Gaussian Processes

Federico Bergamin & Kristoffer H. Madsen

1 Introduction

1.1 Uncertainty in machine learning[1]

Over the last few years, the quality and accuracy of machine learning and deep learning models have dramatically increased. This is mostly due to the ability of training neural networks efficiently. Given this success, people started introducing neural networks as part of the decision making pipeline, sometimes substituting traditional Bayesian techniques. However, the output of modern neural networks is usually a point estimate of the prediction and there are no methods to measure how confident the network is in giving that prediction. This is in contrast with Bayesian techniques that model also the uncertainty of parameters and predictions.

Understanding what a model does not know is a critical part of many machine learning systems, especially because machine learning algorithms are already having an impact to our everyday life. When we are using and deploying these models, it would be quite necessary to know whether the model is certain about its output. This allows a human to intervene for clarification when the model is not sure about its answer. Therefore, there are a lot of advantages from being able to quantify the model uncertainty for different applications. For example, if we train a model to classify images of dogs and cats, when the model gets a cat image as input, it should classify it as a cat with rather high confidence. But what should happen if it gets a duck image as input? We should expect that the model returns a "random" prediction but with very low confidence, or equivalently, with high uncertainty. This is an example of *out of distribution* test data. Indeed, the trained model has been trained using cats and dogs images and it learned how to distinguish among them. But an image of a duck would lie outside the data distribution used to train it.

Uncertainty plays a fundamental role in deploying an AI system in domains that can become life-threatening to humans. These go under the umbrella of *AI safety* or *safe AI*, which is concerned with the safe deployment of AI systems in real-time settings. It includes automated decision making and recommendation systems in life sciences and medicine and autonomous control of self-driving cars and critical systems. Indeed, in all these fields are quite important to say how much the model is confident about its estimate. For example, when a doctor or physician advises a certain treatment to a patient, the patient relies on the confidence of the expert that analyzes his medical record. If you want to substitute this last process by an algorithm there are a lot consequences that one should take into account. Having an algorithm with a high accuracy that outputs a single estimate results in a high bias on the judgement of the expert. For example, if your model gets an out of training distribution test example, it would predict something unreasonable, resulting in a bias for the doctor. Having a measurement of the uncertainty of the model in that specific prediction, can help the expert to understand when the model does not fit the test data and hence will predict at random.

Besides safe AI, there are a lot of applications that rely on uncertainty to be able to learn from small amounts of data. The common problem in those scenarios is that collecting and labeling training data or evaluating them is expensive or time consuming, therefore an efficient exploration of the input space is needed. These applications include Bayesian optimization, active learning, K-bandit problem, and reinforcement learning. In this course we will focus on the first two.

In addition, uncertainty plays an important role also for training and selecting a machine learning model. Indeed, knowing if the model is over-confident or under-confident can help to get better performance from it and, at the same time, if we have trained different models, having insights on

[1] This section is inspired by the first chapter of Yarin Gal's PhD thesis *"Uncertainty in Deep Learning"*.

the uncertainty, can help in deciding the model that best describes the data.

1.2 Problem setting

In order to motivate the importance of measuring uncertainty we should recall the regression problem. Consider a training set of N D-dimensional observations $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ together with corresponding target values $\{y_1, y_2, \ldots ., y_N\}$. A typical assumption in regression, is that the target values are the result of the a deterministic function of the input (which will be what we are trying to learn) and an additive noise term:

$$y = g(\mathbf{x}) + \epsilon \tag{1}$$

Since the goal is to learn a model to make predictions $\hat{y}$ for new, unseen points $\hat{\mathbf{x}}$, we are interested in approximating the true generative process $g(\mathbf{x})$ with a model $f(\mathbf{x}, \mathbf{w})$ that should be able to capture the shape of the true function $g(\mathbf{x})$, where $\mathbf{w}$ is a vector of parameters of our model.

In the usual linear regression setting, we have assumed that the output is given by a linear combination of the input features:

$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D = w_0 + \sum_{j=1}^{D} w_j x_j \tag{2}$$

Notice that the model we defined above is *linear* in terms of both the parameters w and the data x_i. This implies that the set of functions that we can fit using this model is restricted to lines with different slopes. Therefore, we usually consider a non-linear transformation of the input features. This results in modeling the output as a linear combination of fixed non-linear functions of the input variables, given by

$$f(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \tag{3}$$

where $\phi_1(\mathbf{x}), \ldots, \phi_{M-1}(\mathbf{x})$ are $M-1$ *basis functions.* In this case, this is still a linear model with respect the parameters w, but not the data[2]. Therefore, if in the following notes, when we are referring to non-linear regression, we are referring to perform regression using a model that is not linear with respect the data. but that is still linear with respect to the parameters. It is usually convenient, when implementing these methods, to define

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} 1 \\ \phi_1(\mathbf{x}) \\ \vdots \\ \phi_{M-1}(\mathbf{x}) \end{bmatrix}$$

so we can write the non-linear (w.r.t the data) regression model in a more compact way:

$$f(\mathbf{x}, \mathbf{w}) = \boldsymbol{\phi}(\mathbf{x})^T \mathbf{w}$$

where $\mathbf{w} = [w_0, w_1, \ldots, w_{M-1}]$.

The problem with this classical approach for performing regression when the data is non-linear is that it typically returns only one function, the one that best fits the training data. This function will return a single estimate for each test point and we cannot measure how confident the model is in its predictions (see Fig.1 which compares the usual non-linear regression with a probabilistic approach that gets the confidence interval for its predictions). But what would happen if we observe more data? Is this function still good? And what about the other functions that fit the data pretty well? Might one of those become a more appropriate fit of the data better if we add new training points?

[2] Sometimes people refers to this model as doing non-linear regression, but this is still a linear model with respect the parameters.

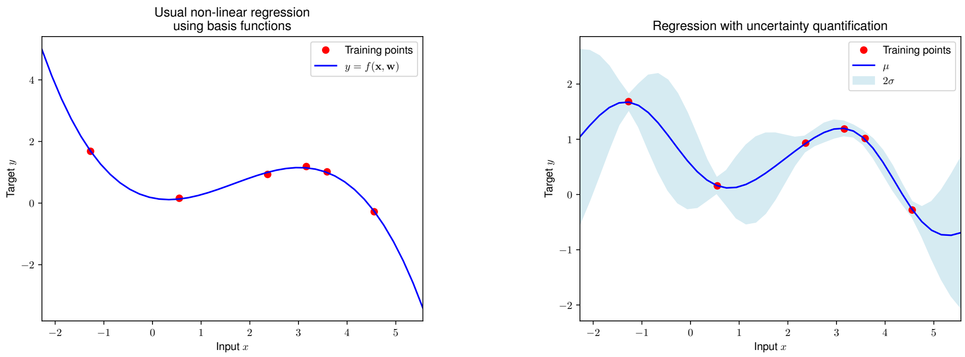

Figure 1: Different approaches for solving the same non-linear regression problem. The dataset is created using `seed`$= 32$ and $y = 0.2 - 0.3x + 0.5x^2 - 0.1x^3 + 0.05*$`np.random.rand()`. (*Left*) Usual non-linear regression using *basis functions* $\boldsymbol{\phi}(\mathbf{x}) = [1, \mathbf{x}, \mathbf{x}^2, \mathbf{x}^3]$. We can see that the result is a single function that fits the data. (*Right*) Probabilistic approach that returns a mean prediction function and posterior intervals. We can notice that the model is confident where there are training points, but it is uncertain in region where there are no points.

As we are going to see in this first lecture we can solve the mentioned open problems using Gaussian Processes. The usual definition you can find in most books and blog posts is that *"A Gaussian process is a probability distribution over functions."*. This probabilistic method provides an alternative approach to regression problems where there are potentially infinitely many functions that can fit the data well as we have seen above. Gaussian Processes assign a probability to each of those functions and the mean of this distribution is the most probable fitting of the data. Since this is a probabilistic approach, a Gaussian Process measures also the confidence of the prediction. They can be used also in the classification setting, but this is beyond the scope of this course. These notes aim to introduce the mathematical intuition of Gaussian Processes by presenting the concepts both in a more intuitive way and in a more mathematical one. We will start by recalling the Gaussian Distribution, in particular focusing on the Conditional Gaussian Distribution, which is at the core of Gaussian Processes. Then we will go through a proof that the assumption made by Gaussian Process is a plausible one and then we will focus on Gaussian Process regression. We will introduce a method to build the distribution over functions and we will present how to obtain the posterior distribution to make predictions.

2 Multivariate Normal Distribution

We first review the definition and the properties of the Multivariate Gaussian distribution. A D-dimensional vector $\mathbf{x} = (x_1, x_2, \dots, x_D)$ is said to be multivariate Gaussian distributed if all linear combinations of $\mathbf{x}$ are Gaussian distributed with some mean (m) and variance (v):

$$y = \mathbf{a}^T\mathbf{x} = a_1x_1 + a_2x_2 + \ldots + a_Dx_D \sim \mathcal{N}(m, v) \tag{4}$$

for all $\mathbf{a} \in \mathbb{R}^D$, where $\mathbf{a} \neq \mathbf{0}$.

