Journal of Machine Learning Research 18 (2017) 1-6 Submitted 10/16; Revised 2/17; Published 4/17

GPflow: A Gaussian Process Library using TensorFlow

Alexander G. de G. Matthews AM554@CAM.AC.UK
Department of Engineering
University of Cambridge
Cambridge, UK

Mark van der Wilk MV310@CAM.AC.UK
Department of Engineering
University of Cambridge
Cambridge, UK

Tom Nickson TRON@ROBOTS.OX.AC.UK
Department of Engineering Science
University of Oxford
Oxford, UK

Keisuke Fujii FUJII@ME.KYOTO-U.AC.JP
Department of Mechanical Engineering and Science
Graduate School of Engineering, Kyoto University, Japan

Alexis Boukouvalas ALEXIS.BOUKOUVALAS@MANCHESTER.AC.UK
Division of Informatics
Manchester University
Oxford Road, Manchester, UK

Pablo León-Villagrá PABLO.LEON@ED.AC.UK
School of Informatics
University of Edinburgh
10 Crichton Street, Edinburgh, UK

Zoubin Ghahramani ZOUBIN@ENG.CAM.AC.UK
Department of Engineering
University of Cambridge
Cambridge, UK

James Hensman JAMES.HENSMAN@LANCASTER.AC.UK
CHICAS, Faculty of Health and Medicine
Lancaster University
Lancaster, UK

Editor: Alexandre Gramfort

Abstract

GPflow is a Gaussian process library that uses TensorFlow for its core computations and Python for its front end.[1] The distinguishing features of GPflow are that it uses variational inference as the primary approximation method, provides concise code through the use of automatic differentiation, has been engineered with a particular emphasis on software testing and is able to exploit GPU hardware.

1. GPflow and TensorFlow are available as open source software under the Apache 2.0 license.

1. Existing Gaussian Process Libraries

Gaussian processes (GPs) are versatile Bayesian nonparametric models using a prior on functions (Rasmussen and Williams, 2006). There are now many publicly available GP libraries ranging in scale from personal projects to major community tools. We do not here give detailed consideration to all libraries, regrettably neglecting, for instance, those that only support Gaussian likelihoods (Ambikasaran et al., 2014) or do not make provision for scalability (Pedregosa et al., 2011). The influential GPML toolbox (Rasmussen and Nickisch, 2010) uses *MATLAB*. It has been widely forked. A key reference for our particular contribution is the GPy library (GPy, since 2012), which is written primarily using Python and Numeric Python (NumPy). GPy has an intuitive object-oriented interface. Another relevant GP library is GPstuff (Vanhatalo et al., 2013) which is also a *MATLAB* library.

2. Objectives for a new library

GPflow is motivated by a set of goals. The software is designed to be fast, particularly at scale. Where approximation inference is necessary we want it to be accurate. We aim to support a variety of kernel and likelihood functions. Another goal is that the implementations are well tested. The software should be made easy to use by an intuitive user interface. Finally it should be easy to extend the software. We argue that there is a way to better meet these simultaneous objectives than existing packages and that it is realised in GPflow.

3. Key features for meeting the objectives

To best meet the key goals of our library, we were led to a project that had all of the following distinguishing features:

(i) The use of variational inference as the primary approximation method to meet the twin challenges of non-conjugacy and scale (Matthews, 2016).

(ii) Relatively concise code which uses automatic differentiation to remove the burden of gradient implementations.

(iii) The ability to leverage GPU hardware for fast computation.

(iv) A clean object-oriented Python front end.

(v) A dedication to testing and open source software principles.

Table 1 gives a summary of which GP libraries possess the distinguishing features we have highlighted. The GPflow interface and Python architecture are heavily influenced by GPy. An important difference between GPflow and GPy is that GPflow uses TensorFlow for its core computations rather than numeric Python. This difference significantly affects the general requirements of the architecture. The GPU functionality in GPy is currently limited to CUDA code for the GPLVM (Dai et al., 2014). By contrast the GPflow implementation is targeted at a broad variety of GPU capability.

Library	Sparse variational inference	Automatic differentiation	GPU demonstrated	OO Python front end	Test coverage
GPML	✓	✗	✗	✗	N\R
GPstuff	Partial	✗	✗	✗	N\R
GPy	✓	✗	GPLVM	✓	49%
GPflow	✓	✓	SVI	✓	99%

Table 1: A summary of the features possessed by existing GP libraries at the time of writing. OO stands for object-oriented. In the GPU column GPLVM denotes GP latent variable model and SVI is Stochastic variational inference. N\R denotes not reported.

