Structured Deep Gaussian Processes

João Perre Carvalho de Brito Orelhas

Masters' Dissertation

Supervisors: Prof. Vera Miguéis (University of Porto)
Prof. Edwin Bonilla (University of New South Wales)

Integrated Master in Industrial Engineering and Management

2018-06-15

Abstract

Deep Gaussian Processes (DGPs) benefit both from the flexible probabilistic approach of Gaussian Processes and the scalability of neural networks. They have already been successfully used for unstructured prediction tasks, but their structured version has not yet been investigated. This thesis proposes a new formulation of DGPs which can accommodate the structuredness of data and make predictions accordingly. This model is based on Random Feature Expansions (RFE) and is trained using Stochastic Variational Inference. It is then evaluated on standard Natural Language Processing (NLP) tasks and shows performance improvements both in the Error Rates and in Uncertainty Quantification compared to other state-of-the-art techniques like Gaussian Processes with Elliptical Sliced Sampling (GP-ESS), Conditional Random Fields (CRFs) and Gaussian Processes with inducing inputs and variational inference. This is also a scalable model, as evidenced by its performance on a dataset with 211,000 words and 11,000 sentences.

Acknowledgements

First, I would like to thank Professor Edwin Bonilla for having the courage to take me in without any background in the field of Gaussian Processes. It was a unique and extremely enriching learning journey, and I am truly grateful Professor Edwin trusted me with this project, as it was a once in a lifetime opportunity which I will treasure forever. Thank you for teaching me, providing me with the tools to learn and for being such an outstanding lead example with your tremendously inspiring motivation, something that I definitely bring with me from my experience there. I also thank the cooperation program Merging Voices, without which this would never have been possible.
I am also very thankful to Professor Vera Miguéis for the thorough review of this work. Your critical thinking and interest even in the smallest intricacies of this project truly contributed both to this thesis and my growth.
To my lab colleague Shuai Zhang: thank you for your friendship and motivation, your hard-work ability and knowledge pushed me further everyday.
To all my friends from FEUP, and especially from Tuna de Engenharia: thank you for making this journey so enjoyable, Porto's student life can be a unique experience and I think I can say I got the best of it.
To the friends that have accompanied me through my life, I am very grateful to have met you and for all the moments of joy we lived.
To my family: thank you for constantly supporting me to pursue my dreams, both here and on the other side of the world. I am very happy that I can count on your presence whenever I need you.
To the friends I made in Sweden, Taiwan and Australia: I am glad to have met so many interesting people in all these places, and I hope to meet every one of you again. Thank you.

João Perre

"Tenho em mim todos os sonhos do mundo."

Álvaro de Campos

Contents

Acronyms and Symbols

ARD	Automatic Relevance Determination
D	Dimensionality of the inputs
DGP	Deep Gaussian Process
GP	Gaussian Process
GPU	Graphical Processing Unit
i	Imaginary number
KL	Kullback-Leibler
LHS	Left hand side
MCMC	Markov Chain Monte Carlo
MRF	Markov Random Fields
MVG	Multivariate Gaussian
N	Number of observations
Nh	Number of hidden layers
NLP	Natural Language Processing
Nseq	Number of sequences
RBF	Radial Basis Functions
RFE	Random Features Expansion
RHS	Right hand side
X	Set of all inputs - matrix of N rows and F columns
$\mathbf{x}$	One input
Y	Set of all outputs
$\mathbf{y}$	One output
$\|v\|$	Number of labels

List of Figures

List of Tables

Chapter 1

Introduction

Machine Learning deals with making predictions based on data. This topic has interested researchers and practitioners for a long time, but it was only with the increase in computational power and boom in available data that Machine Learning methods saw their real potential. From the early successes in the finance world by quant funds, to the more recent applications in image recognition by Facebook, Google and the likes, Machine Learning has proven that it is one of the most important tools of the current world. Not only are such algorithms important to passively analyze the world, they have already been able to actively influence it, as evidenced by their role in elections.

These techniques can be deconstructed in many ways. A typical one is "supervised" and "unsupervised" learning, depending on whether the training dataset is labelled or not. Another one is "probabilistic" and "non-probabilistic". Non-probabilistic methods do not provide any measure of uncertainty, but probabilistic ones do. The most typical criteria to evaluate such techniques is how often they are right or wrong (accuracy or error rate), but many other criteria can be found. Most techniques can be decomposed into a set of inputs, an objective function, a predictive function and some prediction loss measure. However, in the real world, the loss caused by a bad prediction strongly depends on the specific problem. Modelling the failure rate of a spacecraft structure is not the same as recognizing faces online, so the criteria used should depend on the problem being solved, and tolerance for errors should vary accordingly. For situations when freedom of fit and a quantification of uncertainty are desired, Gaussian Processes are a solution to look into.

Gaussian Processes (GPs) and variations have been used for centuries - some of its most famous variations are Wiener Processes, Kalman Filters and Langevin Processes. However, the application of Gaussian Processes to prediction problems only started in the 1940s with time series. Later, in the 1970s, they were applied to meteorology and geostatistics - in particular to mining engineering. Their arrival to statistics itself only came a few years later, with multivariate input regression problems, and although they are a standard tool of statisticians, temporal and spatial applications are still the largely dominant applications. Regarding Machine Learning, they were first studied in the 1990s, at a time when neural networks were the main models used. Around the same time, Neal (1996) showed that, as a matter of fact, under certain conditions, neural networks could

tend to a GP, linking these 2 promising techniques (Snelson, 2007). In the early 2000s there was extensive work on GPs, culminating with the book by Rasmussen and Williams (2006), but more recently, the focus of the academic community has been on neural network related techniques.

Aside from being probabilistic methods that can be used in supervised and unsupervised learning, Gaussian Processes are also "non-parametric". This means that a GP does not have a fixed set of parameters, but instead the parameters grow with the number of training points (Murphy, 2013). These are generally more flexible than parametric methods, as they do not introduce assumptions on the data distribution, like linear or non-linear regression do. However, this increase in flexibility comes at the cost of increased computational costs. The time complexity to train and use a standard GP is $O(N^3)$, where N stands for the number of observations, as it requires the inversion of an $N \times N$ covariance matrix, which is usually done via Cholesky decomposition. For that reason, in order to make GPs scalable, several approximations have been introduced throughout the years. This thesis takes advantage of some of these approximations, and combines work done within the GPs field and the neural networks' field. It bridges these two and brings the properties of both in order to successfully solve problems where outputs have dependencies between each other - Structured Prediction - evidencing GPs can rival and beat state-of-the-art techniques.

1.1 Problem statement

This work introduces a new formulation of Deep Gaussian Processes with the aim of solving Structured Prediction problems - Structured Deep Gaussian Processes (SDGPs). To evaluate this new formulation, a series of questions has been posed:

- How does an unstructured model's architecture have to change to encompass the structuredness of data?
- What is the impact of using different hyperparameters when solving Structured Prediction problems?
- How does this model compare with the current state-of-the-art in the standard metrics (correctness and uncertainty quantification)?

1.2 Overview

This first chapter briefly introduces the topic with the sole objective of familiarizing the reader with GPs on a high level and their place in Machine Learning. Chapters 2 and 3 aim to review what has been done related to the main topic of this thesis: Gaussian Processes and Structured Prediction. Chapter 2 gives a better insight of how and why GPs are used, and introduces their intricate particularities related to the SDGPs model. This includes topics like Deep Gaussian Processes and key approximations to make them scalable like Random Feature Expansions. Chapter 3 outlines the type of problem being solved, Structured Prediction, and solutions that have been proposed

to address this. Chapter 4 explains the full SDGP model, how it is approximated to a tractable model and how it is trained and used. Then, chapter 5 consists of the evaluation of this model on a series of popular benchmark datasets. At last, chapter 6 includes the conclusion and ideas for future work.

Chapter 2

Gaussian Processes

This section aims to explain the key concepts required to grasp how Gaussian Processes are used in the field of Machine Learning, and why they are used.

2.1 Gaussian Processes in Machine Learning

Techniques based on Gaussian Processes (GPs) have the purpose of making predictions, just like regression. For that reason, it is important to compare them to such techniques and highlight why and when they are preferred - the motivation for using GPs. After knowing the advantages of GPs, it is important to define what they are mathematically, their key parameters, and how they can be interpreted. However, knowing the definition is not enough to understand how they are used to make predictions, so the following section will deal with the procedure of using GPs to make predictions. On the other hand, being a Machine Learning technique, it is clear that the predictions should be done after putting the model through a learning scheme. This training/learning stage is explained next. At last, the defining element of GPs, the kernel, will be introduced and some examples of kernels briefly analyzed.

2.1.1 Motivation

The main advantages of GPs are: ability to quantify uncertainty in predictions and flexibility in the functions it can fit. To convey these, an example from industrial engineering will be presented. The context to consider can be an industrial setting with a certain flow rate (the Ys), which depends on certain machine settings (the Xs). Suppose that in the past a variety of parameters has been tried successfully, but now the manager wants to try completely new settings on the machine (a new value of X). It is obviously important to know what will happen to the flow rate under the new circumstances.

Many techniques can be used to answer this question, which revolves around building a predictive function. The most basic one would be obtained by looking at the data distribution and fitting a line through it - linear regression. It would also be possible to increase complexity and fit a non-linear function (a polynomial, sinusoidal, exponential, etc.) - non-linear regression. One could go

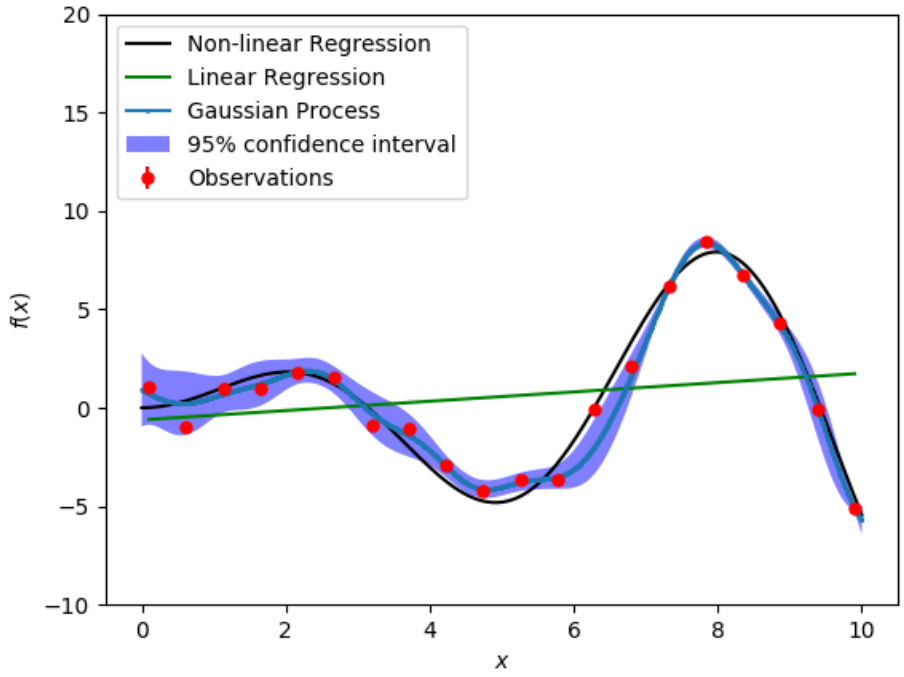

Figure 2.1: GPs compared to regression techniques. The uncertainty measure of the GPs is illustrated by the shaded areas surrounding the mean function of the GP in blue.

even further and consider a network of such continuous functions combined with ON/OFF activation functions in a network - a neural network. However, this would still restrict the fit to a finite set of functions. GPs, on the other hand, allow for more freedom, as it is not necessary to specify the functions that should fit the data. The data is fitted to the best distribution of functions. The impact this has in the ability to make predictions is shown in Figure 2.1. In green, an example of a single parametric function (linear regression) can be seen, in black, a more complex function, and in blue, the Gaussian Process.

The shortcomings of choosing a linear function are clear: the chosen function might not fit the data particularly well and there is no measure of uncertainty. The first problem can be strongly mitigated by using a more advanced method like non-linear regression. However, the lack of an uncertainty measure remains. This means that if a prediction is made for a scenario which is considerably different than previously observed ones, it is not possible to quantify how good or bad that prediction is. This is where GPs prove to be particularly helpful. The advantage of having a measure of uncertainty might not be obvious at first, but what it means is that different predictions have different levels of quality. Predictions for a test point that is considerably far away from the training points should intuitively have less quality than predictions for points that are near the training data. This is what the GP captures and what the shaded areas represent: quality of predictions. This is especially important when the cost of making a bad prediction is particularly high or when there is a considerable amount of missing data. Firstly, regarding

the former, consider the following example of modelling the failure of an airplane engine, aside from a prediction of failure, it is important to know how good or bad such a prediction is. If the model predicts that the engine will not fail but is highly uncertain, it might be better to re-evaluate, run more tests, etc. If a standard non-probabilistic method is used, the analysis would stop at "prediction of non-failure", which would be insufficient. As a matter of fact, GPs are heavily applied in the Aerospace industry (Dodd and Rogers, 2000). Secondly, regarding problems with a lot of missing data, the uncertainty quantification also gains higher relevance, which has been studied by Lu and Tang (2014) and Snoek et al. (2012). This missing data problem occurs in many fields. For example, in medicine, there are thousands of clinical tests available, but patients will only have done a few, making the data available sparse (Clifton et al., 2013; Cheng et al., 2017). Other fields that benefit from this mathematically principled uncertainty quantification are soil mapping (Gonzalez et al., 2008), industrial and mechanical engineering (bearing degradation assessment (Hong and Zhou, 2012)), natural language processing (Cohn et al., 2014) and computer vision (Bratieres et al., 2014).

Knowing the motivations for using GPs, their inner workings will be presented in more detail. First, their definition will be introduced. Then, their training and prediction schemes will be explained.

2.1.2 Definitions

GPs can be interpreted in at least 3 different ways. The first one is as an infinite collection of Gaussian variables, which is the standard definition. The second one is as an infinite-dimensional multivariate Gaussian distribution and the third one is as a normal distribution generalized to functions. Starting with the standard definition, a Gaussian Process is a stochastic process (infinite collection of random variables) where any finite subset of random variables follows a multivariate Gaussian distribution.

Thinking about an infinite collection of random variables might not be natural, so a more intuitive interpretation is brought: GPs as a generalization of the normal distribution. The industrial engineering example will be reintroduced to materialize this transition from the normal distribution to the GP. The nomenclature is the same, that is, Y representing a certain flow rate and X representing the parameters of the machine that is outputting that flow rate. At the beginning of this experiment, suppose only one point has been studied, x_1. It is known that the flow rate Y_1 with parameters x_1 follows a normal distribution with a certain mean and variance:

$$Y_1(x_1) \sim N(\mu, \sigma) \tag{2.1}$$

After some experiments, suppose another set of parameters x_2 has also been studied. Now suppose it is known that $Y(x_1)$ follows a normal, $Y(x_2)$ follows another normal, and they both follow a 2-dimensional multivariate Gaussian distribution:

$$Y(x_1), Y(x_2) \sim N(m(x_1, x_2), \Sigma(x_1, x_2)) \tag{2.2}$$

Continuing this process, after studying N sets of parameters, the output will follow an N-dimensional Gaussian. If $N \to \infty$, then an infinite collection of random variables will arise, with the restriction that every subset of them follows a multivariate Gaussian. Recalling the standard definition, it becomes clear that this is a Gaussian Process. In section 2.2, a graphical representation of this process is shown. In each picture, it is possible to see a certain number of Random Variables (represented by their mean and variance). The number of random variables increases from picture to picture and the last one is the representation of the Gaussian Process, when $N \to \infty$.

The third way of thinking about GPs is as a generalization of the normal distribution to functions. Under their "functional view" (Rasmussen and Williams, 2006), GPs are represented in the following way:

$$f \sim GP(m(x), k(x,x')) \tag{2.3}$$

They rely on a mean function $m(x)$ and on a covariance function $k(x,x')$, but the mean function is frequently set to zero. These covariance functions can take many forms and their choice is subject of thorough study (Shawe-Taylor and Cristianini, 2004). However, generally speaking, the idea is that as similarity between x's increases, similarity between y's also increases (Murphy, 2013). This means that if a prediction is made for an x close to many known x's, then the confidence on that prediction is high, otherwise it is low. This covariance function is responsible for the form of the distribution being modeled, but no specific function is chosen, as it encompasses all the functions. All possible explanations of the data are considered, although a preference is given to better ones. As a matter of fact, this fitting of 'infinite functions' with different weightings has been illustrated by Williams (1998), with the proof that a neural network tends to a GP when its number of nodes tends to infinite. The diagram in Figure 2.3 illustrates the transition from the most simple regression model to a GP. In a regression, there is only one function considered, and it can be extremely simple (linear), or more complex (non-linear). If many functions of different types are combined in a finite network with different weights given to different functions - a neural network - then the ability to represent complex phenomena grows considerably. If the number of functions in such a network grows to infinite, then such a network is equivalent to a Gaussian Process. These transitions are illustrated in Figure 2.3. Their Bayesian nature makes GPs inherently less prone to overfitting, even without any specific regularization schemes (Neal, 1996).

Lastly, an alternative approach to Gaussian Processes is as a type of Bayesian Regression. Please refer to Rasmussen and Williams (2006) for a detailed explanation.

