Comparison of Optimization Techniques for Stateful Black Boxes

Jerome Rasky 42386145, Tristan Rice 25886145

Abstract

Current research into global black box optimization of expensive functions focuses on stateless models such as machine learning hyper parameter optimization. Many real world problems cannot be reset back to a clean state after a single evaluation. In this paper, we evaluate the performance of five popular black box algorithms on two stateful models: governmental budgeting and flying an airplane. We find that current black-box optimization methods do not sufficiently deal with noisy functions, especially when that noise is highly dependent on previous parameters, and dominates the function output.

Introduction

For our project, we were interested in the area of black-box optimization. Many engineering simulations and problems exist in a complex problem space without easy access to gradients, and often where function evaluation itself is extremely expensive. In these kinds of situations, easy solutions like gradient descent are out of scope, and others like Monte Carlo methods are prohibitively expensive.

We wanted to explore the specific case of stateful black-boxes. These are functions whose output is noisy, and whose noise depends on previous parameters. This includes situations like government budgeting, where each year feels the impact of previous years, or flying an airplane, where each second the plane continues to feel the impact of previous control movements.

By exploring various methods on these stateful models, we hope to gain an understanding for how existing methods fare in the presence of stateful models.

Related Work

Pattarwat Chormai (2017) and Chen et al. (2016)

These papers take a reasonably popular approach: build a differentiable model of the function, and perform gradient descent on that model. In particular, the authors of the above papers build a Gaussian process from samples of the black-box function, and then train a long short-term memory recurrent neural network to minimize that Gaussian process. This method has the advantage of learning latent factors that impact optimization, and so are less dependent on the underlying response surface model used. Like other papers, these papers investigate the performance of this method on noiseless functions. The authors in both of these papers make claims about the performance of their methods, which we wanted to evaluate in a slightly context.

Hansen et al. (2010)

There is a considerable amount of prior work in the area of black-box optimization. This paper in particular compares 31 methods against the BBOB-2009 functions, which vary in their complexity

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

and their difficulty in optimization. Generally, this paper found that the best method to use depends heavily on the dimensionality of the problem. In general, the more complex the function, the tougher it is to optimize. We wanted to extend this analysis to noisy functions, which this paper does not address. In particular, we wanted to evaluate current methods against very noisy functions.

Jones, Schonlau, and Welch (1998)

This earlier paper on black-box optimization explores the space of response surfaces, and using those surfaces to select further points of interest. We implement a method along these lines to compare against. This method involves measuring expected improvement, which is a combination of minimum value and error at a certain point on the response surface. Using more robust statistical methods to search the function space of a black-box function makes sense, given the computational cost of evaluating the method itself. We wanted to extend this analysis to stateful black-box functions, as a kind of baseline to compare other popular methods against.

Description and Justification

Our project involves testing various black-box optimization methods on stateful functions. This means our project has essentially two components: the models to train against, and the models to be trained. We selected both on the basis of varying complexity, in order to get a broader understanding of the current space of black-box optimization.

Algorithms

We selected our algorithms on the basis of current state-of-the-art optimization packages, as well as from trends in machine learning. These models vary in their complexity, and very in turn in their performance. For some of these models, we implemented a variant that includes the iteration number as a feature. This essentially turns these models into a time-series model, which is an appropriate approach to take in the case of stateful functions.

Random Search

Random search is one of the simplest black box optimization algorithms. In a typical implementation, it randomly guesses points within the bounds and then takes the smallest value as the best option. Since we are modeling continuous processes, we do not ever take a fixed "best" value and always randomize at each iteration. This is used as a baseline for the other algorithms.

Gradient Boosted Decision Trees (GBDT)

One common solution to black box optimization is to train a model on top of all the sampled points and then minimize over the model to pick the next point. At each step our implementation trains a set of gradient boosted decision trees using the CatBoost library (Dorogush, Ershov, and Gulin 2017). To find the global minimum of the model we pick 100 random points within the search space and then use L-BFGS-B (Byrd et al. 1995) to find the local minimum. We use the best of those 100 local minima as the next search point.

