Having Fun with Gaussian Processes

Emilio Maddalena and Yingzhao Lian

March 2021

1 Setting up the stage

We will use `Python 3`. The package `GPflow 2` will be our easy-to-use toolbox, which implements vanilla GPs, sparse GPs, multi-output GPs, and much more. Take a look at `www.gpflow.org` and `https://gpflow.readthedocs.io/en/develop/` for more details.

In order to make sure everything is in order, try reconstructing a simple $f(x) = \sin(x)$.

We have 30 samples obtained uniformly from $-2\pi \leq x \leq -2\pi$. Also, the dataset is contaminated with zero-mean Gaussian noise with variance 0.1. Note that the GP doesn't have access to this latter value, and will try to estimate it as a hyperparameter.

Run `simple_test.py` and see how well a squared-exponential GP performs in this task.

Observe how its hyperparameters change after the training phase.

What can we learn from a simple sine wave?

What is the minimum number of samples you can have while keeping a "good fit"?

The range of our ground-truth is $[-1, 1]$. Crank up the noise variance to 1 and look at the output dispersion. Run your code several times and check the results. Increase the number of samples and see if you can still get a smooth sinusoidal fit with low uncertainty.

Let's incorporate some prior knowledge into our experiment. Assume you exactly know the noise level. Use the commands

```
my_gp.likelihood.variance.assign(sigma)
set_trainable(my_gp.likelihood.variance, False)
```

to tell it to the optimizer and to keep it fixed throughout the optimization procedure. Can you now get away with less datapoints?

What if you only had a rough idea of the noise variance, and only wanted to bias the optimization results towards it?

2 A slightly more difficult problem

In gt_funcs.py we have defined a number of interesting test functions for you to experiment with. Some are even defined for an arbitrary number of input dimensions.

Run hard_test.py and try to reconstruct the Schwefel function in $d = 2$ dimensions

$$f(x) = 418.98d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|})$$

You will need to pick an appropriate kernel and decide on a minimum number of samples.

3 Playing with sparse models

Run sparse_test.py to get a feeling for how easy/difficult it is to train a sparse Gaussian process. You will need to select the number of inducing points M. Both FITC and VFE are implemented in GPflow respectively as GPRFITC and SGPR. If necessary, provide the exact GP hyperparameters as initial guesses to the SGP training phase. For a good comparison between both techniques, we recommend the very good paper (2016) "Understanding probabilistic sparse Gaussian process approximations" by Bauer, Van der Wilk and Rasmussen.

4 Some additional pointers

What is automatic relevance determination (ARD) and in which scenario is it useful?

Take a closer look at the widely used squared-exponential (SE) kernel, otherwise known as the radial basis function one:

$$k(x, x') = \exp\left(\frac{\|x - x'\|^2}{2\ell^2}\right)$$

which belongs to the class of 'stationary' or 'translational-invariant' kernels. Can you imagine a 1D function that would be very hard to describe with a SE-GP? This is perhaps the number one modeling limitation of this class.

Go back to our sine-wave example and replace the input grid by `xx = np.linspace(xmin * 5, xmax * 5, 1000).reshape(-1, 1)`. Run the code again. This suggests that squared-exponential GPs do not extrapolate well. If extrapolation is what you need, you might want to read about semi-parametric Gaussian processes.

Some active researchers in the field: Carl E. Rasmussen, Richard Turner, Mark van der Wilk, Neil D. Lawrence, Robert B. Gramacy, Nando de Freitas, Andrew Gordon Wilson, Andreas Krause, Matthias Seeger, Thomas Schön.