2.1.3 Using a GP for predictions

The end goal of a Gaussian Process is making a prediction for a certain Y_* based on a certain X_*. The function used for making such predictions is called the predictive posterior and is computed using a conditional likelihood function, $P(Y_* \mid X_*, f)$, where X_* represents the new values of X for which predictions are to be made. In the most basic case, this likelihood function follows a

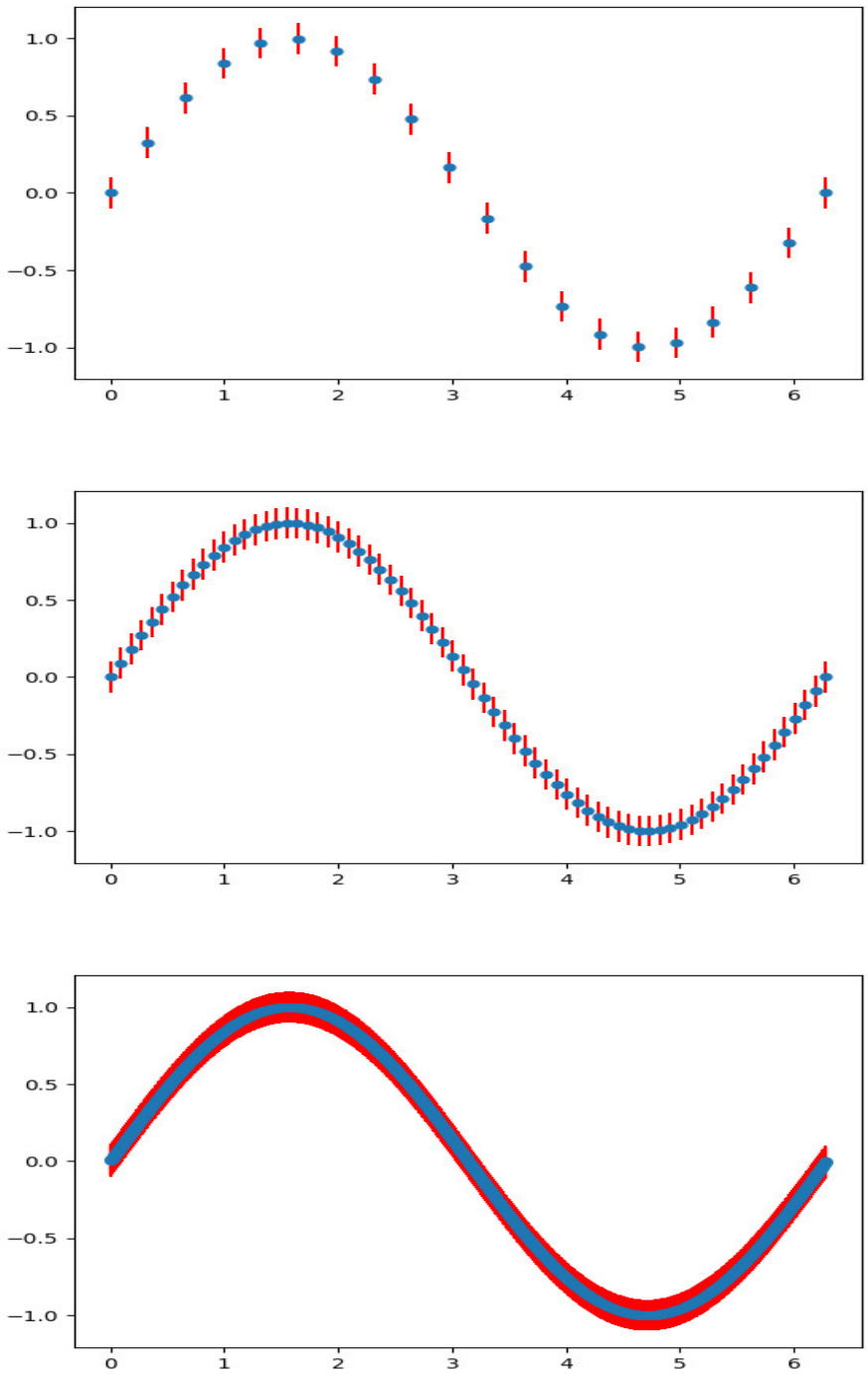

Figure 2.2: From a finite collection of Gaussian variables to a Gaussian Process

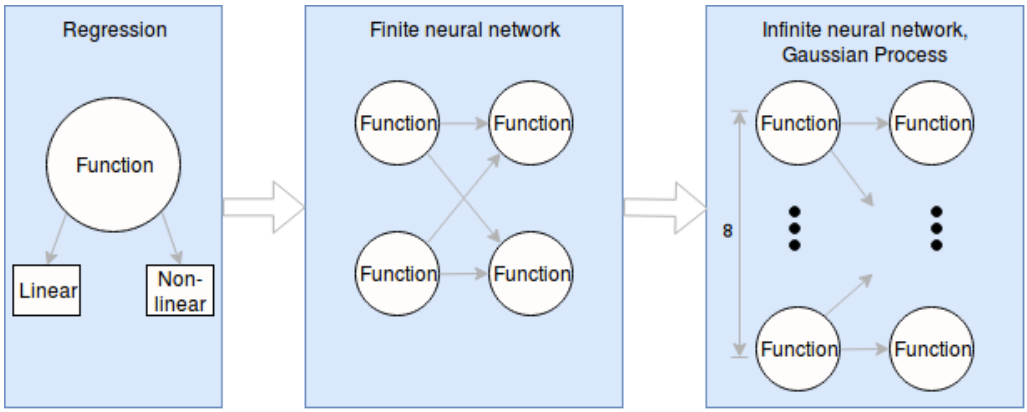

Figure 2.3: Regression, finite neural networks and Gaussian Processes

GP, which works well for some regression problems. The prediction Y_* is related to the observed outputs Y in the following way (Rasmussen and Williams, 2006):

$$\begin{bmatrix} Y \\ Y_* \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right) \tag{2.4}$$

To obtain the predictive posterior Y_*, it is only a matter of conditioning on the rest of the information available (the data and Y):

$$Y_* \mid X, X_*, Y \sim N(\mu, \sum) \tag{2.5}$$

$$\mu = K(X_*,X)[K(X,X)]^{-1}Y \tag{2.6}$$

$$\sum = K(X_*,X_*) - K(X_*,X)[K(X,X)]^{-1}K(X,X_*) \tag{2.7}$$

Where $\sum$ denotes the covariance matrix. A particularity of conditioning Y_* is that $Y_* \mid X_*, X$ follows a multivariate Gaussian distribution, bringing the Gaussian Process from the infinite variables' realm to the finite one. In this Gaussian regression setting, it is possible to compute predictions analytically. However, this is not always the case. For example, in classification, the likelihood $P(Y_* \mid X_*)$ will not be Gaussian, rendering the above method invalid. The standard technique to handle this is to consider a hidden function f which will follow a GP and then "squashing" its output via, for example, a logistic regression. In that case, the likelihood function will be a sigmoid and it will rely on the hidden function f instead of directly on the input X:

$$P(y = +1 \mid X, f) = \sigma(f) \tag{2.8}$$

To make predictions, the hidden functions are integrated out:

$$P(y_* = +1 \mid X, Y, X_*) = \int_{f_*} P(y = +1 \mid X, f) P(f_* \mid X, Y, X_*) df_* \tag{2.9}$$

where:

$$P(f_* \mid X,Y,X_*) = \int_f P(f_* \mid X,Y,f)P(f \mid X,Y)df \tag{2.10}$$

With this non-Gaussian likelihood, it is not possible to obtain the predictive posterior $P(Y_* \mid f_*)$ analytically (unlike in regression with Gaussian likelihood). For that reason, an approximation has to be introduced. The most typical approximations are either deterministic ones or sampling based (Rasmussen and Williams, 2006). On the analytical side, there are many options available like the Laplace approximation (Williams and Barber, 1998) and Expectation Propagation (Minka, 2001). On the sampling side, a popular technique is Markov Chain Monte Carlo (which originated from Physics as the 'Metropolis algorithm' (Metropolis et al., 1953) and later found its application in data analysis (Gelfand and Smith, 1990; Robert and Casella, 2011).

For the problem that this thesis addresses - Structured Prediction, the likelihood functions used are even more complex than the one used in classification, causing these integrals to be also intractable. The technique used to overcome this intractability was Variational Inference (Jordan et al., 1999), and it will be explained in a separate section.

2.1.4 Training a GP - Bayesian Inference

Gaussian Processes are Bayesian models. Being Bayesian can be considered a way of thinking about probabilities, and its competing school of thought is the frequentist view. The Bayesian approach looks at the probability of an outcome as a quantification of belief, while the frequentist approach considers it to be the limit of its relative frequency. This means that the Bayesian approach puts a probabilistic distribution over what is not known, updates such a distribution as more data is gathered and uses the updated distribution in order to make predictions. On the other hand, the frequentist approach uses the best estimator for what he does not know.

A typical coin flip example can be used to illustrate these two views. The probability of getting heads or tails is not known. The frequentist approach believes that such probabilities are fixed, so it does not put a distribution over them. Instead, it chooses a metric (estimator) like "past frequency". The Bayesian approach, on the other hand, believes that since the real probability is not known, it should follow a certain distribution, and as the coin is flipped, that distribution is updated using Bayes theorem.

To illustrate the Bayesian perspective in the context of Gaussian Processes, the industrial engineering example can be brought again. Suppose a GP is used to make predictions about the flow rate of a machine. Before taking any measurements, there is no data to incorporate, so a certain Gaussian Process has to be chosen.

$$P(f) \sim GP(m(x),k(x,x')) \tag{2.11}$$

This prior can incorporate previous knowledge on the mean function $m(x)$ and kernel $k(x,x')$, but it does not account for any observations. At moment zero, since there is no data to use, the prior

is used to make predictions. As time advances, observations are gathered and the posterior can be computed using Bayes theorem like mentioned before.

$$P(f|Y,X) = \frac{P(Y|f,X)P(f \mid X)}{P(Y \mid X)} \tag{2.12}$$

To introduce the typical nomenclature:

- $P(f|Y,X)$ is called "the posterior" and it represents f's distribution after "seeing" the data, that is, *a posteriori*. This posterior feeds the "predictive posterior" introduced before in subsection 2.1.3, $P(Y_* \mid f_*)$, and they should not be confused.
- $P(Y \mid f,X)$ is the likelihood of observing Y given a certain f and X.
- $P(f \mid X)$ is called the prior because it is chosen *a priori* to any observations Y.
- $P(Y \mid X)$ is usually called evidence or marginal likelihood. This can be interpreted as: "the data observed is the evidence of the model this data follows". It is also called "marginal likelihood" because it is the result of a marginalization over the hidden functions f: $P(Y \mid X) = \int P(Y \mid f,X)P(f \mid X)df$.

Using the posterior yields a better prediction as opposed to simply considering the prior, and an illustration of these improvements is represented in Figure 2.4. The predictive functions using the prior are clearly scattered and not particularly useful to model any phenomenon. On the other hand, after collecting data, the predictive function is adjusted and a well defined trend is identified. The margin of error, represented by the shaded areas, is clearly smaller in the posterior, meaning predictions are of higher quality. In some areas, like the rightmost one, the "error bars" are not even visible, which means highly accurate predictions are expected. In other areas, like the leftmost one, since uncertainty is higher, the quality of the predictions is expected to be lower.

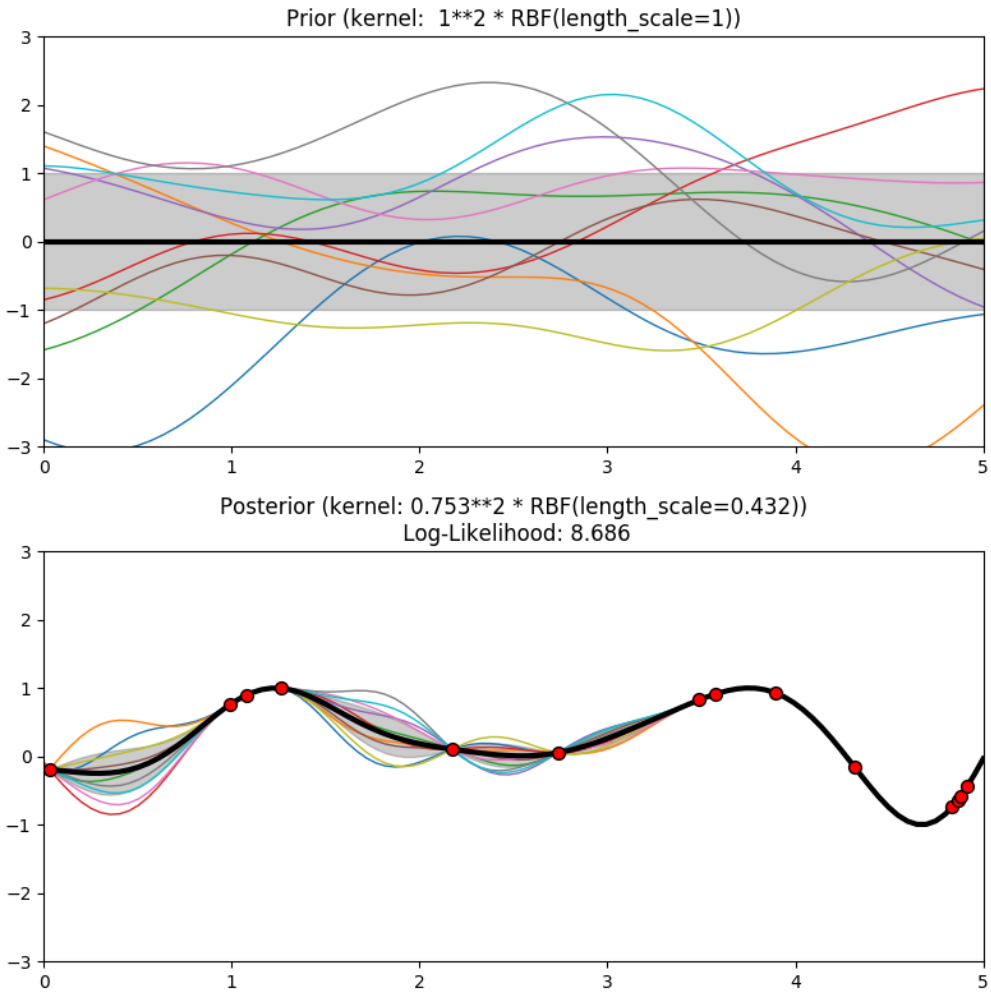

Figure 2.4: Predictions using a Gaussian Prior compared with predictions using a Posterior (with an RBF kernel).

Since the predictive power of the posterior is clearly superior to the prior's, obtaining the posterior can be considered part of training. However, there is still a large margin for improvement. So far, it has been considered that the kernel was completely fixed, but that does not need to be the case. Ideally, all the parameters defining the problem, S, namely the hyperparameters defining the kernel, can be optimized. If it was possible, one would like to maximize $P(S \mid Y,X)$, that is, maximizing the probability of these parameters being the "correct" parameters for the observed data. Since this is usually an intractable problem, the typical choice for objective function is what was previously defined as 'evidence', or 'marginal likelihood': $P(Y|X,S)$, that is, the probability of having observed what was observed, given a certain model with the observed data. Developing the marginal likelihood results in:

$$\log P(Y|X,S) \propto -\left[Y^T\left(K+\sigma^2 I\right)^{-1} Y + \log|\left(K+\sigma^2 I\right)|\right] \tag{2.13}$$

Equation 2.13 makes it evident that the marginal likelihood naturally balances model fit and model

complexity (Rasmussen and Williams, 2006), thereby making it a good choice for objective function.

Having defined a "framework" for training, which is maximizing the marginal likelihood, it is necessary to study how this is conducted. Consider once again how the marginal likelihood can be computed as a marginalization over the latent variables, f:

$$P(Y \mid X) = \int_f P(Y \mid f, X) P(f \mid X) df \tag{2.14}$$

Optimizing this for a regression model with Gaussian likelihood is an analytically tractable problem. However, as mentioned before, structured prediction brings complex likelihoods which make optimizing equation 2.14 intractable. This, once again, prompts the use of the Variational Inference approximation.

Following typical notation and for simplicity, the conditioning on X will be omitted from now on (i.e. $P(Y \mid X)$ will be referred to as $P(Y)$).

2.1.5 Variational Inference

This approach consists of approximating the true posterior distribution $P(f \mid Y)$, the term that brought intractability, with a simpler distribution $q(f)$. This $q(f)$ is called the "variational distribution", or the "variational posterior". Since the purpose of bringing q was to eliminate intractability, q should come from a tractable family. Therefore, the first step is choosing a family of distributions. Then, it is necessary to get more granular and actually compute its parameters, which is done by making the "variational posterior" as close as possible to the "true posterior" (Murphy, 2013). The origin of the name "variational distribution" will be explained first, followed by the explanation of how to obtain q.

The name "variational" comes from "variational calculus", a field of calculus that studies how variations can be used to find extreme points of functionals. A functional is a real-valued function on a vector space V, usually of functions (Rowland, 2018). Knowing that the domain of q is a space of functions, this definition was not unexpected. In plain English, a functional is essentially a function of a function, that is, instead of having a certain value as input, it has a function (Bishop, 2006). Some examples of functionals are entropy: $S[P] = \int P(X) \ln P(X) dX$, the energy functional $E[f] = \int \parallel \nabla f \parallel^2 dA$ (Rowland, 2018) and the cumulative distribution $C[P] = \int P(X) dX$. q is related to "variational calculus" because it is obtained by minimizing a functional, which will become evident in the next paragraph.

Having defined the mathematical object that q is, it is necessary to understand how it is obtained. The end goal is to make q as close to the true posterior $P(f \mid Y)$ as possible. Therefore, the first concept that needs to be defined is "closeness", or similarity. When dealing with vectors, a classical measure of similarity is the n-norm (Manhattan distance for n=1, Euclidean distance for n=2, etc.). However, when dealing with distributions, this concept of norm is not directly applicable. Instead, what is usually used for similarity (or more particularly, dissimilarity) is the

Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951), which is defined in equation 2.15:

$$KL(q(f)||P(f \mid Y)) = \int q(f) \log \frac{q(f)}{P(f \mid Y)} df \tag{2.15}$$

Then, it is necessary to define the family to which q will belong. Some possible choices are: "q will follow a multivariate Gaussian distribution", or "q factorizes over its inputs". After having defined q's form, its parameters have to be estimated. Making q as close as possible to $P(f \mid Y)$ means, ideally, minimizing their KL divergence. However, it is not really possible to minimize that KL divergence directly, as integrals involving the true posterior are the reason for the variational approximation in the first place. For that reason, it is necessary to convert this KL divergence into something else which is tractable to optimize (Blei, 2002).

Starting with the definition of KL divergence:

$$KL(q(f)||P(f \mid Y)) = \int q(f) * \log \frac{q(f)}{P(f \mid Y)} df \tag{2.16}$$

Bayes Theorem can be applied to $P(f \mid Y)$:

$$= \int q(f) * \log \frac{q(f) * P(Y)}{P(f,Y)} df \tag{2.17}$$

This is an expectation over a certain distribution q. Applying logarithm rules:

$$= \mathbb{E}_{q(f)}[\log q(f) - \log P(f,Y) + \log P(Y)] \tag{2.18}$$

Since $P(Y)$ does not depend on f in any way:

$$= \mathbb{E}_{q(f)}[\log q(f) - \log P(f,Y)] + \log P(Y) \tag{2.19}$$

Equation 2.19 holds a relationship between the original KL and the evidence. This is key to understanding how the original KL is related to the quantity that will actually be optimized, but this will be made clearer some paragraphs below.

