View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Harvard University - DASH

Practical Bayesian Optimization of Machine Learning Algorithms

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters.

Citation	Snoek, Jasper, Hugo Larochelle, and Ryan Prescott Adams. 2012. Practical Bayesian optimization of machine learning algorithms. Advances in Neural Information Processing Systems 25: 2960-2968
Accessed	February 19, 2015 1:12:18 PM EST
Citable Link	http://nrs.harvard.edu/urn-3:HUL.InstRepos:11708816
Terms of Use	This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Open Access Policy Articles, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#OAP

(Article begins on next page)

Practical Bayesian Optimization of Machine Learning Algorithms

Jasper Snoek
Dept of Computer Science
University of Toronto
jasper@cs.toronto.edu

Hugo Larochelle
Dept of Computer Science
University of Sherbrooke
hugo.larochelle@usherbrooke.edu

Ryan P. Adams
School of Engineering and Applied Sciences
Harvard University
rpa@seas.harvard.edu

Abstract

The use of machine learning algorithms frequently involves careful tuning of learning parameters and model hyperparameters. Unfortunately, this tuning is often a "black art" requiring expert experience, rules of thumb, or sometimes brute-force search. There is therefore great appeal for automatic approaches that can optimize the performance of any given learning algorithm to the problem at hand. In this work, we consider this problem through the framework of Bayesian optimization, in which a learning algorithm's generalization performance is modeled as a sample from a Gaussian process (GP). We show that certain choices for the nature of the GP, such as the type of kernel and the treatment of its hyperparameters, can play a crucial role in obtaining a good optimizer that can achieve expert-level performance. We describe new algorithms that take into account the variable cost (duration) of learning algorithm experiments and that can leverage the presence of multiple cores for parallel experimentation. We show that these proposed algorithms improve on previous automatic procedures and can reach or *surpass* human expert-level optimization for many algorithms including latent Dirichlet allocation, structured SVMs and convolutional neural networks.

1 Introduction

Machine learning algorithms are rarely parameter-free: parameters controlling the rate of learning or the capacity of the underlying model must often be specified. These parameters are often considered nuisances, making it appealing to develop machine learning algorithms with fewer of them. Another, more flexible take on this issue is to view the optimization of such parameters as a procedure to be automated. Specifically, we could view such tuning as the optimization of an unknown black-box function and invoke algorithms developed for such problems. A good choice is Bayesian optimization [1], which has been shown to outperform other state of the art global optimization algorithms on a number of challenging optimization benchmark functions [2]. For continuous functions, Bayesian optimization typically works by assuming the unknown function was sampled from a Gaussian process and maintains a posterior distribution for this function as observations are made or, in our case, as the results of running learning algorithm experiments with different hyperparameters are observed. To pick the hyperparameters of the next experiment, one can optimize the expected improvement (EI) [1] over the current best result or the Gaussian process upper confidence bound (UCB)[3]. EI and UCB have been shown to be efficient in the number of function evaluations required to find the global optimum of many multimodal black-box functions [4, 3].

Machine learning algorithms, however, have certain characteristics that distinguish them from other black-box optimization problems. First, each function evaluation can require a variable amount of time: training a small neural network with 10 hidden units will take less time than a bigger network with 1000 hidden units. Even without considering duration, the advent of cloud computing makes it possible to quantify economically the cost of requiring large-memory machines for learning, changing the actual cost in dollars of an experiment with a different number of hidden units. Second, machine learning experiments are often run in parallel, on multiple cores or machines. In both situations, the standard sequential approach of GP optimization can be suboptimal.

In this work, we identify good practices for Bayesian optimization of machine learning algorithms. We argue that a fully Bayesian treatment of the underlying GP kernel is preferred to the approach based on optimization of the GP hyperparameters, as previously proposed [5]. Our second contribution is the description of new algorithms for taking into account the variable and unknown cost of experiments or the availability of multiple cores to run experiments in parallel.

Gaussian processes have proven to be useful surrogate models for computer experiments and good practices have been established in this context for sensitivity analysis, calibration and prediction [6]. While these strategies are not considered in the context of optimization, they can be useful to researchers in machine learning who wish to understand better the sensitivity of their models to various hyperparameters. Hutter et al. [7] have developed sequential model-based optimization strategies for the configuration of satisfiability and mixed integer programming solvers using random forests. The machine learning algorithms we consider, however, warrant a fully Bayesian treatment as their expensive nature necessitates minimizing in the number of evaluations. Bayesian optimization strategies have also been used to tune the parameters of Markov chain Monte Carlo algorithms [8]. Recently, Bergstra et al. [5] have explored various strategies for optimizing the hyperparameters of machine learning algorithms. They demonstrated that grid search strategies are inferior to random search [9], and suggested the use of Gaussian process Bayesian optimization, optimizing the hyperparameters of a squared-exponential covariance, and proposed the Tree Parzen Algorithm.

