Machine Learning in the Physical World

Practical Gaussian Processes

Carl Henrik Ek

October 6, 2024

Abstract

Gaussian processes provide a probability measure that allows us to perform statistical inference over the space of functions. While GPs are nice as mathematical objects when we need to implement them in practice we often run into issues. In this worksheet we will do a little bit of a whirlwind tour of a couple of approaches to address these problems. We will look at how we can address the numerical issues that often appear and we will look at approximations to circumvent the computational cost associated with Gaussian processes. Importantly when continuing using these models in the course you are most likely not going to implement them yourself but instead use some of the many excellent software packages that exists. The methods that we describe here are going to show you how these packages implement GPs and it will hopefully give you an idea of the type of thinking that goes into implementation of machine learning models.

So far we have not done any type of *learning* with Gaussian processes. Learning is the process where we adapt the parameters of the model to the data that we observe. For a probablistic model the object here is to fit the distribution of the model such that the data has high likelihood under the model. You can do this at many different levels, you can fit the parameters of the likelihood directly to data, referred to as *maximum likelihood* but in that case you have not taken into account the information in the prior, i.e. you are fitting the data in a completely unconstrained way which is likely to lead to overfitting. Instead we try to marginalise out as many of the parameters as we can to reflect the knowledge that we have of the problem and then optimise the remaining ones. Remember, there will always be parameters left in a model as long as you place a parametrised distribution that you integrate out. For Gaussian processes, in most practical applications, we marginalise out the prior over the function values, $p(f \mid \theta)$ from the likelihood to reach the marginal likelihood,

$$p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}) = \int p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{X}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta},$$

to reach the *marginal likelihood* which does not have $\mathbf{f}$ as parameters. However, this distribution still has the parameters of the prior[1] $\boldsymbol{\theta}$ as a dependency. Learning now implies altering these parameters so that we find the ones that the probability of the observed data is maximised,

$$\boldsymbol{\theta}^* = \mathrm{argmax}_\theta \, p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}).$$

Now let us do this specifically for a zero mean Gaussian process prior. We will focus on the zero mean setting as it is usually not the mean that gives us problems but the covariance function. However, if you want to have a parametrised mean function most of the things that we talk about will be the same. Given that most distributions you will ever work with is in the exponential class the normal approach to this is to maximise the *log marginal likelihood*. If we write this up for a Gaussian process it will have the following form,

$$\begin{aligned}\log p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}) = &\int p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{X}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \\ &- \frac{1}{2} \mathbf{y}^{\mathrm{T}} \left(k(\mathbf{X}, \mathbf{X} + \beta^{-1}\mathbf{I}) \right)^{-1} \mathbf{y} - \frac{1}{2} \log \det \left(k(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I} \right) - \frac{N}{2} \log 2\pi. \end{aligned} \tag{1}$$

[1] sometimes you will hear these refereed to as hyper-parameters.

If we combine the noise variance and the evaluation of the covariance function into a single matrix $\mathbf{K}$ the two terms where the covariance matrix appear is,

$$\log \det \mathbf{K},$$

and in quadratic form together with the data,

$$\mathbf{y}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y}.$$

The parameters that we are interesting in learning are the ones that are part of generating the covariance function.

In this document we will first look at how we can make the operations above suffer less from issues of numerical stability. We will then look at how we can derive an approximation to the expression above to reduce the cubic computational complexity associated with the matrix inverse in the last expression.

1 Numerical Stability

To address the numerical issues related to the two expressions above we are going to exploit that the problem that we work on is actually quite structured. The covariance matrix is in a class of matrices that are called positive-definite matrices meaning they are symmetric, full rank and with all eigen-values positive. This we can exploit to make computations better conditioned. To do so we are going to use the Cholesky decompostion which allows us to write a positive definite matrix as a product of a lower-triangular matrix and its transpose,

$$\mathbf{K} = \mathbf{L}\mathbf{L}^{\mathrm{T}}.$$

Lets first look at how the decomposition can be used to compute the `log determinant` term in the marginal log-likelihood,

$$\log \det \mathbf{K} = \log \det \left(\mathbf{L}\mathbf{L}^{\mathrm{T}}\right) = \log \left(\det \mathbf{L}\right)^2.$$

Now we have to compute the determinant of the Cholesky factor $\mathbf{L}$, this turns out to be very easy as the determinant of a upper/lower diagonal matrix is the product of the values on the diagonal,

$$\det \mathbf{L} = \prod_i^N \ell_{ii}.$$

If we put everything together we get this,

$$\log \det \mathbf{K} = \log \left(\prod_i^N \ell_{ii}\right)^2 = 2\sum_i^N \ell_{ii}.$$

So the `log determinant` of the covariance matrix is simply the sum of the diagonal elements of the Cholesky factor. From our first classes in Computer science we know that summing lots of values of different scale is much better for keeping precision compared to taking the product.