$P(Y)$, the last term in equation 2.19, can be written as an integral:

$$\log P(Y) = \log \int P(Y,f) df$$

$$= \log \int P(Y,f) \frac{q(f)}{q(f)} df$$

$$= \log \mathbb{E}_{q(f)} \left[\frac{P(Y,f)}{q(f)} \right] \tag{2.20}$$

Jensen's inequality states that $f(E[X]) \leq E[f(x)]$ (more details in Appendix A.1). Applying Jensen's inequality to 2.20:

$$\log \mathbb{E}_{q(f)}\left[\frac{P(Y,f)}{q(f)}\right] \geq \mathbb{E}_{q(f)}\left[\log P(Y,f) - \log q(f)\right] \tag{2.21}$$

And this term is called the ELBO, which stands for Evidence Lower Bound, since it is a lower bound of the evidence, $P(Y)$. Knowing that $P(Y,f) = P(Y \mid f)P(f)$:

$$ELBO = \mathbb{E}_{q(f)}\left[\log P(Y \mid f)P(f) - \log q(f)\right]$$

$$= \mathbb{E}_{q(f)}\left[\log P(Y \mid f) - \log \frac{q(f)}{P(f)}\right]$$

Therefore,

$$ELBO = \mathbb{E}_{q(f)}\left[\log P(Y \mid f)\right] - KL(q(f)||P(f))] \tag{2.22}$$

This ELBO might seem unrelated to what was being initially minimized, the KL between the variational approximation and the true posterior. This relationship is evidenced below (Blei, 2002):

$$KL(q(f)||P(f \mid Y)) = \mathbb{E}_{q(f)}\left[\log \frac{q(f)}{P(f \mid Y)}\right]$$

$$= \mathbb{E}_{q(f)}\left[\log q(f)\right] - \mathbb{E}_{q(f)}\left[\log P(f \mid Y)\right]$$

$$= \mathbb{E}_{q(f)}\left[\log q(f)\right] - \mathbb{E}_{q(f)}\left[\log q(f,Y)\right] + \log P(Y)$$

$$= -\left(\mathbb{E}_{q(f)}\left[\log q(f)\right] + \mathbb{E}_{q(f)}\left[\log P(f,Y)\right]\right) + \log P(Y)$$

So,

$$KL(q(f)||P(f \mid Y)) = -ELBO + \log P(Y) \tag{2.23}$$

And now it is clear that minimizing the original KL can be done by maximizing the ELBO. The original KL and this negative ELBO differ only by a constant, $P(Y)$, constant which is bounded by the ELBO. An alternative strategy is starting directly by minimizing the model evidence, $P(Y)$, which was defined as the training objective. This is presented in the Appendix.

The great advantage of considering this ELBO instead of the original objective function is that its terms are computable. The first term is referred to as "Expected Log Likelihood" (L_{ell}) and can be computed using Monte Carlo Sampling and the second term (KL-divergence) is analytically tractable. This ELBO maximization corresponds to the training stage.

One problem that arises when maximizing this ELBO is scalability, for which the solution Stochastic Variational Inference has been proposed (Hoffman et al., 2013). This technique essentially uses stochastic optimization: finding extremes of the objective function by following noisy gradient estimates. This is particularly powerful if the objective function is a sum of independent terms, which makes this technique parallelizable.

2.1.6 Kernels

The kernel to consider is the key aspect of a GP, as it encodes the assumptions about the form of the function being modeled. As a matter of fact, the kernel and its parameters are oftentimes the only thing that the user chooses for a GP (as the mean function is set to zero). For that reason, the choice of the kernel is a subject of high importance. While there are many kernels available (Shawe-Taylor and Cristianini, 2004), the most relevant ones for this thesis are: the squared exponential, the arc-cosine and the linear kernel. These can be considered a dot product between a finite number of basis functions ϕ:

$$k(x,x') = \phi(x) \cdot \phi(x') \tag{2.24}$$

where $x,x' \in X \subset \mathbb{R}$.

If the kernel is translation invariant, then it is considered stationary (Genton, 2002):

$$k(x,x') = K_{stationary}(x-x') \tag{2.25}$$

Such kernels have properties which will become evident when deriving approximations for Gaussian Processes. A commonly used stationary kernel is the "**squared exponential**". In its most basic form:

$$k_{SE}(x,x') \equiv k_{SE}(\delta) = \sigma^2 \exp\left(-\frac{\delta^2}{2}\right) \tag{2.26}$$

where $\delta = x - x'$

It is immediate to notice that this is an infinitely differentiable function and that function smoothness is always guaranteed. This makes this kernel an interesting and useful one.

σ^2 is called the signal variance and it is a scaling parameter. It determines the variation of function values from their mean. Large values allow the function to incorporate outliers, and small values characterize functions that do not deviate much from the mean. Other parameters can also be considered if the complexity is higher. There can be a noise variance σ_{noise}, which has the purpose of modeling noise in the training data. Technically, this is not a part of the covariance function in itself, but it is incorporated in the GP. If this parameter is considerably high, the training data is expected to be noisy. Additionally, there can be a lengthscale parameter l incorporating function smoothness. If this parameter is high, then the function is considered to be smooth, that is, it changes slowly. If different dimensions are expected to have a different impact, then each dimension can have a specific lengthscale parameter l_d - this is called Automatic Relevance

Determination (ARD) (Neal, 1996; MacKay, 1991), and means that different dimensions have different relevances to the output. Including these additions, the squared exponential kernel can be considered in its most complete form:

$$k_{SE}(x,x') \equiv k_{SE}(\delta) = \sum_d \sigma^2 \exp\left(-\frac{\delta_d^2}{2l_d}\right) + \sigma_{noise} \tag{2.27}$$

where δ_d is the difference in x in dimension d.

Another stationary and important kernel is the **arc cosine**, introduced by Williams (1998). This one is more complex and involves evaluating the integral of a product of Heaviside step functions, which are denoted by $\Theta(z) = \frac{1}{2}(1 + sgn(z))$:

$$k(\delta) = 2\int \frac{\exp\left(\frac{-w^2}{2}\right)}{(2\pi)^{d/2}} \Theta(w \cdot x)\Theta(w \cdot y)(w \cdot x)(w \cdot y)^n \tag{2.28}$$

These kernels are related to neural networks and they have been proven to be successful, thereby evidencing that many advantages of neural networks can be brought to kernel-based methods (like GPs) (Weston et al., 2008). Depending on the mappings of the neural network that originated the arc cosine kernel, different arc cosine kernels can be created. However, the one with Heaviside step function mappings is the one that will be considered here (Cho and Saul, 2011).

The last kernel to be explained here is the **linear kernel**. This is a simple kernel given by the inner product summed to an optional constant.

$$k_{lin}(x,x') = x \cdot x' + c \tag{2.29}$$

It is clear that this is a non-stationary kernel, as it is not a function of the distance between x and x'. This was the kernel used by Galliani et al. (2016) for Structured Prediction with GPs.

The impact of the choice of kernel in the priors and posteriors can be seen in Figures 2.5 and 2.6. A new kernel (exponential sine squared) has been brought for a better representation of how diverse GPs can be depending on the kernel used. Essentially, although GP posteriors will always provide a good fit for the data provided, the level of uncertainty and form of the function is heavily influenced by the kernel. For example, in the linear case (Figure 2.5), the mean function is smooth and the uncertainty representation is reduced, but on the exponential sine squared one the exact opposite happens.

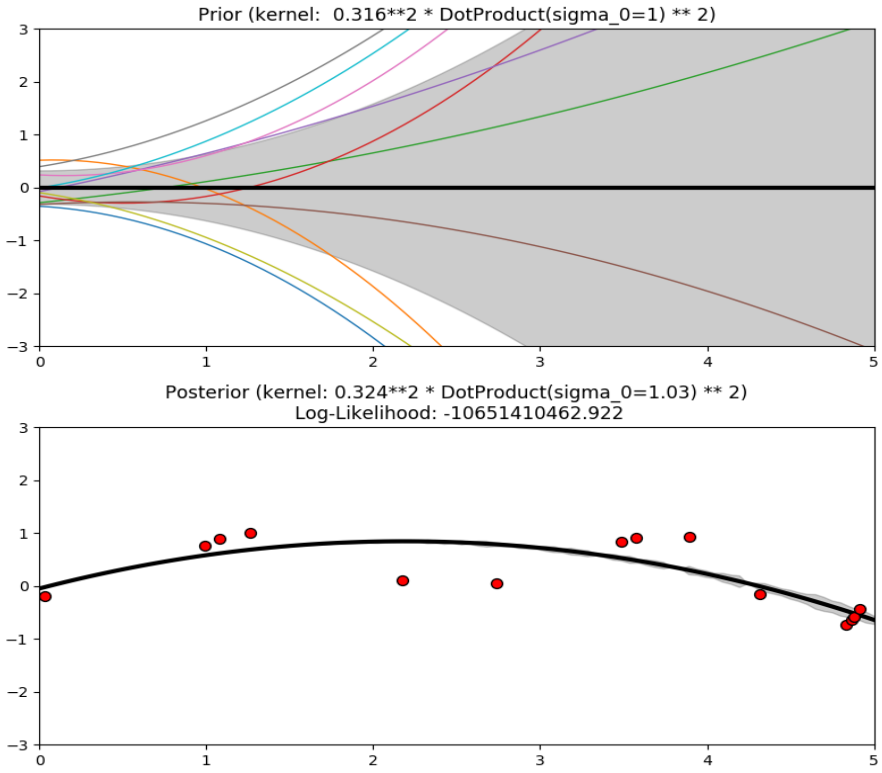

Figure 2.5: GP Prior and Posterior: linear kernel

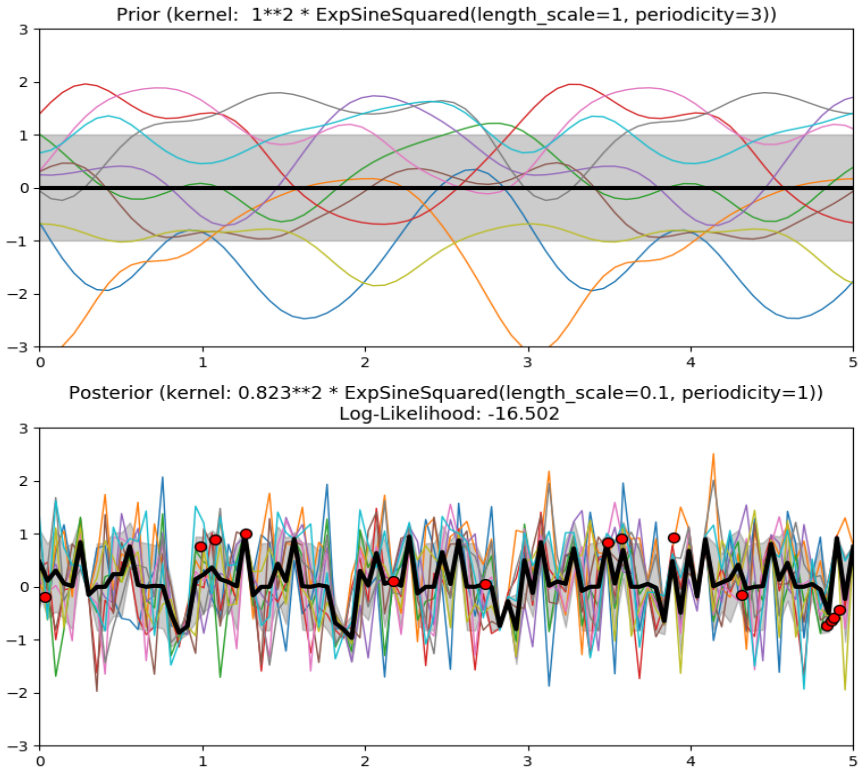

Figure 2.6: GP Prior and Posterior: exponential sine squared kernel

2.2 Deep Gaussian Processes

So far, the architectures considered entailed only one GP (flat GP). However, Damianou and Lawrence (2012) introduced the concept of a deep GP. This introduces complexity to the typical flat model, which naturally allows to represent more interesting phenomena. Not only can the DGP build complex mean functions, but it can also model stochasticity better. While the flat GP considers the uncertainty throughout x roughly the same, the deep one can capture the higher tendency for outliers in some areas, and reduced variances of others.

Put simply, a deep GP is a neural network where the activation functions are Gaussian Processes. This is most easily understood by making the parallel with a neural network in its mathematical form. If a deep neural network is mathematically represented as a composition of functions:

$$g(X) = f_{NL}(f_{NL-1}...(f_2(f_1(X))) \tag{2.30}$$

then, a deep Gaussian Process is represented as a composition of stochastic process:

$$P(Y \mid X) = P(Y \mid f_{NL})P(f_{NL-1} \mid f_{NL-2})...P(f_2 \mid f_1)P(f_1 \mid X) \tag{2.31}$$

Instead of representing a complex function as a composition of simple functions, DGPs represent a complex stochastic process as a composition of simple Gaussian Processes. However, applying a Gaussian Process to another Gaussian Process is certainly not a linear operation, so it is not expected that $P(Y \mid X)$ is still a Gaussian Process - and it is not (Damianou and Lawrence, 2012). Therefore, the training and use of such models is not analytically tractable, which calls for another approximation. The original formulation considered the inducing points inputs approximation (Titsias, 2009).

The inducing points approximation consists of using a smaller set of M "inducing inputs" instead of the full inputs' set X. Such inducing inputs are chosen using Variational Inference. In this approach, Cholesky decompositions are used in order to invert the kernel matrix. Due to their cubic time complexity, these can hinder performance. A Cholesky decomposition assumes that it is possible to find the inverse of a positive definite matrix Mat (symmetric matrix with all positive eigen values (Weisstein, 2018a)) by finding a lower triangular matrix L whose inverse is L^T. Following this assumption, the computations are straightforward, unknown parameters are created for L and then the resulting matrix equation is obtained, which will have $\sum_{i=1}^{M} i$ unknowns and M^2 equations. Of these, $\sum_{i=1}^{M} i$ are linearly independent, which means that by solving for the unknowns, the inverse of Mat is obtained. This results in $\sum_{i=1}^{M} i = M(M+1)/2 = M^2/2 + M$ equations. On the left-hand side (LHS) of these equations, no computations are required, it is simply the value of Mat. However, on the right-hand side (RHS), for each equation a vector product between rows/columns of L and L^T has to be computed, so, it requires M operations. This makes this algorithm $O(M^3)$ in time complexity, which puts an implicit ceiling on the inducing inputs' freedom and consequently limits its scalability.

Other techniques like Expectation Propagation (Bui et al., 2016) and Random Feature Expansions (Cutajar et al., 2016) have also been introduced. The latter was the approach chosen, as it does not employ any Cholesky decomposition. This approximation is explained in the next section.

2.2.1 Random Feature Expansions

Cutajar et al. (2016) introduced Random Feature Expansions (RFE) to each GP of a deep GP network, thereby increasing the scalability of DGPs. In this method, each GP is expanded into a set of functions, which results in a Bayesian neural network which can be trained by stochastic variational inference. This is particularly advantageous when compared to other approaches (like inducing points) because it only requires the computation of matrix products instead of matrix inversion/decomposition. These expansions rely on Bochner's theorem, Fourier Transforms and Monte Carlo sampling (Rahimi and Recht, 2008). The resulting network depends on the kernel of the original GPs. The case of an Radial Basis Function (RBF)/squared exponential kernel will be briefly presented here. If the DGP is a network of GPs, each using an RBF kernel, the parameters to consider are:

- θ^{ℓ}: Set of parameters that characterizes the GP covariances of layer ℓ. It comprises $(\sigma^2)^{\ell}$ and Λ^{ℓ}
- $(\sigma^2)^{\ell}$: Variances of layer ℓ
- Λ^{ℓ}: Lengthscale parameters of layer ℓ that result from using automatic relevance determination in the kernel

According to Bochner's theorem (Edwards, 1979), since the RBF kernel function is a continuous shift-invariant normalized covariance function, if it is positive definite, then it can be written as a Fourier transform of a non-negative measure $P(\omega)$ and vice versa. These ω are typically called spectral frequencies (Cutajar et al., 2016).

Using Bochner's theorem, the kernel is expressed as a Fourier transform and Euler's formula ($e^{ix} = \cos(x) + i\sin(x)$) is applied:

$$k_{RBF}(X, X') = \int P(\omega) exp(i(X - X')^T \omega) d\omega \tag{2.32}$$

$$= \int P(\omega) \left[\cos((X - X')^T \omega) + i\sin((X - X')^T \omega)\right] d\omega \tag{2.33}$$

Where i stands for 'imaginary number' and $P(\omega)$ is a non-negative measure. $P(\omega)$ can be chosen to be $P(\omega) = N(0, I)$.

A quick examination of the LHS combined with the knowledge from the kernel function makes it clear that the LHS has to be real-valued. Looking at the RHS, it also becomes clear that since $P(\omega)$ is non-imaginary, the imaginary part can be dropped from the RHS, thereby resulting in:

$$k_{RBF}(X,X') = \int P(\omega)\cos\left[(X-X')^T\omega\right] d\omega \tag{2.34}$$

The implication of equation 2.34 is that the kernel can be expressed as an expectation over $P(\omega)$, which can be estimated via Monte Carlo sampling. With some algebraic manipulation, equation 2.34 becomes:

$$k_{RBF}(X,X') = \int P(\omega)\left[\cos(X^T\omega)\cos(X'^T\omega) + \sin(X^T\omega)\sin(X'^T\omega)\right] d\omega \tag{2.35}$$

Creating a new function to simplify the notation: $z(x \mid \omega) = \left[\cos(X^T\omega), \sin(X^T\omega)\right]$ and considering the Monte Carlo estimation:

$$k_{RBF}(X,X') \approx \frac{1}{N_{RF}} \sum z(X \mid \omega^{\cdot})^T z(X^T \mid \omega^{\cdot}) \tag{2.36}$$

If N_{RF} tends to infinity, then this representation tends to the original kernel. However, if it does not, this represents something else. Typically, in kernel methods, the 'kernel trick' implicitly projects the objects into a higher dimension and then returns an operation done in that higher dimension whose output belongs to a lower dimension. This comes at the cost of having to evaluate the kernel applied to all pairs of data points. What this approach brings is an explicit mapping of the data to another dimension, which is done with the feature map z - thereby reducing the computational cost (Rahimi and Recht, 2008).

After applying RFE to the DGP, each GP is replaced by a sequence of operations: multiplication by spectral frequencies Ω and multiplication by weights W. It is important to refer how these are linked to the initial parameters (the hyperparameters of each GP).

The term resulting from the multiplication by the spectral frequencies is given by:

$$\Phi_{rbf}^{(\ell)} = \sqrt{\frac{(\sigma^2)^{(\ell)}}{N_{RF}^{(\ell)}}} \left[\cos(F^{(\ell)}\Omega^{(\ell)}), \sin(F^{(\ell)}\Omega^{(\ell)})\right] \tag{2.37}$$

where σ is a parameter from the original network and:

- $N_{RF}^{(\ell)}$: number of random features in layer ℓ
- $F^{(\ell+1)} = \Phi_{rbf}^{(\ell)} W^{(\ell)}$
- $P(\Omega_{.,j}^{(\ell)}) = N(0, (\Lambda^{(\ell)})^{-1})$, evidencing how the spectral frequencies are related to the original model/parameters
- $P(W_{.,i}^{(\ell)}) = N(0, I)$

This new network is graphically represented in Figure 2.7.

Once the network is built, it is just a matter of training it. Notice that the ELBO is the same as in equation 2.22, but now f is the result of a neural network instead of a DGP. To train this network, the chosen method was Stochastic Variational Inference (Hoffman et al., 2013), which is essentially stochastic optimization (Robbins and Monro, 1951) applied to variational inference.