The multivariate Gaussian distribution of the vector $\mathbf{x}$ is usually denoted by $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. $\boldsymbol{\mu}$ is a D-dimensional mean vector that describes the expected value of the distribution and each component describes the mean of the corresponding dimension. $\boldsymbol{\Sigma}$ is a $D \times D$ covariance matrix, which describes the variance along each dimension and describes the correlation between all the random variables. It is a symmetric and positive semi-definite matrix. The diagonal consists of the variance σ_i^2 of the $i-$th variable, while the off-diagonal elements σ_{ij} for $i \neq j$ represent the covariance between the $i-$th and $j-$th variables.

A multivariate Gaussian has the following probability density function:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\} \tag{5}$$

where $|\mathbf{\Sigma}|$ is the determinant of $\mathbf{\Sigma}$.

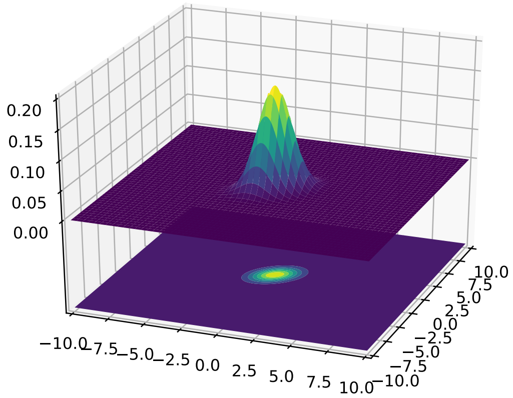

Figure 2: Example of the probability density function of a 2-dimensional Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$, where $\boldsymbol{\mu} = [0, 0]$ and $\mathbf{\Sigma} = [[1, 0.4], [0.4, 1]]$. Above, it is plotted as function of $\mathbf{x} = [x_1, x_2]$, whereas below we can see the same distribution as a contour plot.

There are two main reasons why the Gaussian distribution is frequently used in machine learning and statistics when we try to model real data. First, the *Central Limit Theorem* justifies their use in modelling noise in data. Indeed, we assume that the observations we are working with are usually corrupted by noise, i.e perturbations affecting the measurement process. In general, noise can be considered as the result of the action of different independent random variables. The Central Limit Theorem tells us that, in some situations subject to certain mild conditions, the sum of independent random variables results in a new random variable whose distribution becomes increasingly Gaussian as the number of variables in the sum increases. The second reason is that Gaussians have nice analytical properties that make it easier to manipulate them, as we are going to see in detail. Those allow us to write integrals that involves Gaussian distributions in closed form.

2.1 Conditional and Marginal Gaussian Distribution

An important property of the Gaussian distribution is that its conditional and marginal distribution are still Gaussian. Therefore, we can compute the mean and the covariance matrix in closed form. Consider our D-dimensional vector $\mathbf{x} \in \mathbb{R}^D$ with Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{\Sigma})$. Suppose that its variables have been partitioned in two sets $\mathbf{x}_a$, that contains the first M components, and $\mathbf{x}_b$ that contains the remaining $D - M$ components. We also define the corresponding partitions of the mean $\boldsymbol{\mu}$ and the partitions of the covariance matrix $\mathbf{\Sigma}$. Therefore, we get:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \qquad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \qquad \mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}_{aa} & \mathbf{\Sigma}_{ab} \\ \mathbf{\Sigma}_{ba} & \mathbf{\Sigma}_{bb} \end{pmatrix}$$

The symmetry of the covariance matrix $\mathbf{\Sigma}^T = \mathbf{\Sigma}$ implies that both $\mathbf{\Sigma}_{aa}$ and $\mathbf{\Sigma}_{bb}$ are symmetric and $\mathbf{\Sigma}_{ba} = \mathbf{\Sigma}_{ab}^T$.

Marginal distribution The *marginal distributions*,

$$\begin{aligned} p(\mathbf{x}_a) &= \int_{\mathbf{x}_b} p(\mathbf{x}_a, \mathbf{x}_b | \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{x}_b \\ p(\mathbf{x}_b) &= \int_{\mathbf{x}_a} p(\mathbf{x}_a, \mathbf{x}_b | \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{x}_a \end{aligned} \tag{6}$$

are Gaussian and are defined by the following mean and covariance matrix:

$$\begin{aligned} p(\mathbf{x}_a) &= \mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}) \\ p(\mathbf{x}_b) &= \mathcal{N}(\mathbf{x}_b | \boldsymbol{\mu}_b, \boldsymbol{\Sigma}_{bb}) \end{aligned} \tag{7}$$

Conditional distribution The *conditional distribution* of $\mathbf{x}_a$ given $\mathbf{x}_b$

$$p(\mathbf{x}_a|\mathbf{x}_b) = \frac{p(\mathbf{x}_a, \mathbf{x}_b | \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\int_{\mathbf{x}_a} p(\mathbf{x}_a, \mathbf{x}_b | \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{x}_a} = \frac{p(\mathbf{x}_a, \mathbf{x}_b | \boldsymbol{\mu}, \boldsymbol{\Sigma})}{p(\mathbf{x}_b)} \tag{8}$$

and the one of $\mathbf{x}_b$ given $\mathbf{x}_a$

$$p(\mathbf{x}_b|\mathbf{x}_a) = \frac{p(\mathbf{x}_a, \mathbf{x}_b | \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\int_{\mathbf{x}_b} p(\mathbf{x}_a, \mathbf{x}_b | \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{x}_b} = \frac{p(\mathbf{x}_a, \mathbf{x}_b | \boldsymbol{\mu}, \boldsymbol{\Sigma})}{p(\mathbf{x}_a)} \tag{9}$$

are also Gaussian. We can compute the mean and the covariance matrix of resulting distributions in closed form:

$$p(\mathbf{x}_a|\mathbf{x}_b) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{a|b}, \boldsymbol{\Sigma}_{a|b})$$

where

$$\begin{aligned} \boldsymbol{\mu}_{a|b} &= \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b) \\ \boldsymbol{\Sigma}_{a|b} &= \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba} \end{aligned} \tag{10}$$

Respectively, for $p(\mathbf{x}_b|\mathbf{x}_a)$ we have:

$$p(\mathbf{x}_b|\mathbf{x}_a) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{b|a}, \boldsymbol{\Sigma}_{b|a})$$

where

$$\begin{aligned} \boldsymbol{\mu}_{b|a} &= \boldsymbol{\mu}_b + \boldsymbol{\Sigma}_{ba}\boldsymbol{\Sigma}_{aa}^{-1}(\mathbf{x}_a - \boldsymbol{\mu}_a) \\ \boldsymbol{\Sigma}_{b|a} &= \boldsymbol{\Sigma}_{bb} - \boldsymbol{\Sigma}_{ba}\boldsymbol{\Sigma}_{aa}^{-1}\boldsymbol{\Sigma}_{ab} \end{aligned} \tag{11}$$

These relationship are the key points when we will derive the Gaussian Process regression.

A simple example To get some intuition about those equations, we can think about the case where $\mathbf{x}_a$ and $\mathbf{x}_b$ are independent. From independence, it follows that:

$$\mathrm{Cov}(\mathbf{x}_a, \mathbf{x}_b) = 0 = \boldsymbol{\Sigma}_{ab}$$

We can use the formula above (Eq. 10 and Eq. 11) to derive the conditional distributions $p(\mathbf{x}_a|\mathbf{x}_b)$ and $p(\mathbf{x}_b|\mathbf{x}_a)$. We derive $p(\mathbf{x}_a|\mathbf{x}_b)$, but the same procedure can be applied to $p(\mathbf{x}_b|\mathbf{x}_a)$.

$$\begin{aligned} \boldsymbol{\mu}_{a|b} &= \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b) = \boldsymbol{\mu}_a + 0\boldsymbol{\Sigma}_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b) = \boldsymbol{\mu}_a \\ \boldsymbol{\Sigma}_{a|b} &= \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba} = \boldsymbol{\Sigma}_{aa} - 0\boldsymbol{\Sigma}_{bb}^{-1}0 = \boldsymbol{\Sigma}_{aa} \end{aligned}$$

Therefore, we just derived the fact that $p(\mathbf{x}_a|\mathbf{x}_b) = \mathcal{N}(\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}) = p(\mathbf{x}_a)$. Which we already know because the conditional probability of two independent random variables is equal to the unconditional distribution of the non-conditioned variable.

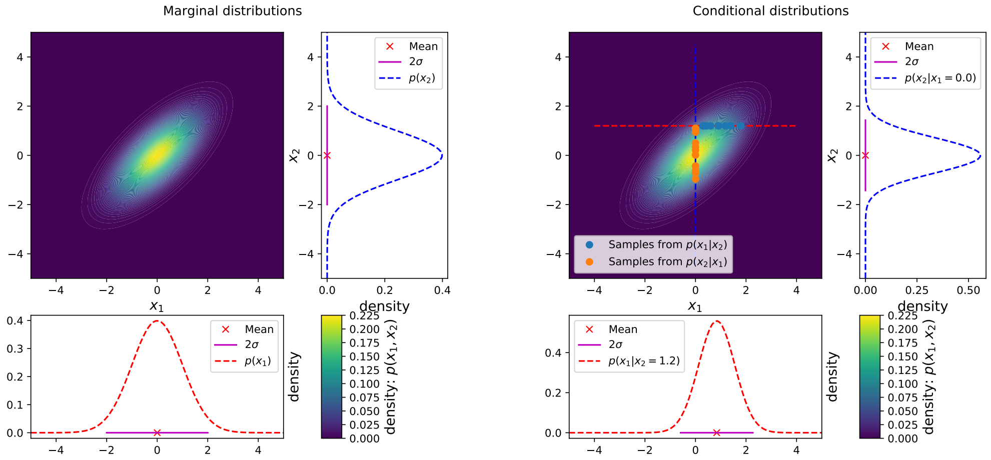

Figure 3: The plots represent the 2-dimensional Gaussian distribution $p(\mathbf{x}_1, \mathbf{x}_2) = \mathcal{N}(\mathbf{x}_1, \mathbf{x}_2|\boldsymbol{\mu}, \mathbf{\Sigma})$, where $\boldsymbol{\mu} = [0, 0]$ and $\mathbf{\Sigma} = [[1, 0.7], [0.7, 1]]$. *Left*: We used Eqn. (7) to compute the marginal distribution from the joint distribution $p(\mathbf{x}_1, \mathbf{x}_2)$. We show also the position of the mean together with the 95% confidence interval. *Right*: We compute the conditional distributions $p(x_1|x_2)$ and $p(x_2|x_1)$ from the joint distribution using Eqn. (10) and Eqn. (11). Also in this case, we report the mean and the 95% confidence interval and we also sample from the conditional.

