

3. Extension to large-scale datasets; we evaluate the scaled GPAR version on two large datasets: an exchange dataset and a temperature dataset. We compare the results against independent GP outputs through the use of log marginal likelihood and visual comparisons.

4.1 Sanity check evaluation

During this section, we perform a sanity check on scaled GPAR to ensure that it behaves the same as standard GPAR in the degenerate case where we use exact pseudo-points; *i.e.* $\mathbf{v} = \mathbf{z}$. To this extend, we compare the models on two small datasets; a synthetic dataset and an EEG dataset.

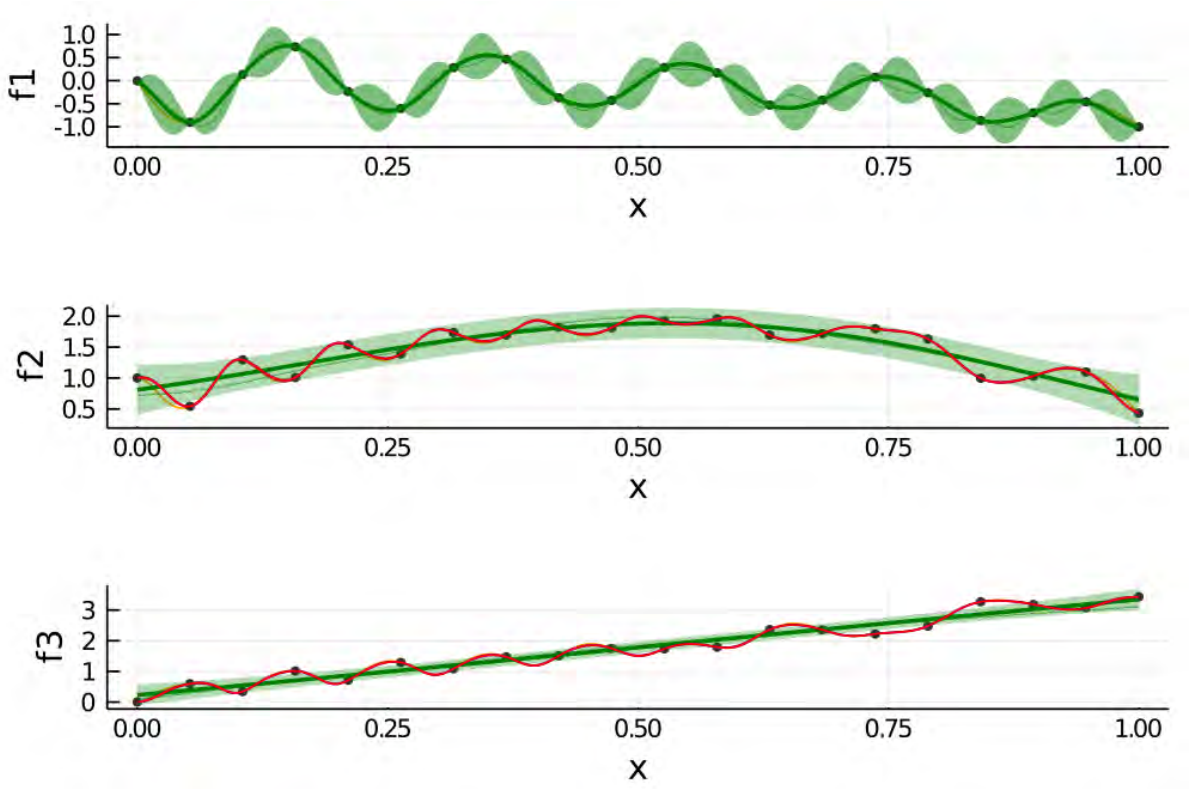

Figure 4.1: Predictions of three models on a small synthetic dataset. The predictions of the independent GP model is shown in green, of the standard GPAR model shown in blue, and of the scaled GPAR model shown in red. The scaled GPAR model's outputs are identical to the ones from the unscaled versions, hence the overlap.