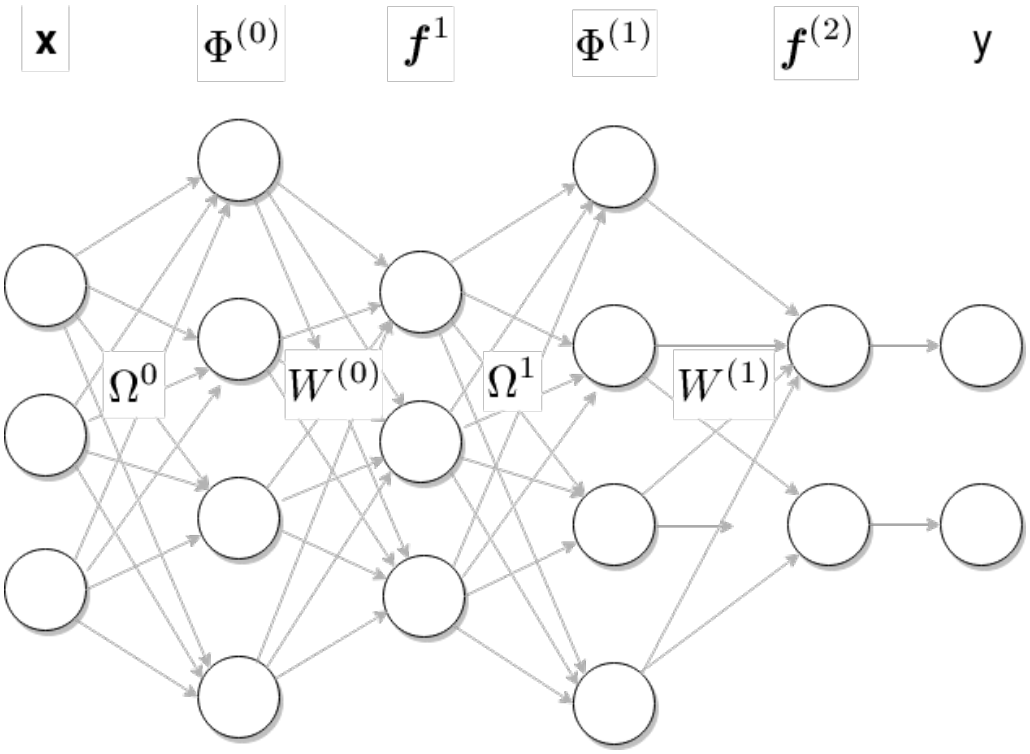

Figure 2.7: Deep Gaussian Process with Random Feature Expansions. Figure reproduced from Cutajar et al. (2016)

Since the objective function (and its gradients) can be decomposed as a sum of independent terms (recall that the L_{ell} decomposes over sequences), this technique is particularly powerful.

However, the gradients of L_{ell} can have considerably high variances, and that hinders optimization (Krauth et al., 2016). In order to reduce the variance of the gradients, the reparameterization trick has been proposed (Kingma and Welling, 2013), and that will also be employed. This consists of reparameterizing the random nodes in the following way:

$$(W^{layer}_{random})_{ij} = \sigma^{layer}_{ij} \varepsilon^{layer}_{random,ij,} + \mu^{layer}_{ij} \tag{2.38}$$

$$\varepsilon^{layer}_{random,ij} \sim N(0,1)$$

where μ^{layer}_{ij} and σ^{layer}_{ij} are variational parameters which are optimized during training. An analogous process is employed for Ω.

Chapter 3

Structured Prediction

This section aims to explain the problem addressed in this thesis - Structured Prediction - and what has been done to solve it. In particular, this thesis deals with discrete output Structured Prediction. Firstly, the problem itself will be defined. Then, popular techniques to solve it will be introduced. At last, work particularly related to this thesis will be presented. This consists of approaches that use Gaussian Processes to solve Structured Prediction problems.

3.1 The problem of Structured Prediction

Structured Prediction occurs when the objects being predicted are structured. Being structured means that observations are not independent between each other. Examples of such objects are sequences, grids, trees, strings, etc. From a more practical perspective, consider word labelling in a sentence, i.e. where each word is given a label ("noun", "verb", "adjective", or "food", "sport", "music" etc., there is considerable freedom for what the labels might be). The label of the 'next word' usually depends not only on its features, but also on the word that comes before it. The main complication with such objects is that likelihoods cannot be factorized over the observations, which implies the elements of a structured object have to be handled simultaneously. For a better grasp of the notation of the problem, consider word classification. For the sentence "Gaussian Processes are the best.", the number of words N is 5. X_i, the input feature matrix, can have features like "starts with capital letter", "number of letters", "is followed by a comma". The output of the model that will allow for classification (y_i) could contain the probability of word i being a verb, a noun, an adjective, or "other". Choosing the features to use is itself a problem and can take practitioners much time.

For a wider grasp of the Structured Prediction problem, it is worth mentioning that this is not limited to sentence analysis and computer vision, although these are typical applications. Another example of a Structured Prediction problem is the prediction of process failure for sequential processes and/or machines, where process can have a broad meaning. If there is a sequence of processes, the failure of one of them is dependent not only on the parameters that define it but also on the adjacent processes.

Regarding the type of structure, there are two, not mutually exclusive, interpretations. The first type is structure between the Y elements. In word classification in a sentence, this materializes as: the classification of word i depends on the classifications of word $i-1$ and/or word $i+1$. This is intuitive - how likely is it that a sentence has many adjectives consecutively? The second type is structure in the Y with respect to the inputs X, or "indexed on X". Using the same example, this materializes as: the classification of word i depends on word $i-1$ and/or word $i+1$ themselves. Bringing another example, some particular verbs might be more likely followed by other verbs than nouns, such as the verbs used in the passive voice, like "has" ("He has done", "She has eaten", etc.).

3.2 Solutions for the Structured Prediction problem

Some of the main techniques for Structured Prediction are: structured perceptron (Collins, 2002), maximum entropy Markov models (McCallum et al., 2000), maximum margin Markov networks (Taskar et al., 2003), structured support vector machines (Tsochantaridis et al., 2005) and conditional random fields (CRFs) (Lafferty et al., 2001). The latter is particularly relevant because it provides likelihood functions that can be generalized for Structured Prediction with GPs (Bratières et al., 2015). Since such likelihoods are used in this thesis, this technique will be explained in more detail.

CRFs can be interpreted as a type of Markov Random Field (MRF) (Murphy, 2013), but they serve a different purpose: Structured Prediction. The key concept with CRFs is that they do not model the distribution over X, $P(X,Y)$, only the one over Y's, $P(Y \mid X)$. For simplicity, suppose elements in both Y and X belong to $\{0,1\}$ (Koller and Friedman, 2009):

$$P(Y \mid X) = \frac{P(Y,X)}{P(X)} \tag{3.1}$$

Since there is no interest in modelling X, an alternative way of computing $P(Y \mid X)$ has to be found. The solution is the introduction of potentials $\hat{P}$ instead of probabilities P, which are then normalized by a factor Z instead of $P(X)$.

$$P(Y \mid X) = \frac{\hat{P}(Y,X)}{Z(X)} \tag{3.2}$$

These potentials are formulas to evaluate the 'desirability' of Y to X. Since they do not necessarily sum to one, they do not satisfy the probability axioms (Weisstein, 2018b) and are therefore called "potentials". These potentials are computed by the product of a set of factors in $\Phi = \{\phi_1(D_1), ..., \phi_k(D_k)\}$, each with a certain scope D_i:

$$\hat{P}(X,Y) = \prod_{i=1}^{k} \phi_i(D_i) \tag{3.3}$$

The normalization constant $Z(X)$ is naturally:

$$Z_\Phi(X) = \sum_Y \hat{P}(X,Y) \tag{3.4}$$

Having defined a framework, the specific factors that will be used should be defined. The simple case of binary classification will be used as a base case. A possibility for the factors is just a log-linear combination of features that have an effect on the dependent variable:

$$\phi_i(x_i,y) = \exp\{w_i 1\{x_i = 1, y = 1\}\} \tag{3.5}$$

where 1 is an indicator function (it is 1 if the condition in brackets is true, and 0 otherwise). So, looking at $P(y \mid x_i)$, if $y = 1$, then $\phi_i(x_i, y = 1) = \exp(w_i x_i)$, and if $y = 0$, $\phi_i(x_i, y = 0) = 1$, which leads to $\hat{P}(X, y = 1) = \exp\{\sum_i (w_i x_i)\}$ and $P(X, y = 0) = 1$. Summing these two to get the normalizing $Z(X)$:

$$P(y = 1 \mid X) = \frac{\exp\{\sum_i (w_i x_i)\}}{1 + \exp\{\sum_i (w_i x_i)\}} = \frac{P(y = 1, X)}{P(y = 1, X) + P(y = 0, X)} \tag{3.6}$$

Which means a certain classification (word label) depends on every X element (each word of the sentence). This simple case can be generalized for higher dimensions, for example by doing binary classification for every label.

Note that the argument of the numerator, $w_i x_i$, was not an analytic implication, it was chosen. Therefore, other functions or more complicated objects could have been chosen. This is precisely what was done by Bratières et al. (2015) for the likelihood function where, instead of considering a parametric, log-linear combination of weighted features, it considers a log-linear combination of GPs with kernels evaluated between cliques, instead of inputs X. A clique is a subset of vertices of an undirected graph such that every two vertices are adjacent. This allows to represent the relation between observations, something that is explained in more detail in the next section.

$$P(Y|X, f) = \frac{\exp(\sum_c f(c, x_c, Y_c))}{\sum_{Y' \in Y} \exp(\sum_c f(c, x_c, Y'_c))} \tag{3.7}$$

3.3 Gaussian Processes for Structured Prediction

Structured Prediction can deal with continuous or discrete outputs. For the problem of structured continuous-output, Twin Gaussian Processes (Bo and Sminchisescu, 2010) propose considering input kernels similar to output kernels. This problem is related to the well developed area of multi-output regression (Bonilla et al., 2008; Álvarez et al., 2010; Álvarez and Lawrence, 2011). However, the problem addressed in this thesis, structured discrete-output, deals with considerably more complex likelihood functions which are considerably expensive to compute throughout training (Galliani et al., 2016). For that reason, different techniques have to be used. CRF-based techniques have a long history of success in these problems. They have been combined with kernel methods (Lafferty et al., 2004), they have been given a Bayesian treatment (Qi et al., 2005), and,

taking advantage of the latter two, more recently Bratières et al. (2015) proposed the GPstruct. The GPstruct considers an MRF likelihood function represented in 3.8.

$$P(Y|X,f) = \frac{\exp(\sum_c f(c,X_c,Y_c))}{\sum_{y' \varepsilon Y} \exp(\sum_c f(c,X_c,y'_c))} \tag{3.8}$$

Where the hidden functions, f, are generically defined as $f \sim GP(0,k(c,c'))$, that is, a GP evaluated between cliques - a clique is a set of nodes such that every two are connected, following undirected graphical models' nomenclature. More practically, the f considered encompasses unary and binary cliques, responsible for the unstructured and structured parts of classification, respectively. The details of the elements that compose f, f_{un} and f_{bin} are presented below:

- f^j_{un} is drawn from a GP and is the vector of unary functions of hidden function j, corresponding to the j^{th} label in the vocabulary:

$$f_{un} \sim GP(0,k(x,x')) \tag{3.9}$$

- f_{bin} is a finite Gaussian (not indexed by X) and introduces the desirability of a certain label being followed by another one. K_{bin} is of dimensions $|v| \times |v|$. This follows the unindexed interpretation of data structuredness defined in section 3.1

$$f_{bin} \sim N(0,K_{bin}) \tag{3.10}$$

- These two hidden functions are considered simultaneously by:

$$p(f) = p(f_{un})p(f_{bin}) = \prod_{j=1}^{|v|} N(f^j_{un};0,K_j)N(f_{bin};0,K_{bin}) \tag{3.11}$$

This is essentially a CRF with a more complex f, a GP and a multivariate Gaussian. In order to compute this likelihood, a forwards-backwards belief propagation algorithm is employed. In order to do inference, Bratières et al. (2015) uses Elliptical Slice Sampling (ESS), which is a Markov Chain Monte Carlo (MCMC) procedure. Although the results are promising on text and video processing tasks, this procedure is not scalable to large datasets and takes advantage of the details of the likelihood considered (Galliani et al., 2016), limiting the model's freedom. Bratieres et al. (2014) also proposed a scalable version of the GPstruct based on ensemble learning, but only presented results for 2-dimensional problems (grids) and not for the standard 1-dimensional benchmarks used by Bratières et al. (2015) and Galliani et al. (2016), so these cannot be used in this thesis for comparison purposes.

Galliani et al. (2016) showed that the GPstruct is actually a subset of a different model that is amenable to Stochastic Variational Inference. This means that by using an alternative formulation of the GPstruct, the likelihood dependence and scalability problems are overcome. This approach is called "gray-box" because it does not require the details of the "structured likelihood" model (it is not called "black-box" because it might require human intervention for some general Structured Prediction problems). The proof of concept is done for linear-chain likelihoods (1 dimensional structures like sentences), as in this thesis.

These are successful applications of GPs to Structured Prediction, but there are at least two roads that can be further explored: the complexity of the unary functions can be increased, via a DGP for example, and the complexity of the binary functions can also be increased, by considering them Gaussian Processes (or even DGPs) indexed on the inputs (X). This thesis looks into that possibility: models for Structured Prediction with more complex priors.

3.4 Performance criteria

3.4.1 Test likelihood

Considering the models presented are probabilistic, they should predict correct classifications with high probabilities. Therefore, the first metric to consider is the "test likelihood", which is broadly defined as the probability of the model making good predictions, thereby representing the so called "uncertainty quantification". There are (at least) two ways of considering this "test likelihood". The simplest one, is referred to as "unstructured test likelihood":

$$L_{unstruct} = \prod_{i=1}^{N} P(Y_i^{test}) \tag{3.12}$$

where N is the number of test observations and $P(Y_i)$ is the probability of the model making a correct prediction for the i-th word. This is called "unstructured" because it does not consider the structuredness of data. What formula 3.12 implies is that the probability of the test data occurring is the probability of each observation occurring, independently of each other.

Considering this thesis deals with structured datasets, and there are likelihood functions that can accommodate this structure, it might make more sense to consider the "structured test likelihood":

$$L_{struct} = \prod_{i=1}^{Nseq} P(Y_{i,1}^{test}, Y_{i,2}^{test}, ..., Y_{i,T_i}^{test}) \tag{3.13}$$

where $Y_{i,n}$ is the n-th word of the i-th sentence and T_i is the number of words of the i-th sentence.

$P(Y_{n,1}^{test}, Y_{n,2}^{test}, ..., Y_{n,T_n}^{test})$ is computed using a specific structured likelihood function like the one in equation 3.8. Edunov et al. (2017) proposed a variety of loss functions, one of which is essentially the L_{struct}, although it is named Seq_{NLL}. Using the terminology in (Edunov et al., 2017), the difference between $L_{unstruct}$ and L_{struct} is whether there is a token-level or a sequence-level focus. This second metric definitely makes sense from a theoretical standpoint, but since the previously reported results involve only $L_{unstruct}$, that will be the one used for model comparison purposes.

3.4.2 Error rate

The error rate is the standard metric for classification problems. What it represents is straightforward, and although probabilistic models do not technically optimize for this, its interpretability makes it a strong metric to use. Additionally, this metric lends itself to any type of Machine

Learning technique (like the non-probabilistic, non-Bayesian alternatives), therefore it provides the benefit of comparability with other models.

In probabilistic models, in order to compute this error rate a zero-one loss scheme is usually employed. Zero-one loss considers that the model's final prediction for each observaation is the class which is predicted with the highest probability, and then evaluates how many are correct ("one") and how many are wrong ("zero"). This is represented as:

$$L(y_{real}, y_{pred}) = I(y_{real} \neq y_{pred})$$

Where $I(a)$ is the indicator function (it equals 1 if a is True and 0 if a is False). The final Error Rate will be the fraction of incorrect predictions from all of the test data.

In this thesis, the performance metrics have been implemented in 5 folds and the reported values are an average across them.

Chapter 4

Models

Two models are proposed in this chapter, the SDGP1 and the SDGP2. These build upon the work done by Galliani et al. (2016) which was explained in section 3.3, but there are three key differences between them:

1. The original full GPs are approximated using Random Feature Expansions
2. The hidden functions can be deep - that is, compositions of GPs
3. Interactions between observations can be more complex (SDGP2 case)

The models presented in section 4.1 consider the original full GPs, that is, GPs without approximations, and the models in section 4.3 are approximated using Random Feature Expansions. These are referred to as "full" and "approximate" models, respectively. While sections 4.1 and 4.3 illustrate how models are used to make predictions, sections 4.2 and 4.4 illustrate how the models are trained. Section 4.5 includes details regarding the implementation of both the training and predicting procedures.

4.1 Full Structured Deep Gaussian Processes

The objective of the model is to make predictions in Y for new observations using a deep network of Gaussian Processes. However, the actual function used to make predictions is the likelihood function: $P(Y_* \mid f)$, which depends on the deep network. A high level view of the model is represented in Figure 4.1 to clarify how these variables and functions interact.

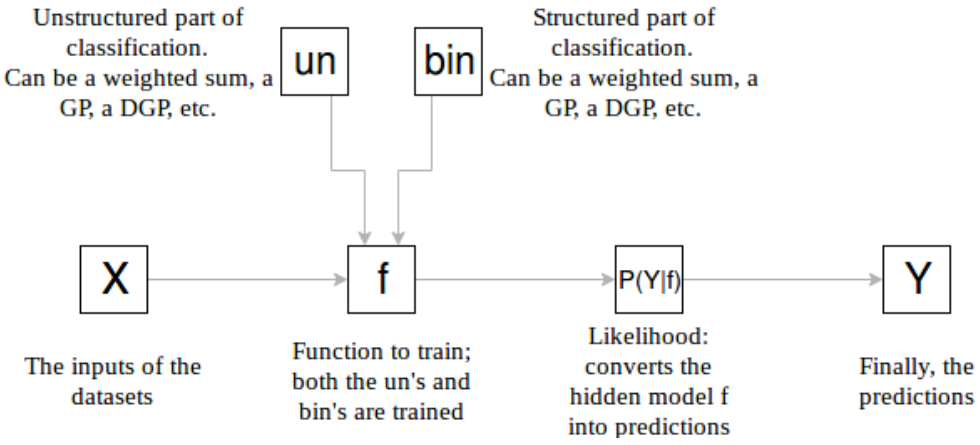

Figure 4.1: SDGP: a high level view

For a better understanding of the models proposed, the following considerations should be regarded:

- Since sequences are considered independent, the likelihood should factorize over sequences: $P(Y_* \mid f) = \prod_{j=1}^{Nseq} P(Y_j \mid f_j)$
- For black-box likelihood models, the key element that differentiates models is f
- f has 2 components, f_{un} and f_{bin}, responsible for individual and pairwise components of classification, respectively
- f has dimensions $N \times |v|$, as it takes an input of N observations X, and outputs, for each of them, the "desirability" of each label. $|v|$ stands for the number of labels. These will be converted into probabilities by the likelihood function.
- f_{un} and f_{bin} can be computed in a variety of ways. This thesis considers 2 possibilities, resulting in 2 models: the SDGP1 and SDGP2. Both consider $f_{un}(x_i)$ a vector with the desirability of each possible label (output) to input x_i: $f_{un}(x_i) = (\tilde{p}_1, \tilde{p}_2, ..., \tilde{p}_{|v|})$, but they handle f_{bin} differently.

