BAYESIAN OPTIMIZATION OF FEL PERFORMANCE AT LCLS*

M. McIntire, T. Cope, D. Ratner, SLAC, Menlo Park, CA 94025, USA
S. Ermon, Stanford University, Stanford, CA 94305, USA

Abstract

The LCLS free-electron laser at SLAC is tuned via a huge number of parameters such as energy and magnet settings. Much of this tuning, including quadrupole magnet settings, is typically done by hand by the LCLS operators. In this paper we introduce an automated tuning system using Bayesian optimization, and describe its application to the optimization of noisy objectives such as FEL performance. We demonstrate with preliminary results from our implementation at LCLS that this system can improve both the speed of tuning procedures as well as the quality of the resulting solution.

INTRODUCTION

In order to reduce its daunting operational costs and increase the availability of the machine for experiments, the LCLS free-electron laser (FEL) [1] is currently undergoing a broad effort to automate and streamline its various procedures. One target for automation is the tuning of quadrupole magnets, which currently requires the attention of an operator and is extremely time consuming. Machine drift over time and noisiness of the physical processes leads to a difficult optimization problem that must be addressed regularly in order to keep the FEL tuned properly. Development of advanced controls is a high priority for accelerators worldwide (see e.g. [2–6]).

Recently DESY has developed the Ocelot tuning framework [7] which facilitates general optimization of machine parameters. Initial tests of Ocelot to control quads at LCLS with e.g. Nelder-Mead optimization [8] have been promising. However, such methods have various pitfalls in our setting. The Nelder-Mead algorithm, for example, is unable to backtrack and cannot deal effectively with the substantial noisiness of the FEL performance feedback. This often results in improper convergence and can require e.g. several restarts of the optimization procedure, which extends tuning time and may not improve performance.

Another drawback of common optimization algorithms is that they are relatively incapable of incorporating physical models and prior knowledge. Ideally, we would like to be able to guide the optimization process with our knowledge about what regions are likely to be promising or how noisy the signal is. In this paper we address this need for flexibility and robustness to noise by introducing the Gaussian process as a highly flexible and powerful model. We then describe Bayesian optimization, which uses this Gaussian process model to optimize efficiently and robustly. We also describe the implementation of this Bayesian optimization framework for FEL performance tuning at LCLS and present the preliminary results of our testing of this and other optimization methods.

GAUSSIAN PROCESSES

Gaussian processes (GPs) provide us with a powerful method of performing inference on a probability distribution over functions. Formally, a GP is a (possibly infinite) set of random variables Y with the property that every finite set of variables $y_1, \cdots, y_n \in Y$ has a multivariate normal distribution. In Gaussian process regression, these random variables represent the dependent variable at corresponding locations in parameter space. For example, suppose y_1 and y_2 are random variables corresponding to output at locations x_1, x_2 in parameter space. Then the distributions of y_1 and y_2, as well as their joint distribution, are all normal. Informally, we might expect that if x_1 and x_2 are near each other in parameter space, then y_1 and y_2 might strongly covary; we will soon make this precise.

A GP prior is defined entirely in terms of its prior mean function and its covariance function. If $\mathcal{X}$ is our parameter space, the prior mean function is a function $f_0 : \mathcal{X} \to \mathbb{R}$. The prior mean is often taken to be zero for computational convenience but can be an arbitrary function, though as is the case in general for Bayesian statistics, a misspecified prior can be harmful. The covariance (or kernel) function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is essentially a difference measure on the parameter space. The covariance function has a dramatic effect on the GP posterior that results from inference over training data.

The key to GP regression lies in this process of training the GP by doing inference over training data. Given training data X with output $\mathbf{y}$, we can evaluate the GP posterior at test points X_* to obtain the resulting multivariate normal distribution; the mean of this distribution is the GP posterior mean at the test points, which is the output for regression. We additionally have the covariance for this distribution, providing not only the uncertainties for individual regression outputs but the full joint covariance for the outputs.

Explicitly, let $K_X = K(X, X)$ be the Gram matrix of the training data, i.e. the kernel function applied to each pair of points in X. We assume that the observed values $\mathbf{y}$ are generated from an underlying 'true' function values $\mathbf{f}$ with normally distributed noise, i.e. $\mathbf{y} = \mathbf{f} + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$. Then we can (as shown in e.g. [9]) write the posterior mean for function values $\mathbf{f}_*$ at X_* as

$$\mathbf{f}_* | X_*, X, \mathbf{y} \sim \mathcal{N}(K(X_*, X)[K_X + \sigma^2 I]^{-1}\mathbf{y}, \\ K(X_*, X_*) - K(X_*, X)[K_X + \sigma^2 I]^{-1} K(X, X_*) \,. \quad (1)$$

Training the GP therefore requires inversion of a matrix of size n if we are given n training points. This can cause a

* Work is supported by Department of Energy Contract No. DE-AC02-76SF00515.

