# Understanding Gaussian Processes: From Theory to Applications

Giovanni Franzese
TU Delft

April 2024

# Preface

The first time I fitted a Gaussian Process (GP) was at the beginning of my PhD, during the COVID-19 pandemic. At the time I was using MATLAB, and I did not have a very thorough understanding of the mathematics that was regulating the fitting or the prediction process. Nevertheless, seeing my GP fitting perfectly my data was a revelation and since then, my love for them has not vanished. I collected my ideas and trained different models in the last years that made my understanding of the fundamentals better and I hope, with these lecture notes, to gently introduce you to the fantastic field of probabilistic model learning. There are many other good sources where to learn on Gaussian Process and these lecture notes are not a substitute for them, but hopefully will give you some tools and intuitions to read and understand better the literature.

Giovanni

Chapter 1

# Introduction

Imagine to model the acceleration of a car given how hard you press the accelerator and the current velocity of the car. This may help in then building a controller that, by levering the model prediction, is able to perform hazardous maneuver successfully or to cruise at a very fuel-efficient pace. Having a good model of the reality, from physics to engineering, makes many engineers busy (and happy).

However, finding good models of anything, from car dynamics to quantum physics is complicated and requires validation and many iterations. For example, we could model something completely from first principle. Modeling something from first principles involves creating a theoretical framework based on fundamental laws, principles, and assumptions, without relying on empirical data. The experimental data are only used to validate the model.

On the other hand, we could also model something completely from a data-driven point of view where the underlying phenomena is too complicated to model from a first principle point of view. In this case we can just rely on generic function approximators, such as Neural Networks or Gaussian Process, and regress the model given a set of data.

Unfortunately, when fitting data-driven models, we are facing many challenges. For example, when modeling the output acceleration we many record data with an accelerometer that ouputs very noisy measure, makeing the inference of the true car acceleration complicated. Moreover, since during execution the car velocity and the throttle will never perfectly match any of the recorded data, how should the regressor interpolate the prediction correctly. Last but not least, what should be the confidence of the model in making prediction when there are not data evidence or when the prediction is too noisy?

Let's understand this!

# Chapter 2

# Linear Regression

## 2.1 Ordinary Linear Regression

(Ordinary) Linear Regression is a statistical model which estimates the linear relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables).

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}\mathbf{w} \tag{2.1}$$

where $\boldsymbol{x} \in \mathbb{R}^{N\times l}$ , $\boldsymbol{y} \in \mathbb{R}^{N\times o}$ and $\mathbf{w} \in \mathbb{R}^{l\times o}$ where $l$ is the number of input features and $o$ is the number of output features and $N$ is the number of observations.

The least share solution of the parameters is

$$\mathbf{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}. \tag{2.2}$$

The matrix to invert has dimension $l \times l$ and a cost $\mathcal{O}(l^3)$.

## 2.2 Kernelized Linear Regression

Kernelized linear regression is a technique that applies the kernel trick to linear regression. It transforms the input features into a higher-dimensional space $\boldsymbol{\phi}$, allowing for non-linear relationships to be captured. The regression is then performed in this transformed space, effectively modeling non-linear patterns in the data while still using a linear model. This approach can be computationally expensive for large datasets due to the need to compute pairwise kernel evaluations. Mathematically,

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{\phi}(\boldsymbol{x})) = \boldsymbol{\phi}(\boldsymbol{x})\mathbf{w} \tag{2.3}$$

where $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^{N\times L}$ , $\boldsymbol{y} \in \mathbb{R}^{N\times o}$ and $\mathbf{w} \in \mathbb{R}^{L\times o}$ where $L$ is the number of new projected input features and $o$ is the number of output features and $N$ is the

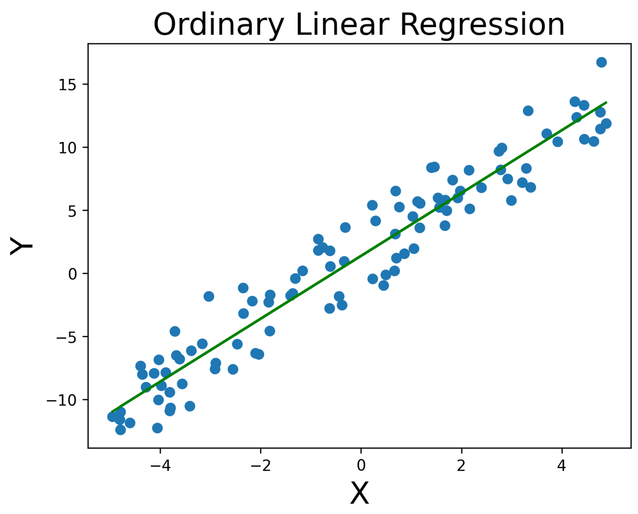


Figure 2.1: Example of Linear Regression on noisy data

number of observations. If the number of extracted features is very big, the calculation of the matrix $\mathbf{w}$, i.e.,

$$\mathbf{w} = (\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{\phi}(\boldsymbol{X}))^{-1}\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{y}. \tag{2.4}$$

will be intractable given the inversion of the matrix. However, given that we only have $N$ data points, the solution parameters solution must be correlated between them and the maximum N parameters can be independent (per output). This is why we can write $\mathbf{w}$ as a linear combination of the data points, such as:

$$\mathbf{w} = \boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{\alpha} \tag{2.5}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^{Nxo}$. Given the definition of $\mathbf{w}$ and the least square solution, we can write that

$$(\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{\phi}(\boldsymbol{X}))^{-1}\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{y} = \boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{\alpha} \tag{2.6}$$

$$(\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{\phi}(\boldsymbol{X}))\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{\alpha} = \boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{y} \tag{2.7}$$

$$\boldsymbol{\phi}(\boldsymbol{X})\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{\phi}(\boldsymbol{X})\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{\alpha} = \boldsymbol{\phi}(\boldsymbol{X})\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{y} \tag{2.8}$$

If we define $\boldsymbol{K} = \boldsymbol{\phi}(\boldsymbol{X})\boldsymbol{\phi}(\boldsymbol{X})^\top$ then

$$\boldsymbol{\alpha} = \boldsymbol{K}^{-1}\boldsymbol{y} \tag{2.9}$$

hence to make prediction on a new point

$$\boldsymbol{f}(\boldsymbol{x}_*) = \boldsymbol{\phi}(\boldsymbol{x}_*)\boldsymbol{\phi}(\boldsymbol{X})^\top \boldsymbol{K}^{-1}\boldsymbol{y} \tag{2.10}$$

$$\boldsymbol{f}(\boldsymbol{x}_*) = \boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X})\boldsymbol{K}^{-1}\boldsymbol{y} \tag{2.11}$$

By writing the dot product between a possibility (infinitely) big $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\phi}(\boldsymbol{x}')$, as a nonlinear function $k(\boldsymbol{x}, \boldsymbol{x}')$ has a significant computational advantage. This is called the **kernel trick**. We can summarize and say that the kernel trick allows algorithms to implicitly operate in a higher-dimensional space by computing the dot product between data points in this space without actually having to compute the transformation of the data points into that higher-dimensional space. Finding the kernel function that corresponds to a particular $\phi$ vector is not simple and vice versa. However, since the dot product is an operator of similarity between vectors, we can say that the kernel between two inputs is telling us how similar the projected vectors would be in these other spaces. The idea on how two vectors are similar in this infinite dimensional space is going to be of use later when we are going to use the kernel to build our covariance matrix in a Gaussian Process. Figure 2.2, shows how we can fit non linear data by using a linear regression technique, thanks to the use of a non linear kernel.

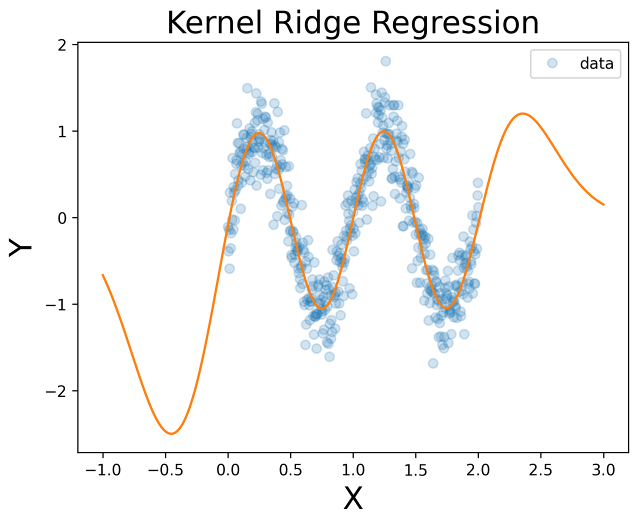


Figure 2.2: Fitting a non-linear function using kernelized linear regression where the kernel is a square exponential kernel.

## 2.3 Bayesian Linear Regression

Bayesian linear regression is a statistical approach to linear regression that incorporates Bayesian principles. Instead of providing a single point estimate for the regression coefficients, it provides a probability distribution over possible values of the coefficients. This distribution is updated based on prior beliefs about the coefficients and observed data, allowing for uncertainty to be quantified. Mathamatically,

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}\mathbf{w} \tag{2.12}$$

or

$$p(\boldsymbol{f}|\boldsymbol{x}, \mathbf{w}) = \mathcal{N}(\boldsymbol{x}\mathbf{w}, 0) \tag{2.13}$$

We assume that the observation of the data is the true function plus as Gaussian noise, i.e.,

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon \tag{2.14}$$

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{2.15}$$

This also means that the likelihood of observing the data is also gaussian distributed with linear mean and a diagonal covariance matrix, i.e.,

$$p(\boldsymbol{y}|\boldsymbol{X}, \mathbf{w}) = \mathcal{N}(\boldsymbol{X}\mathbf{w}, \sigma_n^2 I) \tag{2.16}$$

In order to find the probability distribution of the parameters, starting from a prior distribution, i.e.,

$$\mathbf{w} \sim \mathcal{N}(0, \Sigma_p) \tag{2.17}$$

where $\Sigma_p$ is the prior covariance matrix, we can use Bayes theorem

$$p(\mathbf{w}|\boldsymbol{y}, \boldsymbol{X}) = \frac{p(\boldsymbol{y}|\boldsymbol{X}, \mathbf{w})p(\mathbf{w})}{p(\boldsymbol{y}|\boldsymbol{X})}. \tag{2.18}$$

To predictions, we can marginalize

$$p(\boldsymbol{f}_*|\boldsymbol{X}) = \int p(\boldsymbol{f}_*|\mathbf{w}, \boldsymbol{X})p(\mathbf{w}|\boldsymbol{y}, \boldsymbol{X})d\mathbf{w} \tag{2.19}$$

Figure 2.3, shows the prediction of the linear regression with the predicted uncertainty bounds.

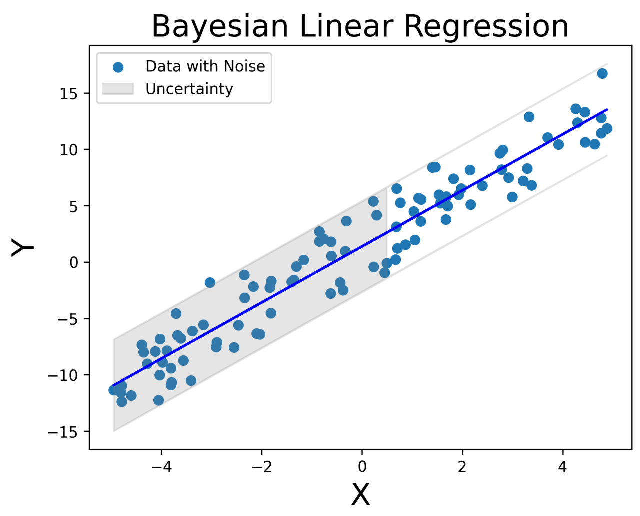


Figure 2.3: Fitting of noisy data using Bayesian Linear Regression

# Chapter 3

# Properties of Gaussian Distributions

## 3.1 Definition Gaussian

The probability density function (PDF) of a univariate Gaussian distribution with mean $\mu$ and variance $\sigma^2$ is given by:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The probability density function (PDF) of a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is given by:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

where $\mathbf{x}$ is the vector of random variables, $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Sigma}$ is the covariance matrix, and $d$ is the dimensionality of $\mathbf{x}$.