3 Gaussian Processes

3.1 A different representation of Multivariate Gaussian samples

> *"Gaussian Process are tricky and it's not the equations that are tricky, but the concepts are."*
>
> David J.C. MacKay

We are going to introduce a new representation for the samples of a multivariate distribution that will (hopefully) allow you to get a better understanding of why Gaussian Processes work. We start considering a bi-variate Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{\Sigma})$, where the mean $\boldsymbol{\mu}$ is a 2-dimensional vector and the covariance $\mathbf{\Sigma}$ is a 2×2 matrix. When we sample from this distribution we get a 2-dimensional vector $\mathbf{x} = [x_1, x_2]$. We will represent it plotting the value of x_1 and x_2 at index 1 and 2, respectively, as shown in Fig. 4.

Using this representation we can observe the effect of the covariance matrix in the samples. It can be noticed from Fig. 5 that considering highly correlated variables x_1 and x_2 (correlation of 0.9 in the example), we obtain bars that "wiggle" slightly and we have that the two endpoints tends to move up and down together. Since these two variables have a positive correlation, it means that x_2 increases as x_1 gets larger and vice-versa. In case of almost independent variables (0.1 correlation), it can happen that the end points move to different direction, because the correlation between them is only weakly informative. As we are going to see in the Gaussian Process section, we are going to prefer correlated variables due to the *Smoothness Assumption*, which is a general way of expressing that if a certain point x produce an output y, a close point x' should produce an output y' close to y. Therefore, in the next experiments that we are going to present to understand intuitively the Gaussian Processes, we are going to use a covariance matrix that reflects the smoothness assumption. We use the indices to order the variables of the multivariate Gaussian and we make each variable more correlated with closer variables, i.e. their indices are close, and less correlated with variables that are further.

Suppose we get more information about the value of x_1. We can use the new representation to

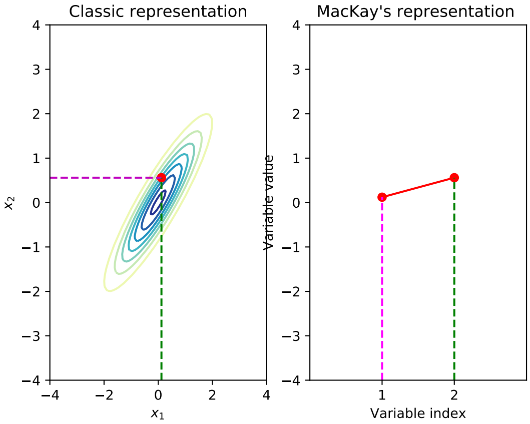

Figure 4: The figures show the usual representation of a sample from a 2-dimensional multivariate Gaussian and a representation based on the new representation introduced by MacKay. We sample from $\mathcal{N}(\mathbf{x}_1, \mathbf{x}_2|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = [0, 0]$ and $\boldsymbol{\Sigma} = [[1, 0.9], [0.9, 1]]$. In the 2-dimensional case, we take the sample and plot the value of x_1 and x_2 at index 1 and 2 connected by a straight line.

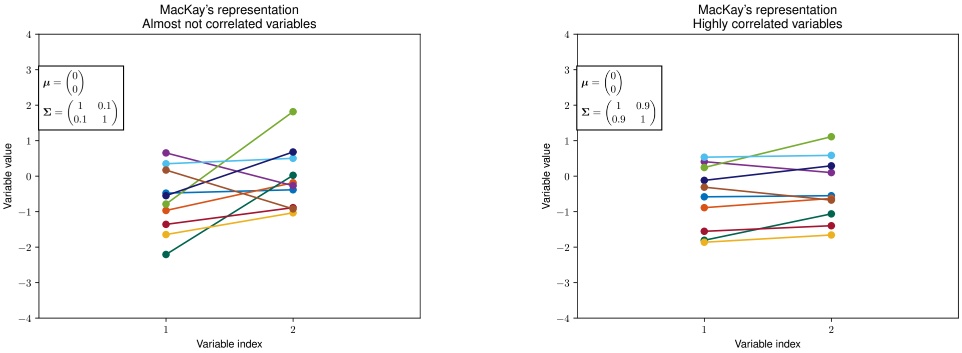

Figure 5: The plots represent various samples from a 2-dimensional normal distribution using the MacKay's bar plot representation with two difference covariance matrices. When x_1 and x_2 are almost independent (*Left*), i.e. the covariance between them is close to 0, the endpoints of the bar plots wiggle more than the case in which the variables are correlated (*Right*).

see what possible values x_2 can get. We can use Eqn. (11) to compute the conditional distribution $p(x_2|x_1)$ and then sample from it. Since we are assuming smoothness and highly correlated variables, the difference between the value of x_1 and the possible values of x_2 is small (See Fig. 6 for an example).

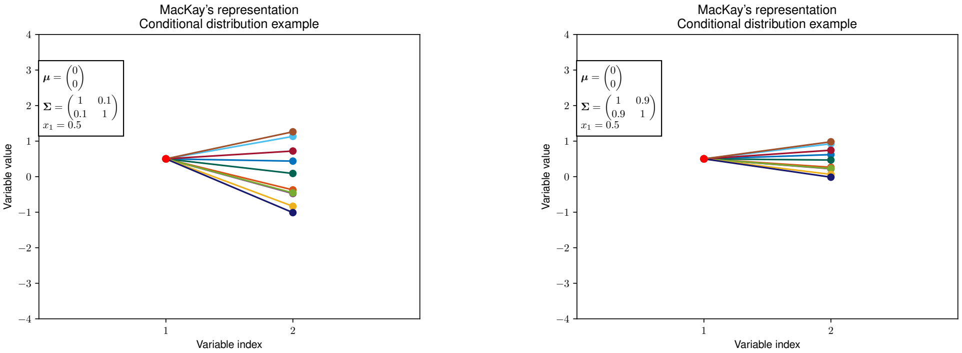

Figure 6: Samples from the conditional distribution in case of almost independent (*Left*) and highly correlated (*Right*) variables.

This simple representation allows us to represent high-dimensional Gaussian samples in a 2D plot. This is the key point to see the connection between multivariate Gaussian and the Gaussian Process regression that we are going to introduce in Section 3. Consider a 20-dimensional Gaussian with a covariance matrix that reflects the smoothness assumption, we can sample $\mathbf{x} = [x_1, x_2, \ldots, x_{20}]$ and plot the value of the variables at indices $[1, 2, \ldots, 20]$. The result is presented in Fig. 7. In addition to that, we can look at what happens to these samples when we condition them on other variables, such as x_1 and x_2, or x_1 and x_{11} for example. Also in this case it is pretty simple and straightforward. We just use the Eqn. (11) to obtain the conditional distribution $p(x_3, x_4, \ldots, x_{20}|x_1, x_2)$ and $p(x_2, x_4, \ldots, x_{20}|x_1, x_{11})$. In Fig. 8 (*Right*) here we notice that the results look like a lot of solutions of some non-linear regression problem where in our training set we have two points x_1 and x_2. Viewing this from the regression problem perspective, in Fig. 8, we have only two points, defined by the tuples $\{(1, x_1), (2, x_2)\}$, where the first element of the tuple is the indices of the Gaussian variable, that in this case becomes our x, and the value of the variable itself, denoted by x_i but that in these perspective is our y of the regression problem. Therefore, we are interested in predicting the values for the points $[3, \ldots, 20]$, which can be considered our test set.

Be careful! In the way we are creating a connection between multivariate Gaussian and Gaussian Process, the notation that can be misleading. Usually in regression we have a dataset of points x with their respective targets y. In this case we have that the x points are given by the indices of the Gaussian variables and the targets by the values of the variables, so in this case $x_1, \ldots, x_{20}$.

In the proposed setting, we can sample a lot of possible different "functions", therefore we can plot the mean and the standard deviation for each prediction for the unknown variables $[x_3, \ldots, x_{20}]$. The results can be observed in Fig. 8 (*Left*). It can be seen, that the standard deviation is lower when we are close to the training observations, whereas it gets higher far from them.