Copyright © 2016 CC-BY-3.0 and by the respective authors

scaling problem for larger data sets; fortunately there is an extensive literature on sparse GPs and methods of approximation for the full GP posterior, e.g. [10–16]. For our purposes, using a sparse GP is necessary. We use the sparse online GP of [10], since the iterative update scheme introduced therein is particularly well-suited to the iterative structure of Bayesian optimization. See [17] for a general, unifying discussion of sparse GP approaches, and [9] for a thorough introduction to GPs and GP regression.

BAYESIAN OPTIMIZATION

Suppose we are given an objective function $g : \mathcal{X} \to \mathbb{R}$ mapping the parameter space $\mathcal{X}$ to a variable that we wish to maximize. In general optimization problems we are allowed to query g at whatever points we wish in order to find a maximum. In motivating Bayesian optimization, we assume that g is expensive to evaluate, e.g. perhaps evaluation requires adjusting machine settings and waiting for the changes to take effect. In this case, spending extra time on computation may be worthwhile if it reduces the number of function evaluations we must make.

Bayesian optimization addresses this trade-off by fully utilizing the observed data to decide where to evaluate g at each step. By maintaining a probabilistic model of the data as e.g. a Gaussian process, we can search parameter space for promising regions to explore via the GP rather than by querying g itself. This exploration is guided by an acquisition function $\alpha : \mathcal{X} \to \mathbb{R}$. Common acquisition functions include e.g. the probability of improvement and the expected improvement [18] at the sampled point. The expected improvement in particular has been shown to perform well in most settings, and has proven convergence properties [19].

Thanks to the simplicity of GPs, we can find an analytic form for the expected improvement at a point $\mathbf{x} \in \mathcal{X}$. We let $\mu : \mathcal{X} \to \mathbb{R}$ represent the GP posterior mean function, and denoting all observed points as X we let

$$y^* = \max_{\mathbf{x}_{obs} \in X} \mu(\mathbf{x}_{obs}) \, .$$

This definition of y^* seems convoluted at first glance, but is necessary due to the noise in our observations. Defining y^* as simply the maximum observation, or even μ evaluated at the location of the maximum observation, fails to account for the possibility that the maximum observation was maximal due to noise.

We then define the improvement at $\mathbf{x}$ as

$$I(\mathbf{x}) = \max(0, \mu(\mathbf{x}) - y^*) \, .$$

The expected value of $I(\mathbf{x})$ can be computed relatively easily given the normal distribution of the GP posterior at $\mathbf{x}$. Define

$$Z = \frac{\mu(\mathbf{x}) - y^*}{\sigma(\mathbf{x})} \, ,$$

and we then have

$$\mathrm{EI}(\mathbf{x}) \equiv E[I(\mathbf{x})] = \sigma(\mathbf{x})[Z\Phi(Z) + \phi(Z)] \, , \qquad (2)$$

Algorithm 1 Bayesian optimization

1: **while** Not converged **do**
2: Compute $\mathbf{x}_{t+1} = \arg\max_{\mathbf{x}}(\alpha(\mathbf{x}))$.
3: Query objective function at $\mathbf{x}_{t+1}$ to get y_{t+1}.
4:
5: Add $(\mathbf{x}_{t+1}, y_{t+1})$ to the model.
6: $t = t + 1$
7: **end while**

where Φ and ϕ respectively denote the CDF and PDF of the standard normal distribution [18].

With an acquisition function defined (in this case, $\alpha = \mathrm{EI}$), the Bayesian optimization procedure becomes straightforward. Pseudocode is given in Figure 1; in iteration t, a new point $\mathbf{x}_{t+1}$ is computed by maximizing the acquisition function over the parameter space. We then query the objective function at $\mathbf{x}_{t+1}$, and incorporate the new information $(\mathbf{x}_{t+1}, y_{t+1})$ into the model (in our case, this means updating the GP with additional training data). This can be repeated for a given number of iterations or until some convergence criterion is met. For a more in-depth introduction to Bayesian optimization, see [20].