You can also use the Cholesky factors in order to do address the term that includes the inverse and making this better conditioned. What we will do is to solve the inverse by solving two systems of linear equations, both who have already been made into upper and lower triangular form therefore being trivial to solve. This might sounds a bit confusing but when you see the math hopefully it becomes obvious. Lets write down a system of linear equations,

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{2}$$

The brute-force way to solve this would be as,

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b},$$

but if we know that $\mathbf{A}$ is positive-semi definite we can exploit this using the Cholesky factors. Lets re-write the system using this knowledge,

$$\mathbf{L}\mathbf{L}^{\mathrm{T}}\mathbf{x} = \mathbf{b}. \tag{3}$$

The matrix $\mathbf{L}$ is now a lower triangular matrix so we can instead look at two system of equations that we can solve, one using forward substitution and then finally one using backward substition. First lets re-write the right-hand side of the equation,

$$\mathbf{b} = \mathbf{L}\mathbf{y}.$$

We can now easily solve for $\mathbf{y}$ by exploiting the lower-triangular structure,

$$\begin{array}{lllllllll} \ell_{1,1}y_1 & & & & & & & = & b_1 \\ \ell_{2,1}y_1 & + & \ell_{2,2}y_2 & & & & & = & b_2 \\ \vdots & & & & & & & & \\ \ell_{n,1}y_1 & + & \ell_{n,2}y_2 & + & \ldots & + & \ell_{n,n}y_n & = & b_n, \end{array} \tag{4}$$

by using $y_1 = \frac{b_1}{\ell_{1,1}}$ and then use this results to solve,

$$y_2 = \frac{b_2 - \ell_{2,1}y_1}{\ell_{2,2}},$$

we can then continue to solve for all of $\mathbf{y}$ from the top to the bottom, this process is called *forward substitution*. This is an intermediate result that we can use to find $\mathbf{x}$. To see how to do this we bring back the left-hand side of the equation,

$$\mathbf{L}\mathbf{L}^{\mathrm{T}}\mathbf{x} = \mathbf{L}\mathbf{y} \tag{5}$$

$$\mathbf{L}^{\mathrm{T}}\mathbf{x} = \mathbf{y}. \tag{6}$$

Now we have the same type of system as we had in the previous case where we have a matrix of coefficients that is now upper-triangular being the transpose of the lower-triangular Cholesky factor,

$$\begin{array}{lllllllll} \ell_{n,1}x_1 & + & \ell_{n,2}x_2 & + & \ldots & + & \ell_{n,n}y_n & = & y_1 \\ \ell_{2,1}x_1 & + & \ell_{2,2}x_2 & & & & & = & y_2 \\ \vdots & & & & & & & & \\ \ell_{1,1}x_n & & & & & & & = & y_n, \end{array} \tag{7}$$

so we can now use the same approach to solve for $\mathbf{x}$ just going from the bottom row upwards instead which is called *back-substitution*. Here we have written the general solution to a system of linear equations using the Cholesky factors, in the scenario where we are looking for the matrix this implies that we are trying to solve,

$$\mathbf{A}\mathbf{x}_i = \mathbf{e}_i,$$

where $\mathbf{e}_i$ is the canonical basis of the vector space. By this approach we have avoided solving the inverse of a matrix and instead found the solution by solving two simpler systems of linear equations by using the Cholesky factors. Obviously we still need to find the decomposition itself and in total the two paths to the solution have the same complexity it is just that the latter is more stable because we are exploiting the structure of the problem.