What does this representation suggest? Suppose that we are interested in getting the predictions for 10 different x-values without having training points. We can use what we have just learnt above. We can take these x-values and construct a 10×10 covariance matrix that describe a 10-dimensional multivariate Gaussian. If we sample from this distribution we get a 10-dimensional vector, where each entry is a possible prediction for that specific x point. In case we have some training set, we can still create the covariance matrix and then compute the conditional distribution and sample from that. There are at least still two open questions. First, we do not know how to construct a covariance matrix from the x-axis points to get prediction for the possible y's. Second, the way we try to visualize a Gaussian process using an alternative representation of a multivariate Gaussian seems to suggest that the x-axis would have to be all integers because these are the indices of the variables of the multivariate Gaussian. However, in a regression setting we want to

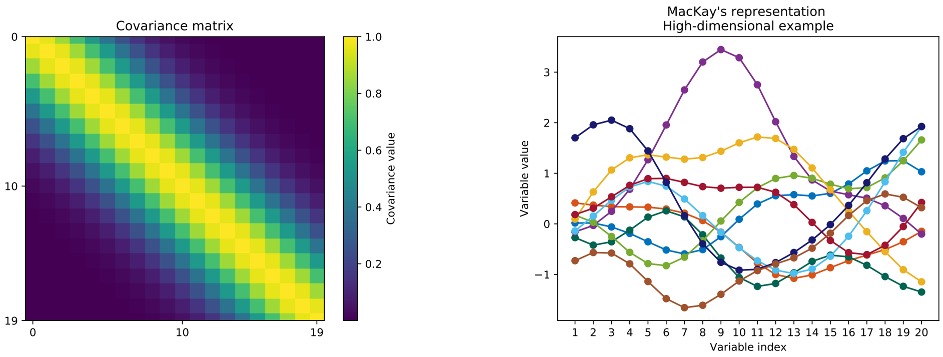

Figure 7: Random samples from a 20-dimensional multivariate Gaussian represented using the MacKay representation. (*Left*) Image representation of the covariance matrix. Notice that the covariance matrix is structured: closer points have higher correlation. The covariance is computed using the indices $1, 2, \ldots, 20$ in a way that closer points are higher correlated. (*Right*) Random samples from the multivariate Gaussian with mean $\mathbf{0}$ and covariance matrix defined in the left plot.

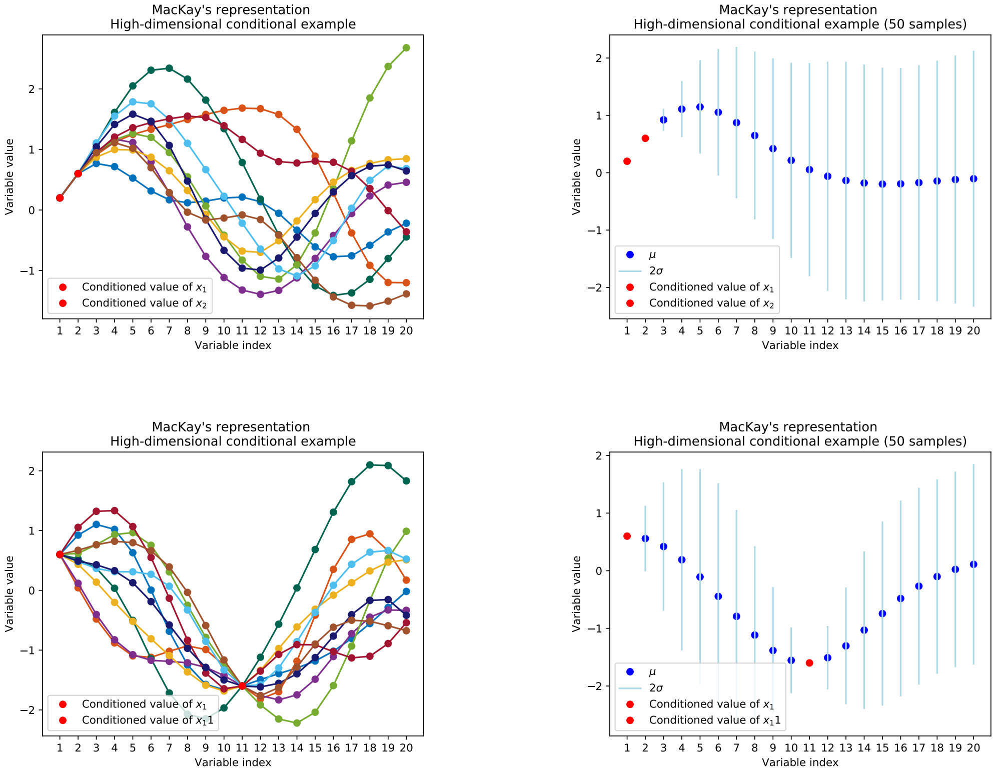

Figure 8: (*Upper plots*) Random samples from the conditional probability $p(x_3, , x_{20}|x_1, x_2)$. (*Upper Left*) The random samples are non-linear function that goes through the conditioned points. (*Upper Right*) Since we can get a lot of samples from the conditional distribution, we get 50 samples and compute the mean and the standard deviation for each variable $x_3, , x_{20}$. We can notice that the points closer to the observations x_1, x_2 has low standard deviation and non-zero mean. Point that are further instead gets higher uncertainty and mean close to zero.(*Bottom plots*) Random samples from the conditional probability $p(x_2, , x_{20}|x_1, x_{11})$. Also in this case we can notice that the standard deviation is lower when it is close to the points we have conditioned on.

model observations with real values, not integers. This is related to the first point, that is, how we compute the covariance matrix. As we are going to see in Section 3.4, in Gaussian Processes we generate the covariance matrix by evaluating a *kernel function*, also often called *covariance function*, pairwise on all the points.

3.2 A more analytical introduction to Gaussian Processes

We can introduce Gaussian Processes in a more analytical way compared to the approach used before, where we use a different representation to create a link between the multivariate Gaussian distribution and regression. In Section 1.2 Eq. (1) we presented the regression problem as:

$$y = f(\mathbf{x}, \mathbf{w}) + \epsilon \tag{12}$$

$$y = \mathbf{w}^T \mathbf{x} + \epsilon \qquad \text{(Linear regression)} \tag{13}$$

where ϵ is the residual error between our prediction and the true observation. We often assume ϵ to be normal distributed, $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Therefore, we have that $y \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2)$ is also Gaussian, i.e. $p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2)$, which means that the targets y are distributed according to a Gaussian distribution.

Suppose we have a dataset $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ of n observations. We have three different methods to learn the parameters $\mathbf{w}$:

- *Maximum Likelihood Estimation* (MLE), where we try to maximize the likelihood of our data given the weights:

$$p(D|\mathbf{w}) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) \tag{14}$$

- *Maximum a Posteriori* (MAP), where instead we try to maximize the posterior probability of the weights $\mathbf{w}$ given the dataset:

$$\begin{aligned} p(\mathbf{w}|D) &= \frac{p(D|\mathbf{w})p(\mathbf{w})}{p(D)} \\ &\propto p(D|\mathbf{w})p(\mathbf{w}) \end{aligned} \tag{15}$$

  where $p(D|\mathbf{w})$ is the *likelihood*, which is what we are maximizing in MLE, and $p(\mathbf{w})$ is the *prior*.

- *Fully Bayesian approach*, where instead of computing a single point estimate for $\mathbf{w}$, as the two previous methods do, which results in a single function, we can fully compute the posterior distribution $p(\mathbf{w}|D)$. As we are going to see next, in Gaussian Process we are going to use a Bayesian approach, but we will be interested in the distribution over function and not over weights.

We have seen that $p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2)$, therefore we have that $p(D|\mathbf{w}) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w})$, which is a product of Gaussian distributions, that is still a Gaussian distribution. Both of the aforementioned methods can be used to learn the parameters of the model. At the end of the day, however, the aim of learning the parameters of a model is for making predictions for future data, i.e. the test set. In addition to that, the predictions we obtain from the learned model depends on a particular set of parameters $\mathbf{w}$, i.e. the one that maximizes the objective. Therefore, instead of trying to learn the parameters $\mathbf{w}$, we can try to model the targets of our test set $\mathbf{y}_*$ directly. Suppose we have only one test point $\mathbf{x}_*$, we are interesting in modeling the distribution $p(y_*|D, \mathbf{x}_*)$, where D is the training set as before. We can notice that in this case we are not committing to a specific model, indeed there is no weights parameters in the distribution that we want to model and there is only one assumption that we are going to make.

We can try to derive an expression for $p(y_*|D, \mathbf{x}_*)$, marginalizing out the models given by the various $\mathbf{w}$:

$$\begin{aligned} p(y_*|D, \mathbf{x}_*) &= \int_{\mathbf{w}} p(y_*, \mathbf{w}|\mathbf{x}_*, D) d\mathbf{w} \\ &= \int_{\mathbf{w}} p(y_*|\mathbf{x}_*, D, \mathbf{w}) p(\mathbf{w}|D) d\mathbf{w} \\ &\text{Since knowing } \mathbf{w} \text{ makes it independent from } D\text{, we can write:} \\ &= \int_{\mathbf{w}} p(y_*|\mathbf{x}_*, \mathbf{w}) p(\mathbf{w}|D) d\mathbf{w} \end{aligned} \tag{16}$$

Unfortunately, this integral is in general intractable in closed form. However, if we assume that both the likelihood $p(D|\mathbf{w})$ and the prior $p(\mathbf{w})$ are Gaussian, we can use the Gaussian properties to immediately infer that $p(\mathbf{w}|D)$ is Gaussian (using Bayes' rule). Regarding $p(y_*|\mathbf{x}_*, D, \mathbf{w})$, as soon we get $\mathbf{w}$, we have that the y_* does not depend on D anymore, so we have $p(y_*|\mathbf{x}_*, \mathbf{w})$ and, in the regression setting we have seen that it is Gaussian. Therefore, we have that the distribution we are trying to model, $p(y_*|D, \mathbf{x}_*)$, is Gaussian.

