Bayesian optimization for hyper-parameter tuning

-ABHINAV JAIN[*], YOGESH SHARMA[+]
(*[*]Department of Electrical Engineering, [+]Department of Aerospace Engineering, IIT Kanpur*)

CS-774 OPTIMIZATION TECHNIQUES

Abstract

Most Machine learning algorithms require an appropriate tuning of model hyper-parameters to enhance the efficiency of the model in obtaining optimal results. In this report the automatic tuning problem within the framework of Bayesian optimization is considered, in which a learning algorithm's generalization performance is modelled as a sample from a Gaussian process (GP). The tractable posterior distribution induced by the GP leads to efficient use of the information gathered by previous experiments, enabling optimal choices about what parameters to try next. Since GPs scale cubically with number of observations, it's not a wise strategy to use it. Therefore, a deep neural network for Bayesian Optimization is studied that scales linearly with number of observations. A well-defined Bayesian optimization leads to results outperforming manual maneuvers in tuning machine learning algorithms. Further we implement the Bayesian optimization algorithm to tune hyper parameters for digit classification using a sparse encoder stacked with 'softmax' classifier.

I. INTRODUCTION

Machine learning algorithms are rarely parameter-free; learning procedures almost always require a set of high-level choices that significantly impact generalization performance. As a practitioner, one is usually able to specify the general framework of an inductive bias much more easily than the particular weighting that it should have relative to training data. As a result, these high-level parameters are often considered a nuisance, making it desirable to develop algorithms with as few of these "knobs" as possible. A more flexible take on this issue is to view the optimization of high-level parameters as a procedure to be automated. Specifically, we could view such tuning as the optimization of an unknown black-box function that reflects generalization performance and invoke algorithms developed for such problems. These optimization problems have a somewhat different flavor than the low-level objectives one often encounters as part of a training procedure: here function evaluations are very expensive. 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. It is desirable to understand how to include a concept of cost into the optimization procedure. In this setting where function evaluations are expensive, it is desirable to spend computational time making better choices about where to seek the best parameters. Bayesian optimization provides an elegant approach and has been shown to outperform other state of the art global optimization algorithms on a number of challenging optimization benchmark functions.

We argue that a fully Bayesian treatment of the GP kernel parameters is of critical importance to robust results, in contrast to the more standard procedure of optimizing hyper-parameters
We also examine the impact of the kernel itself and employ a Matern 5/2 kernel instead of the default choice of the squared-exponential covariance function. We take a look at the available choice of acquisition functions and use the Expected Improvement function to trade-off between exploration and exploitation. Since time cost of evaluation is critical we incorporate it in our approach.

Further, we point out the limitations of a GP and explore a different method to model our priors using deep neural networks. After building a sound understanding of the Bayesian optimization we implement it to tune hyper parameters for a digit classification problem using the publicly available spearmint package for our developed code given an MNIST dataset. We have used the digit classification algorithm using a sparse auto-encoder stacked with a 'softmax' classifier. The improved efficiency is presented under results. Following sections would delve deeper into each of these.

II. BAYESIAN OPTIMIZATION WITH GAUSSIAN PROCESS

Much of the efficiency stems from the ability of Bayesian optimization to incorporate prior belief about the problem to help direct the sampling, and to trade off exploration and exploitation of the search space. It is called Bayesian because it uses the famous "Bayes theorem", which states that the posterior probability of a model (or theory, or hypothesis) M given evidence (or data, or observations) E is proportional to the likelihood of E given M multiplied by the prior probability of M:

$$\mathrm{P(M|E)} \propto \mathrm{P(E|M)P(M)}$$

Inside this simple equation is the key to optimizing the objective function. In Bayesian optimization, the prior represents our belief about the space of possible objective functions. Although the cost function is unknown, it is reasonable to assume that there exists prior knowledge about some of its properties, such as smoothness, and this makes some possible objective functions more plausible than others.

The essential philosophy is to use all of the information available from previous evaluations of f(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(x) are expensive to perform as is the case when it requires training a machine learning algorithm it is easy to justify some extra computation to make better decisions.