## 3.2 Gaussian Conditioning

Given a joint probability $p(\boldsymbol{f}, \boldsymbol{y})$ to be Gaussian, i.e.,

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_f} \\ \boldsymbol{\mu_y} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma_{ff}} & \boldsymbol{\Sigma_{fy}} \\ \boldsymbol{\Sigma_{yf}} & \boldsymbol{\Sigma_{yy}} \end{bmatrix}\right). \tag{3.1}$$

then also we can compute the distribution of any of the vector variables $\boldsymbol{f}$ and $\boldsymbol{y}$ by conditioning the joint distribution and we could prove that the conditioned distribution is still normally distributed, i.e.,

$$\boldsymbol{f}|\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu_{f|y}}, \boldsymbol{\Sigma_{f|y}})$$

where,

$$\boldsymbol{\mu}_{\boldsymbol{f}|\boldsymbol{y}} = \boldsymbol{\mu}_{\boldsymbol{f}} + \boldsymbol{\Sigma}_{\boldsymbol{f}\boldsymbol{y}}\boldsymbol{\Sigma}_{\boldsymbol{y}\boldsymbol{y}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}})$$
$$\boldsymbol{\Sigma}_{\boldsymbol{f}|\boldsymbol{y}} = \boldsymbol{\Sigma}_{\boldsymbol{f}\boldsymbol{f}} - \boldsymbol{\Sigma}_{\boldsymbol{f}\boldsymbol{y}}\boldsymbol{\Sigma}_{\boldsymbol{y}\boldsymbol{y}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{y}\boldsymbol{f}}.$$

For the proof, look at [1].

The conditioning of a Gaussian has a central role in the computation of the predictive distribution when learning a Gaussian Process.

## 3.3 Gaussian Marginalization

$$p(\boldsymbol{y}) = \int p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}) \tag{3.2}$$

If $p(\boldsymbol{y}|\boldsymbol{f})$ is $\mathcal{N}(\boldsymbol{A}\boldsymbol{f}, \boldsymbol{\Sigma}_y)$ and $p(\boldsymbol{f})$ is $\mathcal{N}(\boldsymbol{\mu}_f, \boldsymbol{\Sigma}_f)$ then

$$p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{A}\boldsymbol{\mu}_f, \boldsymbol{A}\boldsymbol{\Sigma}_f\boldsymbol{A}^\top + \boldsymbol{\Sigma}_y) \tag{3.3}$$

For the proof, look at [1]. This single equation explains many of the equation of GPs. For example, if $p(\boldsymbol{y}|\boldsymbol{f})$ is $\mathcal{N}(\boldsymbol{f}, \boldsymbol{I}\sigma_n^2)$ and $\mathcal{N}(0, \boldsymbol{\Sigma}_f)$ we obtain that

$$p(\boldsymbol{y}) = \mathcal{N}(0, \boldsymbol{\Sigma}_f + \boldsymbol{I}\sigma_n^2) \tag{3.4}$$

that is the solution we used when computing the marginal likelihood to optimize the hyperparameters of the kernel.

Similarly, when we computed

$$q(\boldsymbol{f}) := \int p(\boldsymbol{f}|\boldsymbol{u})q(\boldsymbol{u})d\boldsymbol{u}, \tag{3.5}$$

we can compute $p(\boldsymbol{f}|\boldsymbol{u})$ using the conditioning, i.e. $\boldsymbol{f}|\boldsymbol{u}$,

$$p(\boldsymbol{f}|\boldsymbol{u}) = \mathcal{N}K(X,Z)K(Z,Z)^{-1}f, K(X,X) - K(X,Z)K(Z,Z)^{-1}K(Z,X))$$

so in this circumstance $\boldsymbol{A} := K(X,Z)K(Z,Z)^{-1}$ and

$\Sigma_f := K(X,X) - K(X,Z)K(Z,Z)^{-1}K(Z,X)$. If $q(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{m}, \boldsymbol{S})$ and using the definition of marginalization of Gaussians 3.3 we obtain the definition for the prediction of an SVGP, Eq. 8.22. You can try it yourself.

## 3.4 Bayes Theorem with Gaussians

Bayes theorem tells us that

$$p(\boldsymbol{f}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}) \tag{3.6}$$

however, if the likelihood probability and the prior probability are normally distributed, then also the posterior are. So, if $p(\boldsymbol{y}|\boldsymbol{f})$ is $\mathcal{N}(\boldsymbol{A}\boldsymbol{f}, \boldsymbol{\Sigma}_y)$ and $p(\boldsymbol{f})$ is $\mathcal{N}(\boldsymbol{\mu}_f, \boldsymbol{\Sigma}_f)$ then

$$\boldsymbol{f}|\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{R}(\boldsymbol{y} - \boldsymbol{A}\mu_f) + \mu_f, \boldsymbol{\Sigma}_f - \boldsymbol{R}\boldsymbol{A}\boldsymbol{\Sigma}_f^\top) \tag{3.7}$$

where $\boldsymbol{R} = \boldsymbol{\Sigma}_f \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{\Sigma}_f \boldsymbol{A}^\top + \boldsymbol{\Sigma}_{\boldsymbol{y}})^{-1}$. For the proof, look at [1].

When we used Bayes theorem was to find the posterior $\boldsymbol{f}$ given the data $\boldsymbol{y}$ and knowing that $p(\boldsymbol{y}|\boldsymbol{f}) = \mathcal{N}(\boldsymbol{f}, \boldsymbol{I}\sigma_n^2)$ and $p(\boldsymbol{f}) = \mathcal{N}(0, K(\boldsymbol{X}, \boldsymbol{X}))$. Then by setting $\mu_f = 0$, $\boldsymbol{A} = \boldsymbol{I}$, $\boldsymbol{\Sigma}_{\boldsymbol{y}} = \boldsymbol{I}\sigma_n^2$ and $\boldsymbol{\Sigma}_f = K(\boldsymbol{X}, \boldsymbol{X})$ we obtain

$$\boldsymbol{f}|\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{R}\boldsymbol{y}, K(\boldsymbol{X}, \boldsymbol{X}) - \boldsymbol{R}K(\boldsymbol{X}, \boldsymbol{X})^\top) \tag{3.8}$$

where $\boldsymbol{R} = K(\boldsymbol{X}, \boldsymbol{X})(K(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{I}\sigma_n^2)^{-1}$. That is exactly the equation found for the posterior distribution.

## 3.5 Further Reading

1. Bayes' Theorem for Gaussians [1]

# Chapter 4

# Exact Gaussian Process

When learning a model we are estimating the parameters from a (finite) amount of data but are often required to act on a continuous space, e.g., when predicting the robot acceleration in previously unknown states. Therefore, fitting a continuous function to approximate the data is necessary. Since we do not have an infinite amount of data, assumptions on the function need to be made, e.g. linear, non-linear, parametric, non-parametric, etc.

The use of function approximators is not only specific to supervised learning but to any learning field that requires the mapping between inputs and outputs. The difference in how the function is shaped separates different learning fields. Many different regressors could be interpreted as a particular case of a unified model [5], showing the usability of different function approximations.

Since the focus of this lecture is to quantify and exploit uncertainties to increase efficiency and efficacy in robot learning, this chapter will introduce the basic concepts of Bayesian Learning with a particular focus on Gaussian Process Regression. The goal will be to quantify two types of uncertainties, aleatoric and epistemic.

**Aleatoric uncertainty** is also known as stochastic uncertainty and is representative of unknowns that differ each time we run the same experiment. Aleatoric is derived from the Latin alea or dice, referring to a game of chance.

**Epistemic uncertainty** is also known as systematic uncertainty, and is due to things one could in principle know but does not in practice. This may be because a measurement of particular data has been deliberately hidden, i.e. there is not enough data to support a confident prediction.

## 4.1 Gaussian Process Regression

A Gaussian Process (GP) is a generalization of a Gaussian distribution over functions. In other words, a Gaussian process defines a distribution over functions, where any finite number of points from the function's domain follows a multivariate Gaussian distribution. If we want to find this distribution in a fully Bayesian way, then we must define a prior distribution over all the possible functions.

### 4.1.1 Prior

The prior is typically specified as a mean function and a covariance function. The prior distribution represents our beliefs about the functions before observing any data. Our prior belief is that our function is a sample of a Gaussian Process defined by a mean function and a kernel matrix, i.e.,

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')) \tag{4.1}$$

where $k(\boldsymbol{x}, \boldsymbol{x}')$ is the kernel function and $m(\boldsymbol{x})$ is the mean function. Eq. 4.1, is the generalization of a discrete case,

$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma}) \tag{4.2}$$

that can be read as "the vector f is a sample of a multivariate Gaussian distribution with mean $\boldsymbol{m}$ and covariance matrix $\boldsymbol{\Sigma}$ ". In other words, the Gaussian Process is a generalization of the multivariate Gaussian distribution defined over function (not vectors).

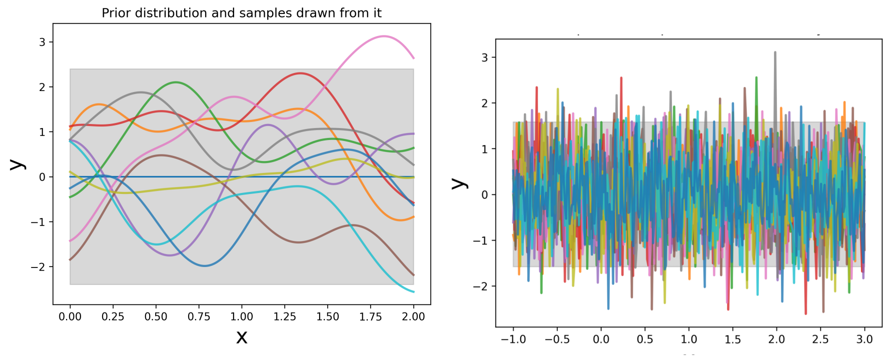
Practically speaking when defining a prior Gaussian distribution, we must define the mean vector and the covariance matrix, while for a prior Gaussian Process we have to define the mean function [1] and the kernel function, which will be used to then build the covariance matrix later on, when creating a multivariate Gaussian distribution out of a Gaussian Process to take into account the discrete nature of the data.

Figure 4.1 shows the prior distribution of the function values f for different input locations x. The shaded area shows the uncertainty of the function value at a certain input position. The coloured outputs are the samples drawn from the Gaussian Process. Although the shaded area of the two Figures may look similar, the samples drawn from them are very different. The reason is that the left prior distribution has a covariance matrix $\Sigma$ [2] has non-zero extra diagonal terms while the right one has only zero extra diagonal terms. When a multivariate Gaussian distribution has a diagonal covariance matrix, it means that all the elements of the distributions are independent. Sampling one time from a $n$-dimensional multivariate Gaussian will generate $n$ independent samples, that could have been sampled independently: we are sampling noise. On the other hand, when the elements on the extra diagonal terms are nonzeros, it means

[1] In the absence of any prior knowledge, the mean function is usually set to zero.