We test two versions of this model. The first has just the model parameters we are interested in, and the second additionally has the iteration number. Adding the iteration number provides a limited way to model the changing hidden state.

Bayesian Optimization (Gaussian Processes)

This method uses Gaussian Processes to build a model of the original function which we then use to find the point to maximize the expected improvement. This is a very popular hyper parameter optimization algorithm and is used as the default at several large companies including Google (Golovin et al. 2017). For our tests, we use the BayesOpt library (Martinez-Cantin 2014).

We test the same two variants of input parameters for this model as we did with GBDT.

Tree-structured Parzen Estimator

A Tree-structured Parzen Estimator (Bergstra et al. 2011) is another algorithm that has fairly decent performance for hyper parameter optimization. We used the HyperOpt library (Bergstra, Yamins, and Cox 2013) with the TPE minimizer.

This model was tested with only the input parameters due to inflexibility of the library.

LSTM based Recurrent Neural Network

We implemented the model as described in Pattarwat Chormai (2017). The problem of black box optimization can be formulated as the problem of finding the sequence containing the minimum value of a black box function. This formulation can be used to fit a Long Short Term Memory neural network. This method essentially uses an LSTM to learn how to minimize the function, rather than using a gradient.

At every step, the rnn $LSTM$ determines the next step to take.

$$\begin{aligned} x_n, h_n &= LSTM(x_{n-1}, y_{n-1}, h_{n-1}) \\ y_n &= \psi(x_n) \end{aligned}$$

Where $\psi(x) = E[GP(x)]$, the expected value of the Gaussian process model at point x.

Our implementation of this method uses Tensorflow (Abadi et al. 2015), making use of its LSTM framework. We also make use of GPflow (Matthews et al. 2017) for creating our Gaussian processes.

This method is obviously heavily dependent on the exact form of the Gaussian process. We started with the simplest case of a "vanilla" Gaussian process using radial basis functions for the kernels. This approach yielded poor results since with only a few data points to train on, most of the function space is predicted to be zero. Using a summed Matern and Linear kernel, performance can beat Bayesian optimization.

Models

To evaluate these algorithms we trained them on two stateful models. Each model has a single function that needs to be optimized with n parameters as well as bounds for the range of values that can be accepted.

Death Rate

Many real life models, such as governmental budgeting, are nearly impossible to evaluate. Instead, we built a stateful model of a governmental budget, and used it as a target of various black-box optimization methods. We created a simple model by using the World Bank Development Indicators and training a Gradient Boosted Decision Tree to predict deathrate based off of expenditures in education, health, R&D and military (Bank 2018). To add a stateful component to this, we added momentum, such that changing the parameters produces lag with respect to the death rate.

While this is a very simple model, it does provide some realistic behaviors. Many large systems have a high latency between cause and effect. We are also primarily interested in highlighting the differences between these algorithms.

We bound the inputs to be within two times the maximum existing value for that expenditure category and greater than zero.

Airplane

The second model is that of flying an airplane. The Python Flight Mechanics Engine is a project attempting to model every aspect of flying a plane (Team 2018). We used it to model flying a plane to a location. The model takes in the throttle as well as the position of the elevator, aileron and rudder and outputs the distance the plane has flown towards the target. This model has numerous stateful variables that need to be modeled including position, rotation, velocity, direction. There are also many nonlinearities due to things like air resistance and gravity.

This model uses the Cessna 172 as a base and bounds the inputs to be match the actual control range of the aircraft.