A. Gaussian Process

A GP is an extension of the multivariate Gaussian distribution to an infinite dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution. Just as a Gaussian distribution is a distribution over a random variable, completely specified by its mean and covariance, a GP is a distribution over functions, completely specified by its mean function, m and covariance function, k:

$$f(x) \sim GP(m(x), K(x, x'))$$

GP models do more than just predicting expected outcome of a particular experiment. In addition to predicting the mean value for how well you expect a neural network to do, they predict a Gaussian distribution using a variance predictor function. We assume that the function f(x) is drawn from a Gaussian process prior and that our observations are of the form $\{\mathrm{x_n}, \mathrm{y_n}\}_{\mathrm{n=1}}^{\mathrm{N}}$ where $\mathrm{y_n} \sim \mathrm{N}\big(\mathrm{f(x_n)},\ \mathrm{v}\big)$, v is the variance of the function observations. It is often useful to intuitively think of a GP as analogous to a function, but instead of returning a scalar f(x) for an arbitrary x, it returns the mean and variance of a normal distribution. For convenience, we assume here that the prior mean is the zero function m(x) = 0; this leaves us with the choice of Kernel Function K(x).

Under the Gaussian process prior, these functions depend on the model solely through its predictive mean function $\mu(x; \{\mathrm{x_n}, \mathrm{y_n}\}, \theta)$ and predictive variance function $\sigma^2(x; \{\mathrm{x_n}, \mathrm{y_n}\}, \theta)$. We use the ARD Matern 5/2 kernel instead of the default choice of square exponential kernel.

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

$$r^2(x,x') = \sum_{d=1}^{D}(x_d - x'_d)^2\,/\theta_d^2$$

GP itself has its own hyper-parameters so a complete Bayesian treatment of the underlying GP kernel is preferred to the approach based on optimization of the GP hyper-parameters. Essentially we don't have any parameters except the choice for similarity matrix which is a categorical choice that needs to be made.

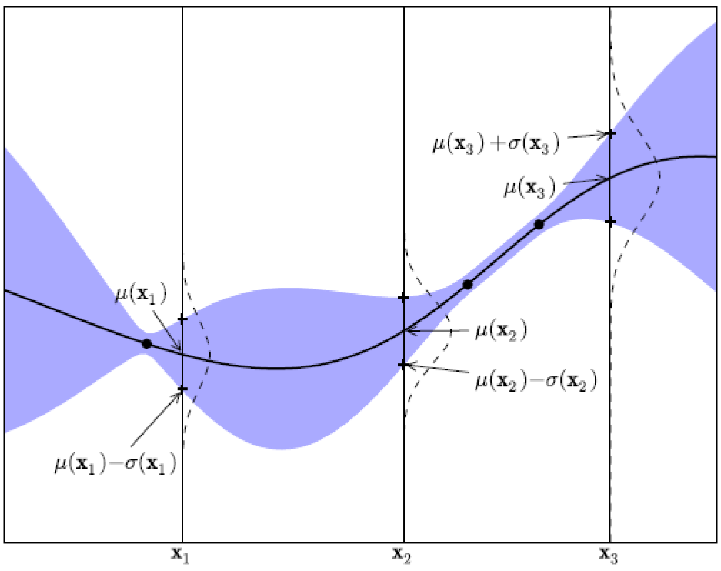

Fig. 1: Simple 1D Gaussian process with three observations. The solid black line is the GP surrogate mean prediction of the objective function given the data, and the shaded area shows the mean plus and minus the variance. The superimposed Gaussians correspond to the GP mean and standard deviation (μ(.) and σ(.)) of prediction at the points, x 1:3. Source: E. Brochu, V. M. Cora and N. D. Freitas, *"A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning"*

B. Acquisition Functions

It is a heuristic function that fits our data and tells us which experiment to run next by giving a score. It trades off between exploration (where variance is high) and exploitation (where mean is low). We select the next point for function value calculation by replacing the objective function with a pseudo objective with

$$x_{next} = arg\,\max_x a(x)$$

Maximizing acquisition function is itself an optimization problem. Our choice for acquisition function results in a tractable form which can be maximized using standard approaches.