4.1.1 Small synthetic dataset

We use the same synthetic dataset as in [Requeima et al., 2018]. The dataset consists of 20 noisy observations taken at regular intervals from the three functions described in Equation 4.1. In order to simulate output inter-dependencies, we set $y_1 = f_1(x)$ and $y_2 = f_2(x, y_1)$.

$$\begin{aligned} f_1(x) &= -\frac{\sin(10\pi(x+1))}{2x+1} - x^4 \\ f_2(x, y_1) &= \cos(y_1)^2 + \sin(3x) \\ f_3(x, y_1, y_2) &= y_2 y_1^2 + 3x \end{aligned} \tag{4.1}$$

The goal of our model is to interpolate between these observations. The results can be seen in Figure 4.1. The predictions of the independent GP model is shown in green, of the standard GPAR model shown in blue, and of the scaled GPAR model shown in red. We

plot the predictions from the GPAR models only for the latter two functions since the first function represents a single-output problem for which the GPAR structure is identical to one of an independent GP.

We notice that the independent GPs are not expressive enough to model functions f_2 and f_3, but GPAR can perfectly recreate them. Furthermore, the scaled GPAR predictions perfectly overlap the ones from its unscaled counterpart, meaning that it performs as expected in this case.

4.1.2 EEG dataset

The EEG dataset consists of 256 readings for each of the seven electrodes placed on a volunteer's scalp; it is the same experiment as in Figure 2.4. The results from four runs comparing the scaled version of GPAR with the unscaled version can be seen in Figure 4.2. We show the standard GPAR predictions in blue, and the scaled GPAR predictions in red. Because of the stochastic nature of GP hyperparameter optimization, we would expect minor differences. This is why we chose to show multiple plots.

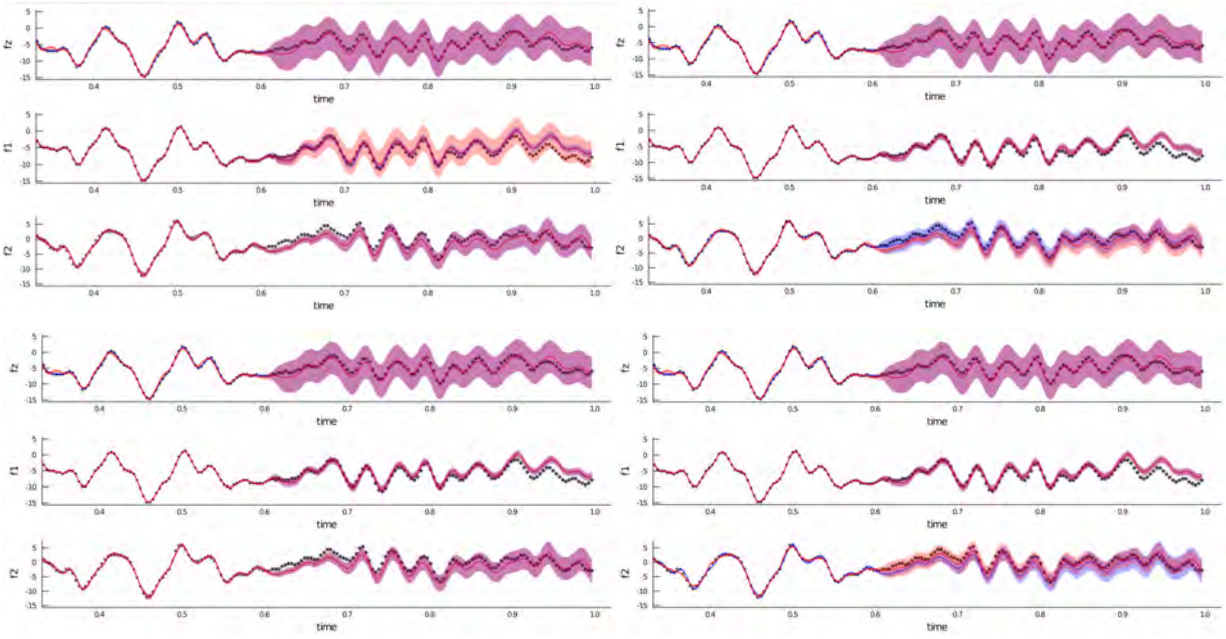

Figure 4.2: Four figures comparing standard GPAR (blue) against scaled GPAR (red) on the EEG dataset. The ground truth is shown in black. The task is the extrapolation of three signals for the last 100 points given the values of the first 156 points and the values of the data from the other sensors. We plot four figures since the results have a high variance because of the underlying stochastic nature of Gaussian Processes. We see that both models perform similarly on this task, even outputting the same distributions in the lower left figure.