The next subsections explain how models SDGP1 and SDGP2 handle f_{un} and f_{bin}.

4.1.1 SDGP1

In the SDGP1, f_{un} follows a DGP and f_{bin} follows a multivariate Gaussian:

$$f_{un} \sim DGP(0, K_{un}) \Leftrightarrow f_{un} = (f_{un}^{Nh} \circ f_{un}^{Nh-1} \circ \ldots \circ f_{un}^{1})(X) \tag{4.1}$$

$$f_{bin} \sim N(0, K_{bin}) \tag{4.2}$$

The relations between the inputs and outputs are represented in Figure 4.2. Each element y_i depends not only on x_i (by means of f_{un}) but also on the adjacent elements y_{i+1} and y_{i-1} (by means

of f_{bin}). For simplicity, it was considered that there was only one GP per unary layer. However, the number of parallel GPs per layer can be another parameter to choose.

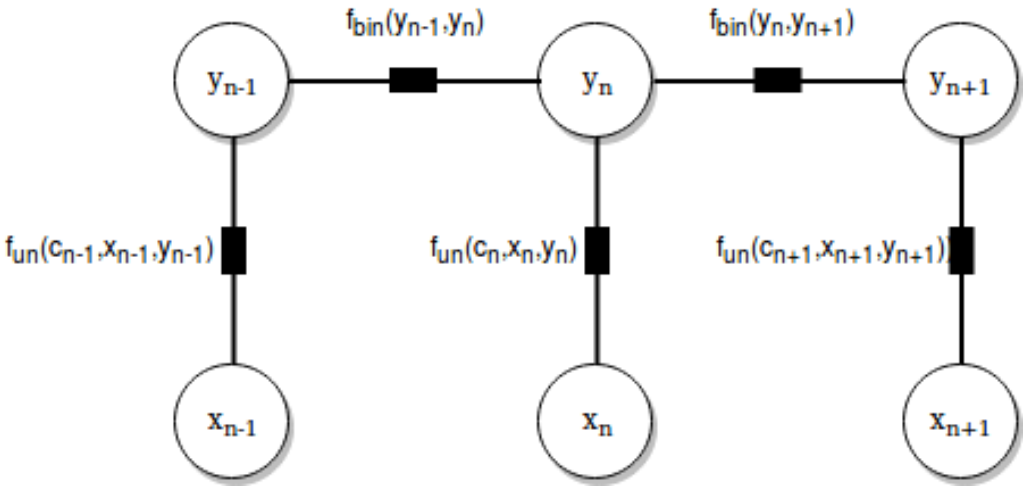

Figure 4.2: Undirected graph for Structured Prediction (SDGP1). Figure reproduced from Bratières et al. (2015)

Figure 4.3 considers the same model (SDGP1) but makes the depth of the unary functions explicit. This representation considers that the input variables are fed to every layer, an approach called "feedforward".

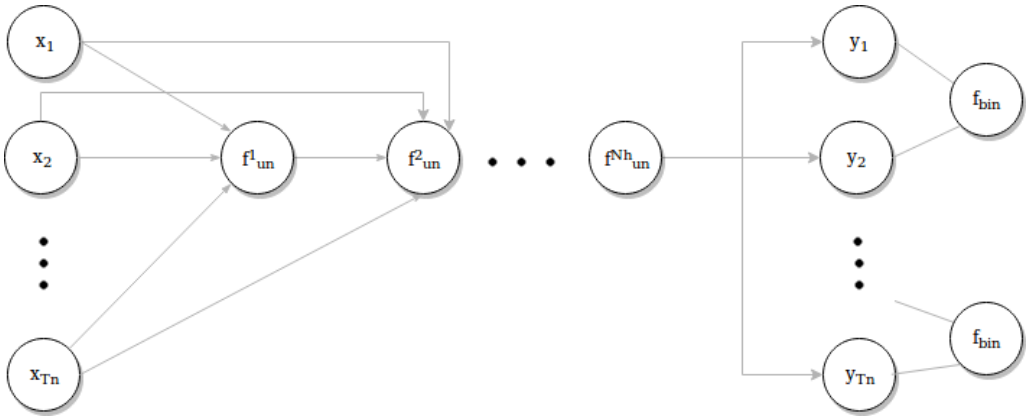

Figure 4.3: Representation of the SDGP1 as a neural network

The implications of equations 4.1 and 4.2 on how the prior probability is computed are:

$$P(f) = P(f_{un})P(f_{bin}) = \prod_{j=1}^{|v|} DGP(0, K_{un,j})N(0, K_{bin}) \tag{4.3}$$

Notice that while a GP becomes a multivariate Gaussian distribution after conditioning on X, this is not (necessarily) the case with a DGP. For that reason, the notation was kept with "DGP", although this is after "seeing" the inputs. The kernels considered here are analogous to the ones

considered by Bratières et al. (2015). Since the unary functions are independent of the binary ones, the kernels are block diagonal matrices.

$$K = Cov(f) = \begin{bmatrix} K_{un} & 0 \\ 0 & K_{bin} \end{bmatrix} \tag{4.4}$$

Additionally, each sequence is independent of each other, so K_{un} is also block diagonal.

$$K_{un} = \begin{bmatrix} K_{un,1} & 0 & \dots & \dots & 0 \\ 0 & K_{un,2} & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & K_{un,|v|-1} & 0 \\ 0 & 0 & \dots & 0 & K_{un,|v|} \end{bmatrix} \tag{4.5}$$

$K_{un,j}$ is the $N \times N$ covariance matrix that is generated by the kernel function after evaluating all the observations (X) with regards to label j.

$$K_{un,i} = \begin{bmatrix} K_{un,i,1,1} & K_{i,2,1} & \dots & K_{un,i,N-1,1} & K_{un,i,N,1} \\ K_{un,i,1,2} & K_{un,i,2,2} & \dots & K_{un,i,N-1,2} & K_{un,i,N,2} \\ \dots & \dots & \dots & \dots & \dots \\ K_{un,i,1,N-1} & K_{un,i,2,N-1} & \dots & K_{un,i,N-1,N-1} & K_{un,i,N,N-1} \\ K_{un,i,1,N} & K_{un,i,2,N} & \dots & K_{un,i,N-1,N} & K_{un,i,N,N} \end{bmatrix} \tag{4.6}$$

After choosing the desired kernel function (squared exponential, arccosine, linear, etc.), it is possible to compute the kernel matrices. As a result, it is possible to compute f, which can feed the likelihood function. This, in turn, will allow to produce predictions.

4.1.2 SDGP2: The unary and binary functions follow independent DGPs.

In this case, f_{bin} is no longer a multivariate Gaussian which relies only on adjacent classifications, it is an indexed function which relies on the inputs, X. The network architecture is represented in Figure 4.4.

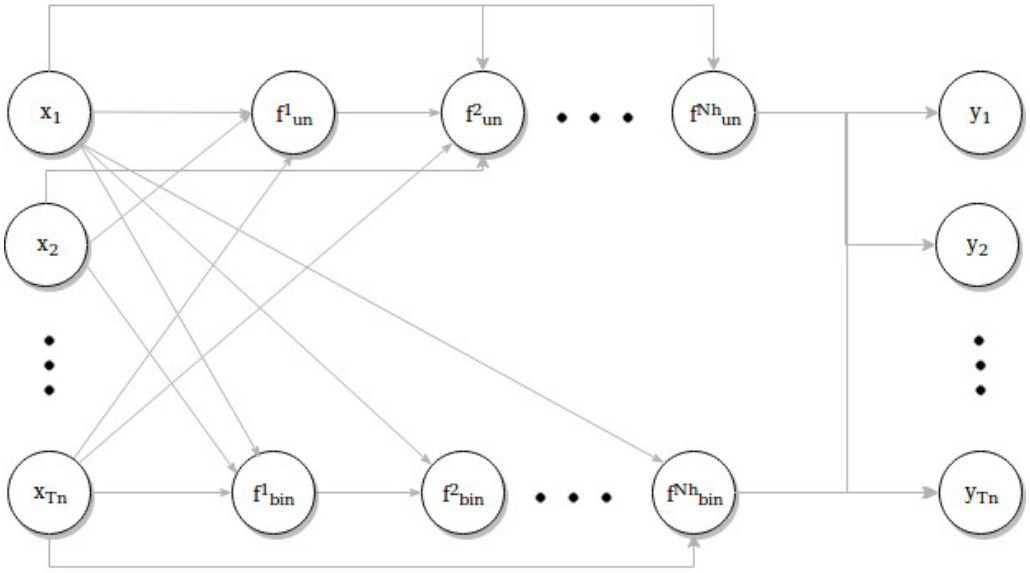

Figure 4.4: Representation of the SDGP2 as a neural network

Both f_{un} and f_{bin} follow Deep Gaussian Process distributions:

$$f_{un} \sim DGP(0, K_{un}) \Leftrightarrow f_{un} = (f_{un}^{N_h} \circ f_{un}^{N_h-1} \circ \ldots \circ f_{un}^{1})(X) \tag{4.7}$$

$$f_{bin} \sim DGP(0, K_{bin}) \Leftrightarrow f_{bin} = (f_{bin}^{N_h} \circ f_{bin}^{N_h-1} \circ \ldots \circ f_{bin}^{1})(X) \tag{4.8}$$

In order to make predictions Y_* for new observations, it is necessary to estimate the covariance parameters of each layer $\theta_{GP\#}^{layer\#}$, something that happens in both the SDGP1 and SDGP2. To do that, the marginal likelihood is maximized with respect to the covariance parameters and network weights. To compute the marginal likelihood, the integral in equation 4.9 has to be computed.

$$P(y \mid X, \theta) = \int P(Y \mid f^{N_h}) P(f^{N_h} \mid f^{N_h-1}, \theta^{N_h-1}) * \ldots * P(f^1 \mid X, \theta^0) df^{N_h} \ldots df^1 \tag{4.9}$$

Since the DGP is not necessarily a GP (or any other tractable stochastic process), the integral in equation 4.9 is intractable. Therefore, in order to train and use this model, a variational distribution will be introduced. Note that since this variational approximation is introduced in order to conduct inference, the model itself is still considered "full".

A third possible architecture would be for f_{un} and f_{bin} to share a DGP, a concept analogous to 2 GPs "sharing a kernel", but this is left for future work.

4.2 Variational Inference and Learning

The main objective of training is estimating (inferring) the parameters of the DGP, that is, the parameters characterizing every GP present. As mentioned before, the learning scheme for this model will be Stochastic Variational Inference, which will approximate $P(f \mid Y)$ with $q(f)$. This

method suits objective functions which can be decomposed as a sum of factors, which is the case of the ELBO, as is evidenced in equation 4.10.

$$ELBO = L_{ell} - KL[q(f)||P(f)] \tag{4.10}$$

This objective function, the ELBO, has an expected likelihood term and a KL-divergence term, just like the unstructured case presented in section 2.2. The only difference is that f now has of two components, a structured and an unstructured one. The KL term can be computed analytically if q and P follow Gaussian distributions (see Appendix A.3). The L_{ell} term, however, has to be estimated. Leveraging the work done by Galliani et al. (2016), an estimate of L_{ell} and of its gradients can be computed.

$$L_{ell} = \sum_{n=1}^{Nseq} L_{ell}^n = \sum_{n=1}^{Nseq} \langle \log P(Y^n | f_{un}^n, f_{bin}^n, \Theta, \Omega) \rangle_{q^n(f_{un}^n) q^n(f_{bin}^n)} \tag{4.11}$$

$$\nabla \lambda_{un} L_{ell}^n = \langle \nabla_{\lambda_{un}} \log q^n(f_{un}^n) \log P(Y^n | f_{un}^n, f_{bin}^n, \Theta, \Omega) \rangle_{q^n(f_{un}^n) q^n(f_{bin}^n)} \tag{4.12}$$

$$\nabla \lambda_{bin} L_{ell}^n = \langle \nabla_{\lambda_{bin}} \log q^n(f^{bin,n}) \log P(Y^n | f_{un}^n, f_{bin}^n, \Theta, \Omega) \rangle_{q^n(f_{un}^n) q^n(f_{bin}^n)} \tag{4.13}$$

Where f^n represents the hidden functions evaluated at sequence n and $\langle \bullet \rangle_q$ represents an expectation over q. The expectations in equations 4.12 and 4.13 are computed with respect to the approximate marginal posterior $q(f^n) = q^n(f_{un}^n) q^n(f_{bin}^n)$.

The Variational Inference strategy is straightforward for flat GPs, but when considering a deep structure, key elements of the procedure break down, like considering q to be a Gaussian distribution (q has to approximate a DGP structure, therefore it cannot simply be a Gaussian). Therefore, this model and gradient estimates are not the ones used in this thesis. In order to actually employ the model, another approximation will be introduced - Random Feature Expansion (Rahimi and Recht, 2008), as well as the reparameterization trick, as mentioned in Chapter 2.

4.3 Approximate Model - Random Feature Expansions

After introducing Random Feature Expansions, the model becomes approximate - it is not a DGP anymore. It is a Bayesian neural network, which is defined as a neural network with a prior distribution on its weights (Neal, 1996). This is the model that this thesis actually implemented. However, since the neural network is derived from the SDGP, there is a correspondence between the network parameters and the original ones. For an RBF kernel, this correspondence is evidenced in equations 4.14 and 4.15.

$$\Phi_{rbf}^{(l)} = \sqrt{\frac{(\sigma^2)^{(l)}}{N_{RF}^{(l)}}} [\cos(f^{(l)} \Omega^{(l)}), \sin(f^{(l)} \Omega^{(l)})] \tag{4.14}$$

$$f^{(l+1)} = \Phi_{rbf}^{(l)} W^{(l)} \tag{4.15}$$

The priors over the spectral frequencies and weights are respectively $P(\Omega_{.j}^{(l)}) = N(0, (\Lambda^{(l)})^{-1})$ and $P(W_{.i}^{(l)}) = N(0, I)$. Each of the matrices $\Omega^{(l)}$ has dimensions $D_{F^{(l)}} \times N_{RF}^{(l)}$ and $W^{(l)}$ has dimensions $2N_{RF}^{(l)} \times D_{F^{(l+1)}}$.

4.4 Approximate Model - Inference and Learning

The learning scheme is analogous to the full model, but now the hidden variables are Ω and W instead of f. There is, however, one relevant nuance to consider. Since now there are two sets of hidden variables, Ω and W, these will be handled one at a time, starting with W. This means that at first Ω will be simply drawn from a prior, in an approach named "prior-fixed". However, Ω could easily be variationally learnt alongside W. If Ω is resampled every time, the approach is called "var-resampled", if it is sampled once and then the random parameter (see section 2.2.1) is kept constant throughout training, the approach is called "var-fixed".

Considering, therefore, an analogous process to the full Gaussian case, the posterior over the hidden variables W has to computed: $P(W \mid Y, X, \Omega, \theta)$. In order to do that, some characteristics of the model should be highlighted.

- The layers are considered independent, that is, $P(W_{un}) = \prod_{l=0}^{Nh,un-1} P(W^{un,l})$ and the same for W_{bin}
- W_{un}, W_{bin} behave just like f_{un}, f_{bin}, they are the new hidden function parameters.
- $q(W)$ is a variational approximation of the posterior and is defined by $q(W) = q(W_{un})q(W_{bin}) = \prod_{i,j,l} N(m_{un}^{i,j,l}, (s^2)_{un}^{i,j,l}) \prod_{i,j,l} N(m_{bin}^{i,j,l}, (s^2)_{bin}^{i,j,l})$ where $m^{\cdot,\cdot,\cdot}$ and $(s^2)^{\cdot,\cdot,\cdot}$ are the variational parameters, and i represents the node from which W "departs" and j represents the node where W "arrives".

Regarding the training scheme, the objective function to consider (ELBO) will be different depending on the architecture considered. A derivation for the new ELBO will be done for the most complex case, the deep model with indexed pairwise variables (SDGP2). This derivation will follow the "prior-fixed" approach, that is, it will consider that the posterior only regards W and not Ω. The ELBO for the other approaches (var-fixed and var-resampled) and for SDGP1 can be easily derived based on this derivation, simply by considering whatever hidden functions are being considering instead of W.

At first, the "spectral frequencies", Ω, will be considered fixed, in an approach called "prior fixed". However, these can also be learnt variationally, just like W.

$$\log P(Y \mid X, \Omega, \Theta) = \log \int P(Y \mid X, W, \Omega, \Theta) P(W) dW$$

$$= \log \int \frac{P(Y \mid X, W, \Omega, \Theta) P(W)}{q(W)} q(W) dW$$

$$= \log E_{q(W)} \frac{P(Y \mid X, W, \Omega, \Theta) P(W)}{q(W)}$$

$$\geq E_{q(W)} \left(\log \frac{P(Y \mid X, W, \Omega, \Theta) P(W)}{q(W)} \right)$$

$$= E_{q(W)} \left(\log P(Y \mid X, W, \Omega, \Theta)]) + E_{q(W)} \log \frac{P(W)}{q(W)} \right)$$

$$= E_{q(W)} \left(\log P(Y \mid X, W, \Omega, \Theta) \right) - KL\left[q(W) || P(W) \right])$$

$$= L_{ell} - KL[q(W)||P(W)] \tag{4.16}$$

The variational distribution is now on the new hidden functions, or, using neural networks' terminology, "hidden layers". In equation 4.16, $q(W) = q(W_{un}) q(W_{bin})$ and $P(W) = P(W_{un}) P(W_{bin})$, so the ELBO can be further decomposed:

$$L_{ell} - KL[q(W_{un})||P(W_{un})] - KL[q(W_{bin})||P(W_{bin})] \tag{4.17}$$

where L_{ell} is computed using a conditional likelihood function (which can be specified by the user, but the one used here is the one provided by Bratières et al. (2015)), and W is computed using the reparameterization trick. Such an objective function lends itself to Stochastic Variational Inference perfectly, as it is a sum of two independent terms, and one of them (L_{ell} term) is a sum of other independent terms itself, as the likelihood parallelizes over sequences.

