Preliminary Evaluation of Hyperopt Algorithms on HPOLib

James Bergstra JAMES.BERGSTRA@UWATERLOO.CA
Brent Komer BRENT.KOMER@UWATERLOO.CA
Chris Eliasmith CELIASMITH@UWATERLOO.CA
Centre for Theoretical Neuroscience
University of Waterloo
David Warde-Farley WARDEFAR@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationelle
Université de Montréal

Abstract

Model selection, also known as hyperparameter tuning, can be viewed as a blackbox optimization problem. Recently the HPOlib benchmarking suite was advanced to facilitate algorithm comparison between hyperparameter optimization algorithms. We compare seven optimization algorithms implemented in the Hyperopt optimization package, including a new annealing-type algorithm and a new family of Gaussian Process-based SMBO methods, on four screening problems from HPOLib. We find that methods based on Gaussian Processes (GPs) are the most call-efficient. Vanilla GP-based methods using stationary RBF kernels and maximum likelihood kernel parameter estimation provide a near-perfect ability to optimize the benchmarks. Despite being slower than more heuristic baselines, a Theano-based GP-SMBO implementation requires at most a few seconds to produce a candidate evaluation point. We compare this vanilla approach to Hybrid Monte-Carlo integration of the kernel lengthscales and fail to find compelling advantages of this more expensive procedure.

1. Introduction

Model selection, also known as hyperparameter tuning, can be viewed as a black box optimization problem. The search domain is all possible joint hyperparameter assignments for a machine learning system (inputs to the black box); the output of the black box is a reflection of model fit, such as validation set [in]accuracy. This view has been used to good effect in recent work (Bergstra et al., 2011; Bergstra and Bengio, 2012; Snoek et al., 2012; Thornton et al., 2012), which demonstrates (across a variety of optimization techniques and machine learning systems) that black box optimization algorithms are not only useful, but often better than the world's best machine learning experts. Recently Eggensperger et al. (2013) proposed a collection of benchmarks (HPOlib) to focus research in this area. Hyperopt (Bergstra et al., 2013) is a Python software package that provides a language for describing structured search domains (especially hyperparameter search domains), and provides implementations of algorithms for searching those domains. This paper describes several algorithms written for Hyperopt, and evaluates them on the fast-running screening benchmarks in HPOlib.

2. Hyperopt Optimization Algorithms

We have written seven optimization algorithms for Hyperopt: `rand`, `anneal`, `tpe`, `tree`, `ucb`, `ei_ml2`, `ei_hmc`.

Random search (`rand`) is included in Hyperopt. Search spaces in Hyperopt are described in terms of directed graphical models, and random search is simply ancestral sampling of the search space itself.

The **annealing** algorithm (`anneal`) is a new addition to Hyperopt. At first it draws points identically to random search, but over time the distributions are heuristically concentrated around the best-performing points. Different kinds of original distributions are concentrated in different ways. Uniform-based distributions are concentrated by moving upper and lower bounds together as $1/t$. Normal-based distributions are tightened by reducing the standard deviation according to $1/t$. Categorical distributions are tightened by interpolating between the original distribution and a zero-entropy distribution that puts all mass on the best-performing option. There are a number of constants governing the convergence schedule of `anneal`, and it is susceptible to a number of failure modes. Nevertheless it can be a very quick algorithm for optimizing a gradient-free function over a hyperopt-style space, and it is typically better than random search in that role. The `anneal` algorithm can be useful both for optimizing an original black box function, and optimizing an acquisition function (e.g. Expected Improvement) in the course of Surrogate (aka Sequential) Model-based Bayesian Optimization (SMBO).

The **TPE** algorithm (`tpe`, Tree of Parzen Estimators) is included in Hyperopt. The algorithm is based on the idea that under certain modeling assumptions, the maximization of Expected Improvement (EI) can be related to the maximization of ratio $\frac{P(X|y<\tau)}{P(X|y\geq\tau)}$ (see Bergstra et al. (2011) for details). In a search space with N dimensions (and no conditional parameters), the TPE algorithm models these densities over the search space with two product-of-marginal [Parzen] estimators. They both have the same form, the model of points from a data set $\mathcal{D}$ with $y < \tau$ is as follows:

$$Q(X|y<\tau) = \prod_i \sum_{(x,y)\in\mathcal{D}|y<\tau} \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{-\frac{(x_{ij}-X_i)^2}{2\sigma_{ij}^2}} \tag{1}$$

TPE inherits SMBO and EI's theoretical ability to escape local optima, and it is almost as quick as `anneal` at suggesting new points in each SMBO iteration.