Many acquisition functions can be interpreted in the framework of Bayesian decision theory as evaluating an expected loss associated with evaluating f at a point x. We then select the point with the lowest expected loss, as usual.

Probability of improvement:
Suppose:

$$f' = min\, f$$

is the minimal value of f observed so far. Probability of improvement evaluates f at the point most likely to improve upon this value. This corresponds to the following utility function associated with evaluating f at a given point x:

$$u(x) = \begin{cases} 0 \; f(x) > f' \\ 1 \; f(x) < f' \end{cases}$$

That is, we receive a unit reward if f(x) turns out to be less than f', and no reward otherwise. The probability of improvement acquisition function is then the expected utility as a function of x:

$$a_{pi}(x) = E[u(x)|x, D] = \int_{-\infty}^{f'} N(f; \mu(x), K(x, x))df$$

The point with the highest probability of improvement (the maximal expected utility) is selected. This is the Bayes action under this loss.

Expected improvement
Here, we get a reward for improving upon the current minimum independent of the size of the improvement. This can sometimes lead to odd behavior, and in practice can get stuck in local optima and underexplore globally. An alternative acquisition function that does account for the size of the improvement is expected improvement. Again suppose that f' is the minimal value of f observed so far. Expected improvement evaluates f at the point that, in expectation, improves upon f' the most. This corresponds to the following utility function:

$$u(x) = max(0, f' - f(x))$$

That is, we receive a reward equal to the "improvement" $f' -$ f(x) if f(x) turns out to be less than f', and no reward otherwise. The expected improvement acquisition function is then the expected utility as a function of x:

$$a_{EI}(x) = E[u(x)|x, D] = \int_{-\infty}^{f'} (f' - f)N(f; \mu(x), K(x, x))df$$
$$= (f' - \mu(x))\varphi\big(f'; \mu(x), K(x, x)\big) + K(x, x)N(f'; \mu(x), K(x, x))$$

The point with the highest expected improvement (the maximal expected utility) is selected. The expected improvement has two components. The first can be increased by reducing the mean function μ(x). The second can be increased by increasing the variance K(x; x). These two terms can be interpreted as explicitly encoding a tradeoff between exploitation (evaluating at points with low mean) and exploration (evaluating at points with high uncertainty). The exploitation–exploration tradeoff is a classic consideration in such problems, and the expected improvement criterion automatically captures both as a result of the Bayesian decision theoretic treatment.

Using acquisition function to travel the search space can be seen with the example presented in Fig 2.

C. Modelling Costs

The time it takes to get to minimum is as important as the number of evaluation steps. 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. Different regions of the parameter space may result in vastly different execution times, due to varying regularization, learning rates, etc. To improve our performance we optimize 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.
Just as we do not know the true objective function f(x), we also do not know the duration function $c(x): X \to R^{+}$. We can nevertheless employ our Gaussian process machinery to model ln(c(x)) alongside f(x). We assume that these functions are independent of each other. 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 x.

$$a_{EI}(x; \{\mathrm{x_n}, \mathrm{y_n}, \mathrm{l_n}\}, \theta, \varphi) = \frac{a_{EI}(x; \{\mathrm{x_n}, \mathrm{y_n}\}, \theta)}{\exp\,(x; \{\mathrm{x_n}, \mathrm{t_n}\}, \varphi)}$$