[2] built using the kernel, it is going to be cleared later how

that the different values of the function are correlated, resulting in samples that look smooth; see Sec. 4.3 to understand how to sample from a multi-variate Gaussian distribution. Each of the colored functions is a sample drawn from the Gaussian Process prior. If we now discretize each of the samples in n points, i.e. $f = [f_1, f_2, f_3, \dots f_n]$, the correlation between each of these points would be given by the kernel functions, i.e. $corr(f1, f2) = k(x1, x2)$. As you can see the correlations between different points of the (sampled) functions are given by the kernel function that uses as input the input locations where we are evaluating the function values. Different examples of kernel functions are introduced in Chapter 5. Chapter 6 will show how to choose the kernel function and how the hyperparameters can be optimally determined. In the following section of this chapter, we assume that we have chosen and optimized the kernel already and use the Bayes theorem to the probability distribution of the posterior, i.e. of the distribution of the function after having observed some data values.



(a) The covariance matrix has non-zero extra diagonal terms.

(b) The covariance matrix has only zero extra diagonal terms.

Figure 4.1: Gaussian Process Prior. The colored functions are samples drawn from the distribution.

### 4.1.2 Gaussian Likelihood

In order to make predictions, we need to define a likelihood function that captures the probability of observing the data given the function values at specific points. The likelihood function in exact GP is assumed to be Gaussian, i.e.

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{4.3}$$

$$p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{X}) \sim \mathcal{N}(f(\boldsymbol{X}), \sigma_n^2 \boldsymbol{I}) \tag{4.4}$$

Differently than of the GP prior, in this case, we are actually assuming that the measurements $y$ are going to be affected by noise, which is why the likelihood function $p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{X})$ is model as a Gaussian that has the mean equal $f$ and a covariance matrix that is the identity matrix times the *likelihood noise*, $\sigma_n$.

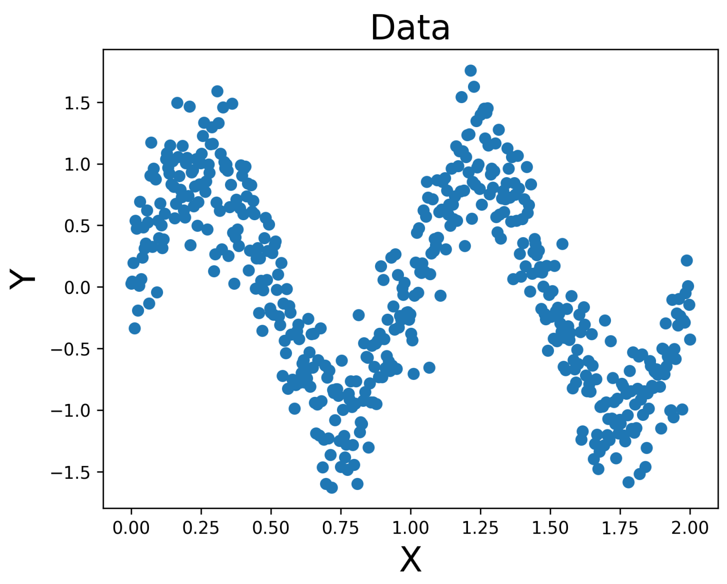


Figure 4.2: Noisy data given as labels to our model

### 4.1.3 Posterior

The posterior combines the prior and the likelihood, which captures the probability of observing the data given the function values at specific points, according to:

$$\overbrace{p(\boldsymbol{f}|\boldsymbol{y},\boldsymbol{X})}^{\text{posterior}} = \frac{\overbrace{p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{X})}^{\text{likelihood}}\overbrace{p(\boldsymbol{f}|\boldsymbol{X})}^{\text{prior}}}{\underbrace{p(\boldsymbol{y}|\boldsymbol{X})}_{\text{marginal likelihood}}} = \mathcal{N}(\boldsymbol{\mu_f},\boldsymbol{\Sigma_f}) \tag{4.5}$$

If the prior and the likelihood are Gaussian distributed, then the posterior is also going to be Gaussian distributed. If the posterior distribution is in the same probability distribution family as the prior probability distribution, the prior and posterior are then called conjugate distributions, and the prior is called a conjugate prior for the likelihood function [3].

The data induce a posterior GP which is specified by a posterior mean function and a posterior covariance function:

$$\boldsymbol{\mu_f} = \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} \tag{4.6}$$

$$\boldsymbol{\Sigma_f} = \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) - \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) \tag{4.7}$$

[3]`https://www.wikiwand.com/en/Conjugate_prior`

To compute the distribution that describes our sampled data $\boldsymbol{y}$ we can solve this integral:

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{y}) = \int p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})d\boldsymbol{f} = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{y}},\boldsymbol{\Sigma}_{\boldsymbol{y}}) \tag{4.8}$$

$$\boldsymbol{\mu}_{\boldsymbol{y}} = \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} \tag{4.9}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{y}} = \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2 - \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})(\boldsymbol{K} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) \tag{4.10}$$

Figure 4.3, shows the posterior distribution of the function $\boldsymbol{f}$ and of $\boldsymbol{y}$. The mean prediction is the same while the uncertainty of y is much larger since we are also adding the likelihood noise to it. It is worth mentioning that $f$ shows the prediction of what is the most likely function before corrupting it with noise during the measurements, while $y$ shows the actual distribution of the data we observed.

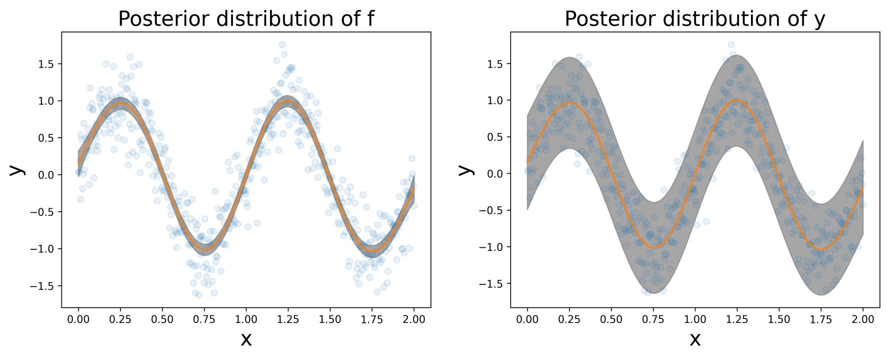


Figure 4.3: Posterior Distribution of $\boldsymbol{f}$ (left) and $\boldsymbol{y}$ (right).

## 4.2 Prediction

Given the calculation of the posterior distribution, the prediction of new output values can be computed as:

$$p(\boldsymbol{f}_*|\boldsymbol{y},\boldsymbol{X}) = \int p(\boldsymbol{f}_*|\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{y},\boldsymbol{X}) \tag{4.11}$$

where we are marginalizing over the posterior distribution of the function, i.e. $p(\boldsymbol{f}|\boldsymbol{y},\boldsymbol{X})d\boldsymbol{f}$, given our the prior model, i.e. $p(\boldsymbol{f}_*|\boldsymbol{f})$.

Considering that $p(\boldsymbol{f}_*,\boldsymbol{f})$ is a Multivariate Gaussian Distribution by definition, then $p(\boldsymbol{f}_*|\boldsymbol{f})$ can be obtained by conditioning with respect to $\boldsymbol{f}$. The distribution of $\boldsymbol{f}$ can be computed with the posterior distribution given the evidence of the data. Given that all the terms of the integral are Gaussians, the integral is also Gaussian. So, if $p(\boldsymbol{f}_*|\boldsymbol{y}, \boldsymbol{X})$ is Gaussian then $p(\boldsymbol{f}_*, \boldsymbol{y}|\boldsymbol{X})$ is a multivariate Gaussian distribution defined as

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I} & \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \\ \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) \end{bmatrix}\right). \tag{4.12}$$

Hence, to make predictions, the mean and the variance of the posterior distribution can be computed as the conditional mean and variance of the multivariate Gaussian distribution, according to:

$$\boldsymbol{\mu}_{\boldsymbol{f}_*} = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} \tag{4.13}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{f}_*} = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \tag{4.14}$$

where $\boldsymbol{X}_*$ are the prediction inputs and $\boldsymbol{X}$ and $\boldsymbol{y}$ are the training input and outputs and the correlations $\boldsymbol{K}$ are computed using a kernel function.

The inversion of the covariance matrix scales with a computation cost of $\mathcal{O}(n^3)$, where $n$ is the number of data points. However, some approximation techniques can be used to reduce the computational cost, see Section 8.

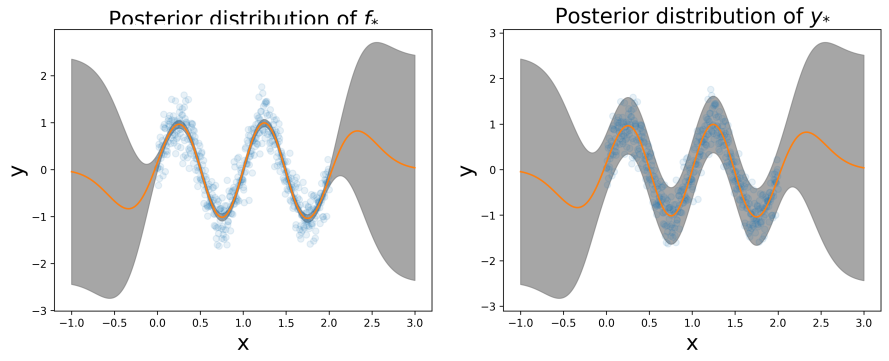


Figure 4.4: Prediction of Gaussian Process posterior and posterior plus likelihood noise

In this plot, we also show the prediction of the Gaussian Process for a range of input that goes beyond the training set. It is possible to notice that the average converges to the zero mean prior when far away from the data and that the *epistemic* uncertainty (lack of knowledge) adds up to the *aleatoric* uncertainty (due to the sensor noise).

## 4.3 Sampling from a Gaussian Process

To sample from a multivariate Gaussian Process the the Cholesky decomposition of the covariance matrix $\boldsymbol{\Sigma}$ is calculated such that

$$\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{\Sigma}$$

where $\boldsymbol{L}$ is a lower triangular matrix. The sample $\boldsymbol{y}_{sample}$ by multiplying $\boldsymbol{z}$, a vector of independent standard normal random variables $\boldsymbol{z}$ is a sample from a Gaussian Distribution, with the Cholesky matrix $\boldsymbol{L}$ and adding the mean vector $\boldsymbol{\mu}$, i.e.,

$$\boldsymbol{y}_{sample} = \boldsymbol{L}\boldsymbol{z} + \boldsymbol{\mu} \tag{4.15}$$

where

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{4.16}$$

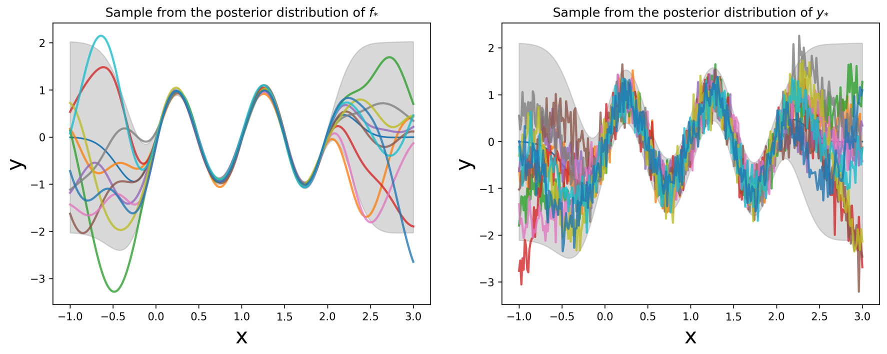


Figure 4.5: Samples from the posterior distribution of $\boldsymbol{f}_*$ and $\boldsymbol{y}_*$

## 4.4 Further Reading

1. A Visual Exploration of Gaussian Processes
2. Chapter 2 of Gaussian Process for Machine Learning [8]

# Chapter 5

# Kernel Design

## 5.1 Kernel Definition

The kernel function specifies the covariance between the function values at two points in the input space. The assumption is that function values that belong to points that are closer to each other in the input space are more likely to be similar. Hence, the kernel function can be interpreted as a measure of similarity between two points in the input space. Given the constraint on the covariance matrix to be semi-positive definite, the kernel function cannot be any function.