The `tree` algorithm is not yet in Hyperopt. Inspired by SMAC (Hutter et al., 2011), it is a SMBO approach with a regression tree mixture surrogate. Trees naturally handle conditional parameters (the screening benchmarks below have none, but real hyperparameter optimization settings have many), are quick to train, and can be updated incrementally from online data. We use a Bagging-style ensemble method. Candidates are drawn by optimizing EI in the surrogate using the `anneal` algorithm.

The `ucb`, `ei_ml2` and `ei_hmc` algorithms are not yet in Hyperopt. These are based on "vanilla" Gaussian Process (GP) SMBO using stationary RBF kernels (in contrast with Matérn kernels advocated in Snoek et al. (2012), and the input-warping technique advanced in Snoek et al. (2014)). The `ucb` and `ei_ml2` algorithms use a single setting of kernel parameters that maximizes marginal likelihood. The `ucb` algorithm generates candidates that maximize the Upper Confidence Bound (UCB) criterion. The `ei_ml2` algorithm generates

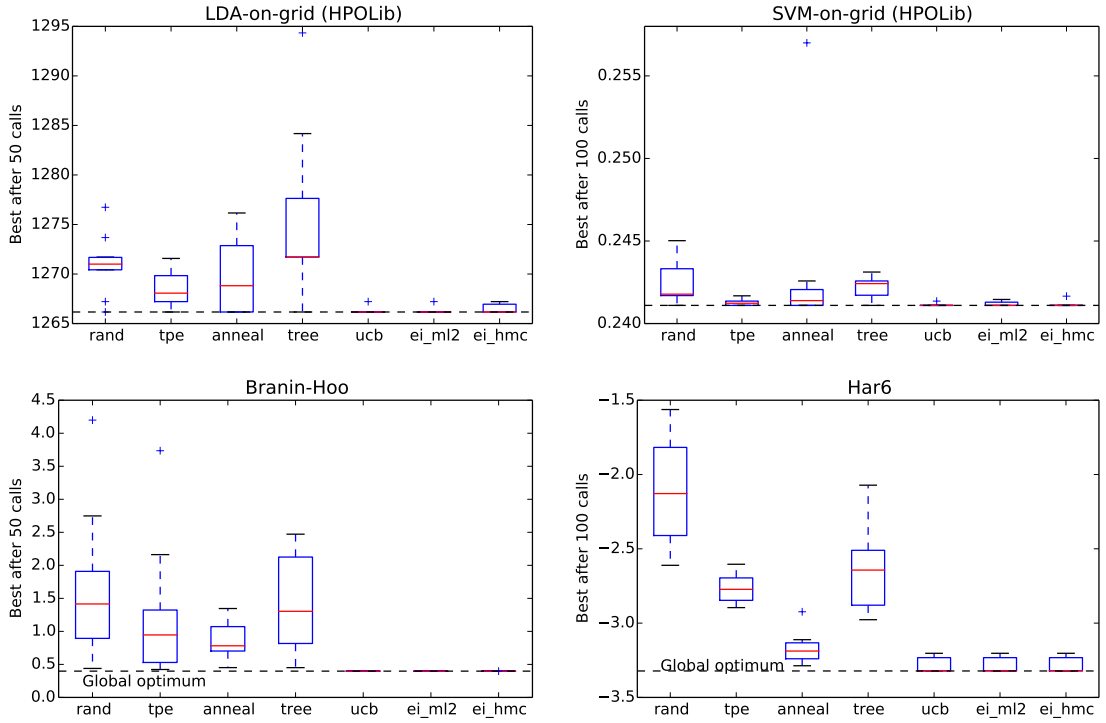

Figure 1: Each boxplot shows the distribution of minimum values found by each algorithm on 10 randomized runs. (Lower is better)

candidates that maximize the EI criterion. The (`ei_hmc`) algorithm averages over multiple kernel parameter settings generated by Hybrid Monte-Carlo sampling (HMC, Neal (1993), as implemented in Desjardins et al. (2010)). Optimization of the surrogate was handled in all cases by gradient descent (L-BFGS-B Zhu et al. (1997) via SciPy (Jones et al., 2001–)) with random restarts. Our implementation was in Theano (Bergstra et al., 2010), which helped with gradient-based optimization and sampling algorithms and provided good evaluation speed.