2 Bayesian Optimization with Gaussian Process Priors

As in other kinds of optimization, in Bayesian optimization we are interested in finding the minimum of a function $f(\mathbf{x})$ on some bounded set $\mathcal{X}$, which we will take to be a subset of $\mathbb{R}^D$. What makes Bayesian optimization different from other procedures is that it constructs a probabilistic model for $f(\mathbf{x})$ and then exploits this model to make decisions about where in $\mathcal{X}$ to next evaluate the function, while integrating out uncertainty. The essential philosophy is to use *all* of the information available from previous evaluations of $f(\mathbf{x})$ and not simply rely on local gradient and Hessian approximations. This results in a procedure that can find the minimum of difficult non-convex functions with relatively few evaluations, at the cost of performing more computation to determine the next point to try. When evaluations of $f(\mathbf{x})$ are expensive to perform — as is the case when it requires training a machine learning algorithm — then it is easy to justify some extra computation to make better decisions. For an overview of the Bayesian optimization formalism, see, e.g., Brochu et al. [10]. In this section we briefly review the general Bayesian optimization approach, before discussing our novel contributions in Section 3.

There are two major choices that must be made when performing Bayesian optimization. First, one must select a prior over functions that will express assumptions about the function being optimized. For this we choose the Gaussian process prior, due to its flexibility and tractability. Second, we must choose an *acquisition function*, which is used to construct a utility function from the model posterior, allowing us to determine the next point to evaluate.

2.1 Gaussian Processes

The Gaussian process (GP) is a convenient and powerful prior distribution on functions, which we will take here to be of the form $f : \mathcal{X} \to \mathbb{R}$. The GP is defined by the property that any finite set of N points $\{\mathbf{x}_n \in \mathcal{X}\}_{n=1}^N$ induces a multivariate Gaussian distribution on $\mathbb{R}^N$. The nth of these points is taken to be the function value $f(\mathbf{x}_n)$, and the elegant marginalization properties of the Gaussian distribution allow us to compute marginals and conditionals in closed form. The support and properties of the resulting distribution on functions are determined by a mean function $m : \mathcal{X} \to \mathbb{R}$ and a positive definite covariance function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. We will discuss the impact of covariance functions in Section 3.1. For an overview of Gaussian processes, see Rasmussen and Williams [11].

2.2 Acquisition Functions for Bayesian Optimization

We assume that the function $f(\mathbf{x})$ is drawn from a Gaussian process prior and that our observations are of the form $\{\mathbf{x}_n, y_n\}_{n=1}^N$, where $y_n \sim \mathcal{N}(f(\mathbf{x}_n), \nu)$ and ν is the variance of noise introduced into the function observations. This prior and these data induce a posterior over functions; the acquisition function, which we denote by $a : \mathcal{X} \to \mathbb{R}^+$, determines what point in $\mathcal{X}$ should be evaluated next via a proxy optimization $\mathbf{x}_{\mathsf{next}} = \operatorname{argmax}_{\mathbf{x}} a(\mathbf{x})$, where several different functions have been proposed. In general, these acquisition functions depend on the previous observations, as well as the GP hyperparameters; we denote this dependence as $a(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta)$. There are several popular choices of acquisition function. Under the Gaussian process prior, these functions depend on the model solely through its predictive mean function $\mu(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta)$ and predictive variance function $\sigma^2(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta)$. In the proceeding, we will denote the best current value as $\mathbf{x}_{\mathsf{best}} = \operatorname{argmin}_{\mathbf{x}_n} f(\mathbf{x}_n)$, $\Phi(\cdot)$ will denote the cumulative distribution function of the standard normal, and $\phi(\cdot)$ will denote the standard normal density function.

