Gaussian processes and Gauss Markov random fields

Tudor Muntianu

December 3, 2020

Abstract

This report gives a brief summary of work done this term while taking David Blei's course, Foundations of Graphical Models, as administered and supervised by Jeremy. The GitHub repo contains all of the code and notebooks that I'll reference here. Some of them are quite messy because they are still being tinkered with.

1 Introduction

1.1 Motivation

The primary motivation for this independent study was the SuperEEG project, which is based largely on Gaussian process regression. Currently the model does not rely on much of the state-of-the-art for GPs largely due to the structure of the problem and the resulting difficulty in expressing it as a single covariance matrix. So the work from this term can be seen as an attempt to take a second crack at this problem while also gaining a (more) solid foundation in graphical models.

So rather than examine any dataset or problem (even ECoG data specifically) in particular, I wanted to explore and learn more about graphical models, GMRfs, GPs, VI, and the software packages that implement them, largely because a SuperEEG 2.0 would call for using something other than NumPy for computation.

1.2 The problem

The basic idea of SuperEEG is to impute spatial data at locally unobserved but globally observed locations, and using this global structure to improve these imputed values. There are not enough papers on this subject, and many of them come from the field of geostatistics where locations are never observed globally. In ECoG at some point every location in sampled, but that is not the case on Earth. This is why GPs can be used in geostatistics, see Alvarez's review [?] which contains some examples, and why SuperEEG's flavor of GPs is novel.

There are, however, some developments which have bled over from computer vision. Deep GMRFs [?] and other inpainting models (which are really just imputing values in images like deep image priors [?] offer some interesting perspectives on this sort of spatial imputation. So what I did here was 1) implement a relatively simple but very powerful GP architecture to learn some variational inference and some other GP-specific methods such as inducing point methods, and 2) reimplement the Siden and Lindsten's deep GMRF model because their code is... neither particularly extensible nor readable. I also picked this model because there are a number of clear future directions that are not as difficult as in the world of theoretical GPs (which is astonishingly deep and convoluted).

1.3 Image classification and GPs

For the GP part of the term, I chose to implement and play with van der Wilk's convolutional Gaussian processe. van der Wilk's paper is significant because of the historical Gaussian process struggle with images. This struggle originates from two facts: 1) GPs are $\mathcal{O}(n^3)$ for training, and require $\mathcal{O}(n^2)$ memory, making dealing with thousands of images times thousands of pixels difficult, and 2) many traditional/commonly used kernels are stationary (translation invariant), making it difficult for them to capture image structure in the way CNNs and other deep architectures can. These limitations are not unique to images, but with MNIST and CIFAR being common datasets, the limitations of GPs are quickly seen when applied to these. However, variational inference and sparse approximations have done much to alleviate the first issue, and newer state-of-the-art methods like deep kernel learning [?] and deep convolutional GPs [?] have been tackling the second.

The way shallow convolutional GPs attack this problem is in much the same way. The model uses inducing points and stochastic VI to make training and inference possible, and the structure of the convolutional kernel allows the stationary radial basis function to also acquire certain non-stationary properties by operating on individual patches and taking weighted sums over them. The exact mechanism of this operation will be described later, but intuitively, it allows the kernel to express relationships between distant pixels while also encoding information about proximal pixels through the RBF. It's a useful (albeit at this point dated) extension of CNNs to GPs much in the same way Siden's deep GMRF paper does the same thing for GMRFs.

1.4 Deep GMRFs and inpainting

A brief introduction to GMRFs is simple: a GMRF is just a random vector from a multivariate Gaussian that satisfies certain conditional independence assumptions. These assumptions can be modified depending on the context, but are always present (hence Gauss Markov). See Rue and Held for more [?]. These models have been largely ignored as they relied largely on the advantage that the representing precision matrices are sparse, and so clever numerical algorithms to operate efficiently on these matrices yielded good results in the past. Obviously this advantage diminishes daily now that we have great autodiff and better/easier sampling tools.

Nevertheless GMRFs are probabalistic graphical models

2 Quick note on GPyTorch and GPFlow

When implementing the conv GPs, I looked at van der Wilk's implementations in GPFlow, a TensorFlow based GP library. I intentionally chose GPyTorch to make it harder for myself (no cheating or copy-paste), but also so that I could compare the two.

While I prefer GPFlow due to the sheer readablity of the existing code, and the large array of models already implemented, I think GPyTorch has a more powerful stack. Integration with Pyro makes MCMC and VI stupid easy (and Stan borderline irrelevant), more sophisticated lazy and approximate covariance calculations are powerful but difficult to leverage for custom models, although the developers respond to requests for help or features quickly. GPyTorch is also seeing significant development activity and funding, so I would expect it to continue to outpace GPFlow from a technical perspective, but perhaps not from a usability perspective.

Both are crucially missing basic explanations of how and where inference lives in the model, kernel, and likelihood objects. This makes building truly custom models from scratch extremely painful the first time around. I spent at least five hours just reading through the codebases in an attempt to decipher the package's architecture. Pyro is much the same way. Once you need to implement a custom loss function or change the way the ELBO is computed docs get thin and examples are not found.