3. Experiments

We compare Hyperopt's algorithms in terms of overall search efficiency (how much computation they can save), and the computational overhead they impose on the search process (how much computation they cost). The four fast-running benchmarks we used from HPOlib were: LDA-on-grid (LDA), SVM-on-grid (SVM), Branin-Hoo (Branin) and Hartmann-6 (Har6). LDA and SVM are pre-computed 3-D grid searches from hyperparameter tuning experiments (grids of $6 \times 6 \times 8 = 288$ and $25 \times 14 \times 4 = 1400$ respectively). Branin is a simple 2-D surface over a 15-unit square with 3 global minima (.398) on a broad valley floor between two steep cliffs. Har6 is a challenging 6-D function over the unit hypercube with at least two local optima (our best optimizers converged reliably to either -3.2 or -3.32).

Table 1: Best values (mean and standard error) based on 10 random repetitions. The top rows correspond to algorithms described in this work and computed by the authors. SMAC, Spearmint, and TPE (*) are reproduced for from Eggensperger et al. (2013). Bold scores are within a 95% confidence interval of the best average score.

Algorithm	LDA	SVM	Branin	Hartmann-6
ei_hmc	1266.7 ± 0.2	$\mathbf{0.241 \pm 0.000}$	$\mathbf{0.398 \pm 0.000}$	$\mathbf{-3.2665 \pm 0.0196}$
ei_ml2	$\mathbf{1266.3 \pm 0.1}$	$\mathbf{0.241 \pm 0.000}$	$\mathbf{0.398 \pm 0.000}$	$\mathbf{-3.2667 \pm 0.0196}$
ucb	$\mathbf{1266.4 \pm 0.1}$	$\mathbf{0.241 \pm 0.000}$	$\mathbf{0.398 \pm 0.000}$	$\mathbf{-3.2667 \pm 0.0196}$
tree	1275.8 ± 2.6	0.242 ± 0.000	1.417 ± 0.241	-2.6508 ± 0.0858
anneal	1269.6 ± 1.2	0.243 ± 0.002	0.861 ± 0.093	-3.1725 ± 0.0333
tpe	1268.3 ± 0.6	$\mathbf{0.241 \pm 0.000}$	1.244 ± 0.326	-2.7654 ± 0.0335
rand	1271.0 ± 0.9	0.242 ± 0.000	1.613 ± 0.362	-2.1121 ± 0.1167
SMAC	1269.6 ± 1.0	$\mathbf{0.241 \pm 0.000}$	0.655 ± 0.090	-2.9770 ± 0.0367
Spearmint	1272.6 ± 3.4	0.246 ± 0.003	$\mathbf{0.398 \pm 0.000}$	$\mathbf{-3.1330 \pm 0.1367}$
TPE (*)	1271.5 ± 1.2	0.242 ± 0.000	0.526 ± 0.043	-2.8230 ± 0.0600

The most important quality in a hyperparameter optimization algorithm is that it can find interesting configurations after a very small number of [expensive] function evaluations. Figure 1 shows typical accuracies observed for each combination of algorithm and benchark function after the number of function evaluations recommended by HPOlib. Figure 2 shows a more temporal picture of how each algorithm tends to approach the best results shown in Figure 1. Table 1 lists the range of minima obtained for each algorithm and problem, and reproduces some results from Eggensperger et al. (2013) for comparison.

It is also important that the optimization itself not require too much time. Firstly, because time spent on the optimization algorithm is time not spent evaluating hyperparameter configurations. Secondly, because slow algorithms are tedious to use. Slowness presents an adoption barrier to practitioners who could benefit from automatic algorithm configuration. Figure 3 shows how many seconds the algorithms in question took to produce recommendations.

4. Discussion

Several lessons learned from the screening benchmarks seem likely to transfer to more computationally expensive real-world configuration problems.

Our "vanilla" Gaussian Process SMBO was very effective: all three variants were perfect or near-perfect on all problems and they also converged fastest. We explore harder problems in future work, where there is room for more sophisticated modeling to make a difference. In terms of speed, the optimization runs were so short that the GP's $O(n^3)$ training time was negligible. Instead, optimization of the surrogate turned out to dominate time spent within the SMBO iterations. Precise optimization of the acquisition function was critical to success on Branin and Har6. Neither UCB nor EI outperformed the other.