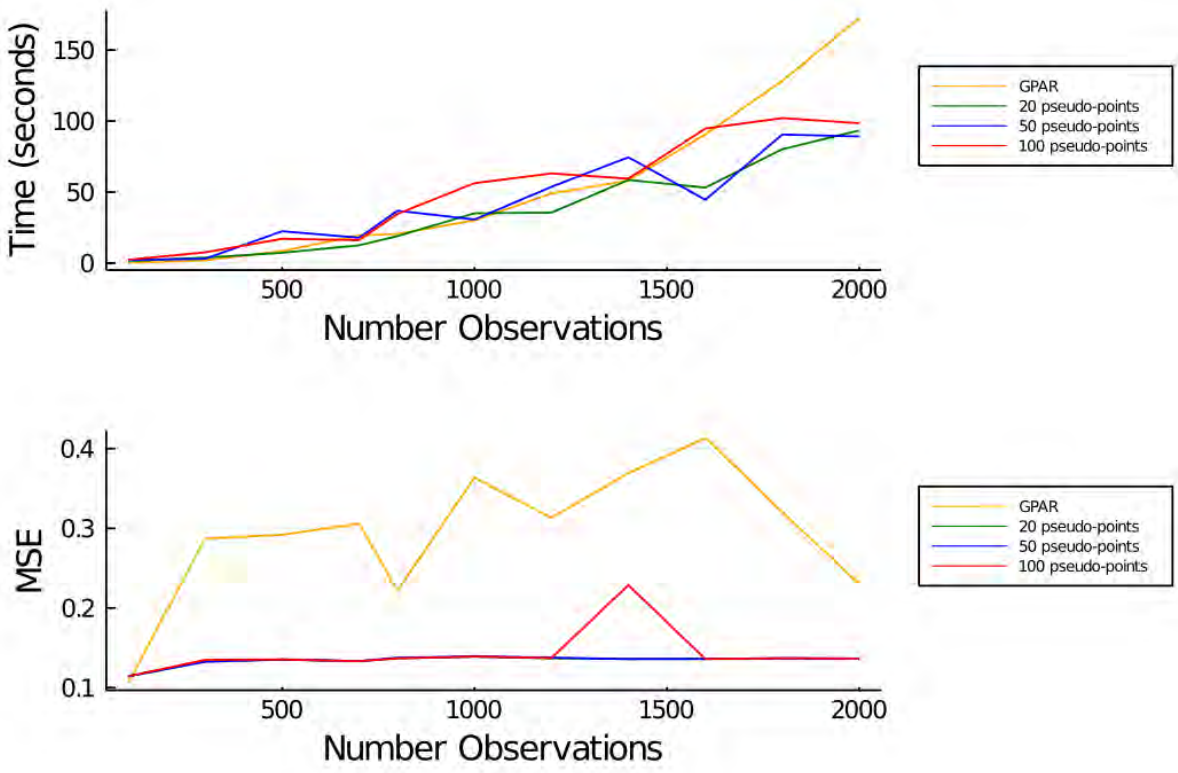

Figure 4.3: Timings and MSE from four models on synthetic dataset. The x axis represents the number of observed variables for which the algorithm is run. We check the algorithm's ability to interpolate, and we also time the process in order to compare the runtimes.

The Euclidian distance between the mean vectors $||\mathbf{f}_{scaled} - \mathbf{f}_{standard}||^2$ is subunitary in almost all cases. We see that the scaled GPAR predictions and error bars are almost identical to the ones from the unscaled version, even being identical in the bottom left figure. Therefore, we can conclude that the scaled GPAR version performs the same as the unscaled version in the case where we use exact pseudo-points.

4.2 Speed-accuracy tradeoff

During this section, we want to check whether the scaled version of GPAR is faster than the standard versions as we add more and more observation data. To check this, we use the small synthetic dataset from Figure 4.1 with two changes:

- We only predict the second function f_2 given the fully observed values of x and $f_1(x)$. We compare the results from the GPAR models against the ground truth by using

the Mean Squared Error.

- In order to achieve coverage over a wider set of possible practical challenges, we change the number of observed points. We do this by extending the observed range whilst keeping the distance between observed variables constant.

The results can be seen in Figure 4.3. Each datapoint has been collected by running the algorithm 10 times and taking the mean. We compare the results from a standard GPAR model against results from three scaled models that use 20, 50, and respectively 100 pseudo-points. The results show that whereas standard GPAR is faster when dealing with less than 500 observations, its runtime increases and it quickly overtakes our models. This is the expected behaviour. Also, GPAR seems to perform worse as we increase the number of observations.