Probability of Improvement One intuitive strategy is to maximize the probability of improving over the best current value [12]. Under the GP this can be computed analytically as

$$a_{\mathsf{PI}}(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta) = \Phi(\gamma(\mathbf{x})), \qquad \gamma(\mathbf{x}) = \frac{f(\mathbf{x}_{\mathsf{best}}) - \mu(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta)}{\sigma(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta)}. \tag{1}$$

Expected Improvement Alternatively, one could choose to maximize the expected improvement (EI) over the current best. This also has closed form under the Gaussian process:

$$a_{\mathsf{EI}}(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta) = \sigma(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta)\,(\gamma(\mathbf{x})\,\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x})\,;\, 0, 1)) \tag{2}$$

GP Upper Confidence Bound A more recent development is the idea of exploiting lower confidence bounds (upper, when considering maximization) to construct acquisition functions that minimize regret over the course of their optimization [3]. These acquisition functions have the form

$$a_{\mathsf{LCB}}(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta) = \mu(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta) - \kappa\,\sigma(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta), \tag{3}$$

with a tunable κ to balance exploitation against exploration.

In this work we will focus on the EI criterion, as it has been shown to be better-behaved than probability of improvement, but unlike the method of GP upper confidence bounds (GP-UCB), it does not require its own tuning parameter. Although the EI algorithm performs well in minimization problems, we wish to note that the regret formalization may be more appropriate in some settings. We perform a direct comparison between our EI-based approach and GP-UCB in Section 4.1.

3 Practical Considerations for Bayesian Optimization of Hyperparameters

Although an elegant framework for optimizing expensive functions, there are several limitations that have prevented it from becoming a widely-used technique for optimizing hyperparameters in machine learning problems. First, it is unclear for practical problems what an appropriate choice is for the covariance function and its associated hyperparameters. Second, as the function evaluation itself may involve a time-consuming optimization procedure, problems may vary significantly in duration and this should be taken into account. Third, optimization algorithms should take advantage of multi-core parallelism in order to map well onto modern computational environments. In this section, we propose solutions to each of these issues.

3.1 Covariance Functions and Treatment of Covariance Hyperparameters

The power of the Gaussian process to express a rich distribution on functions rests solely on the shoulders of the covariance function. While non-degenerate covariance functions correspond to infinite bases, they nevertheless can correspond to strong assumptions regarding likely functions. In particular, the automatic relevance determination (ARD) *squared exponential* kernel

$$K_{\mathsf{SE}}(\mathbf{x}, \mathbf{x}') = \theta_0 \exp\left\{-\frac{1}{2} r^2(\mathbf{x}, \mathbf{x}')\right\} \qquad r^2(\mathbf{x}, \mathbf{x}') = \sum_{d=1}^{D} (x_d - x'_d)^2 / \theta_d^2. \tag{4}$$

is often a default choice for Gaussian process regression. However, sample functions with this covariance function are unrealistically smooth for practical optimization problems. We instead propose

the use of the ARD Matérn $5/2$ kernel:

$$K_{\mathsf{M52}}(\mathbf{x}, \mathbf{x}') = \theta_0 \left(1 + \sqrt{5r^2(\mathbf{x}, \mathbf{x}')} + \frac{5}{3}r^2(\mathbf{x}, \mathbf{x}')\right) \exp\left\{-\sqrt{5r^2(\mathbf{x}, \mathbf{x}')}\right\}. \tag{5}$$

This covariance function results in sample functions which are twice-differentiable, an assumption that corresponds to those made by, e.g., quasi-Newton methods, but without requiring the smoothness of the squared exponential.

After choosing the form of the covariance, we must also manage the hyperparameters that govern its behavior (Note that these "hyperparameters" are distinct from those being subjected to the overall Bayesian optimization.), as well as that of the mean function. For our problems of interest, typically we would have $D + 3$ Gaussian process hyperparameters: D length scales $\theta_{1:D}$, the covariance amplitude θ_0, the observation noise ν, and a constant mean m. The most commonly advocated approach is to use a point estimate of these parameters by optimizing the marginal likelihood under the Gaussian process, $p(\mathbf{y} \mid \{\mathbf{x}_n\}_{n=1}^N, \theta, \nu, m) = \mathcal{N}(\mathbf{y} \mid m\mathbf{1}, \mathbf{\Sigma}_\theta + \nu\mathbf{I})$, where $\mathbf{y} = [y_1, y_2, \cdots, y_N]^\mathsf{T}$, and $\mathbf{\Sigma}_\theta$ is the covariance matrix resulting from the N input points under the hyperparameters θ.