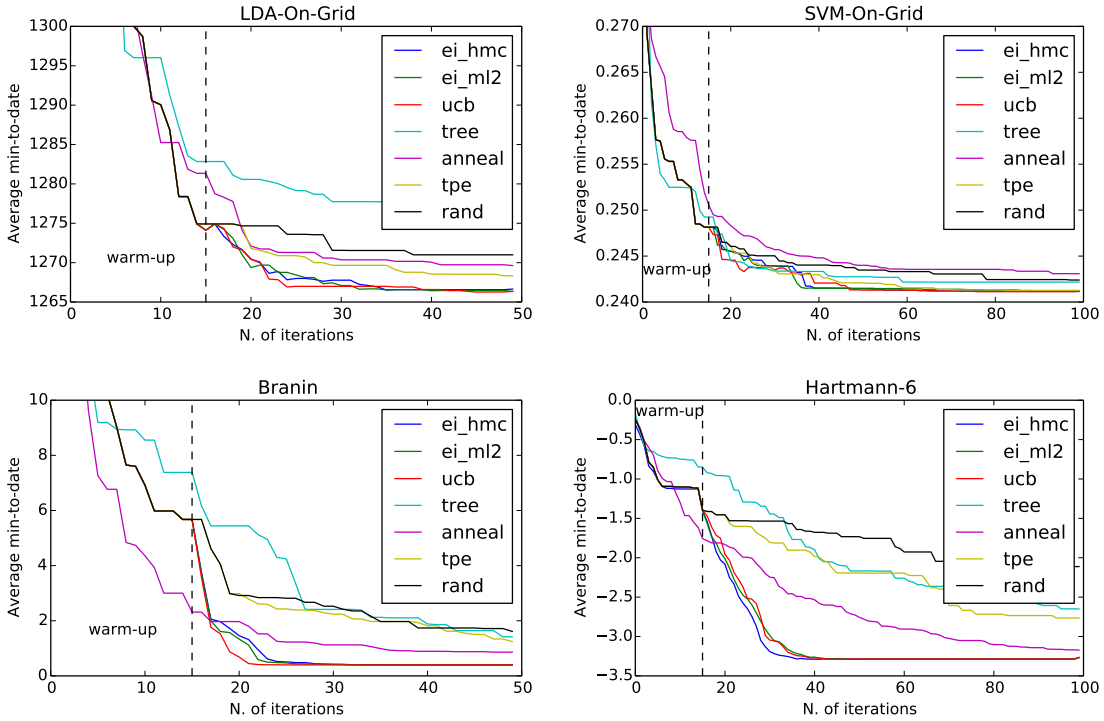

Figure 2: Best-value-to-date at each SMBO iteration, averaging over 10 optimization runs. After an initial "warm-up" period, the GP-based algorithms (ei_ml2, ei_hmc, ucb) make the most rapid convergence to the global optima. (Best viewed in colour.)

The `ei_hmc` algorithm was no better than `ei_ml2` and `ucb` despite being many times slower (Fig. 3). Our hope was that model averaging would help escape local optima in Har6, but this was never observed. Perhaps we were simply not patient enough: in order to maintain a reasonable optimization speed we kept the number of model components very small (8), and tried to compensate by at least decorrelating them (many HMC steps between samples). There appears to be a tradeoff between model accuracy (sampling e.g. length scales) and optimization accuracy, in that it seems doing both *enough* is prohibitively expensive even in these very small problem domains. Further work is required to understand how and when to benefit from HMC.

Although even SMAC is not as accurate as the GP-SMBO in these benchmarks, our own `tree` results are much worse. Our implementation is both slow and ineffective, suggesting that more engineering is necessary. Sample efficiency and variance estimation are paramount in SMBO, perhaps non-Bagging ensemble methods might be more effective.