4.3 Large-scale datasets

During this section, we will evaluate the scaled GPAR version on three large datasets (around $10,000$ observations) and compare the results against the predictions from independent GPs.

Temperature dataset[1]

This dataset consists of tidal height, wind speed, and air temperature readings taken during the month of July 2013 by four weather stations in Bramblemet, Cambermet, Chimet, and Sotonmet, all locations from Southhampton, UK. Our goal is to see whether the scaled GPAR model can retrieve the missing air temperature data between days 12-16 for the Cambermet weather station, using all the other features (including features from the other stations) as input.

The results comparing the predictions from the independent GP model versus the ones from the scaled GPAR model can be seen in Figure 4.4. As we can see, GPAR manages to capture the inter-dependencies of the data, whereas the independent GP model fails.

Exchange rate dataset[2]

The exchange rate dataset consists of the daily exchange rate with respect to the US dollar for the top ten international currencies and three previous metals in the years 1990 - 2015.

[1]The data can be downloaded at http://www. bramblemet.co.uk, http://cambermet.co.uk, http://www.chimet.co.uk, and http://sotonmet.co.uk.

[2]The exchange rates data set can be downloaded at http://fx.sauder.ubc.ca.

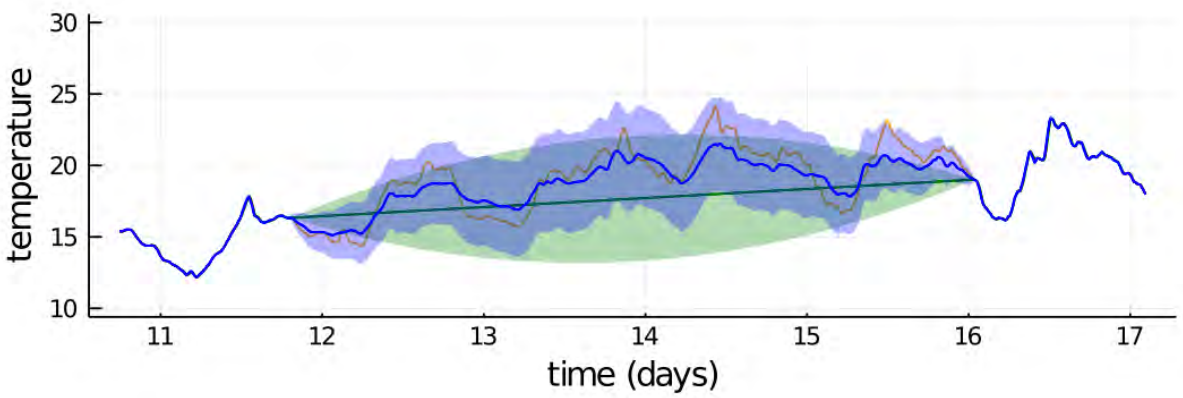

Figure 4.4: Interpolation of missing data on the air temperature in Cambermet during the month of July 2013. We compare two models; one using scaled independent temporal GPs (green), and the scaled GPAR model (blue). The ground truth is displayed in orange. The challenge is to retrieve the missing data between days 12-16. Whereas the best the independent GP can do is draw a line between the ends of the missing parts, the GPAR recognises and models the structure of the data.

Our goal is to predict the exchange rate USD/AUD for the year 2007 using all the other available data (including other exchange rates for the same year).

We ended up using only five inputs; the year, and four exchange rates for CHF, NZD, CAD, and JPY. These were chosen thanks to their high correlation with the USD/AUD exchange rate and thanks to them having little to none missing data. The results can be seen in Figure 4.5. Again, GPAR manages to capture the underlying function whereas the independent GP is not expressive enough to capture any meaningful information.