The above equation shows that we can predict y_* for any $\mathbf{x}_*$ without committing to a specific model, but instead we can average the prediction over all the possible models. Since what we obtain is still a Gaussian distribution, we can get also the uncertainty (or posterior interval) on the prediction. The fact that we know that the result is a Gaussian allow us to skip all the integrals, because we already know how a Gaussian looks like. We know that we have to fit two parameters: a mean and a covariance matrix. Indeed, we know that

$$p(y_*|D, \mathbf{x}_*) = p(\mathbf{y}_*|\mathbf{y}, \mathbf{X}, \mathbf{X}_*) = \mathcal{N}(\boldsymbol{\mu}_{y_*|\mathbf{y}}, \boldsymbol{\Sigma}_{y_*|\mathbf{y}})$$

where $\boldsymbol{\mu}_{y_*|\mathbf{y}}$ and $\boldsymbol{\Sigma}_{y_*|\mathbf{y}}$ would have a form similar to Eq. 10.

This is what happens in Gaussian Process regression. Instead of committing to a specific model, which means learning some parameters $\mathbf{w}$, it starts with the assumption that the distribution of the target values $p(\mathbf{y}, \mathbf{y}^*)$ is Gaussian. Making this assumption upfront, allows us to say that $p(\mathbf{y}^* \mid \mathbf{y})$ is Gaussian, without going through all the steps we did before. But what does it imply that our targets are drawn from a Gaussian distribution? Suppose we have one training observation $(\mathbf{x}_1, y_1)$ and a test observation $(\mathbf{x}_*, y_*)$. If the data is Gaussian distributed it means that there is a Gaussian distribution that tells me that if y_1 and y_* are highly correlated, knowing the value of y_1 would inform about the value of the test target y_*, otherwise, if they are not correlated, we do not gain information. If we know the value of y, then we can get the distribution of y_* computing the conditional Gaussian distribution.

3.3 Putting the pieces together...

We presented two different ways of introducing Gaussian Process regression to achieve some mathematical intuition. The first approach was based on visualization. We introduced a new way to represent samples from a multivariate Gaussian distribution. We saw that we can treat the value of each random variable as the target y of a specific x, in that case given by the index of the variable. If we look at this from the opposite point of view, we can assume that, given some x-values, we can sample their target y from a multivariate distribution. In other words, we can assume that, before observing the training targets, all the targets y are drawn from a Gaussian distribution.

The second approach, instead, was more analytical. Starting from the assumption that the goal of a machine learning model is to get predictions for future data, we decided to model directly the distribution of the targets of the test set. We argued that considering a Gaussian likelihood and a Gaussian prior, we have that the distribution is Gaussian. Assuming directly at the beginning that our targets are Gaussian distributed, as we are going to do in Gaussian Process regression, allows us to reach the same point without having to commit to a specific model.

3.4 Gaussian Process Regression

Let's assume that for now we have noise-free observation, i.e. $y = f(\mathbf{x})$. We have seen that it is reasonable for at least two reasons to assume that, before we observe the training targets, those are drawn from a Gaussian distribution:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \sim \mathcal{N}(\mu, \Sigma) \tag{17}$$

If we center the data, i.e. we subtract the sample mean, we then consider a zero-mean Gaussian distribution, i.e. $\mathcal{N}(0, \Sigma)$. The open question is then how we compute the covariance matrix Σ. Here we can again apply one of the tricks of Gaussian Process. The covariance matrix will not only describe the shape of the distribution, but it is going to determine the characteristics of the function we want to predict. We generate the covariance evaluating a *kernel function*. A kernel, which is also called *covariance function* in the literature, is a positive-definite function of two inputs $\mathbf{x}$, $\mathbf{x}'$. In the Gaussian process context $\mathbf{x}$ and $\mathbf{x}'$ are usually the input observations, therefore they are usually vectors in a Euclidean space. The output of a kernel is a measure of the similarity between two objects. In Gaussian processes, we will use this measure to specify the similarity between the y's, and to compute it we use their respective x. Indeed, for the smoothness assumption, we have that $x_i \approx x_j \implies y_i \approx y_j$. Therefore, you have to remember that we are constructing the covariance matrix of the Gaussian distribution of the y's and to do so, we use their respective x. So formally, we have:

$$\Sigma_{ij} = \mathrm{Cov}(y_i, y_j) = \mathrm{Cov}(f(x_i), f(x_j)) = k(x_i, x_j) \tag{18}$$

There are a lot of possible kernels that can be used in Gaussian Process. Each kernel corresponds to a different set of assumptions that reflects the structure of the function that we wish to model. In these notes we will focus only on a kernel that is considered as the de-facto default kernel, the *Squared-Exponential* (SE) kernel, which is defined by:

$$k_{SE}(x_i, x_j) = \sigma^2 \exp\left(-\frac{\|x_i - x_j\|^2}{2l^2}\right) \tag{19}$$

and has the following two parameters:

- The lengthscale $l > 0$ determines the length of the wiggles in your function. In general, you can extrapolate only up to a lengthscale away from your training data. That is because if there is not data, we will get the chosen prior.
- The output variance $\sigma^2 < 0$, also sometimes called *magnitude*, determines the average distance of the function that we are trying to model from its mean. This can be seen as a scaling factor.

If you have a closer look at this kernel, we can see that its behaviour is dominated by the exponential function. We can see that the both $\|x_i - x_j\|^2$ and the parameter l are always positive, so the function $\exp\left(-\frac{\|x_i - x_j\|^2}{2l^2}\right)$ takes the value from 1, when $\|x_i - x_j\|^2$ is close to 0, and tends towards 0 when $\|x_i - x_j\|^2$ approaches $+\infty$. Therefore, we are enforcing that $y_i = f(x_i)$ and $y_j = f(x_j)$ are highly correlated if the x_i and x_j are close to each other. The closer the x's the higher is the correlation between the y and vice-versa. This perfectly describes the *smoothness assumption* we mentioned above. Indeed, all samples are continuous and infinitely differentiable (has derivatives of any order).

3.4.1 Prior distribution

At this point we answered all remaining open questions. We just learned how to construct the covariance matrix and before we have seen the connection between multivariate Gaussians and regression, and also the connection between multivariate Gaussians and Gaussian process regression. Therefore, we can apply all the tricks introduced in Section 3.1, like sampling some functions from the prior or predicting the value of test points given some training observation. Here we will focus on sampling from the prior, and see how the parameters of the kernel influences the function shapes.

If you think of Figure 7, what we are doing now, is exactly the same. At that point we only justified the use of a structured covariance by the *smoothness assumption*, but we had no recipe on how to construct it. Now, hopefully, it is more clear what we are doing. If we have no training examples, the dimension of the prior distribution depends on the number of test points[3], for example $N_* = |\mathbf{X}_*|$, where the operator $|\mathbf{X}|$ indicates the cardinality of the set $\mathbf{X}$. We build the covariance matrix using the points $x_i \in \mathbf{X}_*$ and the squared-exponential kernel and we usually assume mean vector $\boldsymbol{\mu} = \mathbf{0}$. Therefore, if we sample from this distribution, we get a random vector $\mathbf{y}_*$:

$$\mathbf{y}_* \sim \mathcal{N}(\mathbf{0}, K(\mathbf{X}_*, \mathbf{X}_*)) \tag{20}$$

where $K(\mathbf{X}_*, \mathbf{X}_*)$ denotes the $N_* \times N_*$ matrix of the kernel function evaluated at all pairs of test points. The random vector $\mathbf{y}_*$ will contain possible values for each $x_* \in \mathbf{X}_*$. In Fig. 9 and Fig. 10, we show different samples using different parameters for the lengthscale and kernel variance. It can be noticed that, if we use smaller lengthscale we get more wiggling functions, whereas increasing the kernel variance we get a prior with a larger 95% confidence interval. Notice that in the examples the test points are equidistant, but this need not be the case.

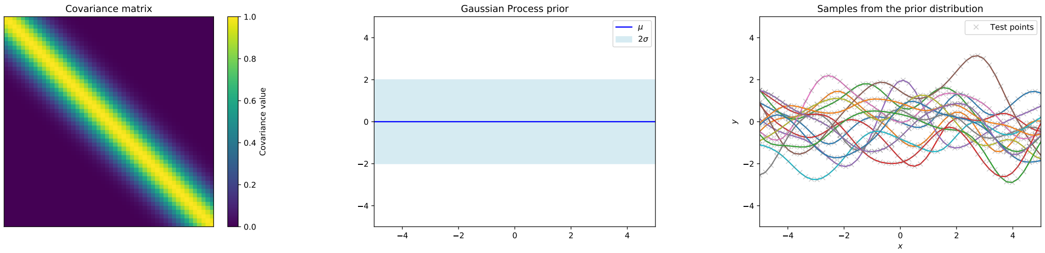

Figure 9: We represent the different steps to take to sample from the prior of a Gaussian process. We use the following parameters fro the square-exponential kernel: `lengthscale = 1`, `kernel_variance = 1`. First, we created the covariance matrix using $N = |\mathbf{X}_{\text{test}}| = 50$ test points (*Left*). The we plot the mean function and the 95% confidence interval (*Center*). Then, form this 50-dimension multivariate Gaussian we sample 15 function, i.e. 15 different values for our test points and plot it (*Right*).