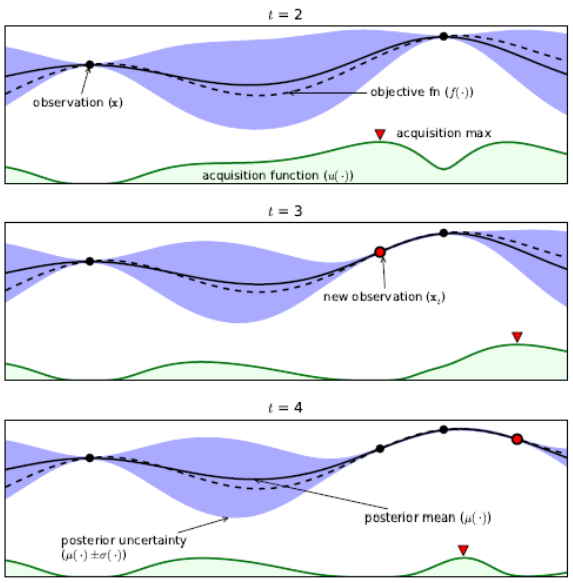

Fig.2: The figures show sampled values of the objective function. The figure also shows the acquisition function in the lower shaded plots. The acquisition is high where the GP predicts a high objective (exploitation) and where the prediction uncertainty is high (exploration): areas with both attributes are sampled first. Note that the area on the far left remains unsampled, as while it has high uncertainty, it is (correctly) predicted to offer little improvement over the highest observation.
Source: E. Brochu, V. M. Cora and N. D. Freitas, *"A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning"*

III. DEEP NETWORK FOR GLOBAL OPTIMIZATION

The basic issue with modelling functions with GPs is that GPs scale cubically with the number of observations as it involves the inversion of a dense covariance matrix; so it becomes challenging to handle objectives whose optimizations requires many evaluations.

This issue can be resolved by the use of neural networks as an alternative to model distribution over functions. Neural network can be used to get low dimensional representation of our input data while still retaining essential features. And then perform regression (called as adaptive basis function regression) with the neural network as the parametric form which results in a model that scales linearly with the number of observations. This approach is called as Deep

Networks for Global Optimization (DNGO) and it shares the advantages that GPs offer which are flexibility and characterization of uncertainty.

DNGO adds a Bayesian linear regressor to the last hidden layer of a deep neural network marginalizing only the output weights of the net while using a point estimate for remaining parameters of the neural network. This results in an adaptive basis regression.

A. Notations

- $\boldsymbol{\varphi}(.) = [\varphi_1(.), \dots\dots, \varphi_D(.)]^T$: Vector of outputs (basis functions) from last hidden layer of network trained on $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, $\mathbf{x}_n \in R^K$ and $y_n \in R$, where D is the size of the last hidden layer
- $\Phi_{nd} = \varphi_d(\mathbf{x}_n)$: output design matrix
- $\mathbf{Y}$**:** stacked target vector
- w_{MAP} : weights governing each layer of the neural network
- $w_{regressor}$: Weights associated with regressor
- $\frac{1}{\beta}$: variance of zero mean noise
- $\boldsymbol{\rho}(\mathbf{x})$ = mean function to capture our prior belief that the function is coarsely approximated by a convex quadratic function centered in the bounded region (i.e. a typical assumption that while searching the boundary of the search space, optimal point lies in the interior of input space)

B. Training

1. Weights and biases of neural network are trained via backpropagation and stochastic gradient descent to get a MAP estimate of all the parameters.

   Input: $\mathbf{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$

   Output: Φ_{nd} and w_{MAP}

2. After training, replace the MAP-parameterized output layer (i.e. the output layer and weights associated with it from w_{MAP}) with a Bayesian linear regressor that captures uncertainty in weights.

   $$\text{Output, } y = w_{regressor}^T * \boldsymbol{\varphi}(\mathbf{x}) + \text{noise}$$

   $$y \sim P(t \mid w_{regressor}, \beta, \mathbf{x}) = N(w_{regressor}^T * \boldsymbol{\varphi}(\mathbf{x}), \frac{1}{\beta})$$

   $$w_{regressor} \sim N(\mathbf{0}, \alpha^{-1}\mathbf{I})$$

   This gives predictive distribution over t as a normal distribution with:

   $$\text{Mean, } \mu(\mathbf{x}; \mathbf{D}, \boldsymbol{\theta}) = \mathbf{m}^T \boldsymbol{\varphi}(\mathbf{x}) + \rho(\mathbf{x})$$

$$\text{Variance, } \sigma^2(\mathbf{x}; \mathbf{D}, \boldsymbol{\theta}) = \boldsymbol{\varphi}(\mathbf{x})^T \mathbf{K}^{-1} \boldsymbol{\varphi}(\mathbf{x}) + \frac{1}{\beta}$$