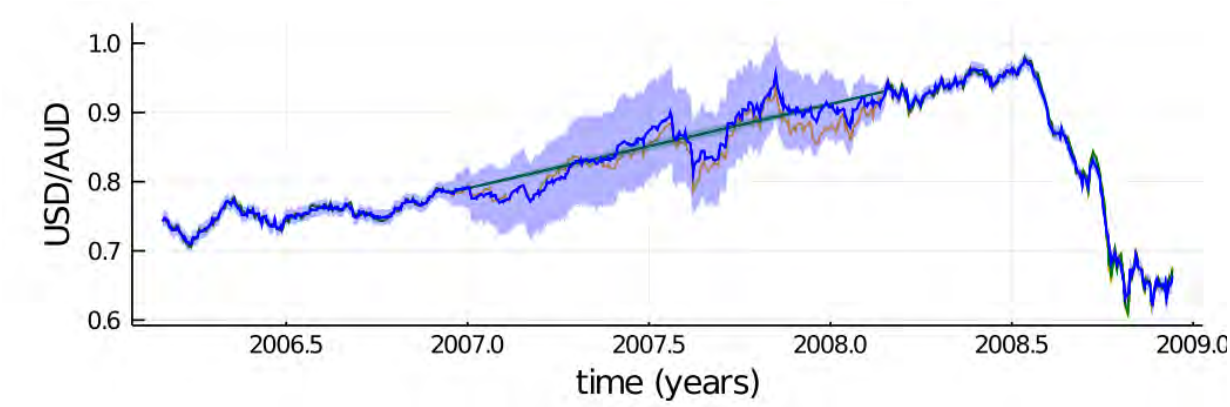

Figure 4.5: Interpolation of missing data on the USD/AUD exchange rate during the year 2007. We compare two models; one using scaled independent temporal GPs (green), and the scaled GPAR model (blue). The ground truth is displayed in orange. The challenge is to retrieve the exchange date for the year 2007, using all the other years and other exchange rates as input.

Chapter 5

Conclusions and further work

Gaussian Processes are powerful tools for tackling single-output supervised learning problems [Williams and Rasmussen, 2006], but they do not transfer their non-linear dependency capture abilities to multi-output problems. The Gaussian Process Autoregressive Regression (GPAR) model [Requeima et al., 2018] addresses this issue, but its cubic in the number of observations runtime complexity means that GPAR cannot be applied to large datasets. A way to scale GPAR has been explored in the original paper; it involves pseudo-point approximations over GPAR, similar to how one would scale a standard GP. However, directly applying pseudo-point approximations to GPAR involves also approximating the time channel through pseudo-points. We wish to avoid approximating the time channel since its lower lengthscale hyperparameters and its (relatively) unbounded domain leads to an ever-increasing number of pseudo-points in order to get a good approximation. The implication of this is that the number of required pseudo-points scale directly with the number of observations, leading to a cubic complexity (albeit still a lower complexity than original).

We introduced a new way to scale GPAR, in which we treat the temporal channel as part of the noise, and then apply DTC. This introduced correlated noise, meaning that the noise matrix is no longer a scaled identity matrix, and thus inverting it becomes an operation with cubic complexity. Our goal is to reduce the complexity from cubic, so requiring to compute a cubic operation is unacceptable. However, we can avoid inverting the noise matrix by converting the temporal GP into a linear gaussian state space model on which we perform filtering and smoothing. This allows us to circumvent the costly inversion operation. Making predictions in this new framework also involves noise matrix inversions, which can again be accelerated using state space techniques.

We compare the time and memory complexities of our model against standard GP and GPAR models in Table 5.1.

We evaluated the model using three evaluation techniques:

	standard GP	unscaled GPAR	scaled GPAR
Time complexity	$\mathrm{O}(O^3N^3)$	$\mathrm{O}(ON^3)$	$\mathrm{O}(OM^2N)$
Memory complexity	$\mathrm{O}(O^2N^2)$	$\mathrm{O}(ON^2)$	$\mathrm{O}(OMN)$

Table 5.1: Time and memory complexities of three GP model classes, where O is the number of output dimensions, M is the number of pseudo-points, and N is the number of observations. We usually have the relation $O < M << N$. We see that the scaled GPAR version is the only one linear in the number of observations; both in terms of time and memory.

1. Sanity check: we evaluated the model in the exact case where the pseudo-points are the same as the inputs. The goal was to check whether the scaled version of GPAR performs the same as GPAR in this degenerate case. We evaluated on two small datasets; a synthetic dataset and the EEG dataset. The results proved that the GPAR version from this thesis is the same as standard GPAR in the degenerate case.

2. Speed improvements; we wanted to check the speed improvement of the new model. To that extent, we evaluated three scaled models with a different number of pseudo-points against a standard GPAR model. We used a synthetic dataset for which we could fix the number of observed points. The evaluation proved that the scaled version is considerably faster than standard GPAR as we add more observation points. We also noticed a slight improvement in the mean squared error.