Har6 is challenging for SMBO approaches because it appears that a good surrogate model of the -3.2-valued local optimum completely hides the -3.32-valued optimum. Our three GP variants got stuck at -3.2 on three of ten runs (although `ucb` and `ei_ml2` got stuck on different runs), and even hundreds of additional iterations made no progress. RBF-GP surrogates are susceptible to inefficient modeling of many response surfaces. Further work

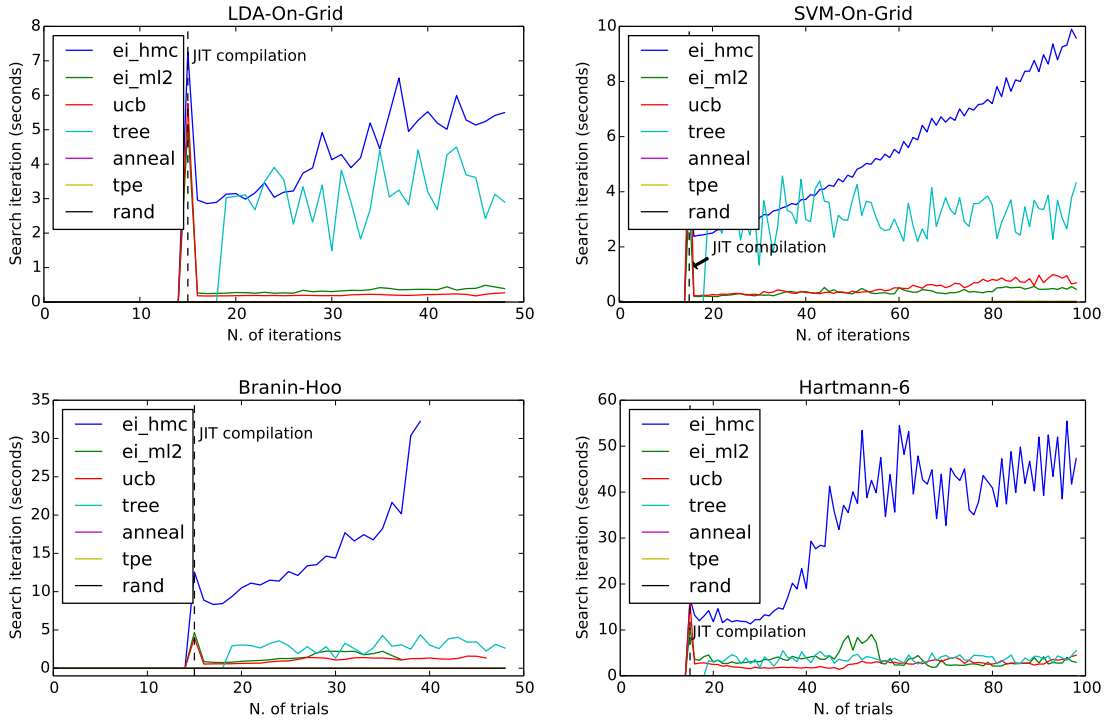

Figure 3: Time spent within each optimization algorithm. Several algorithms' time is negligible on the scale of GP-SMBO approaches. (Best viewed in colour.)

is necessary to determine whether the Har6 function is representative of the challenges of hyperparameter optimization more broadly.

The `anneal` algorithm was better than `rand` as a black box optimizer, and fast enough to provide gradient-free surrogate optimization within SMBO iterations.

5. Conclusion

HPOLib provides an opportunity for machine learning and optimization experts to (a) agree on what the problem of hyperparameter optimization *is*, and (b) measure research progress toward solving that problem. In response to HPOLib's creation, Hyperopt has been extended with a range of new search algorithms, including the fast-to-run `anneal` and three slow-but-efficient GP-SMBO variants. These algorithms perform very well on the four fast-running screening problems in HPOLib, and we look forward to evaluating them on a broader range of optimization problems in future work.

Acknowledgements
This research was supported by the NSERC Banting Fellowship program, the NSERC Engage program and by D-Wave Systems.

References

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NIPS*24*, pages 2546–2554, 2011.

J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms. In *SciPy'13*, 2013.

G. Desjardins, J. Bergstra, and LISA Lab. Deep learning tutorial: Hybrid Monte-Carlo. http://www.deeplearning.net/tutorial/hmc.html, 2010.

K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, 10 December 2013.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION-5*, 2011. Extended version as UBC Tech report TR-2010-10.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/.

R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto, 1993.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

J. Snoek, K. Swersky, R. Zemel, and R. Adams. Input warping for bayesian optimization of non-stationary functions. In *ICML*, 2014.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Automated selection and hyper-parameter optimization of classification algorithms. *CoRR*, abs/1208.3719, 2012.

C. Zhu, R. H. Byrd, and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.