Looking back at the marginal log likelihood we see that the inverse of the covariance matrix appears in a quadratic form. We are going to exploit the fact that we do not actually need to have the inverse in explicit form, we just need the result of the computation using the inverse. This is something we can do using the Cholesky factorisation again,

$$\begin{aligned} \mathbf{y}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y} &= \mathbf{y}^{\mathrm{T}}\mathbf{L}\mathbf{L}^{\mathrm{T}^{-1}}\mathbf{y} \\ &= \mathbf{y}^{\mathrm{T}}\left(\mathbf{L}^{-1}\right)^{\mathrm{T}}\mathbf{L}^{-1}\mathbf{y} \\ &= \left(\mathbf{L}^{-1}\mathbf{y}\right)^{\mathrm{T}}\mathbf{L}^{-1}\mathbf{y} \\ &= \mathbf{z}^{\mathrm{T}}\mathbf{z}. \end{aligned} \tag{8}$$

Now we can solve the simple system $\mathbf{Lz} = \mathbf{y}$ using forward substitution as explained above. So in some ways, rather than storing the full covariance matrix all you really need is the Cholesky factors of the matrix.

1.1 Improving the conditioning of a matrix

Sometimes, you are going to have issues with the co-variance matrix being massively ill-conditioned. This happens when the range of eigen-values are really large. This will effect either an approach based on Cholesky factors or when you apply a general matrix inverse. What you can do in these scenarios is to add a small value to the diagonal to the co-variance matrix before you perform anything involving the inverse. This will in effect just make the function values more independent. This is fine to do as, and will if it has any effect at all, be recovered from as the marginal log-likelihood has the `log-determinant` term which penalises this effect. What you can do is to put whatever code that you have involving the co-variance matrix in a `try {};catch {}` statement and then keep adding a small diagonal matrix till the computation works. If you look into most GP libraries you will see some variant of this. At best you are not going to notice a difference as you are adding such small values so that you are down to errors relating to finite computation and at worst you are going to have to do a few extra loops of training because you are introducing an artificial independence.

2 Approximate Gaussian Processes

To compute the marginal likelihood of the Gaussian process requires inverting a matrix that is the size of the data. As we have already seen this can be numerically tricky. It is also a very expensive process of cubic complexity which severely limits the size of data-sets that we can use. There has been a lot of work over the last few decades focusing on this, way too much to outline in this report. Good papers that outline the idea are these Candela et al., 2005; Snelson et al., 2006. The idea is to reduce the degrees of freedom of the problem and say that rather than using all the data-points that you see as a degrees of freedom instead assume that there is a smaller set of points, usually referred to as *inducing points* that are sufficient to describe all the other points. In some works these points are a subset of the observed data and in other works they are completely hallucinated points with no associated observations. In practice what this means is that we have a set of N function values f that we want to describe and we are going to use a set of M *inducing* function values u to describe them. Due to the marginal property of the Gaussian process this means that we can now write the joint distribution as[2],

$$p(\mathbf{f}, \mathbf{u}) = \mathcal{N}\left(\begin{bmatrix} f \\ \mu_u \end{bmatrix}, \begin{bmatrix} K_{ff} & K_{fu} \\ K_{uf} & K_{uu} \end{bmatrix}\right).$$

We can then use the chain rule and write the joint such as,

$$p(\mathbf{f}, \mathbf{u}) = p(\mathbf{f} \mid \mathbf{u})p(\mathbf{u}).$$

The first term on the left-hand side of the equation is the normal predictive Gaussian distribution which will have the form,

$$p(\mathbf{f} \mid \mathbf{u}) = \mathcal{N}\left(K_{fu}K_{uu}^{-1}\mathbf{u}, K_{ff} - K_{fu}K_{uu}^{-1}K_{uf}\right).$$

We are now going to introduce the likelihood function, however for the inducing locations we do not have observations corresponding to the inducing function values. This means that the only likelihood term we have is,

$$p(\mathbf{y} \mid \mathbf{f}) = \mathcal{N}(\mathbf{y} \mid \mathbf{f}, \beta^{-1}\mathbf{I}).$$

Now let us write up the log marginal likelihood,

$$\begin{aligned} \log p(\mathbf{y}) &= \log \int p(\mathbf{y} \mid \mathbf{f})p(\mathbf{f})\mathrm{d}\mathbf{f} \\ &= \log \int p(\mathbf{y} \mid \mathbf{f})p(\mathbf{f} \mid \mathbf{u})p(\mathbf{u})\mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u}. \end{aligned}$$

[2] where I have omitted the input locations for clarity.