However, for a fully-Bayesian treatment of hyperparameters (summarized here by θ alone), it is desirable to marginalize over hyperparameters and compute the *integrated acquisition function*:

$$\hat{a}(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}) = \int a(\mathbf{x}\,;\, \{\mathbf{x}_n, y_n\}, \theta)\, p(\theta \mid \{\mathbf{x}_n, y_n\}_{n=1}^N)\, \mathrm{d}\theta, \tag{6}$$

where $a(\mathbf{x})$ depends on θ and all of the observations. For probability of improvement and EI, this expectation is the correct generalization to account for uncertainty in hyperparameters. We can therefore blend acquisition functions arising from samples from the posterior over GP hyperparameters and have a Monte Carlo estimate of the integrated expected improvement. These samples can be acquired efficiently using slice sampling, as described in Murray and Adams [13]. As both optimization and Markov chain Monte Carlo are computationally dominated by the cubic cost of solving an N-dimensional linear system (and our function evaluations are assumed to be much more expensive anyway), the fully-Bayesian treatment is sensible and our empirical evaluations bear this out. Figure 1 shows how the integrated expected improvement changes the acquistion function.

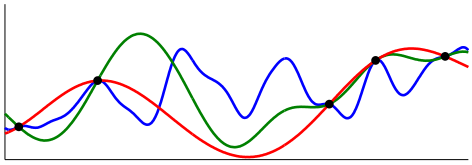
3.2 Modeling Costs

Ultimately, the objective of Bayesian optimization is to find a good setting of our hyperparameters as quickly as possible. Greedy acquisition procedures such as expected improvement try to make the best progress possible in the next function evaluation. From a practial point of view, however, we are not so concerned with function evaluations as with wallclock time. Different regions of the parameter space may result in vastly different execution times, due to varying regularization, learning rates, etc. To improve our performance in terms of wallclock time, we propose optimizing with the *expected improvement per second*, which prefers to acquire points that are not only likely to be good, but that are also likely to be evaluated quickly. This notion of cost can be naturally generalized to other budgeted resources, such as reagents or money.

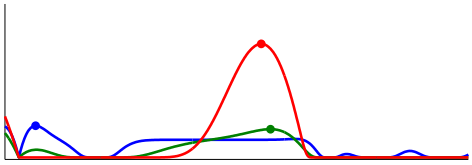
Just as we do not know the true objective function $f(\mathbf{x})$, we also do not know the *duration function* $c(\mathbf{x}) : \mathcal{X} \to \mathbb{R}^+$. We can nevertheless employ our Gaussian process machinery to model $\ln c(\mathbf{x})$ alongside $f(\mathbf{x})$. In this work, we assume that these functions are independent of each other, although their coupling may be usefully captured using GP variants of multi-task learning (e.g., [14, 15]). Under the independence assumption, we can easily compute the predicted expected inverse duration and use it to compute the expected improvement per second as a function of $\mathbf{x}$.

3.3 Monte Carlo Acquisition for Parallelizing Bayesian Optimization

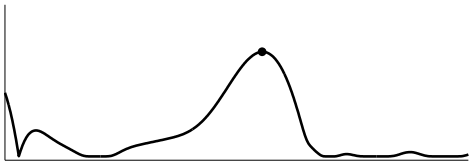
With the advent of multi-core computing, it is natural to ask how we can parallelize our Bayesian optimization procedures. More generally than simply batch parallelism, however, we would like to be able to decide what $\mathbf{x}$ should be evaluated next, even while a set of points are being evaluated. Clearly, we cannot use the same acquisition function again, or we will repeat one of the pending experiments. Ideally, we could perform a roll-out of our acquisition policy, to choose a point that appropriately balanced information gain and exploitation. However, such roll-outs are generally intractable. Instead we propose a sequential strategy that takes advantage of the tractable inference properties of the Gaussian process to compute Monte Carlo estimates of the acquisiton function under different possible results from pending function evaluations.



(a) Posterior samples under varying hyperparameters

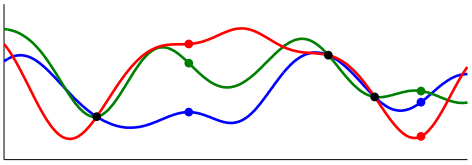


(b) Expected improvement under varying hyperparameters

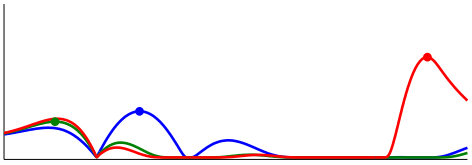


(c) Integrated expected improvement

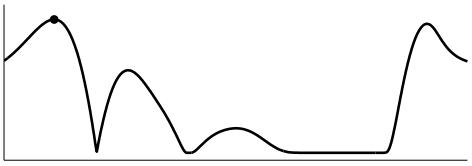
Figure 1: Illustration of integrated expected improvement. (a) Three posterior samples are shown, each with different length scales, after the same five observations. (b) Three expected improvement acquisition functions, with the same data and hyperparameters. The maximum of each is shown. (c) The integrated expected improvement, with its maximum shown.