### 5.1.1 Squared Exponential Kernel

The most common kernel functions are the squared exponential (SE) kernel and the Matern kernel. The SE kernel is infinitely differentiable and it is defined as:

$$k_{SE}(x_i, x_j) = \sigma_v^2 \exp\left(-\frac{1}{2}\left(\frac{(x_i - x_j)^2}{\ell^2}\right)\right) \tag{5.1}$$

where $D$ is the dimensionality of the input space, $x_i$ and $x_j$ are data points in the input space and $\sigma_v^2$ and $\ell_d$ are respectively the vertical (or output) lengthscale and the horizontal lengthscale. The vertical and the horizontal lengthscales are the hyperparameters of the kernel function and they are optimized during the training phase, see Chapter 6.

**Generalization of the SE kernel is the Matern kernel** The Matern kernel is defined as

$$k(x_i, x_j) = \frac{1}{\Gamma(\nu)2^{\nu-1}} \frac{\sqrt{2\nu}}{l}(x_i - x_j)^{2\nu} K_\nu \frac{\sqrt{2\nu}}{l}(x_i - x_j)^2) \tag{5.2}$$

where $K_\nu$ is a modified Bessel function and $\Gamma$ is the gamma function. where $K_\nu$ is a modified Bessel function and $\Gamma$ is the gamma function.

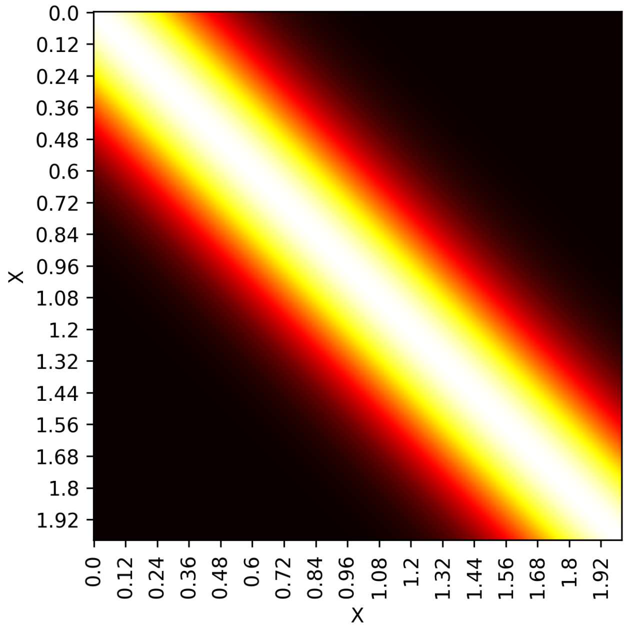


Figure 5.1: Square exponential kernel. The color intensity is proportional to the vertical lengthscale and the width of the band is proportional to the horizontal lenghtscale

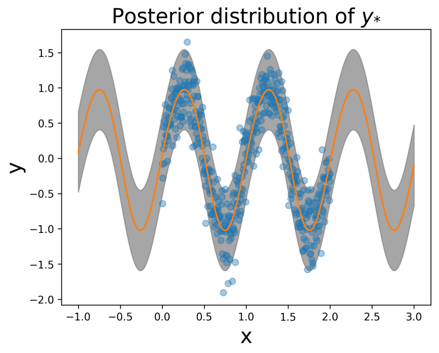
The kernel has an additional parameter $\nu$ which controls the smoothness of the resulting function. The smaller $\nu$, the less smooth the approximated function is. As $\nu$ goes to infinity, the kernel becomes equivalent to the square exponential kernel. Important intermediate values are $\nu = 1.5$ (once differentiable) and $\nu = 2.5$ (twice differentiable).

### 5.1.2 Periodic Kernel

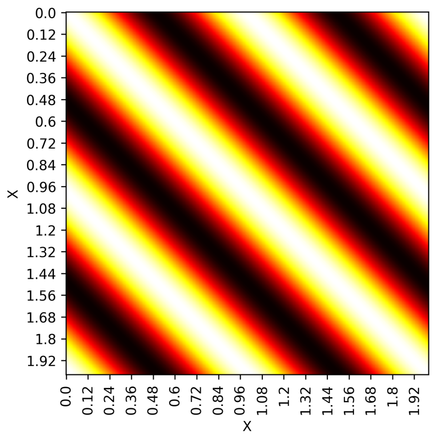
The periodic kernel (derived by David Mackay) allows one to model functions which repeat themselves exactly. The kernel definition is

$$k_{periodic}(x_i, x_j) = \exp\left(-\frac{2\sin^2(\pi(x_i - x_j)^2/p)}{\ell^2}\right) \tag{5.3}$$

where the period $p$ simply determines the distance between repetitions of the function. The lengthscale $\ell$ determines the lengthscale function in the same way as in the SE kernel.



(a) Caption for the first plot



(b) Caption for the second plot

Figure 5.2: Fitting of a noisy sine using a periodic kernel. The kernel matrix shows that inputs are correlated with them close to themselves and the ones that are a distance of a certain periodicity. The prediction of the posterior now does not converge to the prior when going far away from the data but keeps the periodic prediction.

### 5.1.3 Linear Kernel

When the underlying relationship between the input variables and the output is believed to be linear, the linear kernel can be quite effective. In such cases, the linear kernel can capture the linear dependencies between input features. We define the linear kernel as:

$$k_{lin}(x_i, x_j) = \sigma_0^2 + x_i \cdot x_j \tag{5.4}$$

A linear kernel is usually used in combination with other kernels.

### 5.1.4 Kernel Composition

It is possible to sum and multiply kernels between them to create a more complex prior distribution that would generalize better outside the region where data are observed. For example, we could create a kernel that is the sum of linear and periodic kernels, i.e.

$$k_{sum} = k_{lin} + k_{periodic} \tag{5.5}$$

Figure 5.3, shows the prediction of the gaussian process when using the aforementioned kernel combination.

### 5.1.5 Deep Kernel Learning

$$k(x_i, x_j) = \sigma_h^2 \exp\left(-\frac{1}{2}\left(\frac{(\psi(x_i) - \psi(x_j))^2}{\ell^2}\right)\right) \tag{5.6}$$

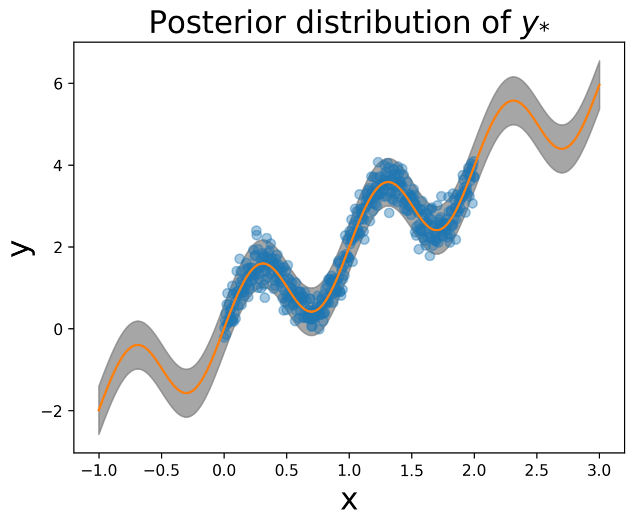


Figure 5.3: Fitting using a combination of linear and sinusoidal kernel

where $\psi$ can be any nonlinear regressor and can be trained with any other of the parameters of the kernel. It has been shown that using of a generic non-linear function to project the input to a latent manifold keeps the kernel to be well defined [2]. Very successful combination of multi-layer-perceptrons and conventional neural networks with Gaussian Process where investigate in the literature to scale the use of GP on high-dimensional inputs like images [9].

# Chapter 6

# Hyperparameters' optimization

The kernel hyperparameter choice and the likelihood noise value will completely change the fitting behavior of the GP. Differently from the least square estimation of the parameters, which tries to minimize the residuals on the output prediction, we rather try to find the parameters that would maximize the probability of sampling our data from the prior distribution. If we consider the square exponential kernel and a Gaussian likelihood function, in the optimization process we need to estimate the horizontal lengthscale, vertical lengthscale, and the likelihood noise. To compute the probability of the data we use the marginal likelihood that is the integral along all the possible output functions sampled from our prior distribution.

## 6.1 Marginal Likelihood Maximization

The marginal likelihood is defined as

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{X})p(\boldsymbol{f}|\boldsymbol{X})d\boldsymbol{f} \tag{6.1}$$

is the marginalization of the likelihood of the data over the whole possible function values. This is also known as the evidence of the data given in the model. Intuitively, this means to find how likely was to sample my data from the prior distribution. This measure of evidence is crucial for the optimization of the hyperparameters of the kernel function because it allows finding that set of hyperparameters that would better fit the data, for example, detecting the right energy of the function (vertical length scale) or the right smoothness (horizontal length scale). When dealing with Gaussian Prior and Gaussian Likelihood, the marginal likelihood can be computed analytically and used to optimize the hyperparameters of the kernel function. Given the exponential nature of the Gaussian distribution, the (natural) logarithm of the marginal likelihood can be obtained as the sum:

$$\log(p(\boldsymbol{y}|\boldsymbol{X})) = \underbrace{-\frac{1}{2}(\boldsymbol{y}^\top \boldsymbol{K}_y^{-1}\boldsymbol{y})}_{\text{data fit cost}} - \underbrace{\frac{1}{2}\log|\boldsymbol{K}_y|}_{\text{complexity term cost}} - \frac{n}{2}\log(2\pi) \tag{6.2}$$

where $\boldsymbol{K}_y = \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I}$ and $n$ is the number of data points. The best set of hyperparameters for the kernel is obtained by the maximization of the marginal log-likelihood (or minimizing the negative marginal log-likelihood). The use of the logarithm allows us to write the product as a sum and to remove the exponential in the gaussian distribution. The marginal likelihood will still tend to favor the least complex model able to explain the data. Practically speaking the optimization process always tries to make the horizontal length scale as big as possible but without reaching the point that the model is too rigid and not able to fit the data well.

Figure 6.1, shows the resulting fitting when we have too little horizontal lenghtscale (on the left) or too big one (on the right). In the first case the model, is very flexible, i.e. it would be able to fit a very non linear function, and we can capture that from the samples drawn from the posterior distribution. On the other hand, in the second case, the model is very rigid, ad all the data are just considered as noise. The marginal likelihood maximisation gives us a cost fuction to optimize and find the best kernel hyperparameters that would describe well the property of our data. However, as any optimization process, we may end up in local minima.