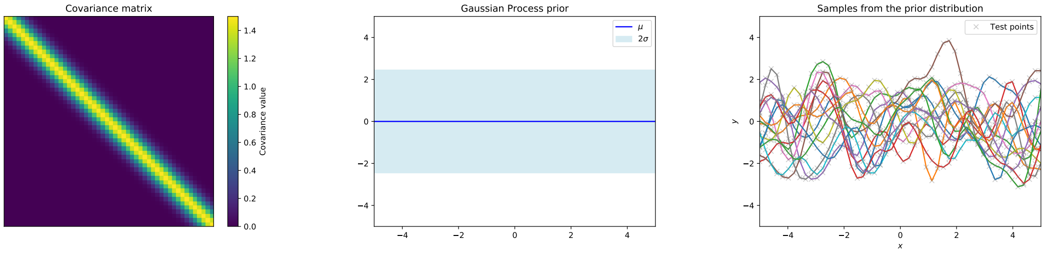

Figure 10: We represent the different steps to take to sample from the prior of a Gaussian process. We use the following parameters fro the square-exponential kernel: `lengthscale = 0.5`, `kernel_variance = 1.5`. First, we created the covariance matrix using $N = |\mathbf{X}_{\text{test}}| = 50$ test points (*Left*). The we plot the mean function and the 95% confidence interval (*Center*). Then, form this 50-dimension multivariate Gaussian we sample 15 function, i.e. 15 different values for our test points and plot it (*Right*).

[3] Note that in this case we are using the typical notation used in Gaussian Process, $\mathbf{X}_*$, to indicate the test set $\mathbf{X}_{\text{test}}$. In the same way, we are using x_* and y_* to denote the test points and respective predictions.

3.4.2 Prediction with noise-free observation

We are not usually interested in sampling random functions from the prior, but instead we want to use the information that training observations give us about the function to be able to predict values for future test points. As a starting point we consider the case in which the training observation are noise free, which is likely an unrealistic assumption in most real-world scenarios. In the non-linear regression setting we present at the beginning of these lecture notes, it means that $y_i = f(x_i, w)$, where the error ϵ does not appear. Before starting doing Gaussian Process regression, it is common practice to standardize the data. This way, each data feature and target will have mean 0 and standard deviation 1. After the standardization, the first step is to define the joint distribution of the training observations $\mathbf{y}$ and the test outputs $\mathbf{y}_*$ according to the prior:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix} \right) \tag{21}$$

where $\mathbf{X}$ is the matrix containing the training points $\mathbf{x}_i$ and $\mathbf{X}_*$ is the test set. If there are N training points and N_* test points then $K(X, X)$ is the $N \times N$ matrix of the kernel evaluated at all training points pairs, $K(X, X_*)$ is the $N \times N_*$ matrix with the kernels evaluated for all pairs created by combining the training points and test points. As before $K(X_*, X_*)$ contains the $N_* \times N_*$ elements obtained by computing the covariance for all pairs of test points (using the covariance function).

In case we do not standardize the data, the mean vector should also be taken into account. In this case the distribution above becomes:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \left(\begin{bmatrix} \boldsymbol{\mu}(\mathbf{X}) \\ \boldsymbol{\mu}(\mathbf{X}_*) \end{bmatrix}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix} \right) \tag{22}$$

To get the posterior, we are interested in the conditional probability of $\mathbf{y}_*|\mathbf{y}$, in this case given by $p(\mathbf{y}_*|\mathbf{y}, \mathbf{X}, \mathbf{X}_*)$, which corresponds to conditioning the joint Gaussian prior distribution on the observations. From Section 2.1 we learned that the resulting distribution is Gaussian and we are also able to compute its conditional mean and conditional covariance. In case of standardized data, we have:

$$\begin{aligned} p(\mathbf{y}_*|\mathbf{y}, \mathbf{X}, \mathbf{X}_*) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}}) \\ \boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}} &= K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} \mathbf{y} \\ \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}} &= K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X}_*) \end{aligned} \tag{23}$$

Otherwise, the mean should be taken into account:

$$\begin{aligned} p(\mathbf{y}_*|\mathbf{y}, \mathbf{X}, \mathbf{X}_*) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}}) \\ \boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}} &= \boldsymbol{\mu}(\mathbf{X}_*) + K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} (\mathbf{y} - \boldsymbol{\mu}(\mathbf{X})) \\ \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}} &= K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X}_*) \end{aligned} \tag{24}$$

As it can be seen in Fig. 11, all different samples go through the training points. If we look at the posterior distribution, we can notice that the uncertainty is really low close to the observation.

3.4.3 Prediction using noisy observation

As previously stated, in real-world applications it is more realistic to not have access to noiseless observations, but rather a noisy version of them. In the regression case, as in Section 1.2, we have $y_i = f(x_i, \mathbf{w}) + \epsilon$, where we also assume that ϵ is additive and independent normal distributed noise, with mean 0 and variance σ_n^2. Taking this into account, the prior of the noisy observation becomes:

$$\Sigma_{ij} = \mathrm{Cov}(y_i, y_j) = k(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{ij} \tag{25}$$

where δ_{ij} is a Kronecker delta which is equal to 1 if and only if $i = j$, and zero otherwise. We can re-write this in matrix notation:

$$\mathrm{Cov}(\mathbf{y}) = K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} \tag{26}$$

The joint distribution of the observed target values $\mathbf{y}$ and the values $\mathbf{y}_*$ at the test points $\mathbf{x}_*$ becomes

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{y}_*\end{bmatrix} \sim \left(\mathbf{0}, \begin{bmatrix}K(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I} & K(\mathbf{X},\mathbf{X}_*)\\ K(\mathbf{X}_*,\mathbf{X}) & K(\mathbf{X}_*,\mathbf{X}_*)\end{bmatrix}\right) \tag{27}$$

and if we do not standardize your data:

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{y}_*\end{bmatrix} \sim \left(\begin{bmatrix}\boldsymbol{\mu}(\mathbf{X})\\ \boldsymbol{\mu}(\mathbf{X}_*)\end{bmatrix}, \begin{bmatrix}K(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I} & K(\mathbf{X},\mathbf{X}_*)\\ K(\mathbf{X}_*,\mathbf{X}) & K(\mathbf{X}_*,\mathbf{X}_*)\end{bmatrix}\right) \tag{28}$$

We can derive the conditional distribution as we did before, and we obtain:

$$\begin{aligned} p(\mathbf{y}_*|\mathbf{y},\mathbf{X},\mathbf{X}_*) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}})\\ \boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}} &= K(\mathbf{X}_*,\mathbf{X})[K(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I}]^{-1}\mathbf{y}\\ \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}} &= K(\mathbf{X}_*,\mathbf{X}_*) - K(\mathbf{X}_*,\mathbf{X})[K(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I}]^{-1}K(\mathbf{X},\mathbf{X}_*) \end{aligned} \tag{29}$$

These are the key predictive equations for Gaussian Process regression. Since it can look a bit confusing, we can re-write it in a more compact notation. If we define $\mathbf{K} = K(\mathbf{X},\mathbf{X})$, $\mathbf{K}_* = K(\mathbf{X},\mathbf{X}_*)$ and $\mathbf{K}_{**} = K(\mathbf{X}_*,\mathbf{X}_*)$, we get:

$$\begin{aligned} p(\mathbf{y}_*|\mathbf{y},\mathbf{X},\mathbf{X}_*) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}})\\ \boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}} &= {\mathbf{K}_*}^T[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{y}\\ \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}} &= \mathbf{K}_{**} - {\mathbf{K}_*}^T[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{K}_* \end{aligned} \tag{30}$$

In case data is not standardized we have to let the target have a non-zero mean, we should be taken into account:

$$\begin{aligned} p(\mathbf{y}_*|\mathbf{y},\mathbf{X},\mathbf{X}_*) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}})\\ \boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}} &= \boldsymbol{\mu}(\mathbf{X}_*) + {\mathbf{K}_*}^T[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}(\mathbf{y}-\boldsymbol{\mu}(\mathbf{X}))\\ \boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}} &= \mathbf{K}_{**} - {\mathbf{K}_*}^T[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{K}_* \end{aligned} \tag{31}$$

3.4.4 Additional kernels

As mentioned earlier the kernel describes all our knowledge and expectations (the prior) about the relation between $\mathbf{X}$ and $\mathbf{y}$, hence the choice of kernel will be extremely important. The squared exponential kernel mentioned earlier is often used due to the simplicity and smoothness. Here, smoothness refers to the functional properties of the functions (as opposed to enforcing more smoothness by increasing the value of l) that can be expressed by the kernel, i.e. in case of the squared exponential kernel the values in the kernel are infinitely differentiable with respect to the values in x. While this assumption has merits due to the simplicity, properties of the resulting functions and computational stability, it may not always be able to express data in real-life scenarios. Hence, the socalled Matérn kernels (or Matérn covariance functions) are popular choice for real data and can be seen as a generalization of the squared exponential kernel

$$K_{\text{Matérn},\nu}(d) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{d}{l}\right)^{\nu} K_\nu\left(\sqrt{2\nu}\frac{d}{l}\right), \tag{32}$$

where d is a distance measure typically the Euclidean distance $\sqrt{||x_i - x_j||^2}$ and Γ is the gamma function, K_ν is the modified Bessel function of the second kind, and l and ν are positive parameters. As ν tends towards ∞ this expression will be equivalent to expression the squared exponential kernel.