(a) Posterior samples after three data



(b) Expected improvement under three fantasies



(c) Expected improvement across fantasies

Figure 2: Illustration of the acquisition with pending evaluations. (a) Three data have been observed and three posterior functions are shown, with "fantasies" for three pending evaluations. (b) Expected improvement, conditioned on the each joint fantasy of the pending outcome. (c) Expected improvement after integrating over the fantasy outcomes.

Consider the situation in which N evaluations have completed, yielding data $\{\mathbf{x}_n, y_n\}_{n=1}^N$, and in which J evaluations are pending at locations $\{\mathbf{x}_j\}_{j=1}^J$. Ideally, we would choose a new point based on the expected acquisition function under all possible outcomes of these pending evaluations:

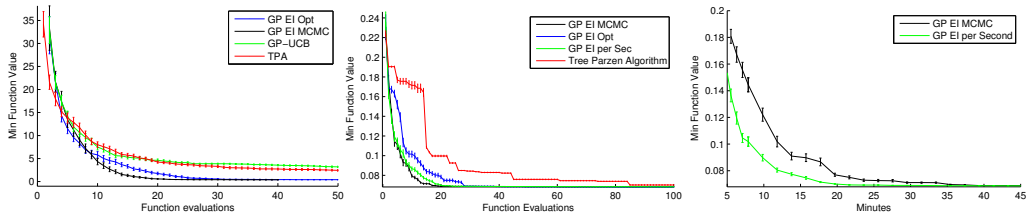
$$\hat{a}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta, \{\mathbf{x}_j\}) = \int_{\mathbb{R}^J} a(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta, \{\mathbf{x}_j, y_j\})\, p(\{y_j\}_{j=1}^J \mid \{\mathbf{x}_j\}_{j=1}^J, \{\mathbf{x}_n, y_n\}_{n=1}^N)\, \mathrm{d}y_1 \cdots \mathrm{d}y_J. \quad (7)$$

This is simply the expectation of $a(\mathbf{x})$ under a J-dimensional Gaussian distribution, whose mean and covariance can easily be computed. As in the covariance hyperparameter case, it is straightforward to use samples from this distribution to compute the expected acquisition and use this to select the next point. Figure 2 shows how this procedure would operate with queued evaluations. We note that a similar approach is touched upon briefly by Ginsbourger and Riche [16], but they view it as too intractable to warrant attention. We have found our Monte Carlo estimation procedure to be highly effective in practice, however, as will be discussed in Section 4.

4 Empirical Analyses

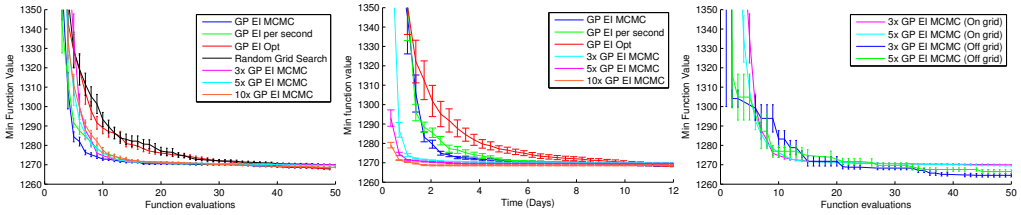
In this section, we empirically analyse[1] the algorithms introduced in this paper and compare to existing strategies and human performance on a number of challenging machine learning problems. We refer to our method of expected improvement while marginalizing GP hyperparameters as "GP EI MCMC", optimizing hyperparameters as "GP EI Opt", EI per second as "GP EI per Second", and N times parallelized GP EI MCMC as "Nx GP EI MCMC". Each results figure plots the progression of $\min_{\mathbf{x}_n} f(\mathbf{x}_n)$ over the number of function evaluations or time, averaged over multiple runs of each algorithm. If not specified otherwise, $\mathbf{x}_{\mathsf{next}} = \operatorname{argmax}_{\mathbf{x}} a(\mathbf{x})$ is computed using gradient-based search with multiple restarts (see supplementary material for details). The code used is made publicly available at `http://www.cs.toronto.edu/~jasper/software.html`.

[1] All experiments were conducted on identical machines using the Amazon EC2 service.



(a) (b) (c)

Figure 3: Comparisons on the Branin-Hoo function (3a) and training logistic regression on MNIST (3b). (3c) shows GP EI MCMC and GP EI per Second from (3b), but in terms of time elapsed.