Having established a desirable set of key design features, the question arises as how best to engineer a GP library to achieve them. A central insight here is that many of the features we highlight are well supported in neural network libraries. Of the available libraries we use TensorFlow (Abadi et al.), as discussed in the next section.

4. Contributing GP Requirements to TensorFlow

In TensorFlow (Abadi et al.) a computation is described as a directed graph where the nodes represent *Operations* (*Ops* for short) and the edges represent *Tensors*. As a directed edge, a Tensor represents the flow of some data between computations. Ops are recognisable mathematical functions such as addition, multiplication etc. *Kernels* (in an unfortunate collision in terminology with the GP literature) are implementations of a given Op on a specific device such as a CPU or GPU. Like almost all modern neural network software, TensorFlow comes with the ability to automatically compute the gradient of an objective function with respect to some parameters. The TensorFlow open source implementation comes with a wealth of GPU kernels for the majority of Ops.

Although we gained significantly from using TensorFlow within GPflow, there were some capabilities that were not yet present in the software which were required for our purposes. We therefore added this functionality to TensorFlow. GP software needs the ability to solve systems of linear equations using common linear algebra algorithms. The differentiation of computational graphs that used the Cholesky decomposition required a new Op, which we contributed. The main part of the code was a C++ implementation of the blocked Cholesky algorithm proposed by Murray (2016). We also contributed code that enabled GPU solving of matrix triangular systems, which in some cases is the bottleneck for approximate inference.

5. Details of GPflow

GPflow supports exact inference where possible, as well as a variety of approximation methods. One source of intractability is non-Gaussian likelihoods, so it is helpful to categorize the available likelihood functionality on this basis. Another major source of intractability is the adverse scaling of GP methods with the number of data points. To this end we support 'variationally sparse' methods which ensure that the approximation is scalable and close

	Gaussian likelihood	**Non-Gaussian likelihood (variational)**	**Non-Gaussian likelihood (MCMC)**
Full covariance	GPR	VGP	GPMC
Variational sparsity	SGPR	SVGP	SGPMC

Table 2: A table showing the inference classes in GPflow. Relevant references are VGP (Opper and Archambeau, 2009), SGPR (Titsias, 2009), SVGP (Hensman et al., 2013, 2015b) and SGPMC (Hensman et al., 2015a).

in a Kullback-Leibler sense to the posterior (Matthews et al., 2016). Whether or not a given inference method uses variational sparsity is another useful way to categorize it. The inference options, which are implemented as classes in GPflow, are summarized in Table 2. Note that all the MCMC based inference methods support a Bayesian prior on the hyper-parameters, whereas all other methods assume a point estimate. Our main MCMC method is Hamiltonian Monte Carlo (Neal, 2010).

We now discuss some architectural considerations. The whole Python component of GPflow is intrinsically objected-oriented. The code for the various inference methods in Table 2 is structured in a class hierarchy, where common code is pulled out into a shared base class. The object-oriented paradigm, whilst very natural for the Python layer of the code, has a different emphasis to that of a computational graph which is arguably closer to a functional concept. The largely functional computational graph, and a object-oriented interface need to live cleanly together in GPflow. This is achieved through the Param class that allows parameters to be both properties of an object that can be manipulated as such and Variables in a TensorFlow graph that can be optimized.

A number of steps have been taken to ensure project quality and usability. All GPflow source code is openly available on GitHub at `http://github.com/GPflow/GPflow`. The web page uses continuous integration to run an automated test suite. The test code coverage for GPflow is higher than similar packages where the code coverage statistics are published, achieving a level of 99% (Table 1). A user manual can be found at `http://gpflow.readthedocs.io`.

6. Timed Experiments

As a scenario, we studied training a multiclass GP classifier on MNIST using stochastic variational inference (Hensman et al., 2015b,a).[2] We compared against GPy. None of the other libraries discussed support this algorithm. Functionally, the algorithms are nearly identical in GPflow and GPy. We did a series of trials measuring the time each package took to perform 50 iterations of the algorithm. The trials included a set of CPU experiments, where we varied the number of threads available to the two packages. For GPflow, we also measured the effect of adding a GPU on top of the maximal number of CPU threads considered. GPy does not presently have a GPU implementation of this algorithm. We