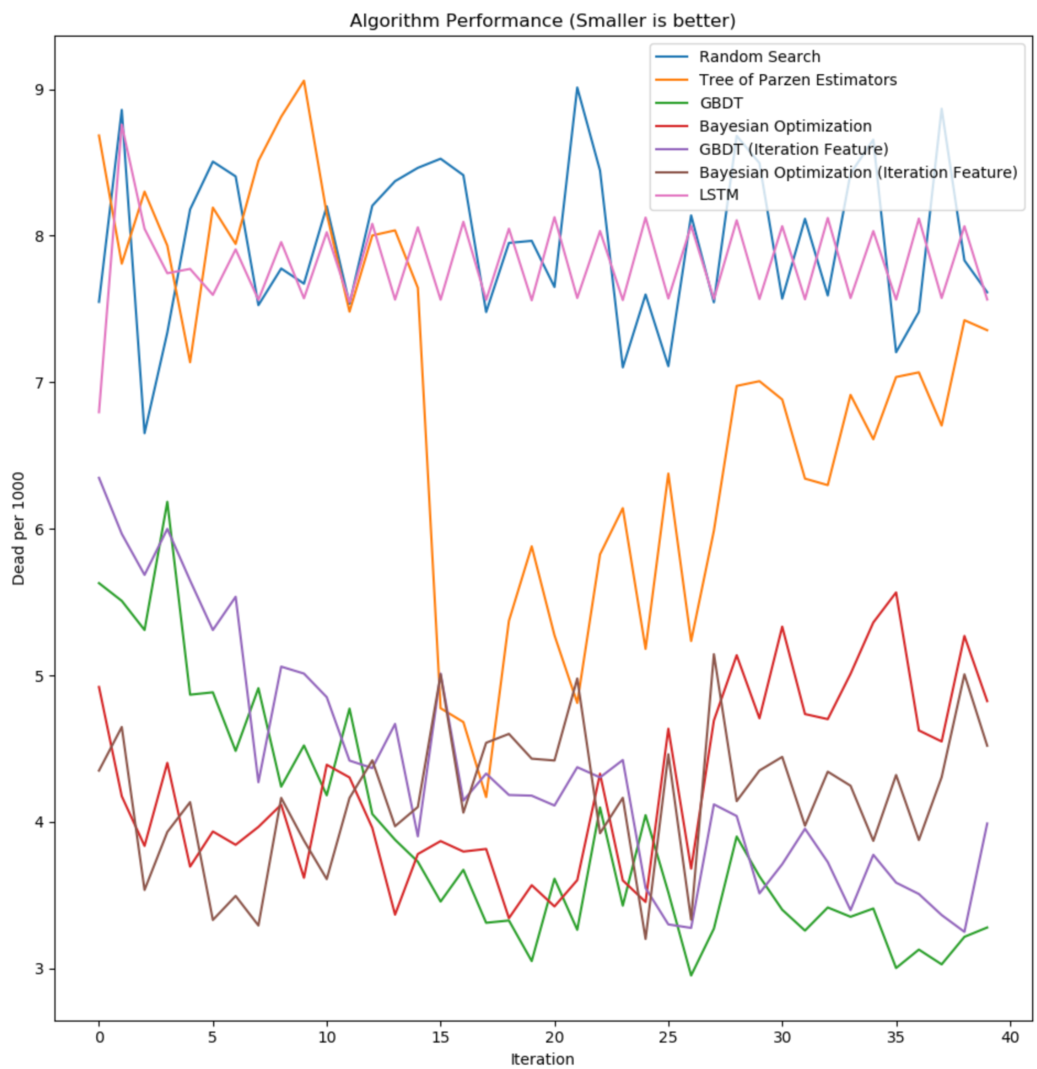
Experiments and Analysis

For all of our experiments, we initialized the models with five "free" random historical data points, and then used those models to predict the next 20 data points, updating the models at each step.

Commonly Bayesian Optimization methods start out with a series of random points to get an initial overview of the space. Our data shows that this leads to very poor performance with stateful models as 5 initial bad points can lead the model to a poor state before being able to recover.