Figure 6.2, shows that when we have only 20 data points, and running the parameter optimization for 20 times, half of the time the solution converges to a good fitt and the other half to just considering the data as pure noise. The more the data, the more stable the kernel parameters optimization will be. Figure 6.3 shows that with 40 points already the kernel parameters consistently converges to a good fit of the noisy sine that generated them.

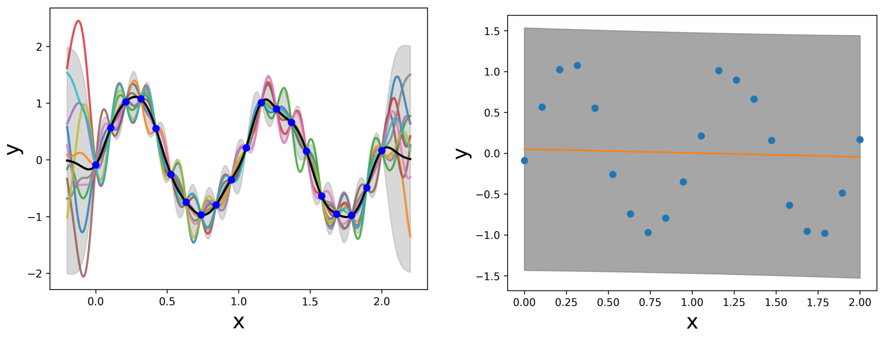
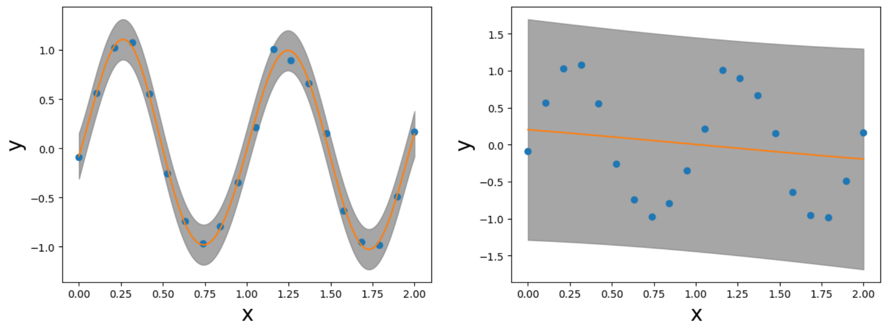


Figure 6.1: Different fitting results with different kernel hyper-parameters. The left plot has a very small horizontal lengthscale, i.e. the model function can be very non-linear, and the right one has a very big length scale, the function can be only very right, it can only fit (almost) linear methods, any oscillations of the data will be considered as noise.



(a) Optimization converges to this solution 12 times over 20 initializations

(b) Optimization converges to this solution 8 times over 20 initializations

Figure 6.2: Optimization of kernel hyperparameters may get stuck to different local minima. The left plot converges to realistic values of the kernel hyperparameters, while the right one converges to a too-rigid model, all the non-linearity are considered as noise.

## 6.2 Automatic Relevance Determination

The input values of the kernel can also have more than one feature vectors, i.e. positions in x,y,z of a robot, and different lengthscales, i.e. normalization values can be used in each of the inputs features to scale the different distances in each dimension. This technique is called Automatic Relevance Determination

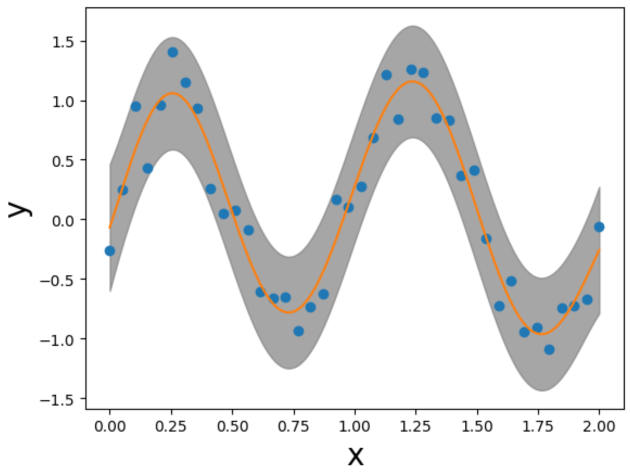


Figure 6.3: With enough samples, the kernel parameters optimization always converge to a good solution.

(ARD). The squared exponential kernel with ARD active looks like this,

$$k(x_i, x_j) = \sigma_v^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{(x_{i,d} - x_{j,d})^2}{\ell_d^2}\right)\right) \tag{6.3}$$

where $D$ is the number of features of the inputs, $x_i$ and $x_j$ are data points in the input space and $\sigma_v^2$ and $\ell_d$ are respectively the vertical lengthscale and the horizontal lengthscale. When the kernel has a different lengthscale for each of the input features, and we optimize the kernel using the marginal likelihood maximization, the optimization process automatically determines the relevance of features in a model, meaning it decides which features are important for predicting the output variable and which can be safely ignored. The optimization process is always trying to find the least complex model (Occam's Razor), hence having one of the lengthscales as big as possible decreases the complexity term without affecting the data fit term. This allows to automatically find which features of the input are not relevant to make predictions. This is why, when a kernel has different lengthscales in different features we say that the Automatic Relevance Determination (ARD) feature is activated.

## 6.3 Questions

1. What are the learnable parameters of a Gaussian Proceess? Are they only the hyperparameters of the kernel?

2. How do we optimize the hyperparameters? Why is it important?

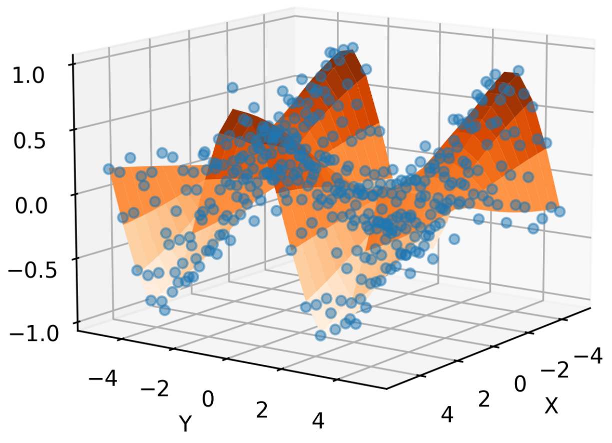


Figure 6.4: Automatic Relevance Determination, Horizontal Lengthscale in x is 7.91, horizontal lengthscale in y is 1.91

## 6.4 Further Reading

1. Kernel Cookbook `https://www.cs.toronto.edu/~duvenaud/cookbook/`
2. Chapter 2.3, Gaussian Process for Machine Learning `https://gaussianprocess.org/gpml/chapters/RW.pdf`

Chapter 7

# Active and Interactive Learning with Gaussian Process

## 7.1 Active Learning with Gaussian Process Regression

Incremental learning is a machine learning paradigm where the model is updated incrementally over time as new data becomes available. The incremental learning process can be divided into two phases: the first phase is the training phase, where the model is trained on a set of data, and the second phase is the incremental phase, where the model is updated with new data. The incremental phase can be further divided into two steps: the first step is the prediction step, where the model is used to predict the output of the new data, and the second step is the update step, where the model is corrected if new labels are provided. The incremental learning process can be used to train a model on a large dataset and then update the model with new data as it becomes available. Active learning is a special case of incremental learning in which a learning algorithm can interactively query a human user (or some other information source), to label new data points with the desired outputs.

Gaussian Process is the perfect candidate to perform active learning given the estimation of the output uncertainty. Imagine wanting to fit a model but labeling the least amount of points to keep the model light and the "supervisor" less busy. The idea is to make predictions on many points of the workspace, such as on a grid, and then only label in the location where the prediction is the most uncertain. This is also the principle behind how Bayesian Optimization works [1].

[1] `https://www.wikiwand.com/en/Bayesian_optimization`

Figure 7.1, depicts the process of active learning of a noisy sine. You can see good fitting already after only as little as twenty samples. In this example, the learner is, after every iteration also estimating the hyperparameters of the kernel, this makes the prediction of the likelihood noise, horizontal and vertical lenghtscale change at every data collection. This is why the uncertainty prediction looks very different in the first steps when we have too little data. This is because there are many local minima in the hyperparameter space. After enough samples, the optimization will always converge to the same (true) values of the kernel hyperparameters.

### 7.1.1 Incremental aggregation of datapoints (post-MLL maximization)

If we consider having enough data to infer the hyperparameters of the kernel already, then, when aggregating more data to the model, only the update of the covariance matrix is performed, while the hyperparameters are kept fixed. To avoid recomputing the inverse of the covariance matrix, a least square update can be performed [7], where given the update covariance matrix,

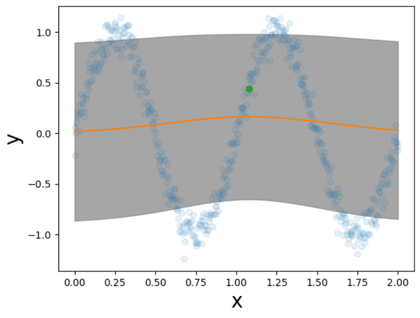
$$\boldsymbol{K}_{n+1} = \begin{bmatrix} \boldsymbol{K}_n(\boldsymbol{X},\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_{new}) \\ \boldsymbol{K}(\boldsymbol{X}_{new},\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_{new},\boldsymbol{X}_{new}) \end{bmatrix} \tag{7.1}$$

the updated inverse covariance matrix becomes:

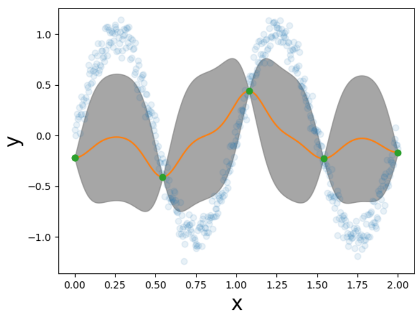
$$\boldsymbol{K}_{n+1}^{-1} = \begin{bmatrix} \boldsymbol{K}_n(\boldsymbol{X},\boldsymbol{X})^{-1} + g\boldsymbol{e}\boldsymbol{e}^\top & -g\boldsymbol{e} \\ -g\boldsymbol{e} & g \end{bmatrix} \tag{7.2}$$

where $\boldsymbol{e} = \boldsymbol{K}_n(\boldsymbol{X},\boldsymbol{X})^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_{new})$ and $g = (\boldsymbol{K}(\boldsymbol{X}_{new},\boldsymbol{X}_{new}) - \boldsymbol{K}(\boldsymbol{X}_{new},\boldsymbol{X})\boldsymbol{e})^{-1}$. In a robotics scenario, where we may explore the workspace to regress a certain function, this (smart) update of the (inverse of the) covariance matrix, allows us to aggregate data to make a confident prediction the next time that point is visited. Of course, we can also decide to not aggregate if the uncertainty on the prediction was little enough.