4.5 Implementation details

The code was implemented in the high-level programming language Python. It makes extensive use of the Tensorflow framework by Abadi et al. (2016), which is a symbolic math library developed by the Google Brain team. This framework allows to build neural networks as computational graphs, which allows for significant computational improvements. Furthermore, this library includes an Automatic Differentiation tool which allows to do optimization simply by providing a function and its parameters. For this thesis, the optimization process or neural network training has been done by feeding the ELBO term to Tensorflow"s Automatic Differentiation scheme. This means that the gradients of the ELBO that have been computed in section 4.4 were not actually hard coded in the model.

The code for this thesis builds upon the code built by Cutajar et al. (2016) and Galliani et al. (2016). The starting point was the former, which had been built considering only "unary" functions and therefore for unstructured classification. A series of changes had to be introduced:

- Processing sequences instead of observations: The first change is the processing of the data. Instead of handling observations individually, they have to be considered as sequences. The datasets cannot be broken apart arbitrarily anymore for mini-batch optimization, sequences must be preserved. The L_{ell} term will be built as a sum of sequential likelihoods (seq_{ell}), and the sequence order is shuffled in every epoch. Galliani et al. (2016) had stored all the matrices in Sparse Tensors (object type in Tensorflow), but this thesis worked with dense tensors.

- Creating a new KL divergence: The KL divergence that the SDGP model considers is different than the ones previously considered. This introduction will affect the objective function (ELBO).

- Creating a new objective function: Although the objective function is the ELBO, since the optimizer does minimization, the objective function will be the ELBO's negative version: $NELBO = -ELBO = KL - L_{ell}$. To quickly evaluate if the optimization is working properly, one should see if the NELBO is decreasing. To identify the root cause of any problem with the ELBO, it is simple to evaluate which term, the KL or the $NLL = -L_{ell}$ is responsible.

- Measuring uncertainty quantification: Since the NLL is already being reported for the aforementioned reasons, the uncertainty quantification measure (test likelihood) will be shown in its NLL form, that is, instead of reporting the test likelihood itself, this implementation reports the negative of its logarithm.

- Order of model implementation: The SDGP models introduce many changes upon the unstructured DGP (Cutajar et al., 2016). For that reason, such changes will be introduced one at a time, to evaluate their impact. Firstly, the f_{bin} will be introduced and only one layer will be considered. This will evaluate how good the Random Features approximation is compared to the inducing inputs one (Galliani et al., 2016). Then, this network will be made deep, in order to evaluate how the additional freedom of a deep network materializes in performance. At last, the indexing on X will be introduced in order to compare how this type of indexing improves the performance compared with the Y-based indexing.

4.5.1 Unindexed hidden functions (SDGP1)

In this case, f_{bin} is introduced "between the Y"s". f_{bin} is a $|v|x|v|$ matrix with the desirabilities of adjacent label combinations, and it is obtained from a multivariate Gaussian. The parameters of this Gaussian are set to "trainable". This is expected to improve the quality of predictions, which will be seen in the increase of the L_{ell} term, and the penalization comes in the form of an additional KL loss. This new KL loss between f_{bin} and its variational posterior is computed using the source function provided by Galliani et al. (2016).

Additionally, the memory problems caused by large datasets should be addressed. While the flat SDGP1 ran smoothly, when the depth was increased and the feedforward introduced, each node went from a few dimensions to thousands due to the large dimensionality of the inputs. For

that reason, a good solution was to reduce the dimensionality of the data. The dimensionality was reduced from 2×10^6 to 100, by multiplying the input data by a weights matrix, and optimizing the weights: $X_{reduced} = X_{original} M_{reducer}$, where $X_{original}$ is the original input matrix of dimensions $N \times D$, $M_{reducer}$ is a matrix of variational parameters of dimensions $D \times 100$ and $X_{reduced}$ is the reduced inputs matrix, with, consequently, dimensions $N \times 100$. This reduction solves the memory issues, but the freedom of fit is naturally affected, and this can introduce overfitting in the model.

4.5.2 Indexed hidden functions (SDGP2)

This presents a more complex case, as there is a considerable increase in the number of hidden functions. The new hidden pairwise functions (W and Ω) have to be created in parallel to the previous unary ones, and these will have different dimensions, as f_{bin} and f_{un} also have different dimensions. Although f_{bin} is the result of a neural network, in the end its dimensions will be $T^n \times |v|^2$, which are then reshaped to $T^n \times |v| \times |v|$. The ELBO will then have to include all this.

Regarding the KL term, since f_{bin} is of the same type as f_{un} and they are parallel (and therefore independent), the same function used to compute the KL loss of f_{un} can be used for f_{bin}. The difference is in the input and output dimensions but this does not pose a problem.

Regarding the L_{ell} term, there is now a problem, since there is no likelihood function available that accommodates an f_{bin} whose dimensions are $T^n \times |v| \times |v|$. The original structured likelihood provided by Bratières et al. (2015) had to be altered into a new "indexed structured likelihood", which, for every element X conditions on that element and considers only the portion of f_{bin} of size $|v| \times |v|$ that corresponds to that X.

After introducing these changes in the network, the KL computation, the likelihood (which affects predictions and the L_{ell} term), the model is ready to be trained and used.

4.5.3 Hyperparameters

Most model hyperparameters to consider have already been shown in the model formulation: number of hidden layers, number of parallel GPs per layer, the kernel in the original GPs, learning strategy for the Ω (prior-fixed, var-fixed, var-resampled), the number of random features in the expansion, the use of Automatic Relevance Determination (or not), the number of Monte Carlo samples in train and test, learning rate, the optimizer that Tensorflow should use and the duration of the experiment (in time or epochs). However, there are 3 particular situations related to the implementation itself that require an additional explanation.

In the original implementation (Cutajar et al., 2016), the network parameters (Θ) are not initialized as trainable variables, as that would be unstable in terms of optimization. Instead, they are set to fixed for a number of initial epochs. Additionally, although the number of Monte Carlo (MC) samples to use in training is an input for the user (and hyperparameter), only one MC sample is considered during the first half of the experiment duration. Both strategies are kept in this thesis.

Secondly, for large datasets it is not computationally feasible to compute the true likelihood function (Bratières et al., 2015). For that reason, an approximation of the likelihood called Piecewise Pseudolikelihood (Sutton and McCallum, 2007) has been implemented, thereby making it possible to run this model on large datasets.

A more detailed explanation of all the hyperparameters can be found in the Appendix A.4.

Chapter 5

Results and discussion

As mentioned in chapter 4, two models have been introduced, the SDGP1 and the SDGP2. As mentioned in the same chapter, there are three key factors that these models bring, and the objective of this section is to evaluate the impact of them both individually and combined. For the SDGP1, two different architectures were used (flat and deep) and for the SDGP2, only the flat case was analyzed.
Regarding the term of comparison for the SDGPs models, the main goal is to beat the state-of-the-art models. However, since there are many ways of looking at these models, there can be many terms of comparison. For a better and holistic understanding of the functioning of SDGPs models, these should be considered. SDGPs can be thought of as multi-label classification models that bring structure to deep models, or multi-label classification models that bring depth to structured models, so all those perspectives should be analyzed. The baselines considered and the corresponding expected performances are:

- Standard unstructured multi-label classification technique (Multinomial Logistic Regression): This is expected to have the worst performance from all the models, as it is the most basic technique and does not account for structure. The SDGP is expected to beat this benchmark.

- Unstructured DGPs (Cutajar et al., 2016): the SDGP brings structure to such models and is therefore expected to beat this benchmark.

- State-of-the-art structured multi-label classification technique (CRF): the SDGP is, in a way, a complex CRF, so it is expected to beat this benchmark.

- Structured flat GP (Bratières et al., 2015; Galliani et al., 2016): *ceteris paribus*, that is, if the only change was introducing depth, then the SDGP would be expected to beat non-deep structures, at least for appropriate amounts of data. However, there are many intricacies that break the *ceteris paribus* assumption. In this field, these models, alongside CRFs, represent the best that has been achieved in performance (in the datasets used for model evaluation).

In order to compare these models, it is important that the performance metrics are always computed in the same way. For every model, the datasets have been divided into 5 folds and an average over the performance metrics in each one of them has been computed. These performance metrics are computed after 4 hours of training for each fold, following the approach done by Galliani et al. (2016).
The rest of the chapter contains 3 sections:

1. a description of the datasets considered
2. how the hyperparameter tuning was performed - a look into model training
3. an evaluation and comparison between the aforementioned models and the SDGPs - a look into model testing

5.1 Datasets

The purpose of these experiments is to analyze whether SDGPs can compete with the state-of-the-art techniques. For that reason, the benchmarks used were the same as used by Bratières et al. (2015) and Galliani et al. (2016). These represent typical NLP problems and the source is the CRF++ toolbox [1]. In particular, the datasets are: Base NP, Chunking, Segmentation and Japanese NE. A description is presented below (Srijith et al., 2014).

Base NP is a noun phrase identification problem. A noun phrase consists of a noun (or pronoun) and the modifiers that affect it. For example: "the beautiful person" or "that man over there" are noun phrases. There are 3 labels for words, depending on whether they are the first word of a noun phrase, a non-first word of a noun phrase, or if they do not belong to a noun phrase.

Chunking is a word labelling problem. Here, the task is to identify whether certain constituents belong to a noun-phrase, a verb-phrase, and others.

Segmentation identifies segments in Chinese ideograms. The labels are "B" and "I", for "beginning of word" and "inside a word", and each unit of the Chinese text has one label.

Japanese NE is a named entity recognition problem. It labels occurrences of named entities in text. Examples of named entities are: "animal", "person", "building", "organization", etc. and here, each Japanese word is assigned a label with the named entity it represents.

In addition to the small datasets shown above, an additional large dataset was considered. **ConLL** is the large dataset used to evaluate the scalability of the model. This dataset will be handled differently than the others. Due to its dimensions, no cross-fold validation will be employed, only a train and test split is considered.

These are all sets of structured objects, where each structured object is composed of T_n observations, and N is the total number of observations, where $N = \sum_{n=1}^{Nseq} T_n$ and $Nseq$ is the number of structured objects.

[1] This was developed by Taku Kudo and can be found at https://taku910.github.io/crfpp/

Table 5.1: Datasets' information

	BaseNP	Chunk.	Segm.	Jap.	ConLL
Labels	3	14	2	17	23
Features	6438	29764	1386	102,799	2,035,523
Sentences	300	100	36	100	8936/2012
Avg. training words	3740	1156	942	1315	211,727

5.2 Hyperparameter tuning

Although the model can accommodate 16 parameters from the user, at a first stage it would not be sensible to optimize them all. Some parameters have already been studied for other problems, so those will be taken from the literature (Cutajar et al., 2016; Galliani et al., 2016), and others are more problem-specific, so they will be tuned. The hyperparameters of interest were identified as: the learning strategy, the learning rate, the kernel of the original GPs and the number of random features. The number of layers is not considered a hyperparameter, since models with different layers are considered models with different architectures, so they will be analyzed separately. The learning strategy is also not a hyperparameter *per se*, since it represents a structural change on the way inference is conducted.

5.2.1 SDGP

The hyperparameter tuning was done considering the SDGP1 flat model and the Base NP dataset. The hyperparameter tuning can be done for every dataset, but considering there are 5 degrees of freedom and it takes 80 hours on one Graphical Processing Unit (GPU) to run the four benchmarks, for research purposes, the hyperparameters were chosen based on the largest of them - Base NP. It should be noted that since all the problems are similarly formulated NLP tasks, it is expected that the optimal hyperparameters for each task do not differ drastically. Due to the time required to run experiments, the set of possible values for the hyperparameters that can be tried is also considerably reduced. For that reason, the experience of the research team was paramount to sensibly choose good candidates for the parameters.

In order to verify if the optimization is working properly, the evolution of the ELBO was considered. In order to select the best hyperparameters, the ones providing the lowest Error Rate were chosen. The evolution of the Error Rate is presented alongside the objective function's evolution (in its negative form, the NELBO), and the evolution of $L_{unstruct}$ can be found in the Appendix. The starting parameters for the flat SDGP are based on the recommended parameters for classification by Cutajar et al. (2016), but these quickly proved to be sub-optimal. Table 5.2 splits the parameters into the ones that are related to the original SDGP model itself, the ones that are related to the implementation of the SDGP model, and the ones that are specific of the resulting

neural network (and its optimization strategy). The parameters of interest were altered one by one, and the ones that produced the best performance were kept. This process is illustrated below.

Table 5.2: Initial hyperparameters

Model		**Implementation**		**Neural network**	
# layers	1	Ω fixed (epochs)	1000	Learn Rate	0.01
Learn strategy	var-fixed	Θ fixed (epochs)	4000	Optimizer	adam
ARD	True	# train samples	100	Batch size	200
Random Features	1000	# test samples	100		
GPs / layer	1	Duration (min)	60		

Learning Strategy and Learning Rate Figure 5.1 illustrates the optimization process with these parameters. In addition to the variable being optimized (NELBO), the evolution of the Error Rate is also presented. Looking at the evolution of the NELBO, it is clear that there is an optimization problem, given that this quantity is supposed to be minimized. Two paths can be followed at this point. The simplest approach is to consider a more stable learning scheme: prior-fixed. The more complicated approach is analyzing the hyperparameters characterizing the results in Figure 5.1 and tuning them until the optimization flows smoothly. Both approaches were applied. The optimization process considering the prior-fixed learning strategy is illustrated in Figure 5.2. As predicted, there were no optimization problems with this strategy. However, since var-fixed has been reported to have better results than prior-fixed (Cutajar et al., 2016), an effort to optimize var-fixed correctly should be conducted. The hyperparameter most related to the learning process itself and that can affect how the algorithm gets stuck on local optima is the learning rate. For that reason, this parameter was reduced from 0.01 to 0.001, and this is considered the beginning of the hyperparameter tuning process, so only the dataset base NP will be considered from now on. Additionally, differences between different folds will now also be evident, and the results for each of the 5 folds will be illustrated. The resulting optimization process is represented in Figure 5.3. This reduction in learning rate was enough to solve the optimization problem, and the var-fixed learning strategy can be further used. The performance metrics are compared between var-fixed and prior-fixed in Table 5.3, and var-fixed outperforms indeed prior-fixed, validating the results presented by Cutajar et al. (2016).

Table 5.3: Performance comparison between learning strategies - prior-fixed and var-fixed

	Error Rate	$NLL_{unstruct}$
Prior-fixed	6.60%	732
Var-fixed	5.18%	550

Kernels For the results to be perfectly comparable to the ones presented by Galliani et al. (2016), a linear kernel should be used. Unfortunately, in order to apply Random Feature Expansions, the

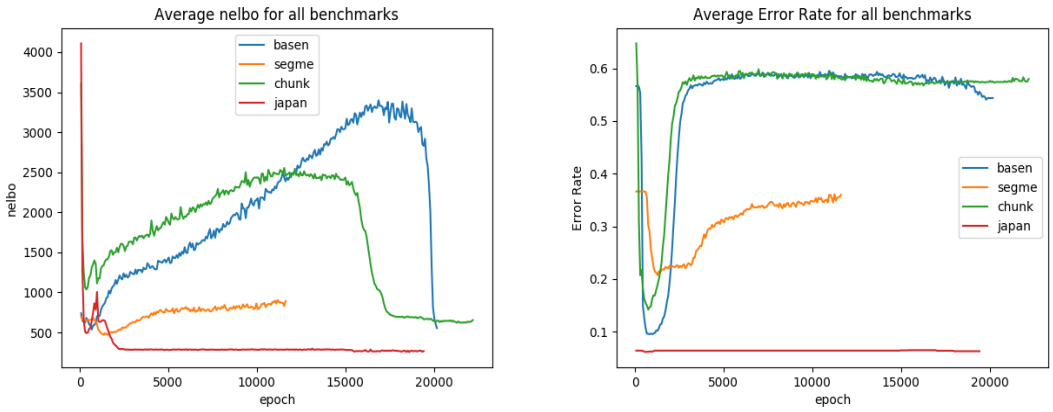

Figure 5.1: Structured experiments with original parameters

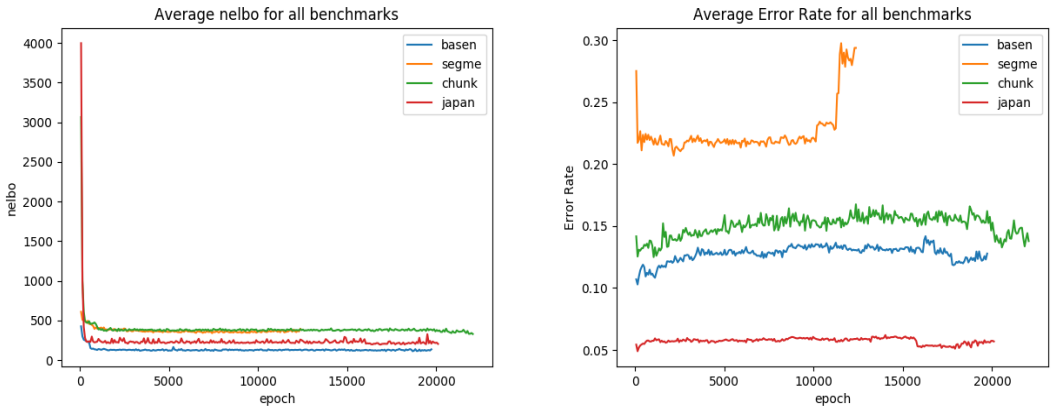

Figure 5.2: Structured experiments with prior-fixed as learning strategy

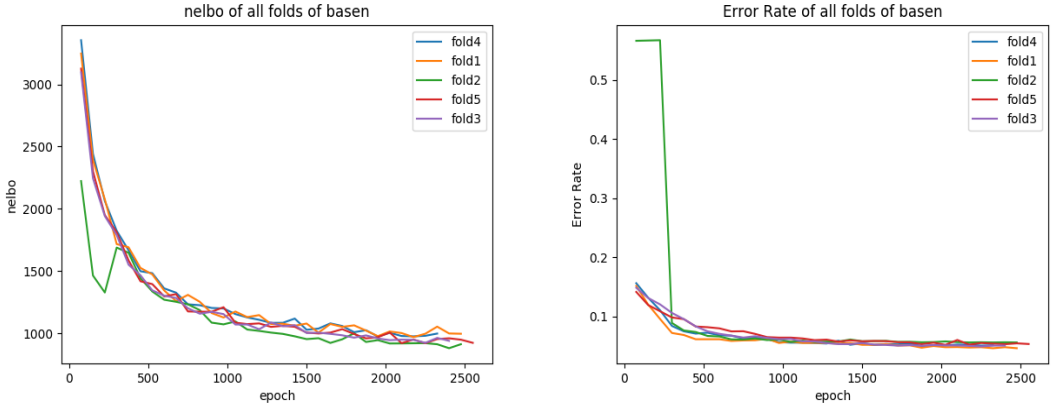

Figure 5.3: Experiments with var-fixed and LR=0.001

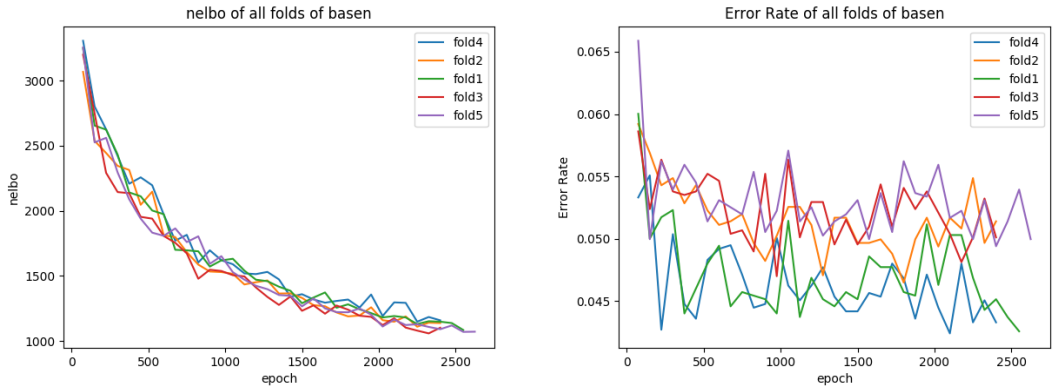

Figure 5.4: Structured experiments with var-fixed, LR=001 and kernel=arc-cosine

kernel has to be strongly stationary (Rahimi and Recht, 2008), which implies linear kernels cannot be used. Still, the RFE has been formulated for the arc-cosine kernel as well, aside from RBF, so this kernel should also be tested. Figure 5.4 illustrates the results of introducing the arc-cosine kernel - an improvement in Error Rate (to 4.75%) and NLL_{struct} (to 552).