$$\lim_{\nu\to\infty} K_{\text{Matérn},\nu}(d) = K_{\text{SE}}(d) = \sigma^2 \exp\left(-\frac{d^2}{2l^2}\right). \tag{33}$$

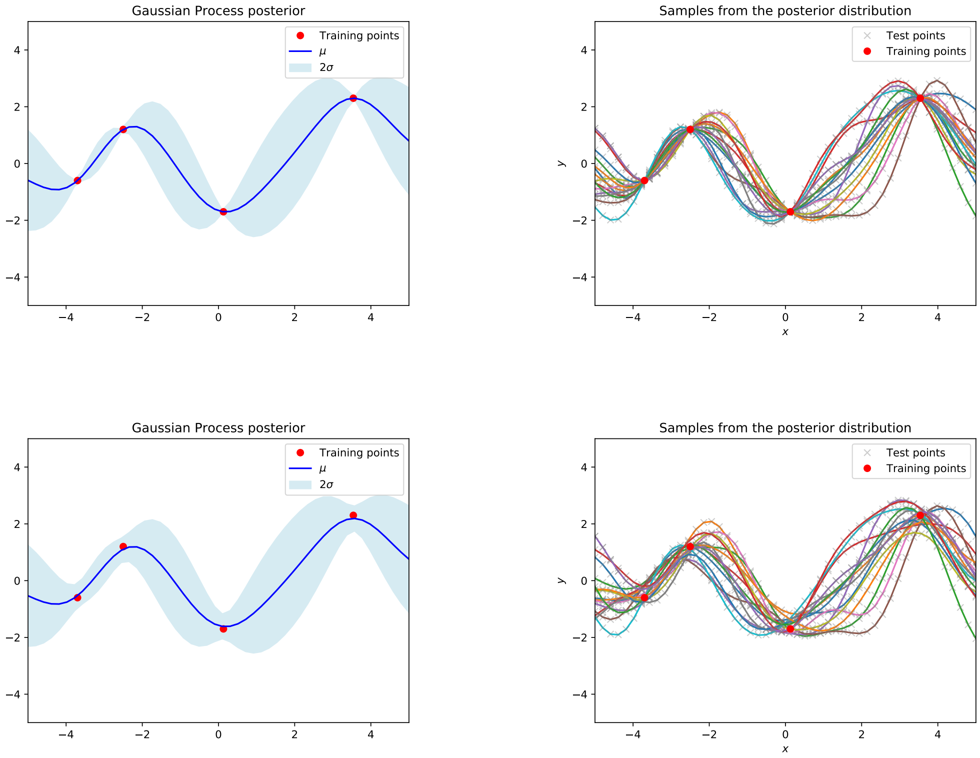

Figure 11: (*Upper plots*) Gaussian Process regression posterior with noise-free observations. We can see that the uncertainty is zero where we have observation and, at the same time, every sample goes exactly through the training points. (*Bottom plots*) Posterior distribution with noisy observations. In this case, we have that we have uncertainty also around the observation, and we have that some sample does not go through the training points.

For half-integer values of ν, i.e. $(\nu = p + 1/2,\ p \in \mathbb{N}^+)$, the kernel can be simplified such that the Bessel function evaluation can be omitted and written as a product of an exponential and a polynomial

$$K_{\text{Matérn},p+1/2}(d) = \sigma^2 \exp\left(-\frac{\sqrt{2p+1}d}{l}\right) \frac{p!}{(2p)!} \sum_{i=0}^{p} \frac{(p+i)!}{i!(p-i)!} \left(\frac{2\sqrt{2p+1}d}{l}\right)^{p-i}. \tag{34}$$

Hence, for $\nu = 5/2$ which is a popular choice in practical settings the kernel becomes

$$K_{\text{Matérn},5/2}(d) = \sigma^2 \left(1 + \frac{\sqrt{5}d}{l} + \frac{5d^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}d}{l}\right). \tag{35}$$

The Matérn kernel is no longer infinitely differentiable but rather $\nu - 1$ times differentiable. The Matérn kernel family is a flexible means to model most smooth functions; however, additional kernels exist allowing an even more expressive function repertoire for Gaussian process regression. Imagine one seeks to model a periodic function, where we want to be able to express that function values separated by exactly one period are also highly correlated. Here, the periodic kernel can be useful

$$k_{\text{periodic}}(d) = \sigma^2 \exp\left(-\frac{2\sin^2(\pi d/p)}{\ell^2}\right) \tag{36}$$

Another popular choice of kernel is the socalled linear kernel, this will allow fitting straight lines to the data and is defined as

$$k_{\text{linear}}(x_i, x_j) = \sigma_a^2 + \sigma_b^2(x_i - b)(x_j - b). \tag{37}$$

Note that the linear kernel depends on the absolute value of x and hence cannot be expressed as a function of the relative distance between the points, hence this kernel is non-stationary. Additionally, one should be aware that the inversion of the training kernel require $\mathcal{O}(N^3)$ operations, which means that Gaussian processes scale rather badly when the number of training samples is increased unless tricks (outside the scope of this course) to reduce the complexity are considered. For the linear kernel in particular, in which we are simply performing Bayesian linear regression, inference can actually be done in $\mathcal{O}(N)$ operations, which is why the naive algorithm considered here should not be considered for this case.

3.4.5 Combining kernels

A very powerful property of kernels is that they can easily be combined to allow more flexible modelling. For example addition or multiplication of two valid kernels will lead to another valid kernel typically with additional parameters. An example would be the multiplication of a squared exponential and a periodic kernel

$$K_{\text{periodic}}(d)K_{\text{SE}}(d) = \sigma^2 \exp\left(-\frac{2\sin^2(\pi d/p)}{\ell^2}\right)\exp\left(-\frac{d^2}{2\ell^2}\right), \tag{38}$$

where the two σ's are combined into one and that we have here assumed that the two length scale are the same. This kernel is also known as the local periodic kernel and will allow modelling of functions that are only locally periodic. Another example of where using multiple kernels is useful would be in cases where the input is multidimensional, and where smoothness over different length scales are expected in each dimension. In such a case it might be useful to define a kernel that uses the distance measure only for one dimension and then multiply several of these to effective obtain different length scales for each dimension. If the length scales are known this is equivalent to scaling each input prior to calculation of the kernel parameters; however, in cases where one seeks to identify the length scale optimizing the kernel parameters for the training data as described in a later section may allow the length scales to be inferred automatically.

3.4.6 Step-by-step implementation of the Gaussian Process regression algorithm

Here, we present a practical implementation of the Gaussian Process regression algorithm (see Algorithm 1). We are going to use the Cholesky decomposition, instead of inverting directly the matrices[4] because it is faster and numerically more stable to compute the inverse of a lower triangular matrix than a normal one. The Cholesky decomposition is used to decompose a symmetric, positive definite matrix $\mathbf{A}$ into a product of a lower triangular matrix $\mathbf{L}$ and its transpose:

$$\mathbf{L}\mathbf{L}^T = \mathbf{A} \tag{39}$$

where $\mathbf{L}$ is called the Cholesky factor. This decomposition is useful when we want to solve linear systems that involve symmetric, positive definite matrices, such as solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$. In our settings, if we consider the mean that we want to compute

$$\boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}} = \mathbf{K}_*{}^T[\mathbf{K} + \sigma_n^2\mathbf{I}]^{-1}\mathbf{y}$$

we can notice that we can write the term

$$[\mathbf{K} + \sigma_n^2\mathbf{I}]^{-1}\mathbf{y}$$

as the result of the following linear system

$$[\mathbf{K} + \sigma_n^2\mathbf{I}]\boldsymbol{\alpha} = \mathbf{y}$$

[4] In case you want to implement the algorithm using the direct inversion, we suggest you to use the `numpy.linalg.pinv` function from the NumPy library.

$$\implies \boldsymbol{\alpha} = [\mathbf{K} + \sigma_n^2 \mathbf{I}]^{-1}\mathbf{y}$$