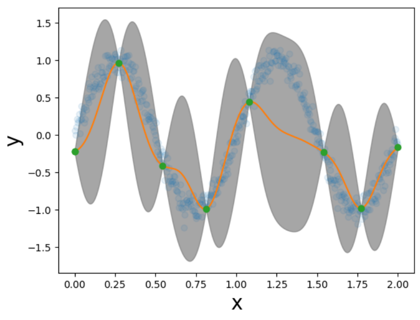
## 7.2 Smart aggregation rule and interactive learning



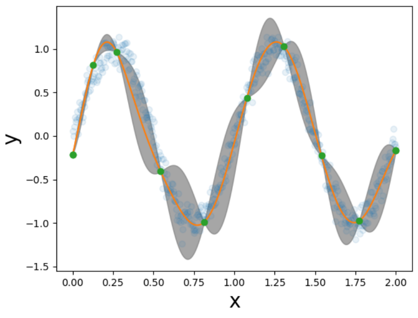
(a) One sample



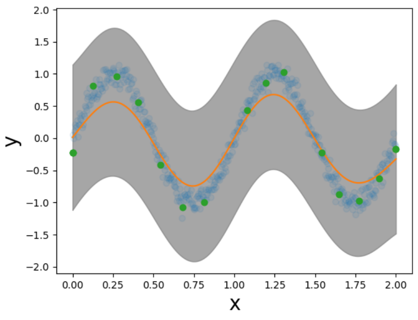
(b) Six samples



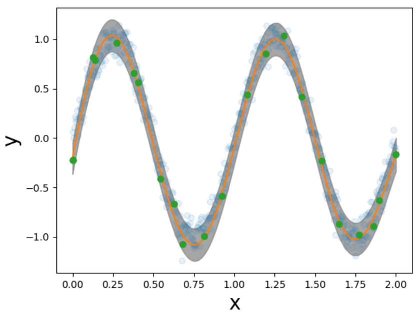
(c) Nine samples



(d) Twelve Samples



(e) Eighteen samples



(f) Twenty four samples

Figure 7.1: Active Learning using Gaussian Process. The kernel is an RBF kernel.

# Chapter 8

# Approximated Gaussian Process

When dealing with many data points, the computational cost of the inversion of the covariance matrix becomes prohibitive. We now seek to find a way to approximate the prior or the posterior that would make the optimization and the prediction with a Gaussian Process faster but without losing accuracy.

## 8.1 Sparse Gaussian Process

We start with the idea of having some extra (latent) variables $\boldsymbol{u}$ at a certain input location $\boldsymbol{Z}$ that we call *inducing variables*, $p(\boldsymbol{f}, \boldsymbol{f_*}, \boldsymbol{u})$ is still gaussian distributed. This extra variable could always be marginalized out given the (consistency) property of Gaussian Process, i.e.

$$p(\boldsymbol{f_*}, \boldsymbol{f}) = \int p(\boldsymbol{f_*}, \boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{u})d\boldsymbol{u} = \int p(\boldsymbol{f_*}|\boldsymbol{f}, \boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{u})d\boldsymbol{u} \tag{8.1}$$

Now, we introduce the fundamental approximation which gives rise to almost all sparse approximations. We approximate the joint prior by assuming that $\boldsymbol{f_*}$ and $\boldsymbol{f}$ are conditionally independent given $\boldsymbol{u}$, i.e.,

$$p(\boldsymbol{f}_*, \boldsymbol{f}) \simeq q(\boldsymbol{f}_*, \boldsymbol{f}) = \int p(\boldsymbol{f}_*|\boldsymbol{u})q(\boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{u})\boldsymbol{du}. \tag{8.2}$$

The name *inducing variable* is motivated by the fact that $\boldsymbol{f}$ and $\boldsymbol{f_*}$ can only communicate through $\boldsymbol{u}$, and then $\boldsymbol{u}$ therefore induces the dependencies between training and test cases. By using this assumption, the joint probability distribution becomes [4],

$$p(\boldsymbol{f}, \boldsymbol{f}_*) \simeq q(\boldsymbol{f}, \boldsymbol{f}_*) = \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{Q}(\boldsymbol{X}, \boldsymbol{X}) & \boldsymbol{Q}(\boldsymbol{X}, \boldsymbol{X}_*) \\ \boldsymbol{Q}(\boldsymbol{X}_*, \boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) \end{bmatrix}\right), \tag{8.3}$$

where $\boldsymbol{K}$ is the prior kernel matrix and $\boldsymbol{Q}$ is the Nyström approximation of the kernel matrix defined as

$$\boldsymbol{Q}(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{Z})\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z})^{-1}\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{x'}) \tag{8.4}$$

where $\boldsymbol{Z}$ is a vector of dimension $m$ with $m << n$, with $n$ being the number of training points. The number and the location of the inducing inputs determine the flexibility of the approximated GP.

**How do we find the approximated kernel matrix?** If $\boldsymbol{u}$ is a sufficient statistic for the data $\boldsymbol{y}$ then we know that when making a prediction for the variance of $\boldsymbol{y}$ given $\boldsymbol{u}$ we obtain

$$\boldsymbol{\Sigma} = \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) - \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{Z})\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z})^{-1}\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{X}) := 0 \tag{8.5}$$

hence the only way to obtain that is to change the prior kernel matrix, i.e.,

$$\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) := \boldsymbol{Q} := \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{Z})\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z})^{-1}\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{X}) \tag{8.6}$$

this explains the definition 8.4.

### 8.1.1 Making prediction with the sparse model

The mean and the covariance of the posterior distribution after modifying the prior using sparse matrices now becomes:

$$\boldsymbol{\mu}_{f_*} = \boldsymbol{Q}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{Q}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} \tag{8.7}$$

$$\boldsymbol{\Sigma}_{f_*} = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{Q}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{Q}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{Q}(\boldsymbol{X}, \boldsymbol{X}_*) \tag{8.8}$$

There is a problem with the inversion of this matrix, i.e. it looks like still have to invert a $(n \times n)$ matrix, where $n$ is the number of training points. However, given the sparse nature of the approximated covariance matrix and using the Woodbury matrix identity [1] , we can write the prediction in a new data point as:

$$\boldsymbol{\mu} = \sigma_n^{-2}\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{Z})\boldsymbol{\Gamma}\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{X})\boldsymbol{y} \tag{8.9}$$

$$\boldsymbol{\Sigma} = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{Q}(\boldsymbol{X}_*, \boldsymbol{X}_*) + \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{Z})\boldsymbol{\Gamma}\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{X}_*) \tag{8.10}$$

where

$$\boldsymbol{\Gamma} = (\sigma_n^{-2}\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{X})\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{Z}) + \boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z}))^{-1}. \tag{8.11}$$

Please notice that after using the Woodbury matrix identity, the final prediction does not have to invert a big matrix of dimension $n \times n$ but only a matrix of dimension $m \times m$. These can generate a big sped-up in the computation of the inverse of the (approximate) covariance matrix. In particular during training, where this inversion has to be computed for any step of the optimization process. However, we still have not discussed how to select the inducing points and in particular, how to optimize their location to maximize the likelihood of the observed data.

[1] `https://www.wikiwand.com/en/Woodbury_matrix_identity`

### 8.1.2 Training the kernel parameters and inducing points location

The inducing points' location can be initialized from a subset of the training samples. For example, if we have thousands of samples, we can select (randomly) one hundred of them and internalize the inducing inputs with that value. However, the initial location may not be ideal, hence we may want to optimize their location by treating them as parameters of the kernel. One could maximize the marginal likelihood (also called the evidence) with respect to $\boldsymbol{Z}$. Remembering that

$$q(\boldsymbol{y}) = \int p(\boldsymbol{y}|\boldsymbol{f})q(\boldsymbol{f})d\boldsymbol{f} \tag{8.12}$$

where $q(\boldsymbol{f}) = \int q(\boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{u})d\boldsymbol{u}$, then,

$$\log(q(\boldsymbol{y}|\boldsymbol{X})) = -\frac{1}{2}\boldsymbol{y}^\top(\boldsymbol{Q}(\boldsymbol{X},\boldsymbol{X})+\sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{Q}(\boldsymbol{X},\boldsymbol{X})+\sigma_n^2\boldsymbol{I}| - \frac{n}{2}\log(2\pi) \tag{8.13}$$

where we must remember that $\boldsymbol{Q}(\boldsymbol{X},\boldsymbol{X})$ is *induced* from $\boldsymbol{Z}$. By maximizing the likelihood, we find the best parameters for the prior distribution, including the inducing inputs' locations, that maximize the likelihood that our data are sampled from. Although this is nice and simple, it is shown in practive that it may lead to overfitting.

Fortunately, as shown in [6], by maximizing also another term, i.e.,

$$-\frac{tr(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) - \boldsymbol{Q}(\boldsymbol{X},\boldsymbol{X}))}{2\sigma_n^2} \tag{8.14}$$

the fitting performance may increase. If the trace is zero, it implies that the inducing variables become sufficient statistics and we can reproduce exactly the full GP prediction. The quantity is a lower bound of the true log marginal likelihood for any value of the inducing inputs, i.e.,

$$\log(p(\boldsymbol{y}|\boldsymbol{X})) \geq \log(q(\boldsymbol{y}|\boldsymbol{X})) - \frac{tr(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) - \boldsymbol{Q}(\boldsymbol{X},\boldsymbol{X}))}{2\sigma_n^2} \tag{8.15}$$

The idea behind finding a true lower bound of the marginal likelihood is that by maximizing the lower bound we are indirectly maximizing both of them. The point is that many times the true marginal likelihood is impossible to calculate. The goal of many statisticians is to find the most computationally, yet close (or tight) lower bound. Additionally, the trace, in the lower bound, corresponds to the squared error of predicting the training latent values $\boldsymbol{f}$ from the inducing variables $\boldsymbol{u}$. When the trace term is zero, the Nyström approximation is exact, which means that the approximated distribution matches exactly the true posterior distribution. Figure 8.1 shows the prediction of the sparse Gaussian Process where only 5 inducing inputs are selected and displayed as red arrows.

Although this is nice and simple, during the lower bound maximization we are still pre and post-multiplying the matrix $\boldsymbol{Q}(\boldsymbol{X},\boldsymbol{X})^{-1}$ by the whole dataset label $\boldsymbol{y}$. The lower bound calculation scales with a complexity cost of $\mathcal{O}(nm^2)$

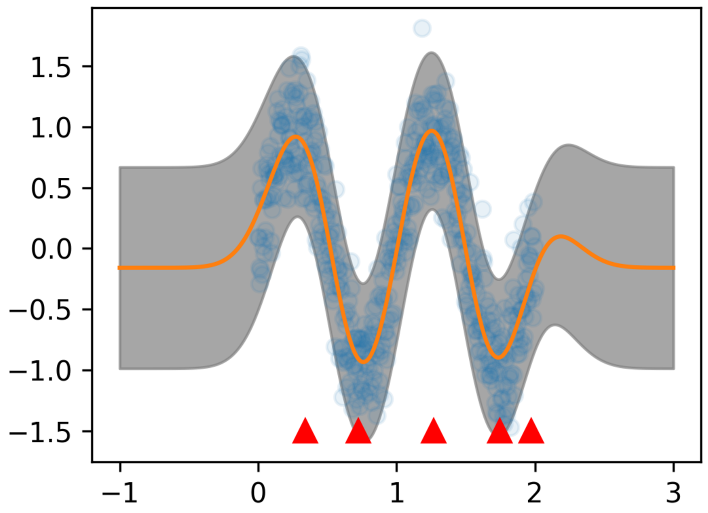


Figure 8.1: Fitting a noisy sine with a Sparse Gaussian Process

so, when having many, i.e. millions, of datapoint, this would not scale. We want to find a method that does not require to use of the whole batch of data for every iteration of the optimization but it can work also with mini-batches of the dataset, like when training a neural network.

## 8.2 Stochastic Variational Gaussian Process

A more computationally scalable bound can be achieved by keeping the additional variational distribution as

$$q(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{m}, \boldsymbol{S}) \tag{8.16}$$

where every inducing output of $\boldsymbol{u}$ correspond to an inducing input $\boldsymbol{Z}$ and $\boldsymbol{m}$ and $\boldsymbol{S}$ are respectively the mean and the covariance matrix.