Random Feature Expansions Then, it is still possible to increase the number of random features to make the approximation closer to a GP. In the next experiments, an increase from 1000 to 2000 random features was considered. The results can be seen in Figure 5.5.

The "right" number of random features is one that makes the network "close enough" to a GP. When an increase in random features does not lead to an increase in performance, it probably means the "right" number of features has been found. For that reason, once such a value is found, this thesis employs it in all the GP expansions, which will imply that all the layers will have the same number of nodes. This does not have to be the case, though, as it would be possible to employ lower values for certain GPs and higher values for others.
The results with 2000 Random Features are similar to the results with 1000, even slightly worse.

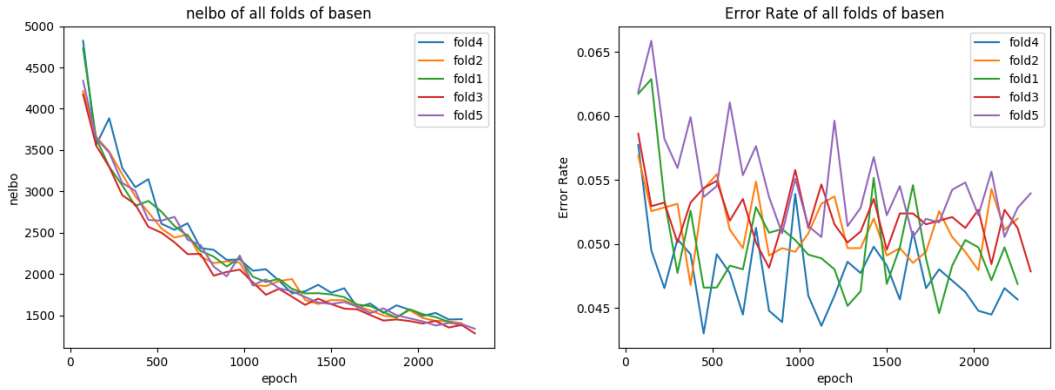

Figure 5.5: Structured experiments with var-fixed, LR=001, kernel=arc-cosine and 2000 random features

This means that 1000 random features are probably already high enough to fully characterize a GP, and there is no necessity to further increase them. Since an increase in random features brings complexity to the model, the final choice was 1000.

Table 5.4: SDGP1 Performance on baseNP after hyperparameter tuning

	Error Rate	NLL_{struct}
Learn Rate = 0.001	5.18%	550
Kernel = Arccos	4.75%	552
RFF = 2000	4.93%	572

After this analysis, the model parameters are set.

5.2.2 Multinomial Logistic Regression

Regarding the Logistic Regression (Multinomial Logistic Regression), an important parameter to be chosen is the regularization constant. In this context, regularization consists of adding extra information in order to prevent overfitting. This materializes in penalizing the objective function by a "regularization term" which is amplified by a constant.

$$obj = min_{\Theta}(PredictionLoss(\Theta) + \frac{1}{C}RegularizationLoss(\Theta))$$

Typically, in order to choose the hyperparameters, the original dataset is split into: training, validation and testing. However, in this case, if this model is to be comparable to the others, then it has to use 80% of the data for training and 20% for testing, which means an independent validation set cannot be created. The strategy used to bypass this problem was using various values for C, splitting the training data on "training-training" and "training-validation", choosing the C with the best Error Rate on the validation set, and then evaluating the whole model by training on the whole training data and testing on the test data. Initially, the standard parameter for C was 1, for which the results are presented in table 5.5.

Table 5.5: Multinomial Logistic Regression performance on different datasets

	BaseNP	Chunk.	Segm.	Jap.
Error Rate	5.02%	8.53%	14.93%	5.50%
$NLL_{unstruct}$	559	476	251	608

To study the effect of C and to try to improve the results, for each dataset, the training data was split into train (80%) and validation (20%) and, for each value of C ($C \in \{0.01, 0.1, 1, 1, 10\}$), this validation set was evaluated. The C responsible for the best performance was the chosen one. Although the best parameters were not always 1, the results after this validation procedure are considerably similar to the initial ones, and are shown in the next section in table 5.7, so it

appears that using a regularization constant different than 1 does not have a significant impact on the results.

5.3 Models comparison

The results will be reported first for each technique (and all datasets) and in the end a table with the top results from every technique will also be presented.

5.3.1 Small datasets

State-of-the-art models

The initial benchmarks come from previously studied models and reported results. These models are Conditional Random Fields (CRFs), GPs with Elliptical Slice Sampling (GP-ESS) (Bratières et al., 2015) and GPs with unindexed pairwise functions with Variational Inference (GP-Var) (Galliani et al., 2016). The best results for Error Rate and NLL are shown in table 5.6, as well as the model that originated them.

Table 5.6: Current state-of-the-art performance on different datasets

	Error Rate	$NLL_{unstruct}$
Base NP	5.10% (ESS)	603 (GP-Var)
Chunking	8.50% (ESS and CRF)	407 (GP-Var)
Segmentation	14.5% (GP-Var)	253 (CRF)
Japanese NE	5.20% (CRF)	339 (GP-Var)

There is not one single model that beats all, but it seems clear that the advantage of the GP-Var lies within better quantification of certainty, as evidenced by the $NLL_{unstruct}$.

Standard unstructured multi-label technique (Multinomial Logistic Regression)

This model produced surprisingly good results surprisingly quick. While all the other models were set to run for 4 hours, this one was only trained for some seconds (it just followed sci-kit learn's procedure) and achieved results comparable to the state-of-the-art. This suggests that either the data is not highly structured after all, or that the so called structured models are not really capturing the structuredness well. The results that are presented in bold outperform the previously reported results.

Table 5.7: Multinomial Logistic Regression performance on different datasets after regularization optimization

	BaseNP	Chunk.	Segm.	Jap.
Error Rate	**5.08**%	8.67%	14.99%	5.34%
$NLL_{unstruct}$	624	454	254	645

Unstructured DGPs

These experiments consisted of considering the DGP model (Damianou and Lawrence, 2012) but with only one layer - a flat GP. This is equivalent to having a single GP expanded via Random Feature Expansions. The expectation was that this would yield considerably worse results compared to the state-of-the-art, since this does not incorporate the structuredness of the data in any way. The resulting Error Rate and NLL test are presented in table 5.8. Although these results are worse than the ones reported in the state-of-the-art, the difference is not substantial. The NLL on the Segmentation dataset is even better than what was previously reported. This, once again, suggests that either the datasets are not highly structured or the state-of-the-art models are not considering this structure properly.

Table 5.8: Flat GP with RFE: Performance on different datasets

	Error Rate	$NLL_{unstruct}$
Base NP	5.58%	646
Chunking	9.89%	431
Segmentation	16.28%	**249**
Japanese NE	5.98%	370

Unindexed SDGP (SDGP1)

In order to evaluate the impact of using Random Feature Expansions, the first architecture to be analyzed is a flat one (1 hidden layer) - the results can be seen in Table 5.9. This model provides a general improvement on the Unstructured DGP and also on the previously studied structured models. In terms of NLL it is also clearly better than the Multinomial Logistic Regression, but in terms of Error Rate the Multinomial Logistic Regression is similar, or even better. For 5 out of the 8 reported quantities, this model represents an improvement over the state-of-the-art. This is not a representation of the power of deep models yet, this only represents how powerful Random Feature Expansions are (the first factor in chapter 4).

Table 5.9: Unindexed flat SDGP performance on different datasets

	Error Rate	$NLL_{unstruct}$	NLL_{struct}
Base NP	**4.96%**	**566**	555
Chunking	9.30%	**386**	491
Segmentation	16.98%	331	399
Japanese NE	**5.06%**	**318**	388

However, the deep architecture is the one expected to achieve the best results. It should be noted, however, that deep architectures with small amounts of data are prone to overfitting, due to the high number of parameters that need to be optimized with only a few inputs. Therefore, it is possible that the deep network underperforms compared to the flat one in the small datasets. However, this should not be the case for large datasets. The next model is a 10 layered SDGP1, it is analogous to the flat SDGP but instead of having a flat structure it has a deep one with 10 hidden layers.

Table 5.10: Unindexed SDGP1 (10 layers) performance on different datasets

	Error Rate	$NLL_{unstruct}$	NLL_{struct}
Base NP	6.67%	804	918
Chunking	13.16%	560	759
Segmentation	17.72%	271	333
Japanese NE	6.78%	631	657

As hypothesized, the deep model does not outperform the flat model on small datasets, as there is a high potential for overfitting considering the number of parameters to optimize and reduced size of data available. A model with 5 layers was also tested and its results are presented in the Appendix.

Indexed SDGP (SDGP2)

Under the hypothesis that the interactions between adjacent labels are more dependent on the inputs (X) than on the labels themselves, this should be the most promising model. The objective is to evaluate the indexing on the flat structure, and if it represents an improvement over the flat SDGP1, following the analysis to deeper structures. However, the results, which are presented in Table 5.11 are not extremely satisfying, and they are clearly worse than the Unindexed SDGP1.

Table 5.11: Flat Indexed SGP performance on different datasets

	Error Rate	$NLL_{unstruct}$	NLL_{struct}
Base NP	7.17%	1865	2672
Chunking	9.88%	713	1045
Segmentation	22.08%	1923	3228
Japanese NE	5.89%	1314	940

There is a variety of plausible justifications for this. The first one is that the interactions depend indeed more on the labels than on the inputs, which is the implicit hypothesis in the structured likelihood function originally presented (Bratières et al., 2015). However, since this model has a considerably higher number of variables, perhaps it would require different hyperparameters (the ones used were the same as in the unindexed SGP), or it might also be due to overfitting. Considering the poor performance of the Indexed SDGP with a flat structure the analysis of the deep indexed SDGP was not followed.

Final comparison between models

The SDGP1 model is competitive with other techniques. For 6 out of the 8 metrics of interest, models implemented in this thesis achieved better results than the ones reported in the state-of-the-art, although the improvements were not significant. Since these datasets have been studied for a long time, one possibility is that the performances achieved are close to the best that can possibly be done. Since so many different approaches have been tried, from Bayesian Deep Networks (this one) to full Gaussian Processes and CRFs, this explanation is plausible. On the other hand, it was also seen that the performances on these datasets using different models is never completely disparate, which can suggest insufficient structure within the data, or insufficient ability to capture such structure by the structured models. The latter is not very likely due to the long term success of some techniques like CRFs. A similar study between the models used in this thesis but with different datasets would clear these doubts.

Table 5.12: Current state-of-the-art performance on different datasets

	Error Rate	$NLL_{unstruct}$
Base NP	**4.96% (SDGP1)**	**566 (SDGP1)**
Chunking	8.50% (ESS and CRF)	**386 (SDGP1)**
Segmentation	14.5% (GP-Var)	**249 (DGP)**
Japanese NE	**5.06% (SDGP1)**	**318 (SDGP1)**

5.3.2 Large datasets

To evaluate the scalability of this model, a large dataset was studied. This dataset was evaluated with the most promising models, based on the results reported in the previous subsection: flat

SDGP1 and the Multinomial Logistic Regression, and also on the deep SDGP1 model. The results are reported in table 5.13.

Table 5.13: Performance of different models on large datasets

	Error Rate	$NLL_{unstruct}$	NLL_{struct}
Multinomial Logistic Regression	4.22%	7897	-
Flat SDGP1	7.19%	13476	10717
Deep SDGP1	4.17%	7222	4942

The Error Rate and $NLL_{unstruct}$ for the SDGP1 seem to be the best reported so far, but the difference to the Multinomial Logistic Regression is, once again, not substantial. It is definitely puzzling that a basic unstructured classification model manages to almost outperform every Structured Prediction algorithm. Like before, there are 2 potential takeaways here, either the data is not highly structured, or the data is indeed highly structured but the SDGP model (and the other models reported) is not fully capturing this.

Chapter 6

Conclusion

This thesis introduced a new formulation of Deep Gaussian Processes for the problem of Structured Prediction. Structured Deep Gaussian Processes combine the non-parametric flexibility and probabilistic approach of Gaussian Processes with the scalability of neural networks, thereby bridging the gap between these two techniques.

Full Deep Gaussian Processes consider a sequence of Gaussian Processes to make predictions, but this model is converted into a Bayesian neural network using Random Feature Expansions in order to make it scalable. Different variations have been considered based on: i) type of dependencies being modeled (whether the structuredness in Y depends on adjacent Y elements - SDGP1, or on adjacent X elements - SDGP2), and ii) model characteristics (number of hidden layers, random feature expansions, and others). The SDGP1 model was the central focus of this study and it proved to be the highest performing model, with Error Rates and Uncertainty Quantification that match and outperform other state-of-the-art methods like CRFs, Structured GPs with Elliptical Sliced Sampling and GPs with Automated Variational Inference, both in small and large datasets. It was identified that the best performing kernel is the arc-cosine one. One thousand random features provides a good enough approximation, and learning rates of 0.01 can cause optimization issues, so the recommended value is 0.001. Additionally, the general consensus that deep structures suit large datasets better was also verified, as the flat model outperformed the deep one on the small datasets, but the deep one clearly outperformed the flat one on the large dataset considered.

The SDGP2 was also explored, but to a lesser extent - its flat version was applied to small experiments. This occurred both due to the limited amount of time available for this project and due to the low performance achieved in comparison with both the SDGP1 and the other benchmarks.

A surprising and relevant takeaway from this study was that considerably complex models like SDGPs should be carefully analyzed before being employed in a problem, as there can be simpler and better alternatives. While a deep Bayesian structure can, in principle, identify highly complex relationships and provide a lot of insights regarding the uncertainty of the predictions, simpler methods like the Multinomial Logistic Regression might be capable of achieving comparable results in considerably less time (a few seconds instead of hours to achieve the same performance metrics).

For future work, a number of directions, theoretical and practical, can be further explored. On the theoretical side, the SDGP2 (and the SDGP1 as well, but to a clearly lesser extent) can be further developed in terms of: 1) dimensionality reduction and 2) hyperparameter tuning. Although dimensionality reduction is a requirement for this model to run, the particular scheme implemented was not especially tailored to the problem and/or dataset, so using a more advanced one like Principal Component Analysis (F.R.S., 1901) or Non-negative Matrix Factorization (Sra and Dhillon, 2006) might improve both the scalability of this model and its performance. Due to time constraints, the hyperparameters were chosen based on one of the small experiments with SDGP1. However, SDGP2 introduces a considerable amount of variables by introducing a parallel network and every experiment is different, especially when considering a large one compared to small ones. Therefore, tailoring the hyperparameters to the specific dataset and model being analyzed is expected to bring improvements over the reported performance. Additionally, considering the bridging between GPs and neural networks, techniques from the latter can be further employed, namely dynamic learning rates, different nodes (random features) for each GP/layer, perhaps even considering dropout, although GPs usually do not need additional regularization schemes. Additionally, different architectures can be considered. For example, in the indexed model, SDGP2, making the unary and binary networks share nodes ("shared kernels") would reduce the number of variables considerably, which could result in a performance improvement. Lastly, a more extensive study and comparison between these and other state-of-the-art models using other datasets is also recommended. Throughout this thesis it became clear that the increase in performance brought by structured models was not as high as expected, and a potential reason for that is lack of structuredness in the datasets, so different datasets should also be studied.

On the practical side, although the SDGP1 model has achieved competitive results in text processing tasks, more innovative applications could be considered. Structured prediction can have a considerably diverse span of applications, especially in the industrial world, where sequential processes and their optimization are widespread. Throughout this thesis, a series of typical industrial examples have been introduced with the aim of motivating practitioners to employ these methods in their fields. In this context, a study comparing the best structured prediction models in the probabilistic field (like CRFs and GPs) with the best ones from the non-probabilistic field (like Convolutional Neural Networks) could provide a better insight into the trade off between uncertainty quantification and computational costs from the practical point of view.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *ArXiv e-prints*.

Álvarez, M. A. and Lawrence, N. D. (2011). Computationally efficient convolved multiple output gaussian processes. *J. Mach. Learn. Res.*, 12:1459–1500.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blei, D. M. (2002). Variational inference.

Bo, L. and Sminchisescu, C. (2010). Twin gaussian processes for structured prediction. *International Journal of Computer Vision*, 87:28–52.

Bonilla, E. V., Chai, K. M., and Williams, C. (2008). Multi-task gaussian process prediction. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*, pages 153–160. Curran Associates, Inc.

Bratieres, S., Quadrianto, N., Nowozin, S., and Ghahramani, Z. (2014). Scalable gaussian process structured prediction for grid factor graph applications. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 334–342, Bejing, China. PMLR.

Bratières, S., Quadrianto, N., and Ghahramani, Z. (2015). Gpstruct: Bayesian structured prediction using gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7):1514–1520.

Bui, T. D., Hernández-Lobato, J. M., Hernández-Lobato, D., Li, Y., and Turner, R. E. (2016). Deep gaussian processes for regression using approximate expectation propagation. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1472–1481. JMLR.org.

Cheng, L.-F., Darnell, G., Chivers, C., E Draugelis, M., Li, K., and E Engelhardt, B. (2017). Sparse Multi-Output Gaussian Processes for Medical Time Series Prediction. *ArXiv e-prints*.