3. Large datasets; we compared scaled GPAR against independent GPs on large datasets (around $10,000$ observations). We used two datasets, one containing air temperature data from fours weather stations, and one containing the exchange rates for international currencies. In both cases, we observed that scaled GPAR manages to capture the inter-dependencies of the dataset and to give useful predictions. In contrast, the independent GPs did not perform as well, not managing to capture any meaningful information.

In conclusion, we implemented a way to scale the Gaussian Process Autoregressive Regression model, which allows us to use the powerful GP on large-scale datasets. We evaluated the scaled model and reached the conclusion that it is faster than GPAR and performs better than an independent GP, and it also is the same as standard GPAR in the case where we use exact pseudo-points. Further work which could be done on this project includes:

- Optimizing the pseudo-point locations $\mathbf{z}$. At the moment we just use regularly spaced locations, but the exact locations can also be trained in a greedy fashion.

- Using FITC as out pseudo-point approximation algorithm instead of DTC.
- Adding the ability to use other temporal kernels such as squared exponential.
- Expand the model to handle non-Gaussian likelihood functions by introducing approximations to thee likelihood.
- Improve modularity by allowing posterior GPs to be used as prior GPs.

Bibliography

[Andrews, 2001] Andrews, A. P. (2001). *Kalman Filtering: Theory and Practice Using MATLAB.* Wiley.

[Bulla, 2006] Bulla, J. (2006). Application of hidden markov models and hidden semi-markov models to financial time series. *University Library of Munich, Germany, MPRA Paper*.

[Calandra et al., 2016] Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. (2016). Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345. IEEE.

[Dai et al., 2017] Dai, Z., Alvarez, M., and Lawrence, N. (2017). Efficient modeling of latent information in supervised learning using gaussian processes. In *Advances in Neural Information Processing Systems*, pages 5131–5139.

[Doucet, 2010] Doucet, A. (2010). A note on efficient conditional simulation of gaussian distributions. *Departments of Computer Science and Statistics, University of British Columbia*, 4.

[Duvenaud et al., 2013] Duvenaud, D., Lloyd, J., Grosse, R., Tenenbaum, J., and Zoubin, G. (2013). Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, pages 1166–1174.

[Eubank and Wang, 2002] Eubank, R. and Wang, S. (2002). The equivalence between the cholesky decomposition and the kalman filter. *The American Statistician*, 56(1):39–43.

[Hartikainen and Särkkä, 2010] Hartikainen, J. and Särkkä, S. (2010). Kalman filtering and smoothing solutions to temporal gaussian process regression models. In *2010 IEEE international workshop on machine learning for signal processing*, pages 379–384. IEEE.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

[Lee et al., 2017] Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2017). Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*.

[MacKay, 1998] MacKay, D. J. (1998). Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166.

[Matérn, 1960] Matérn, B. (1960). Spatial variation-stochastic models and their application to some problems in forest surveys and other sampling investigations. meddelanden fran statens skogsforskningsintitut, almaenna foerlaget, stockholm. (1986), 49 (5).

[Matthews et al., 2016] Matthews, A. G. d. G., Hensman, J., Turner, R., and Ghahramani, Z. (2016). On sparse variational methods and the kullback-leibler divergence between stochastic processes. *Journal of Machine Learning Research*, 51:231–239.

[Matthews et al., 2018] Matthews, A. G. d. G., Rowland, M., Hron, J., Turner, R. E., and Ghahramani, Z. (2018). Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*.

[Nguyen et al., 2014] Nguyen, T. V., Bonilla, E. V., et al. (2014). Collaborative multi-output gaussian processes. In *UAI*, pages 643–652.

[Øksendal, 2003] Øksendal, B. (2003). Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer.

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

[Rauch et al., 1965] Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450.

[Requeima et al., 2018] Requeima, J., Tebbutt, W., Bruinsma, W., and Turner, R. E. (2018). The gaussian process autoregressive regression model (gpar). *arXiv preprint arXiv:1802.07182*.

[Särkkä, 2013] Särkkä, S. (2013). *Bayesian filtering and smoothing*, volume 3. Cambridge University Press.

[Särkkä et al., 2006] Särkkä, S. et al. (2006). *Recursive Bayesian inference on stochastic differential equations*. Helsinki University of Technology.