We can define the approximate posterior as

$$q(\boldsymbol{f}) := \int p(\boldsymbol{f}|\boldsymbol{u})q(\boldsymbol{u})d\boldsymbol{u} \tag{8.17}$$

to approximate the posterior distribution.

Rember that before for the sparse gaussian process we had a different definition:

$$q(\boldsymbol{f}) := \int q(\boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{u})d\boldsymbol{u} \tag{8.18}$$

### 8.2.1 Variational Distribution fitting

In the paper [3], a novel lower bound to the true marginal log-likelihood is proposed, to fit the variational distribution by maximizing the likelihood of fitting the data given as label, i.e.,

$$\begin{aligned}\log(p(\boldsymbol{y}|\boldsymbol{X})) \geq \sum_{i=1}^{n} \Bigg\{ & \log \mathcal{N}(y_i|\boldsymbol{K}(x_i,\boldsymbol{Z})\boldsymbol{K}(\boldsymbol{Z},\boldsymbol{Z})^{-1}\boldsymbol{m},\sigma_n^2) + \\ & -\frac{1}{2}\sigma_n^{-2}\big(\boldsymbol{K}(x_i,x_i) - \boldsymbol{K}(x_i,\boldsymbol{Z})\boldsymbol{K}(\boldsymbol{Z},\boldsymbol{Z})^{-1}\boldsymbol{K}(\boldsymbol{Z},x_i)\big) + \\ & -\frac{1}{2}\mathrm{tr}\big(\sigma_n^2 \boldsymbol{S}\boldsymbol{K}(\boldsymbol{Z},\boldsymbol{Z})^{-1}\boldsymbol{K}(\boldsymbol{Z},x_i)\boldsymbol{K}(x_i,\boldsymbol{Z})\boldsymbol{K}(\boldsymbol{Z},\boldsymbol{Z})^{-1}\big)\Bigg\} + \\ & - \mathrm{KL}(q(\boldsymbol{u})||p(\mathbf{u}))\end{aligned} \tag{8.19}$$

The key property of this new lower bound of the true marginal log-likelihood is that it can be written as a sum of terms, this implies that when we are searching for the optimal inducing inputs ($Z$) and variational parameter ($m$ and $S$), we can just sum over a (small) batch of our dataset, so the computational is only proportional to the how big is the batch size we are using for the update of the parameters. However, it is worth noticing that we still need to compute the inverse of the covariance evaluated on the inducing points, this has a computational cost of $\mathcal{O}(m^3)$. So it only scales with the amount of inducing points, not the data.

To compute the Kullback-Leibler (KL) divergence between a multivariate Gaussian distribution $\mathcal{N}(\mathbf{m},\mathbf{S})$ and a Gaussian distribution $\mathcal{N}(\mathbf{0},\mathbf{K})$, first compute the inverse and determinant of the covariance matrices $\mathbf{S}$ and $\mathbf{K}$, denoted as $\mathbf{S}^{-1}$, $\mathbf{K}^{-1}$, $|\mathbf{S}|$, and $|\mathbf{K}|$ respectively. Then, the KL divergence can be calculated as follows:

$$\mathrm{KL}(q(\boldsymbol{u})||p(\boldsymbol{u})) = \frac{1}{2}\left(\mathrm{tr}(\mathbf{K}(\boldsymbol{Z},\boldsymbol{Z})^{-1}\mathbf{S}) + (\mathbf{m}^T\mathbf{K}(\boldsymbol{Z},\boldsymbol{Z})^{-1}\mathbf{m}) - \log\left(\frac{|\mathbf{K}(\boldsymbol{Z},\boldsymbol{Z})|}{|\mathbf{S}|}\right)\right), \tag{8.20}$$

The fact that we are still able to maximize the (lower bound of the) marginal log-likelihood of the data without having to use the whole set of labels at every iteration opens the door to fit GPs on (very) big datasets without losing the probabilistic interpretation of the inference of the function and the inference of the prediction.

### 8.2.2 Predicting with Stochastic Variational Gaussian Process

To make predictions on a test point and using the definition of the approximate posterior (8.17) and the prediction integral for a standard GP, see Eq. (4.11)[2], the predictive distribution can be computed as:

$$\begin{aligned} p(\boldsymbol{f}_*|\boldsymbol{y}) &= \int p(\boldsymbol{f}_*|\boldsymbol{f},\boldsymbol{u})p(\boldsymbol{f},\boldsymbol{u}|\boldsymbol{y})d\boldsymbol{f}d\boldsymbol{u} \\ &\approx \int p(\boldsymbol{f}_*|\boldsymbol{f},\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u})q(\boldsymbol{u})d\boldsymbol{f}d\boldsymbol{u} \\ &= \int p(\boldsymbol{f}_*|\boldsymbol{u})q(\boldsymbol{u})d\boldsymbol{u} \end{aligned} \tag{8.21}$$

The integral is tractable, and we can compute the mean and variance of a test point in $\mathcal{O}(m^2)$.

The result of this integral is

$$p(\boldsymbol{f_*}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{Am}, \boldsymbol{K}(\boldsymbol{X_*},\boldsymbol{X_*}) + \boldsymbol{A}(\boldsymbol{S} - \boldsymbol{K}(\boldsymbol{Z},\boldsymbol{Z}))\boldsymbol{A}^\top) \tag{8.22}$$

where $\boldsymbol{A} = \boldsymbol{K}(\boldsymbol{X_*},\boldsymbol{Z})\boldsymbol{K}(\boldsymbol{Z},\boldsymbol{Z})^{-1}$.

As you can notice, also in the prediction, we do not have the inputs or the output label anywhere. The prediction time will not be affected directly by the dimension of the training set, but only by the mean and variance of the approximated distribution $q(\boldsymbol{u})$.

## 8.3 Sparse GP is a spacial case of a SVGP

Given the kernel approximation $\boldsymbol{Q}$, we can compute the distribution of the inducing points, $\mathcal{N}(\boldsymbol{u}|\boldsymbol{m},\boldsymbol{S})$, in closed form, given the data $\boldsymbol{y}$, i.e.,

$$q(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{K}(\boldsymbol{Z},\boldsymbol{X})[\boldsymbol{Q} + I\sigma_n^2]^{-1}\boldsymbol{y}, \boldsymbol{K}(\boldsymbol{Z},\boldsymbol{Z}) - \boldsymbol{K}(\boldsymbol{Z},\boldsymbol{X})[\boldsymbol{Q} + I\sigma_n^2]^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{Z})). \tag{8.23}$$

to make a prediction we now have to solve the integral

$$q(\boldsymbol{f_*}) := \int p(\boldsymbol{f_*}|\boldsymbol{u})q(\boldsymbol{u})d\boldsymbol{u}. \tag{8.24}$$

By using the solution for Gaussians of Eq. 8.22 we obtain Eq. 8.7 and 8.8. The integral to compute the posterior as a function of the approximate distribution $q(\boldsymbol{u})$ is the only thing we need. That approximate distribution is computed using the usual condition rule given the observation of the data, while the covariance matrix of the data is approximated with $\boldsymbol{Q}$ (plus the likelihood noise). Remember that we found the approximation of the prior covariance matrix $\boldsymbol{Q}$

[2]Also remember that $p(f,u|y) = p(f|u,y)p(u|y) \approx p(f|u)q(u)$, considering that f is independent from y (but only depends on u) and $p(u|y)$ is approximated with our variational distribution $q(u)$.

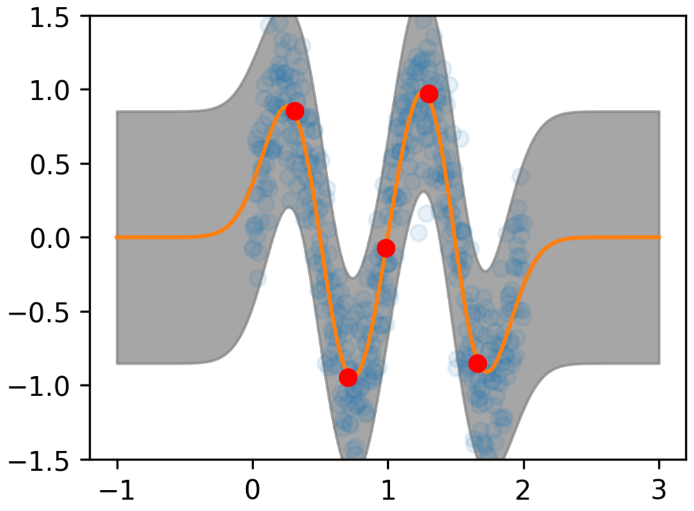


Figure 8.2: Fitting of noisy sine using Stochastic Variational Gaussian Process.

by imposing that if we would have had to predict $\boldsymbol{y}$ from $\boldsymbol{u}$ we should have no residual uncertainty. Then, we use that approximation prior to conditioning on the data and find the distribution of the inducing point $p(\boldsymbol{u})$.

We can conclude, that the Sparse GP is a special case of SVGP, where the variational distribution is computed in closed form from the data. This can be used as a test to validate if the training of $\boldsymbol{m}$ and $\boldsymbol{S}$ of the variational distribution, converges to the excepted mean and variance values given the evidence of our data. The advantage of using an SVGP is that while training, in order to find the inducing points and the hyperparameters of the kernel, we do not need to rely on the complete batch of the data, but we can do mini batching. This can speed up the inducing locations and hyperparameters when we are dealing with a hundred thousand datapoints. In the paper [3], it is also underlined how, when using the optimal mean and variance estimated using all the data, the variational lower bound of the SVGP becomes the same as the sparse GP.

## 8.4 Further Reading

1. Approximation Methods for Gaussian Process Regression, Chapter 3 [4]
2. Variational model selection for sparse Gaussian process regression [6]
3. Gaussian Process for Big Data [3]

# Chapter 9

# Multi-Output Gaussian Process Regression

In the previous chapters, we talked about modeling a single function variable $\boldsymbol{f}$ that can also be dependent on input with more than one feature, like when we talked about Automatic Relevance Determination.

However, we never considered the case of multi-input-multi-output modeling using Gaussian Process Regression. This is indeed possible and it is very useful when we are dealing with multiple outputs like when we are modeling the dynamics of a robot that has more degree of freedom. Let's recall how we make predictions for out posterior distribution

$$\boldsymbol{\mu}_{\boldsymbol{f}_*} = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} \tag{9.1}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{f}_*} = \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*) \tag{9.2}$$

and let's consider the square exponential kernel as our kernel, i.e.,

$$k_{SE}(x_i, x_j) = \sigma_p^2 \exp\left(-\frac{1}{2}\left(\frac{(x_i - x_j)^2}{\ell^2}\right)\right). \tag{9.3}$$

When dealing with multiple outputs, each of them can be modeled as an independent Gaussian Process with different kernel functions and hyperparameters. This simply means that for each output we fit a single output Gaussian Process model. This is the most generic solution but it is very computationally and memory inefficient.

However, if we consider that the kernel function can also be shared between the different outputs. This is useful when the different outputs can be considered sampled from the same generative process, e.g. the velocity of a robot in different cartesian directions.

**Sharing horizontal lenghtscales** When modeling multiple outputs, like velocities, it makes sense to share the horizontal length scales of the kernel, given that the smoothness of the motion can be independent of the cartesian direction.