Cho, Y. and Saul, L. K. (2011). Analysis and Extension of Arc-Cosine Kernels for Large Margin Classification. *ArXiv e-prints*.

Clifton, L., Clifton, D. A., Pimentel, M. A. F., Watkinson, P. J., and Tarassenko, L. (2013). Gaussian processes for personalized e-health monitoring with wearable sensors. *IEEE Transactions on Biomedical Engineering*, 60(1):193–197.

Cohn, T., Preotiuc-Pietro, D., and Lawrence, N. D. (2014). Gaussian processes for natural language processing. In *ACL (Tutorial Abstracts)*, pages 1–3. The Association for Computer Linguistics.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Cutajar, K., Bonilla, E. V., Michiardi, P., and Filippone, M. (2016). Random Feature Expansions for Deep Gaussian Processes. *ArXiv e-prints*.

Damianou, A. C. and Lawrence, N. D. (2012). Deep Gaussian Processes. *ArXiv e-prints*.

Dodd, T. and Rogers, E. (2000). Aerospace applications of gaussian processes, hilbert spaces and wavelets. In *IEE Seminar on Model Validation for Plant Control and Condition Monitoring (Ref. No. 2000/044)*, pages 8/1–8/3.

Edunov, S., Ott, M., Auli, M., Grangier, D., and Ranzato, M. (2017). Classical Structured Prediction Losses for Sequence to Sequence Learning. *ArXiv e-prints*.

Edwards, R. E. (1979). *Positive Definite Functions and Bochner's Theorem*, pages 148–154. Springer New York, New York, NY.

F.R.S., K. P. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

Galliani, P., Dezfouli, A., Bonilla, E. V., and Quadrianto, N. (2016). Gray-box inference for structured Gaussian process models. *ArXiv e-prints*.

Gelfand, A. E. and Smith, A. F. M. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409.

Genton, M. G. (2002). Classes of kernels for machine learning: A statistics perspective. *J. Mach. Learn. Res.*, 2:299–312.

Gonzalez, J. P., Jarvis, A., Cook, S. E., Oberthür, T., Rincon-Romero, M., Bagnell, J. A., and Dias, M. B. (2008). *Digital Soil Mapping of Soil Properties in Honduras Using Readily Available Biophysical Datasets and Gaussian Processes*, pages 367–380. Springer Netherlands, Dordrecht.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347.

Hong, S. and Zhou, Z. (2012). Application of gaussian process regression for bearing degradation assessment. In *2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012)*, pages 644–648.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Mach. Learn.*, 37(2):183–233.

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *ArXiv e-prints*.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

Krauth, K., Bonilla, E. V., Cutajar, K., and Filippone, M. (2016). AutoGP: Exploring the Capabilities and Limitations of Gaussian Process Models. *ArXiv e-prints*.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86.

Lafferty, J., Zhu, X., and Liu, Y. (2004). Kernel conditional random fields: Representation and clique selection. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 64–, New York, NY, USA. ACM.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Lu, C. and Tang, X. (2014). Surpassing Human-Level Face Verification Performance on LFW with GaussianFace. *ArXiv e-prints*.

MacKay, D. J. (1991). Bayesian interpolation. *Neural Computation*, 4:415–447.

McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 591–598, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21:1087–1092.

Minka, T. P. (2001). Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 362–369, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Murphy, K. P. (2013). *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.].

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg.

Qi, Y., Szummer, M., and Minka, T. (2005). Bayesian conditional random fields. *Journal of Machine Learning Research (JMLR), AI & Statistics*, pages 269–276.

Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*, pages 1177–1184. Curran Associates, Inc.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407.

Robert, C. and Casella, G. (2011). A short history of markov chain monte carlo: Subjective recollections from incomplete data. *Statist. Sci.*, 26(1):102–115.

Rowland, T. (2018). Functional. From MathWorld—A Wolfram Web Resource. Last visited on 04/4/2018.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA.

Snelson, E. (2007). Flexible and efficient gaussian process models for machine learning.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc.

Sra, S. and Dhillon, I. S. (2006). Generalized nonnegative matrix approximations with bregman divergences. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 283–290. MIT Press.

Srijith, P. K., Balamurugan, P., and Shevade, S. (2014). Gaussian Process Pseudo-Likelihood Models for Sequence Labeling. *ArXiv e-prints*.

Sutton, C. A. and McCallum, A. D. (2007). Piecewise pseudolikelihood for efficient training of conditional random fields. In *ICML*.

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin markov networks. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, pages 25–32, Cambridge, MA, USA. MIT Press.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse gaussian processes. In *In Artificial Intelligence and Statistics 12*, pages 567–574.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484.

Weisstein, E. W. (2018a). Positive definite matrix. From MathWorld—A Wolfram Web Resource. Last visited on 06/4/2018.

Weisstein, E. W. (2018b). Probability axioms. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/ProbabilityAxioms.html. visited on May 24th 2018.

Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1168–1175, New York, NY, USA. ACM.

Williams, C. K. I. (1998). Computation with infinite neural networks. *Neural Comput.*, 10(5):1203–1216.

Williams, C. K. I. and Barber, D. (1998). Bayesian classification with gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351.

Álvarez, M., Luengo, D., Titsias, M., and Lawrence, N. (2010). Efficient multioutput gaussian processes through variational inducing kernels. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 25–32, Chia Laguna Resort, Sardinia, Italy. PMLR.

Appendix A

Appendix

A.1 Jensen's Inequality

Jensen inequality is the generalization of the statement "the secant of a convex function lies above its graph". Mathematically, this is stated in equation A.1 and its 2-dimensional version can be visualized in picture A.1.

$$f(E[X]) \leq E[f(x)] \tag{A.1}$$

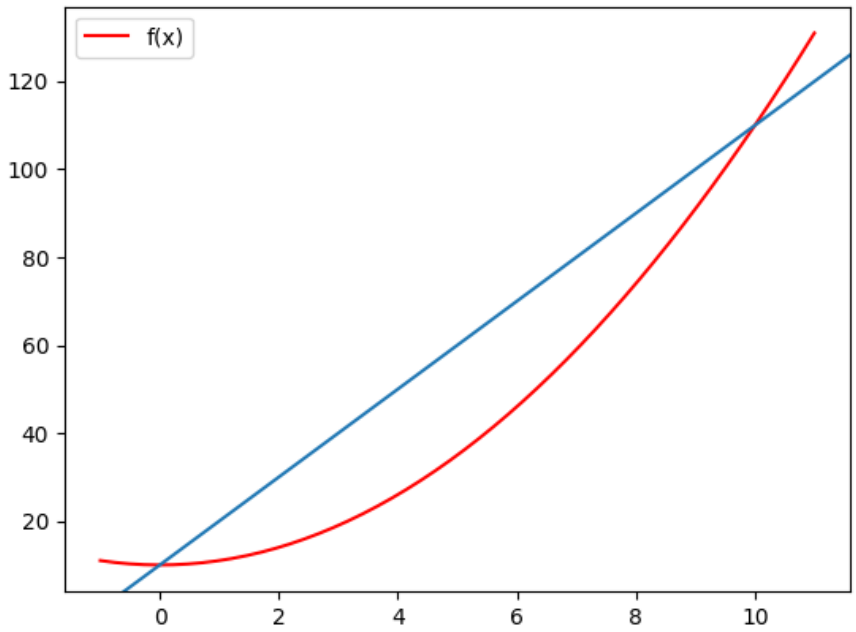

Figure A.1: Illustration of Jensen's Inequality in 2 dimensions. The red line, representing function the convex function f, is below its secant, represented by a blue line.

A.2 Alternative derivation of the generic ELBO

At first, it might not be immediate how the introduction of this new functional q is related to the evidence. Consider the evidence equation again, now with the X's removed for notation simplicity:

$$P(Y) = \int P(Y \mid f) * P(f) * df$$

A couple of things are to be noted: 1) if a KL-divergence is to appear, a logarithm will have to be applied 2) if $q(f)$ is to be related to the objective function, then $q(f)$ will also have to be introduced. Therefore, consider $\log P(Y)$ instead of $P(Y)$ and multiply and divide by $q(f)$ inside the integral:

$$\log P(Y) = \log[\int \frac{P(Y \mid f)P(f)}{q(f)} q(f) df]$$

Having done these operations, one ends up with the logarithm of an expectation over $q(f)$:

$$\log P(Y) = \log[E_{q(f)} \int \frac{P(Y \mid f)P(f)}{q(f)}]$$

So far, this does not seem to have helped much, as this expression is not really tractable as well. As a result, this is where 'Jensen's Inequality' turns out to be helpful. This inequality allows to bound the evidence by another computable term.

$$\log P(Y) = \log[E_{q(f)} \int \frac{P(Y \mid f)P(f)}{q(f)}] \geq E_{q(f)} \log \int \frac{P(Y \mid f)P(f)}{q(f)}$$

Now, simple logarithm rules and the integral operator's linearity eliminate this complicated term:

$$= E_{q(f)}(\log[P(Y \mid f)]) + E_{q(f)}(\log[\frac{P(f)}{q(f)}])$$

Which is a sum between the expected log likelihood and the KL divergence between the variational posterior and the prior.

$$= L_{ell} - KL[q(F)||p(F)]$$

A.3 Expression for KL divergence between Gaussians

Given $P_1(X) = N(\mu_1, \sigma_1^2)$ and $P_2(X) = N(\mu_2, \sigma_2^2)$:

$$KL(P_1(X)||P_2(X)) = \frac{1}{2}[\log(\frac{\sigma_2^2}{\sigma_1^2}) - 1 + \frac{\sigma_1^2}{\sigma_2^2} + \frac{(\mu_1 - \mu_2)^2}{\sigma_2^2}]$$

A.4 Model hyperparameters

- NL: Number of hidden layers
- DF: Number of parallel GPs per layer
- Kernel to consider: The expansions have been defined for the RBF and arccosine.
- Iterations to fix Ω and Θ: These parameters are set until a certain number of iterations after which they start being optimized. This was introduced to solve optimization problems.

- Learning strategy for Ω: the spectral frequencies can be learnt variationally (with fixed randomness or not - var_fixed or var_resampled) or simply drawn from a prior (prior_fixed). Cutajar et al. (2016) has shown that var_resampled appears to be too unstable, so for that reason only prior_fixed and var_fixed have been considered.

- RFF: Number of Random Features. The higher this value is, the closer the model is to a full GP. However, as it increases, more computational power and memory are required.

- Learning Rate: The rate at which the model is updated. If this value is too high, optimization problems can occur, as it might jump around local minima. If it is too low though, it might take too long to optimize.

- Automatic Relevance Determination (Neal, 1996): ARD is related to the way the kernel considers each dimension of the inputs. In essence, it allows to automatically determine the relevance of each dimension. If a dimension does not affect the output, throughout the optimization its weight will tend to 0. All the experiments were done with ARD.

- Number of Monte Carlo samples: The expected log likelihood was computed with Monte Carlo sampling, both for training and testing. For that reason, the number of samples needs to be specified. At the beginning, only 1 training MC sample is used, and this is then increased to 100, following Cutajar et al. (2016)

- Optimizer to be used: The GP is converted to a neural network which is then optimized. The optimization technique/optimizer is specified with this parameter, which is then fed to TensorFlow which will conduct the optimization. The one considered in all the experiments was Adam.

- Pseudo-Likelihood: Computing the true likelihood is a computational challenge in itself (Bratières et al., 2015). For small datasets, this does not pose any significant barriers to training, but in large datasets, these computations can make the training of the model unfeasible. For that reason, an approximation of the likelihood (piecewise pseudo-likelihood (Sutton and McCallum, 2007)) that runs significantly faster was also introduced for large datasets. This is the parameter that defines that, and it was not present in the original DGP (Cutajar et al., 2016).

- Display step: Number of iterations after which the model should stop training and evaluate the test data.

- Duration: Number of minutes the experiments should last.

A.5 Evolution of $L_{unstruct}$ during hyperparameter tuning

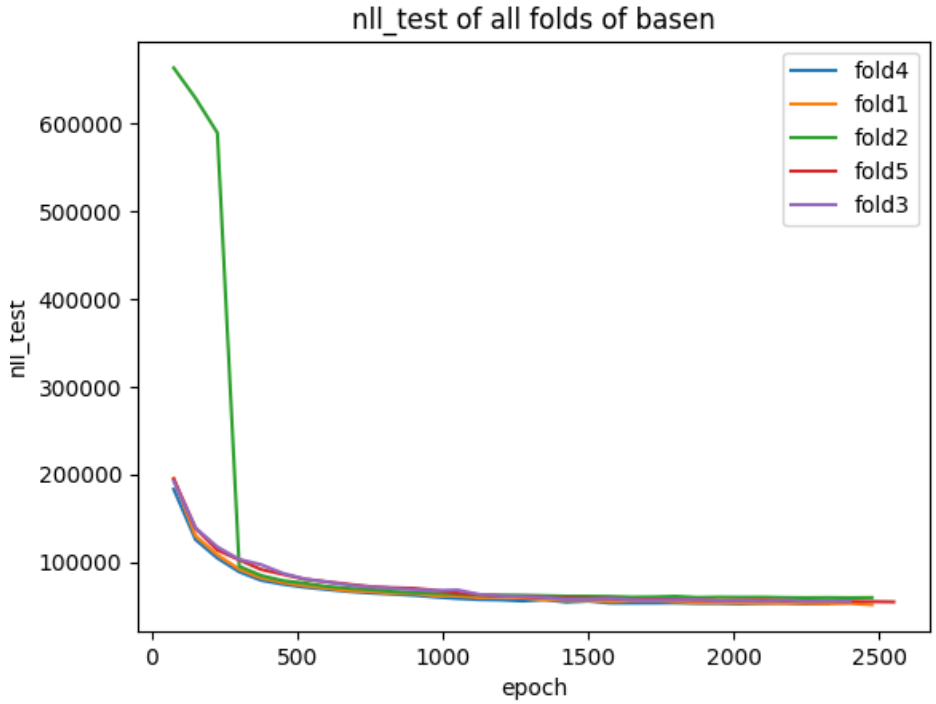

Figure A.2: Experiments with var-fixed and LR=0.001, kernel=RBF and 1000 Random Features Expansions

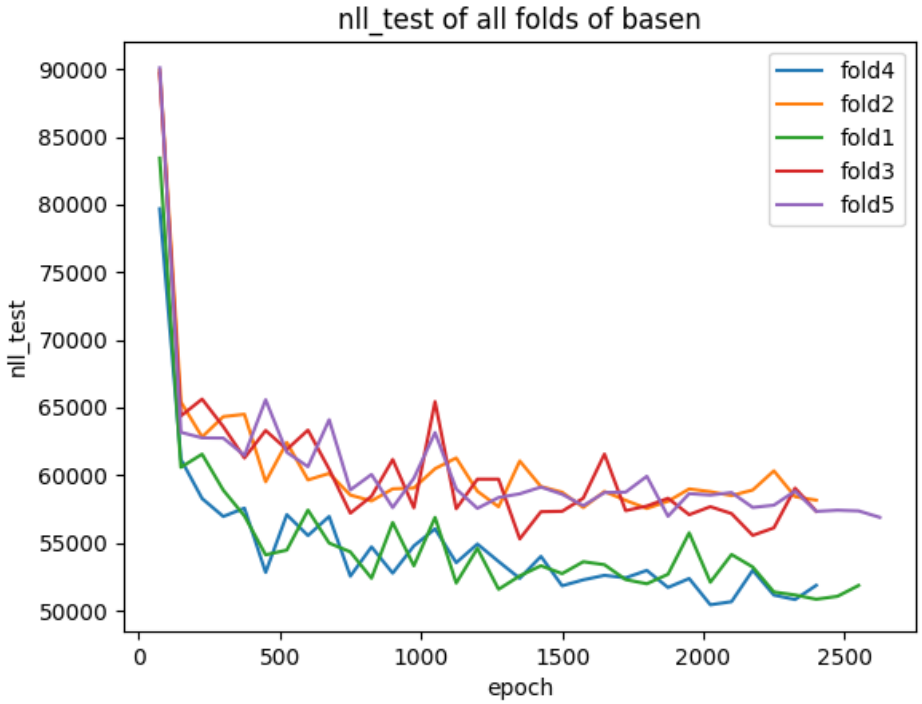

Figure A.3: Structured experiments with var-fixed, LR=001 and kernel=arc-cosine and 1000 Random Features Expansions

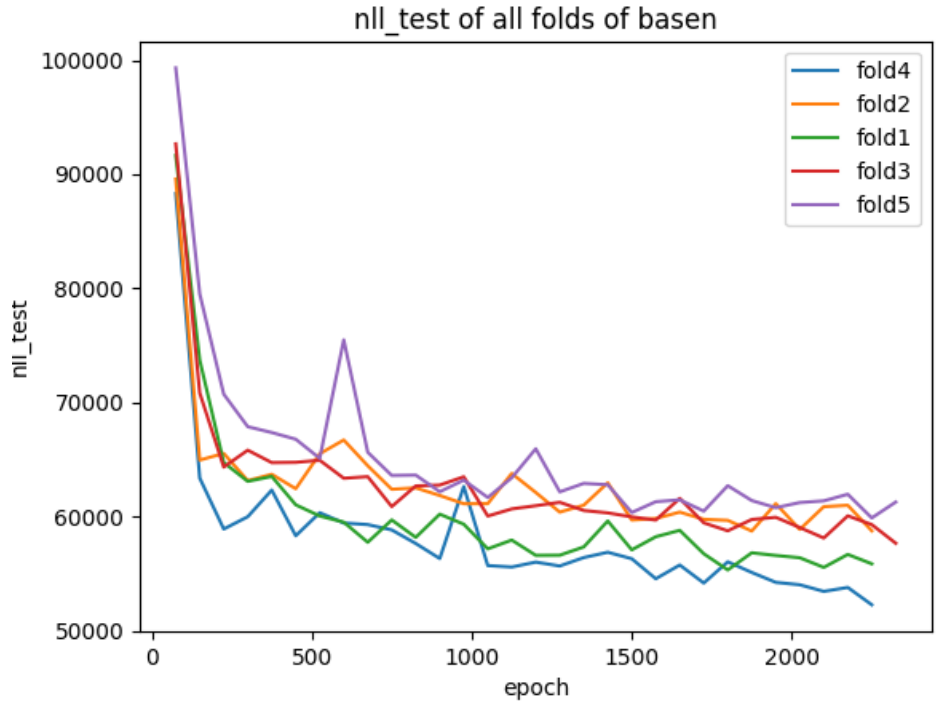

Figure A.4: Structured experiments with var-fixed, LR=001, kernel=arc-cosine and 2000 Random Features Expansions

A.6 Unindexed SDGP1 with 5 layers

Table A.1: Unindexed SDGP1 (5 layers) performance on different datasets

	Error Rate	$NLL_{unstruct}$	NLL_{struct}
Base NP	7.14%	1693	1456
Chunking	12.02%	518	721
Segmentation	16.55%	263	314
Japanese NE	6.78%	530	560