2. Code for these timing experiments can be found at `http://github.com/gpflow/GPflowBenchmarks`

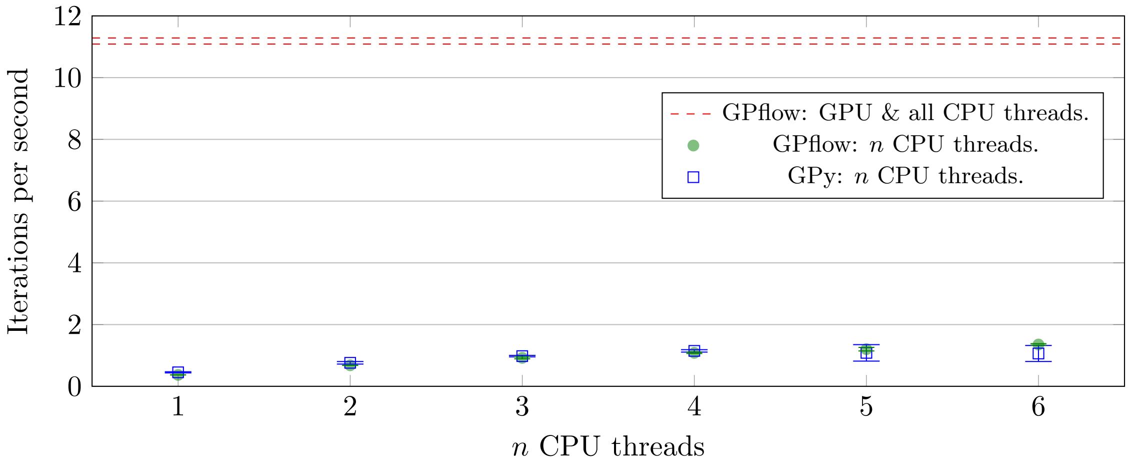

Figure 1: A comparison of iterations of stochastic variational inference per second on the MNIST data set for GPflow and GPy. Error bars shown represent one standard deviation computed from five repeats of the experiment.

used a Linux workstation Intel Core I7-4930K CPU clocked at 3.40GHz an NVIDIA GM200 Geforce GTX Titan X GPU.

The results of the timing experiments are shown in Figure 1. For the CPU experiments, the speeds for GPflow and GPy are similar. It can be seen that the increase in speed from adding a GPU is considerable. These gains could make a significant difference to the work flow of a researcher on this topic. Based on the measurements we have made, training using GPflow with 6 CPU threads would take approximately 41 hours or just under 2 days. Adding a GPU would currently reduce the training time to about 5 hours.

Acknowledgments

We acknowledge contributions from Valentine Svensson, Dan Marthaler, David J. Harris, Rasmus Munk Larsen and Eugene Brevdo. We acknowledge EPSRC grants EP/I036575/1, EP/N014162/1 and EP/N510129/1. James Hensman was supported by an MRC fellowship.

References

Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems.

Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W. Hogg, and Michael O'Neil. Fast direct methods for Gaussian processes and the analysis of NASA Kepler mission data. *arXiv preprint 1403.6015*, 2014.

Zhenwen Dai, Andreas Damianou, James Hensman, and Neil Lawrence. Gaussian process models with parallelization and GPU acceleration. *arXiv preprint 1410.4984*, 2014.

GPy. GPy: A Gaussian process framework in Python. `http://github.com/SheffieldML/GPy`, since 2012.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. In *UAI*, July 2013.

James Hensman, Alexander G. de G. Matthews, Maurizio Filippone, and Zoubin Ghahramani. MCMC for variationally sparse Gaussian processes. In *NIPS*, December 2015a.

James Hensman, Alexander G. de G. Matthews, and Zoubin Ghahramani. Scalable variational Gaussian process classification. In *AISTATS*, May 2015b.

Alexander G. de G. Matthews. *Scalable Gaussian Process Inference using Variational Methods.* PhD thesis, University of Cambridge, 2016.

Alexander G. de G. Matthews, James Hensman, Richard E. Turner, and Zoubin Ghahramani. On Sparse variational methods and the Kullback-Leibler divergence between stochastic processes. In *AISTATS*, May 2016.

Iain Murray. Differentiation of the Cholesky decomposition. *arXiv preprint 1602.07527*, February 2016.

Radford M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162, 2010.

Manfred Opper and Cédric Archambeau. The variational Gaussian approximation revisited. *Neural Computation*, 21(3):786–792, 2009.

Fabian Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Carl E. Rasmussen and Hannes Nickisch. Gaussian Processes for machine learning (GPML) toolbox. *Journal of Machine Learning Research*, 11:3011–3015, 11 2010.

Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, 2006.

Michalis K. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *AISTATS*, April 2009.

Jarno Vanhatalo, Jaakko Riihimäki, Jouni Hartikainen, Pasi Jylänki, Ville Tolvanen, and Aki Vehtari. GPstuff: Bayesian modeling with Gaussian processes. *Journal of Machine Learning Research*, 14(1):1175–1179, 2013.