(a) (b) (c)

Figure 4: Different strategies of optimization on the Online LDA problem compared in terms of function evaluations (4a), walltime (4b) and constrained to a grid or not (4c).

4.1 Branin-Hoo and Logistic Regression

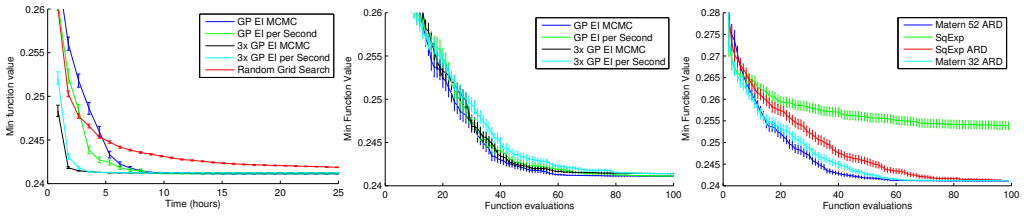
We first compare to standard approaches and the recent Tree Parzen Algorithm[2] (TPA) of Bergstra et al. [5] on two standard problems. The Branin-Hoo function is a common benchmark for Bayesian optimization techniques [2] that is defined over $x \in \mathbb{R}^2$ where $0 \leq x_1 \leq 15$ and $-5 \leq x_2 \leq 15$. We also compare to TPA on a logistic regression classification task on the popular MNIST data. The algorithm requires choosing four hyperparameters, the learning rate for stochastic gradient descent, on a log scale from 0 to 1, the ℓ_2 regularization parameter, between 0 and 1, the mini batch size, from 20 to 2000 and the number of learning epochs, from 5 to 2000. Each algorithm was run on the Branin-Hoo and logistic regression problems 100 and 10 times respectively and mean and standard error are reported. The results of these analyses are presented in Figures 3a and 3b in terms of the number of times the function is evaluated. On Branin-Hoo, integrating over hyperparameters is superior to using a point estimate and the GP EI significantly outperforms TPA, finding the minimum in less than half as many evaluations, in both cases. For logistic regression, 3b and 3c show that although EI per second is less efficient in function evaluations it outperforms standard EI in time.

4.2 Online LDA

Latent Dirichlet Allocation (LDA) is a directed graphical model for documents in which words are generated from a mixture of multinomial "topic" distributions. Variational Bayes is a popular paradigm for learning and, recently, Hoffman et al. [17] proposed an online learning approach in that context. Online LDA requires 2 learning parameters, τ_0 and κ, that control the learning rate $\rho_t = (\tau_0 + t)^{-\kappa}$ used to update the variational parameters of LDA based on the t^{th} minibatch of document word count vectors. The size of the minibatch is also a third parameter that must be chosen. Hoffman et al. [17] relied on an exhaustive grid search of size $6 \times 6 \times 8$, for a total of 288 hyperparameter configurations.

We used the code made publically available by Hoffman et al. [17] to run experiments with online LDA on a collection of Wikipedia articles. We downloaded a random set of 249 560 articles, split into training, validation and test sets of size 200 000, 24 560 and 25 000 respectively. The documents are represented as vectors of word counts from a vocabulary of 7702 words. As reported in Hoffman et al. [17], we used a lower bound on the per word perplexity of the validation set documents as the performance measure. One must also specify the number of topics and the hyperparameters η for the symmetric Dirichlet prior over the topic distributions and α for the symmetric Dirichlet prior over the per document topic mixing weights. We followed Hoffman et al. [17] and used 100 topics and $\eta = \alpha = 0.01$ in our experiments in order to emulate their analysis and repeated exactly the grid

[2]Using the publicly available code from `https://github.com/jaberg/hyperopt/wiki`



(a) (b) (c)

Figure 5: A comparison of various strategies for optimizing the hyperparameters of M3E models on the protein motif finding task in terms of walltime (5a), function evaluations (5b) and different covariance functions(5c).

search reported in the paper[3]. Each online LDA evaluation generally took between five to ten hours to converge, thus the grid search requires approximately 60 to 120 processor days to complete.

In Figures 4a and 4b we compare our various strategies of optimization over the same grid on this expensive problem. That is, the algorithms were restricted to only the exact parameter settings as evaluated by the grid search. Each optimization was then repeated 100 times (each time picking two different random experiments to initialize the optimization with) and the mean and standard error are reported[4]. Figure 4c also presents a 5 run average of optimization with 3 and 5 times parallelized GP EI MCMC, but without restricting the new parameter setting to be on the prespecified grid (see supplementary material for details). A comparison with their "on grid" versions is illustrated.