[Sarkka et al., 2013] Sarkka, S., Solin, A., and Hartikainen, J. (2013). Spatiotemporal learning via infinite-dimensional bayesian filtering and smoothing: A look at gaussian process regression through kalman filtering. *IEEE Signal Processing Magazine*, 30(4):51–61.

[Seeger et al., 2003] Seeger, M., Williams, C., and Lawrence, N. (2003). Fast forward selection to speed up sparse gaussian process regression. Technical report.

[Solin et al., 2016] Solin, A. et al. (2016). Stochastic differential equation methods for spatio-temporal gaussian process regression.

[Titsias, 2009] Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574.

[Williams and Rasmussen, 2006] Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

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

[Wilson et al., 2016] Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. (2016). Stochastic variational deep kernel learning. In *Advances in Neural Information Processing Systems*, pages 2586–2594.

Appendix A

Project Proposal

Tudor Paraschivescu
Churchill
tp423

Machine Learning and Machine Intelligence Project Proposal

Multi-output Gaussian Process Regression at Scale

3rd of April, 2020

Project Originator and Supervisors: Dr Richard E. Turner, Will Tebbutt, and Wessel Bruinsma

Proposal body.

Multi-output regression problems are a subset of supervised learning problems where we try to infer multiple scalar outputs from a given input. Usually, the outputs in such problems exhibit inter-dependencies, so a performant model will take those dependencies into account during training and inference.

Gaussian Process models are useful when doing regression as they directly capture model uncertainty and allow the user to input prior knowledge into the model through the selection of kernel functions. However, naively extending GP models to multi-output regression problems leads to a blow-up in complexity and limited representation.

A clean, practical way to extend GP models to multi-output problems is through the use of Gaussian Process Autoregressive Regression (GPAR) models [Requeima et al., 2018]. They work by using the product rule to decompose the joint distribution over the outputs into a set of conditionals which we model by standard GPs as in Equation A.1. The unknown functions $f_{1:M}$ are what we model using Gaussian Processes.

$$p(y_{1:M}(x)) = \underbrace{p(y_1(x))}_{f_1} * \underbrace{p(y_2(x)|y_1(x))}_{f_2} \dots \underbrace{p(y_M(x)|y_{1:M-1})}_{f_M} \tag{A.1}$$

GPAR achieved state-of-the-art performance so far, but up until now only exact inference was used, so all the tasks have been small-scale. The aim of this project is to extend GPAR to larger models using approximate inference through methods such as pseudo-points[Seeger et al., 2003] and state-space[Sarkka et al., 2013] approximations.

Deterministic Training Conditiona (DTC)

DTC can be thought of as a poor approximation to the variational free energy method of Titsias [Titsias, 2009]. We don't use the VFE because the trace term seems to be a bottleneck and leads to quadratic complexity in the number of data points when combined with the state-space approximation.

DTC is used to approximate models of the form

$$\begin{aligned} f &\sim \mathcal{GP}(0, \mathcal{K}(v, v')) \\ \epsilon &\sim \mathcal{GP}(0, \sigma^2 \delta(v - v')) \\ y(v) &= f(v) + \epsilon(v) \end{aligned} \tag{A.2}$$

We turn GPAR into a model similar to the ones suitable for DTC by decoupling the GP kernels for previous inputs from the ones for time, and then treat the time GPs as noise.

State-space approximation

GPs for time-series can be represented as Linear Gaussian State Space models which allow for O(N) computation of log marginal likelihoods. We will use state-space approximations in combination with DTC.

Measurements and success criteria

We will measure the success of the project by running tests on various multi-outputs real-life datasets and comparing the log marginal likelihood against other multi-output models.

The project will be a success if GPAR is successfully scaled to long multi-output time-series problems through the aforementioned methods.

Workplan

The work plan consists of 4 stages as follows:

1. **Literature review**: getting familiar with the existing literature is the starting point. This part should be mostly done by the end of April
2. **Implementation**: we will implement three models as follows:
   (a) Implementation of exact GPAR inference.
   (b) Implementation of a vanilla pseudo-point approximation technique. This will most likely be based on the variational free energy technique [Titsias, 2009]. The code is fairly similar to DTC and this approach should prove to be more efficient than DTC.
   (c) Implementation of pseudo-point approximation using DTC and combining it with state-space approximations.

   The implementation part should be finished by the start of July.
3. **Evaluation** involves benchmarking the three models using log marginal likelihood, held-out log-likelihood, and Root mean squared error on the held-out data. All of these metrics will be considered when judging the speed-accuracy tradeoff of the models.