Therefore, to solve it in a more stable way, we can compute the Cholesky decomposition of $\mathbf{K}+\sigma_n^2\mathbf{I}$, so we have:

$$\mathbf{K} + \sigma_n^2\mathbf{I} = \mathbf{L}\mathbf{L}^T$$

If we rewrite

$$\boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}} = \mathbf{K}_*{}^T[\mathbf{K} + \sigma_n^2\mathbf{I}]^{-1}\mathbf{y} = \mathbf{K}_*{}^T\boldsymbol{\alpha}$$

we have that

$$\boldsymbol{\alpha} = [\mathbf{K} + \sigma_n^2\mathbf{I}]^{-1}\mathbf{y} = \mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{y}$$

because we have that $[\mathbf{K} + \sigma_n^2 I]^{-1} = (\mathbf{L}\mathbf{L}^T)^{-1} = \mathbf{L}^{-T}\mathbf{L}^{-1}$ due to the properties that $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$. Here we can see that $\mathbf{L}^{-1}\mathbf{y} = \mathbf{m}$ is a linear system and also $\mathbf{L}^{-T}\mathbf{m} = \boldsymbol{\alpha}$ is another linear system. Here we use the *backslash operator* $\backslash$ to indicate the solution of linear system[5]. If we use it, we can write:

$$\boldsymbol{\alpha} = \mathbf{L}^T\backslash(\mathbf{L}\backslash\mathbf{y})$$

We can also go through the computation needed to compute the covariance of the posterior distribution:

$$\boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}} = \mathbf{K}_{**} - \mathbf{K}_*{}^T[\mathbf{K} + \sigma_n^2\mathbf{I}]^{-1}\mathbf{K}_*$$

and if we focus on the right term of the subtraction. We can substitute $\mathbf{K} + \sigma_n^2\mathbf{I} = \mathbf{L}\mathbf{L}^T$ in the right term:

$$\mathbf{K}_*{}^T[\mathbf{K} + \sigma_n^2\mathbf{I}]^{-1}\mathbf{K}_* = \mathbf{K}_*{}^T(\mathbf{L}\mathbf{L}^T)^{-1}\mathbf{K}_*$$

and using the same properties of the inverse matrices we get

$$\mathbf{K}_*{}^T(\mathbf{L}\mathbf{L}^T)^{-1}\mathbf{K}_* = \mathbf{K}_*{}^T\mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{K}_*$$

If we use the property of the transpose of the product of two matrices $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$ we can write

$$\mathbf{K}_*{}^T\mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{K}_* = (\mathbf{L}^{-1}\mathbf{K}_*)^T\mathbf{L}^{-1}\mathbf{K}_*$$

where we notice that we can compute the same quantities once. As we have seen above, $\mathbf{L}^{-1}\mathbf{K}_*$ is the solution, $\mathbf{v}$, to the following linear system:

$$\mathbf{L}\mathbf{v} = \mathbf{K}_*$$

$$\implies \mathbf{v} = \mathbf{L}^{-1}\mathbf{K}_*$$

and using the *backslash operator* we have

$$\mathbf{v} = \mathbf{L}\backslash\mathbf{K}_*$$

With this result in mind, we can compute the covariance of the posterior in the following way:

$$\boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}} = \mathbf{K}_{**} - \mathbf{v}^T\mathbf{v}$$

The algorithm is represented in a more compact and readable way below.

Algorithm 1: Gaussian Process regression

```
Start computing $\mathbf{K} = K(\mathbf{X},\mathbf{X})$, $\mathbf{K}_* = K(\mathbf{X},\mathbf{X}_*)$ and $\mathbf{K}_{**} = K(\mathbf{X}_*,\mathbf{X}_*)$;
$\mathbf{L}$ = Cholesky$(\mathbf{K} + \sigma_n^2\mathbf{I})$;
$\boldsymbol{\alpha} = \mathbf{L}^T\backslash(\mathbf{L}\backslash\mathbf{y})$;
$\boldsymbol{\mu}_{\mathbf{y}_*|\mathbf{y}} = \mathbf{K}_*{}^T\boldsymbol{\alpha}$;
$\mathbf{v} = \mathbf{L}\backslash\mathbf{K}_*$;
$\boldsymbol{\Sigma}_{\mathbf{y}_*|\mathbf{y}} = \mathbf{K}_{**} - \mathbf{v}^T\mathbf{v}$
```

[5] In Python, this can be easily done with the function `numpy.linalg.solve`.

Why should we standardize our data? There are a number of reasons why we should standardize the data. The first reason is to match the prior assumption of the targets having mean 0. In addition to that, we have seen that the kernel function has a scale parameter, the *kernel variance*, therefore having standardize inputs we should get a better estimation of it when we are going to optimize the parameters. The most important reason, however, is that to compute the conditional distribution, we are computing the inverse of matrices and this can create numerical problems for ill-conditioned matrices.

3.4.7 Parameter optimization

There is still one key issue open. We have three hyperparameters to tune in the Gaussian Process regression: the *lengthscale* l, the *kernel variance* σ^2 and the *noise variance* σ_n^2 of the observations. One way to choose the right values for these hyperparameters is to choose the ones that maximize the log-likelihood of the training data.

$$\log p(\mathbf{y}|\mathbf{X}) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K} + \sigma_n^2\mathbf{I}) = -\frac{1}{2}\mathbf{y}^T[\mathbf{K} + \sigma_n^2\mathbf{I}]^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K} + \sigma_n^2\mathbf{I}| - \frac{N}{2}\log(2\pi) \tag{40}$$

where $\mathbf{K} = K(\mathbf{X}, \mathbf{X})$ as before, and $|\mathbf{K} + \sigma_n^2\mathbf{I}|$ is the determinant[6] of the matrix $\mathbf{K} + \sigma_n^2\mathbf{I}$.

If we use Algorithm 1, we can use the properties that the determinant of a positive definite symmetric matrix can be calculated efficiently by using its Cholesky factor $\mathbf{L}$. Indeed, we have:

$$|\mathbf{K} + \sigma_n^2\mathbf{I}| = |\mathbf{L}\mathbf{L}^T| = |\mathbf{L}||\mathbf{L}^T| = |\mathbf{L}|^2 = \prod_{i=1}^{n} L_{ii}^2 \tag{41}$$

because L is lower triangular, and in case of the log-determinant, this becomes

$$\log|\mathbf{K} + \sigma_n^2\mathbf{I}| = 2\sum_{i=1}^{n}\log L_{ii} \tag{42}$$

Therefore, we can compute the log-likelihood in the following way:

$$\log p(\mathbf{y}|\mathbf{X}) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K} + \sigma_n^2\mathbf{I}) = -\frac{1}{2}\mathbf{y}^T\boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{N}{2}\log(2\pi) \tag{43}$$

where we are summing up the logarithms of the elements of the diagonal of the matrix $\mathbf{L}$.

Tricks to optimize the parameters using Gradient Descent While most libraries use Quasi-Newton methods, like the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, for simplicity we will here rely on gradient descent for optimization of the hyperparameters of the Guassian Process. Remember that we want to optimize the lengthscale, the output variance and the noise variance. When we have only a few points it might happen that the optimization process tries to explain the points using the noise variance. Therefore, the mean function of the Gaussian process will result in a straight (with some slope) line, because we get an high value for the lengthscale, and not a function that goes close to the points we have, which is what we would like to get, especially if we apply GPs to Bayesian Optimization. There are two possible solution to avoid this to happen.

One possible way to avoid this to happen, is pretty simple. We can optimize only the lengthscale and the output variance, while we keep the noise variance fixed to a really small value. The second possibility, instead, is to put a prior on the value of the lengtscale, that is $l \sim p(l)$. Therefore, our optimization problem becomes:

$$\{l, o, n\} \in \arg\max_{l,o,n}\{p(l, o, n \mid D\} = \arg\max_{l,o,n}\left\{\prod_{i=1}^{n}\left[p(D_i \mid l, o, n)\right]p(l)\right\}$$

[6] In Python you can compute the determinant using the function `numpy.linalg.slogdet`.

where l is the lengthscale, o is the output variance and n the noise variance. Taking the logarithm we get:

$$\{l, o, n\} \in \arg\max_{l,o,n} \left\{ \sum_{i=1}^{n} \left[\log p(D_i \mid l, o, n) \right] + \log p(l) \right\}$$

This is the new function we should maximize (or minimize the negated function) in the optimization process.

As an example, we will here present how to compute the logarithm of the prior, when we use a Log-normal prior for the lengthscale. The Log-Normal distribution is a continuous distribution of a random variable whose logarithm is normally distributed. The probability density function is given by:

$$p(l) = \frac{1}{l} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln l - \mu)^2}{2\sigma^2} \right)$$

We can derive the logarithm of the density function, $\log p(l)$:

$$\begin{aligned} p(l) &= \log \left(\frac{1}{l} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln l - \mu)^2}{2\sigma^2} \right) \right) \\ &= \log \frac{1}{l} + \log \left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln l - \mu)^2}{2\sigma^2} \right) \right) \\ &= -\log(l) + \log \mathcal{N}(\ln l \mid \mu, \sigma^2) \end{aligned}$$

References and useful links

The following references were used to write the following notes. These references are also meant to be used to get a different introduction to the topic of Gaussian Process regression and/or uncertainty in machine learning.

- *“Gaussian Processes for Machine Learning”*, Carl Edward Rasmussen and Chris Williams, the MIT Press, 2006. You can get the online version for free.

- *“Uncertainty in deep learning”*, Yarin Gal, Diss. PhD thesis, University of Cambridge, 2016. You can access it here

- *“A Visual Exploration of Gaussian Processes”*, by Görtler et al. Distill, 2019. Interactive publication of Gaussian Processes.

- The following YouTube video lectures:

  - “Machine learning - Introduction to Gaussian processes”, lecture by Nando De Freitas at University of British Columbia. Link
  - “ML Tutorial: Gaussian Processes”, lecture by Richard Turner at Imperial College London. Link
  - “Gaussian Process Basics”, lecture by David Mackay. Link
  - “Machine Learning Lecture 26 ”Gaussian Processes”, lecture by Kilian Weinberger at Cornell University. Link