Our system for Bayesian optimization uses a variant of the common squared exponential kernel for its GP model. The squared exponential kernel K_e is defined as

$$K_e(\mathbf{x}_1, \mathbf{x}_2) = \theta_1^2 \exp\Big(- \frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\theta_2^2} \Big) \, .$$

Here θ_1 and θ_2 are hyperparameters that influence the covariance function. The role of θ_1 as a scaling factor is relatively clear, while θ_2 is a length-scale parameter that effectively determines how far points can be from each other before they become uncorrelated; see [9] for more discussion. Our covariance function K is of the form

$$K(\mathbf{x}_1, \mathbf{x}_2) = \theta^2 \exp\Big(- \frac{1}{2}(\mathbf{x}_1 - \mathbf{x}_2)^\top \Lambda^{-1} (\mathbf{x}_1 - \mathbf{x}_2) \Big) \, , \qquad (3)$$

where Λ is a diagonal matrix containing (squared) length-scale parameters for each dimension of the data. This allows us more flexibility than the squared exponential kernel, since we can separately handle the scaling of each dimension.

Choosing values for these hyperparameters is a critical consideration. Many methods exist for hyperparameter selection [9], including maximizing the probability of the data given the model as well as cross-validation. However, these approaches assume that the data is presented all at once, allowing for computations on the entire data set. In our setting, this is not the case; instead, data points arrive one at a time and the performance of our optimizer is directly tied to how well the model fits the data at each iteration.

A benefit of our choice of covariance function (3) is the physical interpretability of its parameters. This allows us to choose appropriate hyperparameters without observing a full data set, instead using our knowledge of the physical system. Our method for computing hyperparameters is described more fully below.

Copyright © 2016 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-147-2

INTEGRATION INTO LCLS

At LCLS, the python software framework OCELOT [7] has recently been integrated and used for online optimization of various parameters on the accelerator (most notably the X-ray pulse intensity gas detector) with the Nelder-Mead algorithm [8] and other optimization algorithms. Using OCELOT, we have developed a general optimization application used to manage the interface between optimization algorithms and the accelerator control system (see [21]). This application has been expanded to allow testing of Bayesian optimization using GPs which incorporates existing tools for safe and streamlined development on a live machine. Integration into the existing UI interface enables fast development of new optimization approaches using the general interface for algorithm testing.

Our system for Bayesian optimization has been integrated into the LCLS control system by creation of a basic getter and setter wrapper for data requests between the optimizer and LCLS controls. During an optimization run devices are chosen, an optimization algorithm is selected, and initialization options are set from a PyQt UI. After the optimization is complete, the data and the final GP model parameters are archived for later analysis and initialization of future optimizers.

The GP model used in Bayesian optimization requires information about the machine to construct an initial model. To address this requirement (which is unique to the Bayesian optimization approach as compared to e.g. Nelder-Mead optimization), the interface was built to allow for the use of historical data to initialize a model, or to load a GP model directly from a file. For early testing, we have also implemented a method that initializes the GP model from the first few iterations of a Nelder-Mead optimizer, which begins by moving each device to explore the parameter space. This method of initialization allows us to quickly run a GP scan without first loading and formatting historical data to construct a model, and generates fresh information about the machine response.

On initialization of the GP we also must specify hyperparameters for its covariance function. These hyperparameters are generated using historical data of typical device ranges for a given beam energy. More specifically, data for each device used in the optimization is collected and binned according to beam energy. This data is then used to define the length scale hyperparameters Λ in terms of the device ranges, e.g. as 50% of the difference between the 90th and 10th percentile values of the device. These length scales directly affect the spatial resolution of the GP model, which we will demonstrate shortly.

In addition to the hyperparameter Λ, we must also specify the noise variance σ^2 and the coefficient θ from equations (1) and (3) respectively. We can set these parameters directly as functions of live measurements of the FEL beam jitter, so that σ gives us an appropriate measure of the noisiness of the feedback during optimization. The hyperparameter θ also can be set as a function of beam jitter.