We are now going to proceed to derive a bound on the integral above. We are going to exploit something called the *Jensen's Inequality*. This inequality relates the value of a convex function of an integral, with that of the integral of the convex function. Specifically it states that,

$$f(\int g \ \mathrm{d}x) \leq \int f \circ g \ \mathrm{d}x,$$

if f is a convex function. To get an intuition for this where we just consider two points,

$$f(\lambda x_0 + (1-\lambda)x_1) \leq f(\lambda f(x_0) + (1-\lambda)f(x_1)),$$

where $\lambda \in [0, 1]$. The inequality is a generalisation of this argument. In this case the function that we are going to use is the logarithm which is a concave function so the relationship is flipped,

$$\log(\int g \ \mathrm{d}x) \geq \int \log(g) \ \mathrm{d}x.$$

Using the logarithm as the concave function we are now going to re-write the log-marginal likelihood,

$$\begin{aligned}\log p(\mathbf{y}) &= \log \int p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u}) \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u} \\ &= \log \int \frac{q(\mathbf{f})q(\mathbf{u})}{q(\mathbf{f})q(\mathbf{u})} p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u}) \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u},\end{aligned}$$

where $q(\mathbf{f})$ and $q(\mathbf{u})$ are probability distributions. This might seem like quite an arbitrary thing to do but we will now apply the inequality to the expression above and hopefully it will start to make sense.

$$\begin{aligned}\log p(\mathbf{y}) &= \log \int \frac{q(\mathbf{f})q(\mathbf{u})}{q(\mathbf{f})q(\mathbf{u})} p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u}) \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u} \\ &\geq \int q(\mathbf{f})q(\mathbf{u}) \log \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u})}{q(\mathbf{f})q(\mathbf{u})} \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u}\end{aligned}$$

The $q(\cdot)$ distributions are free distributions that we can choose however we want. However, you can show that by making these distributions the true posterior distributions the bound will actually be tight [3]. This is the basis for an approximate inference scheme called *variational inference* Blei et al., 2016 where we pick parametrised distributions, called variational distributions, $q(\cdot)$ and then try to maximise the right side of the expression above, as this is maximising a *lower bound* on the marginal likelihood. So what should we choose the variational distributions to be? Lets start with $q(\mathbf{f})$, the bound will be tight if,

$$q(\mathbf{f}) = p(\mathbf{f} \mid \mathbf{u}, \mathbf{y}),$$

where $\mathbf{y}$ are noisy observations of the true function values $\mathbf{f}$. Now let us try to remember why we introduced the inducing outputs $\mathbf{u}$, we aimed to choose them so that they are as representative of the function we want to learn. The most extreme case of this is if they are *perfectly* representing $\mathbf{f}$ which in statistical terms would mean that they are *sufficient statistics* for $\mathbf{f}$. If this was the case it would imply that,

$$p(\mathbf{f} \mid \mathbf{u}, \mathbf{y}) = p(\mathbf{f} \mid \mathbf{u}).$$

So we are going to choose the right-hand side of the equation above to be our variational distribution. Lets see what happens when we plug this into the derivation of the log-marginal likelihood,

$$\begin{aligned}\log p(\mathbf{y}) &\geq \int q(\mathbf{f})q(\mathbf{u}) \log \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u})}{q(\mathbf{f})q(\mathbf{u})} \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u} \\ &= \int p(\mathbf{f} \mid \mathbf{u})q(\mathbf{u}) \log \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{u}) p(\mathbf{u})}{p(\mathbf{f} \mid \mathbf{u})q(\mathbf{u})} \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u}, \\ &= \int p(\mathbf{f} \mid \mathbf{u})q(\mathbf{u}) \log \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{u})}{q(\mathbf{u})} \mathrm{d}\mathbf{f}\mathrm{d}\mathbf{u} \\ &= \int p(\mathbf{f} \mid \mathbf{u}) \log p(\mathbf{y} \mid \mathbf{f}) \mathrm{d}\mathbf{f} + \int q(\mathbf{u}) \log \frac{p(\mathbf{u})}{q(\mathbf{u})} \mathrm{d}\mathbf{u}.\end{aligned}$$

[3] feel free to do this to show that it is true

What we have now derived is a lower bound on the log marginal likelihood where we have made the assumption that the inducing points are sufficient statistics for the whole function. The key thing why this is useful is that the approach above only requires inversion of a matrix the size of the inducing points, if we have m inducing points and n data points the complexity of the above computation is $\mathcal{O}(nm^2)$ compared to $\mathcal{O}(n^3)$ of the direct approach. This approach was initially proposed in Titsias, 2009 and there have been lots of different follow-up works from this. What remains to be decided is the form of $q(\mathbf{u})$. In the original paper ibid. it was shown how the optimal inducing points could be derived directly. However, in a later paper Hensman et al., 2013 it was shown that by making a slightly looser bound and representing $q(\mathbf{u})$ explicitly it was possible to formulate a stochastic optimisation procedure which allowed for scaling to much larger data-sets.