Death Rate

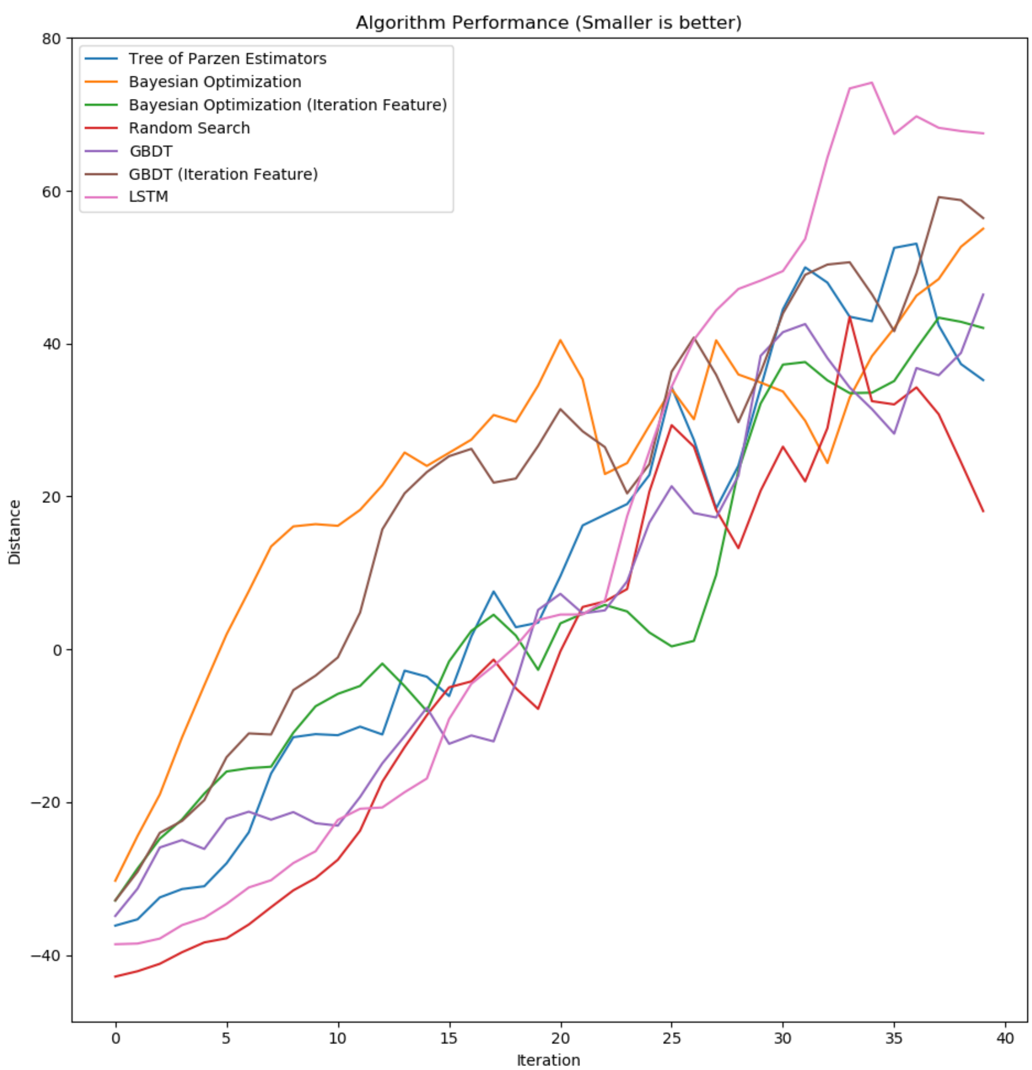
The death rate model is the simpler of the two to model. In general, all methods except for LSTM performed on average better than random. LSTM in particular only sometimes managed to learn something about how to minimize the actual function, which meant on average performance like random. In this case, gradient-boosted decision trees performed the best, followed by Bayesian optimization.