Clearly integrating over hyperparameters is superior to using a point estimate in this case. While GP EI MCMC is the most efficient in terms of function evaluations, we see that parallelized GP EI MCMC finds the best parameters in significantly less time. Finally, in Figure 4c we see that the parallelized GP EI MCMC algorithms find a significantly better minimum value than was found in the grid search used by Hoffman et al. [17] while running a fraction of the number of experiments.

4.3 Motif Finding with Structured Support Vector Machines

In this example, we consider optimizing the learning parameters of Max-Margin Min-Entropy (M3E) Models [18], which include Latent Structured Support Vector Machines [19] as a special case. Latent structured SVMs outperform SVMs on problems where they can explicitly model problem-dependent hidden variables. A popular example task is the binary classification of protein DNA sequences [18, 20, 19]. The hidden variable to be modeled is the unknown location of particular subsequences, or *motifs*, that are indicators of positive sequences.

Setting the hyperparameters, such as the regularisation term, C, of structured SVMs remains a challenge and these are typically set through a time consuming grid search procedure as is done in [18, 19]. Indeed, Kumar et al. [20] avoided hyperparameter selection for this task as it was too computationally expensive. However, Miller et al. [18] demonstrate that results depend highly on the setting of the parameters, which differ for each protein. M3E models introduce an entropy term, parameterized by α, which enables the model to outperform latent structured SVMs. This additional performance, however, comes at the expense of an additional problem-dependent hyperparameter. We emulate the experiments of Miller et al. [18] for one protein with approximately 40 000 sequences. We explore 25 settings of the parameter C, on a log scale from 10^{-1} to 10^{6}, 14 settings of α, on a log scale from 0.1 to 5 and the model convergence tolerance, $\epsilon \in \{10^{-4},10^{-3},10^{-2},10^{-1}\}$. We ran a grid search over the 1400 possible combinations of these parameters, evaluating each over 5 random 50-50 training and test splits.

In Figures 5a and 5b, we compare the randomized grid search to GP EI MCMC, GP EI per Second and their 3x parallelized versions, all constrained to the same points on the grid. Each algorithm was repeated 100 times and the mean and standard error are shown. We observe that the Bayesian optimization strategies are considerably more efficient than grid search which is the status quo. In this case, GP EI MCMC is superior to GP EI per Second in terms of function evaluations but GP EI per Second finds better parameters faster than GP EI MCMC as it learns to use a less strict con-

[3] i.e. the only difference was the randomly sampled collection of articles in the data set and the choice of the vocabulary. We ran each evaluation for 10 hours or until convergence.

[4] The restriction of the search to the same grid was chosen for efficiency reasons: it allowed us to repeat the experiments several times efficiently, by first computing all function evaluations over the whole grid and reusing these values within each repeated experiment.

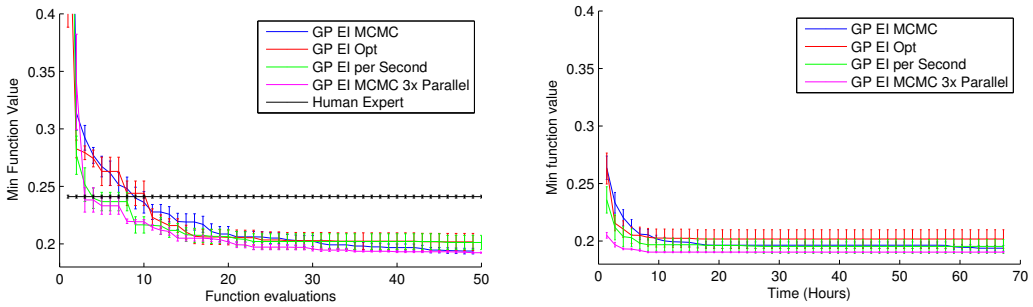


Figure 6: Validation error on the CIFAR-10 data for different optimization strategies.

vergence tolerance early on while exploring the other parameters. Indeed, 3x GP EI per second, is the least efficient in terms of function evaluations but finds better parameters faster than all the other algorithms. Figure 5c compares the use of various covariance functions in GP EI MCMC optimization on this problem[5], again repeating the optimization 100 times. It is clear that the selection of an appropriate covariance significantly affects performance and the estimation of length scale parameters is critical. The assumption of the infinite differentiability as imposed by the commonly used squared exponential is too restrictive for this problem.