3 Software Packages

As you have probably noticed there is a lot of challenges associated with performing learning and inference with Gaussian processes. We have now seen an overview some of the approaches we can do to make computations more stable and how we can use approximations to make things faster and more scalable. This being said there is a lot more trickery to things than we can cover here so while you now know Gaussian processes from a theoretical perspective and have and idea of how we have to work with them from now on we will use external packages to work with GPs. There is a plethora of packages out there all with different pros and cons. Rather than telling you to use a specific one I'm just going to give you a list of a few of the packages as I am sure you are much better suited to choose what to work with than me.

GPy GPy is extension of the Gaussian processes `Matlab` library GPMat that was built by Prof. Neil D. Lawrence and his group at the University of Sheffield. It was developed before `autodiff` became a big thing which have both pros and cons. The positive is that it is probably the most nummerically stable software around as derivations have been done by hand and all the nummerical tricks have been applied. The negative is that because of the lack of an autodiff packages that takes care of the computational graph all the passing of variables have to be done within the software. This means that it is quite tedious to expand the software with new components and its a lot of code that can be quite overwhelming to get your head around.

GPFlow With the introduction of the first larger autodiff packages some of the developers of GPy ported the software to `Tensorflow`. It is quite a mature package as it built on the work done in GPy and most of the nummerical issues from Tensorflow has been ironed out. The negative is that you have to work within Tensorflow.

GPJax Jax was initially started as a way of using autodiff without the additional bloat that Tensorflow introduced. The idea was that it was supposed to be `numpy` with auto-diff. GPJax is a toolbox that includes a lot of the tools needed to work with GP models. While it is not as extensive as GPFlow or GPTorch it feels a lot more lightweight and adaptable compared to the others.

GPTorch For those of you who are used to `Torch` this is a GP toolbox using Torch as a backend. The documentation is very good and while not as extensive as GPFlow I find it easier to work with.

4 Summary

In the previous worksheet we tried to "sell" Gaussian processes to you, hopefully we managed to convince you that they are excellent models that allows for principled uncertainty quantification over the space of functions. But the no-free lunch theorem applies to everything we do so clearly there is a dark side to GPs. Compared to neural networks which lacks theory and understanding but our simple to implement Gaussian processes have the complete opposite characteristics. Here we tried to give you a couple of small set of hint on how we have to reason and think to make GPs work in practice, as you probably noticed that boring course you took in Numerical Analysis sometime in your undergraduate days is indeed going to turn out to be very relevant when working in machine learning. The field of approximate inference is a very active one

and the field is very much trying to come up with the best way of performing inference in these models. There are lots of things we haven't covered but hopefully seeing the basic variational bound gives you an idea of how this work is done.

The motivation for deriving the variational bound was to address the computational complexity of the marginal likelihood for models that were analytically tractable. However, very often we have to work with models that are not analytically tractable, for example when we have non-gaussian likelihoods or when we want to integrate out the inputs from the model. Both of these scenarios forces us to do approximations as it is not possible to compute the marginal likelihood in closed form. An excellent presentation on non-gaussian likelihoods can be found here URL and a good place to read about integrating out the input locations[4] can be found here Damianou, 2015.

References

Blei, David M., Alp Kucukelbir, and Jon D. McAuliffe (2016). "Variational Inference: a Review for Statisticians." In: *CoRR*. arXiv: `1601.00670 [stat.CO]`.

Candela, Joaquin Quiñonero and Carl Edward Rasmussen (2005). "A Unifying View of Sparse Approximate Gaussian Process Regression." In: *Journal of Machine Learning Research* 6, pp. 1939–1959.

Damianou, Andreas C (Feb. 2015). "Deep Gaussian Processes and Variational Propagation of Uncertainty." PhD thesis. University of Sheffield.

Hensman, James, N Fusi, and Neil D Lawrence (2013). "Gaussian Processes for Big Data." In: *Uncertainty in Artificial Intelligence*.

Snelson, E and Zoubin Ghahramani (2006). "Sparse Gaussian processes using pseudo-inputs." In: *Advances in Neural Information Processing Systems* 18, p. 1257.

Titsias, Michalis (2009). "Variational Learning of Inducing Variables in Sparse Gaussian Processes." In: *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*. Ed. by David van Dyk and Max Welling. Vol. 5. Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, pp. 567–574.

[4] which leads to composite Gaussian proceeses