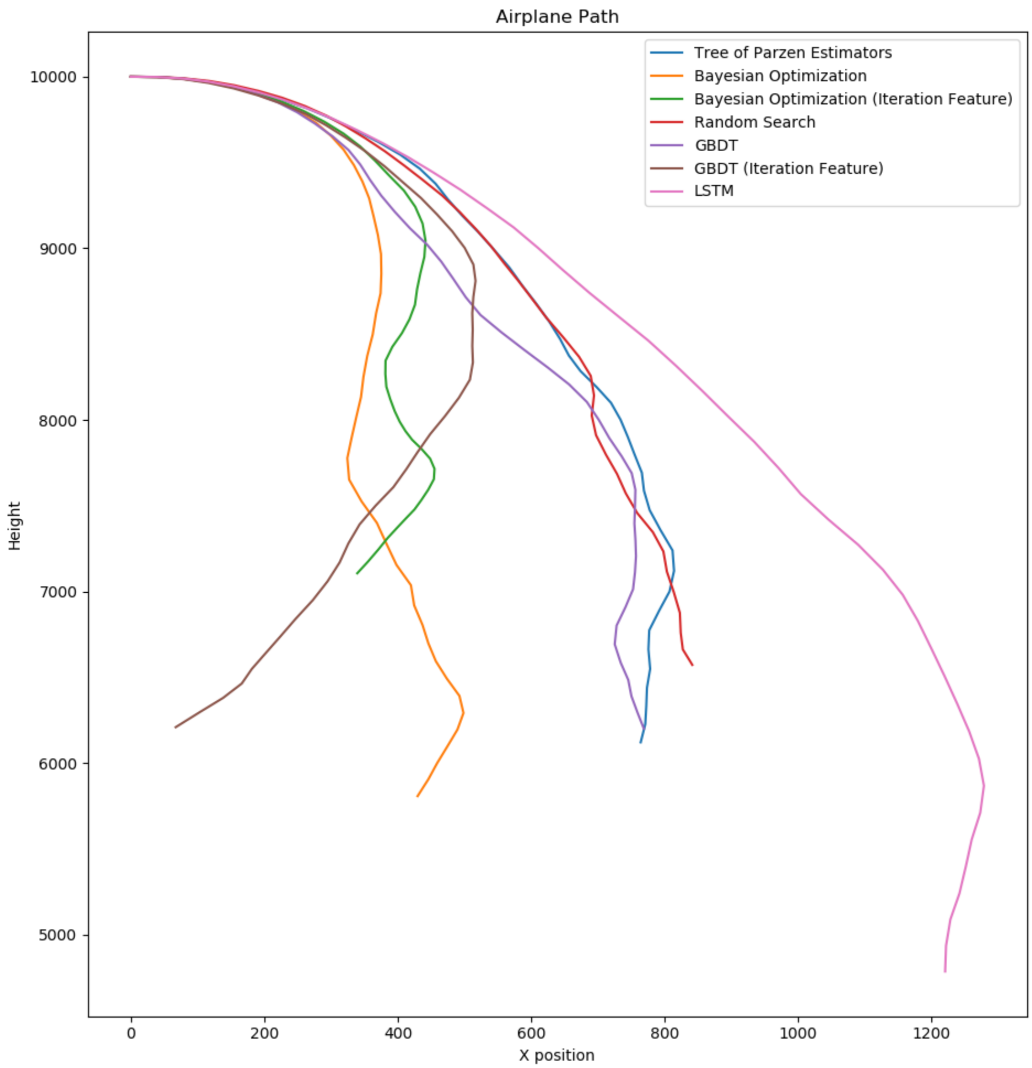


Method	Dead per 1000	Final	Time (s)
Bayesian Optimization	169.943	4.82427	270.191
Bayesian Optimization (Iteration Feature)	166.688	4.51994	230.342
GBDT	156.194	3.2796	126.613
GBDT (Iteration Feature)	174.163	3.98914	133.977
LSTM	312.352	7.56452	120.985
Random Search	318.699	7.6137	0.157484
Tree of Parzen Estimators	274.014	7.35505	0.740803

Airplane

With the airplane model being more complex, it makes sense that every method performed worse. Generally, no method performed better than random on our loss function, although the different methods we tried resulted in different paths taken by the plane. In particular, LSTM managed to fly the plane the furthest. The same effect is present here as well: five historical points is too few to accurately learn much about the behavior of the function.