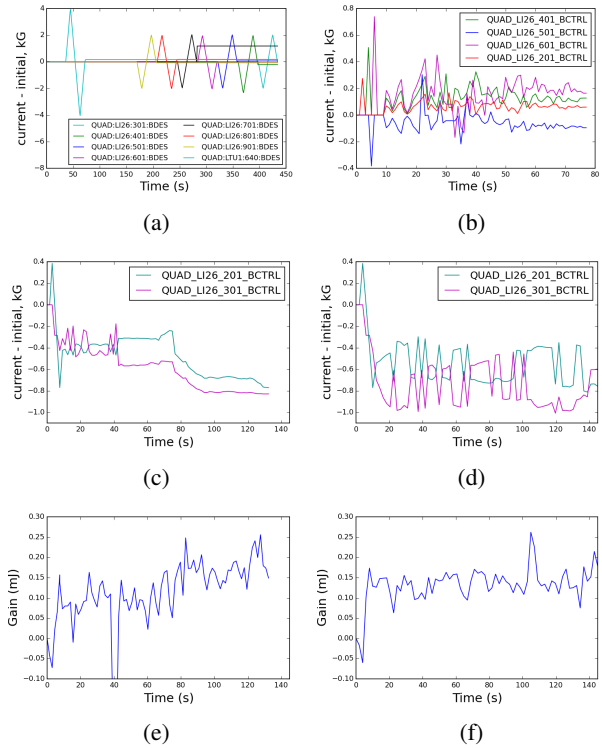

Figure 1: Comparison between methods of optimization. (a) The current method of hand-tuning quad settings requires several minutes of tunings quads one-by-one. (b) A Nelder-Mead optimization, which explores initially and then narrows its focus. (c-f) A comparison between Bayesian optimization using different length scales, done consecutively and with the same machine settings. (c) and (e) show quad settings and pulse energy over time using small length scales, while (d) and (f) show the same using doubled length scales.

PRELIMINARY RESULTS

Initial tests have been conducted using Bayesian optimization. First tests show that this method can achieve faster optimization than hand tuning and other optimization methods (examples of which are shown in Figures 1a and 1b). The results of Bayesian optimization appear to depend strongly on the hyperparameters of the optimizer.

This importance is demonstrated in Fig. 1c and 1d, which show quad settings from two optimization runs performed back-to-back to minimize the effect of machine drift, using the same initial settings. In Fig. 1d, the length scales from the previous run were doubled, which leads to much faster initial movement and gains as seen by the figures directly below (1e and 1f), which show the pulse energy during the optimization. However, these larger length scales inhibit the fine-tuning ability of the optimizer.

We are still in the initial stages of testing our Bayesian optimization system on the live machine, and are gathering data to improve our understanding of hyperparameter selection and model initialization before a fully automated version is used in standard tuning procedures. Our findings thus far leave us optimistic that substantial improvements can be made over current optimization tools.

Copyright © 2016 CC-BY-3.0 and by the respective authors

REFERENCES

[1] P. Emma et al. First lasing and operation of an ångstrom-wavelength free-electron laser. *Nature Photonics*, 4:641 – 647, 2010.

[2] R. Bartolini, M. Apollonio, and I. P. S. Martin. Multiobjective genetic algorithm optimization of the beam dynamics in linac drivers for free electron lasers. *Phys. Rev. ST Accel. Beams*, 15:030701, Mar 2012.

[3] Alicia Hofler, Bal ša Terzić, Matthew Kramer, Anton Zvezdin, Vasiliy Morozov, Yves Roblin, Fanglei Lin, and Colin Jarvis. Innovative applications of genetic algorithms to problems in accelerator physics. *Phys. Rev. ST Accel. Beams*, 16:010101, Jan 2013.

[4] Alexander Scheinker, Xiaoying Pang, and Larry Rybarcyk. Model-independent particle accelerator tuning. *Phys. Rev. ST Accel. Beams*, 16:102803, Oct 2013.

[5] Xiaobiao Huang, Jeff Corbett, James Safranek, and Juhao Wu. An algorithm for online optimization of accelerators. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 726:77 – 83, 2013.

[6] Sandra G. Biedron, Auralee Edelen, and Stephen Milton. Advanced controls for accelerators. In *High-Brightness Sources and Light-Driven Interactions*, page EM9A.3. Optical Society of America, 2016.

[7] I. Agapov, G. Geloni, S. Tomin, and I. Zagorodnov. Ocelot: A software framework for synchrotron light source and {FEL} studies. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 768:151 – 156, 2014.

[8] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[9] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[10] Lehel Csató. *Gaussian Processes - Iterative Sparse Approximations*. PhD thesis, Aston University, 2002.

[11] Lehel Csató and Manfred Opper. Sparse on-line Gaussian processes. *Neural Computation*, 14(3):641–668, 2002.

[12] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264. MIT press, 2006.

[13] Neil Lawrence, Matthias Seeger, and Ralf Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems 15*, pages 625–632. MIT Press, January 2003.

[14] James Hensman, Nicoló Fusi, and Neil D. Lawrence. Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, 2013.

[15] Michalis K. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *In Artificial Intelligence and Statistics 12*, pages 567–574, 2009.

[16] Matthias Seeger, Christopher K. I. Williams, and Neil D. Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *Workshop on AI and Statistics 9*, 2003.

[17] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

[18] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492, December 1998.

[19] Adam D. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12:2879–2904, November 2011.

[20] 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. eprint arXiv:1012.2599, arXiv.org, December 2010.

[21] S. Tomin, G. Geloni, I. Agapov, I. Zagorodnov, Ye. Fomin, Yu. Krylov, A. Valintinov, W. Colocho, T. M. Cope, A. Egger, and D. Ratner. Progress in automatic software-based optimization of accelerator performance. In *Proceedings of IPAC 2016*.

Copyright © 2016 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-147-2