   The evaluation part will be intertwined with the implementation part when possible, and thus should not take more than three weeks. The aim is to finish the evaluation by mid-late July.
4. **Writing**: assuming everything goes to plan, a full month will be dedicated to this part. Some part of the writing will also be done during the literature review stage.

Resource declaration

I will use my personal computer for development, training, evaluation, and write-up. In the case that this proves insufficient, I will switch to using MLMI GPUs or some form of cloud computing (AWS or Google Cloud).

Bibliography

[Andrews, 2001] Andrews, A. P. (2001). *Kalman Filtering: Theory and Practice Using MATLAB.* Wiley.

[Bulla, 2006] Bulla, J. (2006). Application of hidden markov models and hidden semi-markov models to financial time series. *University Library of Munich, Germany, MPRA Paper*.

[Calandra et al., 2016] Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. (2016). Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345. IEEE.

[Dai et al., 2017] Dai, Z., Alvarez, M., and Lawrence, N. (2017). Efficient modeling of latent information in supervised learning using gaussian processes. In *Advances in Neural Information Processing Systems*, pages 5131–5139.

[Doucet, 2010] Doucet, A. (2010). A note on efficient conditional simulation of gaussian distributions. *Departments of Computer Science and Statistics, University of British Columbia*, 4.

[Duvenaud et al., 2013] Duvenaud, D., Lloyd, J., Grosse, R., Tenenbaum, J., and Zoubin, G. (2013). Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, pages 1166–1174.

[Eubank and Wang, 2002] Eubank, R. and Wang, S. (2002). The equivalence between the cholesky decomposition and the kalman filter. *The American Statistician*, 56(1):39–43.

[Hartikainen and Särkkä, 2010] Hartikainen, J. and Särkkä, S. (2010). Kalman filtering and smoothing solutions to temporal gaussian process regression models. In *2010 IEEE international workshop on machine learning for signal processing*, pages 379–384. IEEE.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

[Lee et al., 2017] Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2017). Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*.

[MacKay, 1998] MacKay, D. J. (1998). Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166.

[Matérn, 1960] Matérn, B. (1960). Spatial variation-stochastic models and their application to some problems in forest surveys and other sampling investigations. meddelanden fran statens skogsforskningsintitut, almaenna foerlaget, stockholm. (1986), 49 (5).

[Matthews et al., 2016] Matthews, A. G. d. G., Hensman, J., Turner, R., and Ghahramani, Z. (2016). On sparse variational methods and the kullback-leibler divergence between stochastic processes. *Journal of Machine Learning Research*, 51:231–239.

[Matthews et al., 2018] Matthews, A. G. d. G., Rowland, M., Hron, J., Turner, R. E., and Ghahramani, Z. (2018). Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*.

[Nguyen et al., 2014] Nguyen, T. V., Bonilla, E. V., et al. (2014). Collaborative multi-output gaussian processes. In *UAI*, pages 643–652.

[Øksendal, 2003] Øksendal, B. (2003). Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer.

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

[Rauch et al., 1965] Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450.

[Requeima et al., 2018] Requeima, J., Tebbutt, W., Bruinsma, W., and Turner, R. E. (2018). The gaussian process autoregressive regression model (gpar). *arXiv preprint arXiv:1802.07182*.

[Särkkä, 2013] Särkkä, S. (2013). *Bayesian filtering and smoothing*, volume 3. Cambridge University Press.

[Särkkä et al., 2006] Särkkä, S. et al. (2006). *Recursive Bayesian inference on stochastic differential equations*. Helsinki University of Technology.

[Sarkka et al., 2013] Sarkka, S., Solin, A., and Hartikainen, J. (2013). Spatiotemporal learning via infinite-dimensional bayesian filtering and smoothing: A look at gaussian process regression through kalman filtering. *IEEE Signal Processing Magazine*, 30(4):51–61.

[Seeger et al., 2003] Seeger, M., Williams, C., and Lawrence, N. (2003). Fast forward selection to speed up sparse gaussian process regression. Technical report.

[Solin et al., 2016] Solin, A. et al. (2016). Stochastic differential equation methods for spatio-temporal gaussian process regression.

[Titsias, 2009] Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574.

[Williams and Rasmussen, 2006] Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

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

[Wilson et al., 2016] Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. (2016). Stochastic variational deep kernel learning. In *Advances in Neural Information Processing Systems*, pages 2586–2594.