Where,

$$\mathbf{m} = \beta \mathbf{K}^{-1} \boldsymbol{\Phi}^T (\mathbf{y} - \rho(\mathbf{x})) \in R^D$$

$$\mathbf{K} = \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi} + \alpha \mathbf{I} \in R^{D \times D}$$

IV. IMPLEMENTATION

We carried out the experiment of running Bayesian Optimization on Sparse Auto-encoder embedded Softmax classifier to tune its hyper-parameters. The complete model was run on publicly available MNIST digits data-set to predict labels of the digits. Both the training and test set have a size of 15298 samples with each input vector having size = 28*28. The digits had labels from 1-5.

Sparse Auto-encoder was used to get a low dimensional feature representation of the input data-set. It should be noted that any classifier other than Softmax like SVMs can be used. But for the sake of simplicity and not digress from our main objective of implementing Bayesian optimization, we stick to Softmax Classifier. The hyper-parameter tuning was done for following parameters of the classification module:

- Number of units in hidden layer
- Sparsity parameter which is the desired average activation of every hidden unit
- Weight decay parameter in the overall cost function
- Weight of Sparsity penalty term

These formed the input vector to our Bayesian Optimization module and the objective which we tried to minimize was the classification error. Publicly available package Spearmint was used to perform the optimization.

Stage	**Accuracy (%)**	**Hidden Layer Size**	**Sparsity Parameter**	**Weight decay parameter**	**Sparsity Penalty**
Initial	97.11	100	0.1	0.001	1
After 5 iterations	97.3657	206	0.33887	0.00819	3
After 10 iterations	97.7775	300	0.1	0.003	3
Hard-coded (publicly available)	97.68	200	0.3	0.0055	3

Table 1: Results with hyper parameter values

Since, number of hyper-parameters to be tuned were not large and number of iterations that model was run for small, execution of DNGO model and requirement of parallelizing Bayesian Optimization was not required.

A Matlab/Python wrapper is required to make calls from Spearmint to the model being optimized along with a configuration file with ranges of each hyper-parameter to be tuned, that is the boundaries of the space where Spearmint will search for parameter values. Thus any non-expert just needs to know the search space, the range where he/she believe the hyper-parameters could fall.

V. CONCLUSION

In this report, Bayesian Optimization for hyper parameter tuning was studied with two models: Gaussian Process model and the DNGO model. DNGO model scaled linearly with number of observations offering the great advantage of time saving over GPs. We explained why Bayesian optimization is needed and how it achieves the objective of getting approximate but good hyper-parameter values. We demonstrated the working by implementing Bayesian Optimization based on Gaussian Processes on a classification task. Here, the main objective was to demonstrate that how a non-expert can still work on any Machine Learning model with very little knowledge. Alongside, we also showed the model can gave an accuracy which in many cases beat the hard-coded expert values.

Although, it seems that Bayesian Optimization is a kind of 'one-for-all' solution to every Machine Learning tuning problem; but in some cases, it might be necessary to develop models specific to the problem at hand. Also, at some point in the hierarchy of Bayesian assumptions, user have to have some prior beliefs to incorporate in the model in the form of Meta-prior or Meta-meta prior. Nevertheless, Bayesian Optimization does help the user by providing good approximates to the parameters being tuned, greatly saving the time required to delve deeper in the cumbersome task of choosing the values.

VI. REFERENCES

[1] J. Snoek, H. Larochelle *"Practical Bayesian Optimization of Machine Learning Algorithms"*

[2] E. Brochu, V. M. Cora and N. D. Freitas, *"A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning"*

[3] Shengbo Guo, *"Bayesian Recommender Systems: Models and Algorithms"*

[4] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, Md. M. A. Patwary, Prabhat, R. P. Adams, *"Scalable Bayesian Optimization Using Deep Neural Networks"*

[5] https://github.com/JasperSnoek/spearmint

[6] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams and N. d. Freitas *"Taking the Human Out of the Loop:A Review of Bayesian Optimization"*

[7] https://www.youtube.com/watch?v=a79klpzaPgY