4.4 Convolutional Networks on CIFAR-10

Neural networks and deep learning methods notoriously require careful tuning of numerous hyperparameters. Multi-layer convolutional neural networks are an example of such a model for which a thorough exploration of architechtures and hyperparameters is beneficial, as demonstrated in Saxe et al. [21], but often computationally prohibitive. While Saxe et al. [21] demonstrate a methodology for efficiently exploring model architechtures, numerous hyperparameters, such as regularisation parameters, remain. In this empirical analysis, we tune nine hyperparameters of a three-layer convolutional network [22] on the CIFAR-10 benchmark dataset using the code provided [6]. This model has been carefully tuned by a human expert [22] to achieve a highly competitive result of 18% test error on the unaugmented data, which matches the published state of the art result [23] on CIFAR-10. The parameters we explore include the number of epochs to run the model, the learning rate, four weight costs (one for each layer and the softmax output weights), and the width, scale and power of the response normalization on the pooling layers of the network.

We optimize over the nine parameters for each strategy on a withheld validation set and report the mean validation error and standard error over five separate randomly initialized runs. Results are presented in Figure 6 and contrasted with the average results achieved using the best parameters found by the expert. The best hyperparameters found by the GP EI MCMC approach achieve an error on the *test set* of 14.98%, which is over 3% better than the expert and the state of the art on CIFAR-10. The same procedure was repeated on the CIFAR-10 data augmented with horizontal reflections and translations, similarly improving on the expert from 11% to 9.5% test error and achieving to our knowledge the lowest error reported on the competitive CIFAR-10 benchmark.

5 Conclusion

We presented methods for performing Bayesian optimization for hyperparameter selection of general machine learning algorithms. We introduced a fully Bayesian treatment for EI, and algorithms for dealing with variable time regimes and running experiments in parallel. The effectiveness of our approaches were demonstrated on three challenging recently published problems spanning different areas of machine learning. The resulting Bayesian optimization finds better hyperparameters significantly faster than the approaches used by the authors and *surpasses* a human expert at selecting hyperparameters on the competitive CIFAR-10 dataset, beating the state of the art by over 3%.

Acknowledgements

The authors thank Alex Krizhevsky for making his neural network code available, and George Dahl for valuable feedback. This work was funded by DARPA Young Faculty Award N66001-12-1-4219, NSERC and an Amazon AWS in Research grant.

[5] See also the supplementary material for comparisons on other problems.

[6] Available at: `http://code.google.com/p/cuda-convnet/`

References

[1] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129, 1978.

[2] D.R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

[3] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[4] Adam D. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, (3-4):2879–2904, 2011.

[5] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Bálázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 25*. 2011.

[6] Marc C. Kennedy and Anthony O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3), 2001.

[7] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization 5*, 2011.

[8] Nimalan Mahendran, Ziyu Wang, Firas Hamze, and Nando de Freitas. Adaptive mcmc with bayesian optimization. In *AISTATS*, 2012.

[9] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[10] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *pre-print*, 2010. arXiv:1012.2599.

[11] Carl E. Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[12] H. J. Kushner. A new method for locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86, 1964.

[13] Iain Murray and Ryan P. Adams. Slice sampling covariance hyperparameters of latent Gaussian models. In *Advances in Neural Information Processing Systems 24*, pages 1723–1731. 2010.

[14] Yee Whye Teh, Matthias Seeger, and Michael I. Jordan. Semiparametric latent factor models. In *AISTATS*, 2005.

[15] Edwin V. Bonilla, Kian Ming A. Chai, and Christopher K. I. Williams. Multi-task Gaussian process prediction. In *Advances in Neural Information Processing Systems 22*, 2008.

[16] David Ginsbourger and Rodolphe Le Riche. Dealing with asynchronicity in parallel Gaussian process based global optimization. `http://hal.archives-ouvertes.fr/hal-00507632`, 2010.

[17] Matthew Hoffman, David M. Blei, and Francis Bach. Online learning for latent Dirichlet allocation. In *Advances in Neural Information Processing Systems 24*, 2010.

[18] Kevin Miller, M. Pawan Kumar, Benjamin Packer, Danny Goodman, and Daphne Koller. Max-margin min-entropy models. In *AISTATS*, 2012.

[19] Chun-Nam John Yu and Thorsten Joachims. Learning structural SVMs with latent variables. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

[20] M. Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems 25*. 2010.

[21] Andrew Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Technical report, Department of Computer Science, University of Toronto*, 2009.

[23] Adam Coates and Andrew Y. Ng. Selecting receptive fields in deep networks. In *Advances in Neural Information Processing Systems 25*. 2011.