Software Engineering Assignment

WASP Software Engineering Course Module 2024

Vincent Molin
molinv@chalmers.se
Chalmers University of Technology

September 3, 2024

Introduction

Very broadly, my research is concerned with method development for statistical inference problems. Starting out I worked on deep learning driven methods for Bayesian inverse problems, where we considered using generative neural networks as priors in for instance medical imaging problems. We can think of the learned prior distributions as being supported on a low-dimensional manifold in the data space. Efficient sampling of such a resulting posterior in some sense requires *smooth* coordinates, which motivated my work on stating the learning problem more explicitly as a manifold learning problem with suitable regularizations.

Going forward, and presently I'm working less on neural network related topics. Instead, currently I am mostly working on methods related to curves in spaces of probability measures, with applications to global optimization and inference. Here we numerically and theoretically make use of optimal transport and apply this to both optimization and inference problems.

More from an implementation point of view, my research relies pretty heavily on automatic differentiation and, ideally, good GPU support. This pre-empted switching language and supporting libraries 2-3 years back from Julia to JAX which, at least at that time, had the best support for higher mixed order automatic differentiation. Most important I assume to set the scene for the rest of this is that, in all my research projects so far, I am essentially the sole programmer mostly due to the theoretical nature of most work we do.

As an example, piecewise deterministic Markov processes (PDMPs) is a recurring theme throughout my work, constituting the main class of stochastic processes on which our MCMC-like methods rely. To this end I have written and often use my JAX-library for specifying and simulating these stochastic processes.

Robert's lectures

Code review practices

Code review is something that I've passively come in contact with a little bit in the open source community. Today I work a lot on prototypes and toy problems, which can obfuscate the usefulness of for instance test procedures. However, I like abstracting out code I write and then make use of in several places to openly available packages. As time goes on, I naturally try to make them better in some way than previous code I've published. Mostly I think the quality is improved by experience, but I've also explicitly tried incorporating more tests and the like. I found the different flavors of code review expectations at Google pretty interesting. Before I had a much looser grip on precisely what code review dealt with, I think I had mostly come into contact with the *Maintaining norms*-aspect.

Testing

Much of the research code I come into contact with, typically published on GitHub to accompany a paper, is little more than scripts that hopefully replicates the findings presented in the paper. In many cases this is typically not maintained for very long, if at all. In these situations, I think it is common to not write any tests at all.

Since I work a lot with stochastic algorithms, finding good test oracles can be a pain. Ideally, one would generate a sufficient amount of samples from some a priori known distribution and compute sufficient metrics from this to guarantee that the samples correspond to the correct distribution. In practice, this becomes very hard specifically for the type of distributions where one applies MCMC-methods resulting in measuring performance of the algorithm on simple distributions with large enough computation expended as to hopefully not fail the tests due to randomness.

When I wrote my PDMP library I tried this with a little bit of testing sprinkled through-out, mainly unit tests for all functions. The algorithmic flexibility of JAX comes at a cost of fairly restrictive rules for conditional expressions and requires pure functions. This lead to a problem for me when writing some fairly low level implementations of solutions to an integral equation. I sketched out the algorithm first in a more permissive language, and then later used this implementation as my test oracle. Since I had introduced a bug in the test oracle, this cemented a bug into my library for a pretty long time.

Guest lectures

Behavioral software engineering

Traditional top-down organizational structures can obfuscate some of the day-to-day interactions that impact the product of a team. One aspect of behavioral software engineering is to consider more the psychological and social relations that affect an engineer[1].

While I am not working on software in this setting, much of my research is impacted by the relations to people in my vicinity, and I found it interesting that this has emerged as a field of study. For instance, for me it can be very fruitful to discuss non-work related things during my lunch break.

Integration of ML in safety critical products

Lenberg also discussed integration of data-driven techniques in critical software, something which at the moment is difficult due to making rigorous testing and performance guarantees tricky.

I worked a bit with ML integration in medical imaging. It was very clear from the outset that this was an exploratory study, in the sense that presently it would be impossible to guarantee that the integrated deep learning method was capturing the correct distribution. Live deployment of such a system in for instance cancer imaging seems very irresponsible today, in the same way that I imagine it is for air traffic control system.

Paper discussions

We are going to discuss two articles from the CAIN conference held in 2024, one leaning more towards software engineering and the other dealing with hyperparameter optimization.