Method	Distance	Final	Time (s)
Bayesian Optimization	951.158	55.0508	186.141
Bayesian Optimization (Iteration Feature)	254.179	42.0604	100.786
GBDT	170.037	46.4255	266.083
GBDT (Iteration Feature)	828.039	56.438	186.172
LSTM	483.213	67.5271	200.627
Random Search	-45.2105	18.0915	65.4368
Tree of Parzen Estimators	386.751	35.2345	87.0953

Discussion and Future Work

We found that existing methods are poor at dealing with complex, very stateful models. This is especially obvious in the case of flying an airplane: many models did worse than random. In the simpler case of a momentum method, where the output of the function is not very impacted by previous iterations, the black-box methods we implemented perform better.

This is in line with the related work in the area. Many papers, such as Hansen et al. (2010), are restricted to noiseless functions. The most direct approach to tackle this problem would be to create a model of the noise itself. In the case of highly stateful functions, this model would necessarily be complex. Given the methods we implemented do not attempt to model the noise in the functions they optimize, it is not surprising that they perform poorly when the function is highly stateful.

Our results are a modest sampling of methods, but clearly a larger sample of more methods would be even better. Our model of government budgeting is a simple momentum model, but in reality, a more complex, multi-year momentum in a higher number of dimensions would be a more accurate model of the impact of government budgets.

One area for future exploration is time series models in particular. We decided to investigate general black-box optimization algorithms which generalized poorly to stateful and time-series functions.

References

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, et al. 2015. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." https://www.tensorflow.org/.

Bank, The World. 2018. "Development Indicators." http://data.worldbank.org/indicator.

Bergstra, James S, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. "Algorithms for Hyper-Parameter Optimization." In *Advances in Neural Information Processing Systems*, 2546–54.

Bergstra, James, Dan Yamins, and David D Cox. 2013. "Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms." In *Proceedings of the 12th Python in Science Conference*, 13–20. Citeseer.

Byrd, Richard H, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. 1995. "A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM Journal on Scientific Computing* 16 (5). SIAM:1190–1208.

Chen, Yutian, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando de Freitas. 2016. "Learning to Learn Without Gradient Descent by Gradient Descent." *arXiv Preprint arXiv:1611.03824*.

Dorogush, Anna Veronika, Vasily Ershov, and Andrey Gulin. 2017. "CatBoost: Gradient Boosting with Categorical Features Support." *NIPS 17*. http://learningsys.org/nips17/assets/papers/paper_11.pdf.

Golovin, Daniel, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. 2017. "Google Vizier: A Service for Black-Box Optimization." In *Proceedings of the 23rd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 1487–95. KDD '17. New York, NY, USA: ACM. https://doi.org/10.1145/3097983.3098043.

Hansen, Nikolaus, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. 2010. "Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking Bbob-2009." In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, 1689–96. ACM.

Jones, Donald R, Matthias Schonlau, and William J Welch. 1998. "Efficient Global Optimization of Expensive Black-Box Functions." *Journal of Global Optimization* 13 (4). Springer:455–92.

Martinez-Cantin, Ruben. 2014. "BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits." *Journal of Machine Learning Research* 15:3915–9. http://jmlr.org/papers/v15/martinezcantin14a.html.

Matthews, Alexander G. de G., Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. 2017. "GPflow: A Gaussian process library using TensorFlow." *Journal of Machine Learning Research* 18 (40):1–6. http://jmlr.org/papers/v18/16-537.html.

Pattarwat Chormai, Raphael Holca-Lammare, Felix Sattler. 2017. "Black Box Optimization Using Recurrent Neural Networks," July. http://www.ni.tu-berlin.de/fileadmin/fg215/teaching/nnproject/black-box-optimization-final.pdf.

Team, AeroPython. 2018. "Python Flight Mechanics Engine." https://github.com/AeroPython/PyFME.