**Sharing output lengthscales** However, when sharing the parameters of the kernel among different models, we must be careful. Let's imagine we are modeling the acceleration and the velocity output as a function of the robot's position. We are modeling, then, 2 outputs as a function of one input. We could share all the kernel parameters among the outputs and pretend to be happy with the result. This is fundamentally wrong. Suppose now you make a prediction, with mean and output both for velocity and acceleration, hence, the confidence bounds are $[\mu - 2\sigma, \mu + 2\sigma]$. The problem with this is that the unit of measure of the output standard deviation, i.e.$\sigma$, is set by the output length scale of the kernel. This means that if we are modeling output variables that have different units of measure, for sure we cannot share the output lengthscale. This is because we would not be able to set uniquely the unit of measure of the output lenghtscale. Moreover, if we share the output lengthscales among outputs that have a different level of energy, e.g. the first output data only varying between $[-1, 1]$ and the other between $[-10, 10]$ we will end up with a miscalibrated prior uncertainty. This implies that when we go "far away" from the data, the prediction uncertainty can be over-estimated or under-estimated, for one output or another.

**Sharing likelihood noise** Additionally, if we have different noise levels of the output signals, for example, because one of the sensors is much more noisy than the other, we can also not share the likelihood of noise among the outputs. Again, sharing the noise will make the uncertainty to be miss-calibrated.

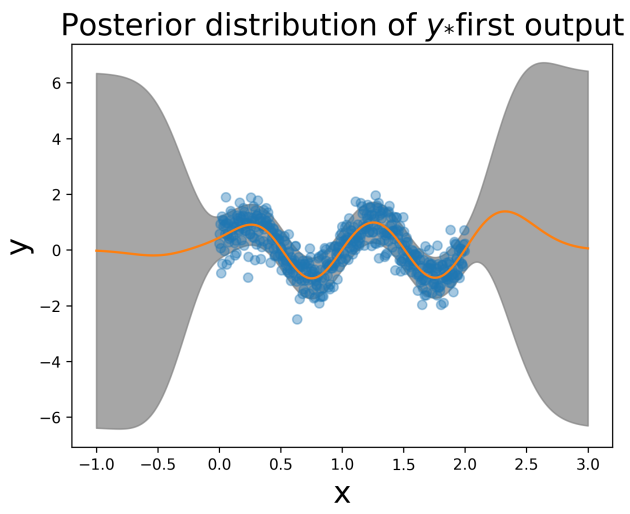
Figure 9.1 summarizes what we just said. In the figure, we are trying to fit two outputs, (a) a cosine that is very noisy and (b) a sine that is less noisy but oscillates between larger values. After fitting and maximizing the marginal log-likelihood of both outputs simultaneously, we notice that the fitting converges to a likelihood noise that underestimates the noise of function (a) and overestimates the likelihood noise of function (b). With deeper concern, we also notice that the maximum uncertainty, at which the model converges outside the region of the data, is well-calibrated for function (b) but overestimates the prior uncertainty of function (a). Although this may seem not important, having well-calibrated (epistemic) uncertainty when going into an unexplored region of the workspace can make the difference between a usable and unusable uncertainty prediction.

**Sharing the inducing inputs** When fitting sparse models, if we have many outputs, having different inducing points for each of the outputs can be intractable. A good option is to share the inducing points among the different outputs. This can speed up the fitting of your models and does not have many disadvantages.

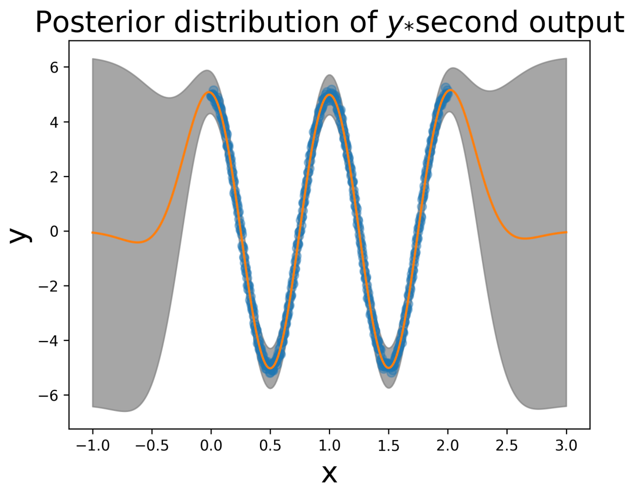
Figure **??**, depicts the vector field learned from the red demonstrations. In this example, we are fitting

$$\Delta x = p(f|x) \tag{9.4}$$

where the probability distribution is a Multi-output sparse Gaussian Process where the inducing points and the input lengthscales are shared among the



(a) Noisy Sine that oscillates between -1 and 1



(b) Cosine that oscillates between -5 and 5

Figure 9.1: Sharing likelihood noise and vertical lengthscale may induce to miscalibrated uncertainties

different outputs while the output lengthscales and noise is not shared. In the plot, you can see that when going far away from the data, the uncertainty of the model is growing, telling us that the prediction of the model is going to be less confident.

# Chapter 10

# Exercises

## 10.1 Definition

Write down the definition of a univariate and multivariate Gaussian Probability Distribution and explain each of the terms. Why is this distribution so successful in statistics?

What is the difference between the conditioning and the marginalization of a multi-dimensional Gaussian? Write down any necessary equation and help yourself with drawings.

## 10.2 Bayes Theorem

### 10.2.1 Definition

Find the correct answer and motivate it

1. $$likelihood \propto \frac{posterior}{prior} \tag{10.1}$$

2. $$posterior \propto \frac{prior}{likelihood} \tag{10.2}$$

3. $$posterior = \frac{likelihood * prior}{marginallikelihood} \tag{10.3}$$

### 10.2.2 Complete the equation

$$p(f^*|f)p(f) = \tag{10.4}$$

$$p(f^*|f)p(f|y) = \tag{10.5}$$

$$\int (p(f^*|f)p(f|y))df = \tag{10.6}$$

$$p(f^*, f|y) = \frac{p(y|f)\cdot}{} = \tag{10.7}$$

$$p(f^*|y) = \int ( \qquad )df = \tag{10.8}$$

## 10.3 Gaussian Process

### 10.3.1 What is a Gaussian Process?

Choose the right answer or answers.

1. It is a function approximator that uses a set of basis functions, which are centered on specific points in the input space, to transform the input data into a higher-dimensional space where the mapping between the inputs and outputs can be learned more easily. The output of the approximator is then a weighted sum of the outputs of the basis functions, where the weights are learned during training
2. It is a statistical model used to represent a probability distribution of a dataset that is assumed to be generated from several Gaussian distributions.
3. It is a type of network that incorporates Bayesian inference into the model. The weights and biases are treated as random variables with probability distributions that are updated as new data is observed. This allows for uncertainty to be quantified and incorporated into the model, which can be particularly useful in settings where the amount of data is limited or the data is noisy.
4. It is a collection of random variables, any finite number of which have a joint Gaussian distribution. It is a generalization of a Gaussian distribution to functions.
5. It is a variant of linear regression that utilizes a kernel function to transform the original input features into a higher-dimensional space, allowing for nonlinear relationships to be captured by a linear model.

### 10.3.2 Kernel Definition

Given the Square Exponential Kernel:

$$k(x_i, x_j) = \sigma_f^2 \exp(-\frac{(x_i - x_j)^2}{\sigma_l^2}) \tag{10.9}$$

Prove that it is a well-defined kernel. Example of well-defined kernels:

$$k(x_i, x_j) = x_i^\top x_j; \tag{10.10}$$

$$k(x_i, x_j) = c\tilde{k}(x_i, x_j); \tag{10.11}$$

where $c$ is a constant.

$$k(x_i, x_j) = f(x_i)\tilde{k}(x_i, x_j)f(x_j); \tag{10.12}$$

where $f$ is a generic function

$$k(x_i, x_j) = \exp(\tilde{k}(x_i, x_j)); \tag{10.13}$$

### 10.3.3 Marginal Likelihood

Maximum likelihood estimation (MLE) is a method that seeks to find the parameters of a statistical model that maximizes the likelihood of observing the data that we have. Essentially, MLE involves finding the set of parameter values that make the observed data most probable. Discuss the truth of any of the following statements

1. Maximum likelihood cannot be used in Gaussian Process Regression because there are no parameters
2. Maximum likelihood is a convex optimization problem
3. Maximum likelihood is a non differentiable function
4. Maximum likelihood cannot be used in practice because we only know how to minimize functions

### 10.3.4 Conditional Posterior

Write down the equation that we use for making predictions (mean and variance) with a Gaussian Process Regression (conditional posterior) at one point.

1. Explain the dimensional of each of the vectors and their meaning.
2. How any of the elements of the vector are calculated
3. Predict the mean and the variance of

$$\lim_{x \to \infty} \mu(x), \sigma(x) \tag{10.14}$$

### 10.3.5 Sparse Approximation of Gaussian Process

Define and discuss if any of the statements are true or false

1. Sparse Approximation of Gaussian Process is a technique used to speed up the computation of Gaussian Process regression for large datasets.
2. In a Sparse Approximation of Gaussian Process, the user needs to select a subset of the original training data.
3. Sparse Approximations are useful for making faster predictions but not training.

### 10.3.6 Exam question

You work for Dyson and you are programming software for autonomous robots. Specifically, you are programming vacuum cleaners that enable end-users to create custom cleaning policies based on human demonstrations. The robot's position in **a room** is determined using a Wi-Fi antenna, and you are tasked with developing a steering strategy for the robot. You have decided to use a Gaussian Process with the SE-ARD kernel, with the robot's position as input and the steering angle as output.

a) When you selected the SE-ARD kernel, you noticed an unusual result in the optimization of the lengthscales for the x, y, and z inputs. Explain why this happened, and write down the equation of the kernel. Additionally, describe the log marginalization likelihood and the significance of each term.

b) After training the robot, it was mistakenly placed in an unexplored region of the room. What do you expect the Gaussian Process to predict, and why?

c) How can you prompt the robot to ask for more data, and what is the name of the field of research that deals with this issue?

d) If the human demonstrator provides new data in the workspace, which matrix dimensions will change? Do you need to invert the entire covariance matrix? What is the name of the field of research that addresses this issue?

e) After recording data for an hour, your database has become too large, and your Gaussian Process is too slow to train. What type of approximation would you use, and what are the hyperparameters of this method? Finally, describe how to select these hyperparameters.

# Bibliography

[1] Chris Bracegirdle. "Bayes' Theorem for Gaussians". In: *http://web4.cs.ucl.ac.uk/staff/C.Bracegirdle/home.php* (2010).

[2] Roberto Calandra et al. "Manifold Gaussian processes for regression". In: *2016 International joint conference on neural networks (IJCNN)*. IEEE. 2016, pp. 3338–3345.

[3] James Hensman, Nicolo Fusi, and Neil D Lawrence. "Gaussian processes for big data". In: *arXiv preprint arXiv:1309.6835* (2013).

[4] Joaquin Quinonero-Candela, Carl Edward Rasmussen, and Christopher KI Williams. "Approximation methods for Gaussian process regression". In: (2007).

[5] Freek Stulp and Olivier Sigaud. "Many regression algorithms, one unified model: A review". In: *Neural Networks* 69 (2015), pp. 60–79.

[6] Michalis Titsias. "Variational learning of inducing variables in sparse Gaussian processes". In: *Artificial intelligence and statistics*. PMLR. 2009, pp. 567–574.

[7] Steven Van Vaerenbergh et al. "Fixed-budget kernel recursive least-squares". In: *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2010, pp. 1882–1885.

[8] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[9] Andrew G Wilson et al. "Stochastic variational deep kernel learning". In: *Advances in neural information processing systems* 29 (2016).