Paper I: Investigating the impact of SOLID design principles on machine learning code understanding[2]

In this paper, Cabral et al. investigate if incorporating the SOLID design principles, for object-oriented code, in machine learning code bases improves the readability of said code. This is motivated by the observation that machine learning code is written by people with diverse backgrounds, and possibly little formal software engineering education, from which they deduce that much ML code does not adhere to a set of best design principles. The key findings in the paper are that at least some of the SOLID principles leads to easier understanding for new readers of the code. As discussed in the course, ML code can already lead to a lot of technical debt by design, and hence improving the code quality in general is in some sense especially important.

I am personally a bit wary of object-oriented programming, and by design I find the line a bit blurry in a language such as Python. In JAX this is to some extent both alleviated and made difficult since all compiled functions needs to be pure. This makes code that does not separate clearly between data and functions pretty unwieldy and possibly hard to optimize. Still, I find that for function compositions that much of the design philosophy still applies. Indeed, one is in some sense also more or less forced by the language to later wrap the inner functional parts in classes.

While at first glance a rigid framework such as JAX seems like a poor match for a larger project, if not only for the popularity of and the experience of people working with competing frameworks, I think there are many benefits to using functional versions of stochastic optimizers. For instance, training and inference code is very sensitive to the pseudo-random number generator and seed used, which can lead to wildly different behavior for different initial states. The stateless nature of functional code then makes the *easiest* solution to any ML code reproducible.

Since the best, in my opinion, automatic differentiation libraries still are heavily Python-based I would be surprised if there are no readability gains from applying tried design principles such as SOLID. While the foundational FAANG-maintained libraries, such as PyTorch, JAX and TensorFlow seem well written, many of the important community-maintained supporting libraries can be of varying quality.

Paper II: A Combinatorial Approach to Hyperparameter Optimization[3]

In this paper, Khadka et al. propose using *$t-$way testing* to lower the computational complexity of hyperparameter sweeps. Given an $n-$argument function f, the idea behind $t-$way testing is that most illegal behavior is caused by one, or up to a few, of the arguments of the function. If this is the case, it is not necessary to exhaustively try every possible combination of inputs but instead sufficient to try all possible combinations of t variables. The case when $t = 2$ is also known as pairwise testing. This allows one to construct a much smaller set of test inputs than the full list of possible combinations. For hyperparameter optimization, the authors apply this as an alternative to for instance grid-based search, which up to some discretization corresponds to the full test of combinations. By only trying variations of $2 - 3$ hyperparameters, a finer search of the grid can be accomplished, granted that the hypothesis is true.

The idea that for instance only certain pairs of hyperparameters impact the empirical risk of the converged model seems pretty ambitious but interesting. In my language, this means that the map from hyperparameters to risk $\theta \mapsto R$ *factorizes*, in the sense that this map is in some sense a combination of maps from smaller spaces. When this is actually the case, and when the function is related to a probability distribution, there are classes of PDMPs which parallelize *very* efficiently.

In the experiments in the paper, the author finds that their approach is able to find fairly good configurations while still probably violating the true nature of the risk map. It would be interesting to quantify the error made by such an assumption for realistic problems, since the computational gains in practice can be pretty massive. For exact inference, this is a hard sell but there are plenty of applications where it is less important to generate samples correctly. Specifically this is the case when one uses sampling methods for optimization, a classical example of which is for instance simulated annealing[4], a topic I am currently working on.

Clearly hyperparameter optimization can be an important part of a larger ML pipeline for certain model classes. In a larger venture, I am however less convinced that the ideas presented here improves the state of the art.

References

1. Per Lenberg, Robert Feldt, and Lars Göran Wallgren. Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and Software*, 107:15–37, 2015.

2. Raphael Cabral, Marcos Kalinowski, Maria Teresa Baldassarre, Hugo Villamizar, Tatiana Escovedo, and Hélio Lopes. Investigating the impact of solid design principles on machine learning code understanding. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*, pages 7–17, 2024.

3. Krishna Khadka, Jaganmohan Chandrasekaran, Yu Lei, Raghu N. Kacker, and D. Richard Kuhn. A combinatorial approach to hyperparameter optimization. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, CAIN '24, page 140–149, New York, NY, USA, 2024. Association for Computing Machinery.

4. Tzuu-Shuh Chiang, Chii-Ruey Hwang, and Shuenn Jyi Sheu. Diffusion for global optimization in $\mathbb{R}^n$. *SIAM Journal on Control and Optimization*, 25(3):737–753, 1987.