Stochastic Optimization and Machine Learning: Cross-Validation for Cross-Entropy Method

Anirban Chaudhuri
Massachusetts Institute of Technology
Cambridge, MA 02155, USA
anirbanc@mit.edu

David Wolpert
Santa Fe Institute
Santa Fe, NM 87501, USA
http://davidwolpert.weebly.com

Brendan Tracey
Santa Fe Institute
Santa Fe, NM 87501, USA
btracey@santafe.edu

Abstract

We explore using machine learning techniques to adaptively learn the optimal hyperparameters of a stochastic optimizer as it runs. Specifically, we investigate using multiple importance sampling to weight previously gathered samples of an objective function and combining with cross-validation to update the exploration / exploitation hyperparameter. We employ this on the Cross-Entropy method as it is finding the optimum of a function. Computer experiments show that this improves performance of the Cross-Entropy method beyond using any fixed value for that hyperparameter. The techniques outlined in this work are applicable to any optimization algorithms which operate on probability distributions.

1 Introduction

Suppose we want to find the minimum of an an objective function $G(x \in \mathcal{X})$ where $\mathcal{X} \subseteq \mathbb{R}^d$. Stochastic optimizers iteratively update a sampling distribution $q_\theta(x)$ at each stage t, and sample the time-t distribution to get the next set of x at which to sample $G(.)$. The goal is to update this $q_\theta(x)$ as the optimizer runs, based on the dataset of samples seen so far, to quickly concentrate on x that have low values of $G(x)$. Examples of stochastic optimizers include Genetic Algorithms [7], Simulated Annealing [3], estimation of distribution algorithms (EDA) [6] and probability collectives [14]. This work is applicable to all estimation of distribution [2, 5, 6, 8, 9, 10, 11] class of stochastic optimizers. The method uses importance sampling to exploit the fact that EDAs probabilistically generate their samples in order to do hyperparameter optimization without generating new samples.

Such optimizers typically control how $q_\theta(x)$ is generated from the dataset at each stage of the algorithm using hyperparameters that manage the exploration / exploitation tradeoff. Typically the hyperparameters remain fixed throughout the optimization process or are updated according to some pre-fixed schedule. In this work we explore the alternative of using machine learning (ML) techniques to regularly update the hyperparameters based on the then-current dataset. Specifically, we use cross-validation combined with multiple importance sampling [12] to resample the then-current data set to estimate how best to set the hyperparameters at each iteration.

We use numerical experiments to show that this use of cross-validation improves the performance of the Cross-Entropy (CE) method [1, 4, 10]. Interestingly, we find that there is a major improvement if we allow the cross-validation to occasionally choose a value of the hyperparameter which, used throughout the run, would result in horrible performance. We also consider a modification of the original CE algorithm where $q_\theta(x)$ is updated based on the entire current dataset (rather than just the most recent samples as in the original algorithm). This modification is based on replacing the simple-sampling Monte Carlo estimates in the original CE method with multiple importance sampling estimates [12].

The work was presented at the NIPS workshop on Optimizing the Optimizers. Barcelona, Spain, 2016.

2 The multiple importance sampling extension of the cross-entropy method

Let $\mathbf{X}$ be random realizations generated from q_θ, and define $\lambda^* = \min_{x \in \mathcal{X}} G(x)$. In the CE method, rather than directly search for a minimizer of $G(x)$, we search for the θ that maximizes $l(\lambda) = \mathbb{P}_{q_\theta}(G(\mathbf{X}) \leq \lambda) = \mathbb{E}_{q_\theta}[I_{\{G(\mathbf{X}) \leq \lambda\}}]$. By gradually shrinking λ to λ^*, the θ's that maximize $l(\lambda)$ should concentrate more and more tightly about the minimizer of $G(.)$. The challenge is that if we start with λ very close to λ^* then potentially $\{G(\mathbf{X}) \leq \lambda\}$ is a very rare event and estimating l becomes a non-trivial problem.

To address this challenge, the CE method starts with θ set to some θ_0. Then at each iteration $t \geq 0$, a set of samples is drawn from $q_{\theta_{t-1}}$, and λ_t is set to the worst value of the best κ percentile of these samples. (We call that subset of n_{e_t} samples the "elite samples", $\{x_{e_t}\}$.) θ_t is then set in order to minimize the KL divergence from q_{θ_t} to the normalized indicator distribution given by

$$p_{\lambda_t} \propto \Theta_{\lambda_t} := \begin{cases} 1, G(x) \leq \lambda_t; \\ 0, otherwise. \end{cases} \tag{1}$$

i.e., to find

$$\theta_t^* = \underset{\theta}{\operatorname{argmin}}\, KL(p_{\lambda_t} || q_\theta) = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{p_{\lambda_t}} \left[\ln \left(\frac{p_{\lambda_t}(x)}{q_\theta(x)} \right) \right] = \underset{\theta}{\operatorname{argmax}} \int_{x \in \mathcal{X}} p_{\lambda_t}(x) \ln(q_\theta(x)) dx. \tag{2}$$

We estimate θ_t^* by finding the θ_t that this way minimizes the importance sampling estimate of the integral in Eq. 2. (Note that we only need to consider the elite samples to do this since p_{λ_t} is zero for the other samples.) Since the normalization constant is not relevant to the solution of the optimization problem, we can use Θ_{λ_t} — whose value is 1 for the elite samples — in place of p_{λ_t} in the Monte Carlo estimation.

In the original CE method only the best κ percentile of the most recently generated samples are used to estimate $\theta^*(t)$, and those samples are all given a weight of $1/N_{e_t}$. Here we instead take the best κ percentile of all the samples generated so far, and use multiple importance sampling to combine them [12]. This is an unbiased estimator [12], just like conventional importance sampling, which often has far smaller variance. To construct the associated estimate for θ_t^*, first define

$$\bar{q}_t(x) = \sum_{j=1}^{k_t} \frac{n_t^{(j)}}{N_{e_t}} q_{e_t}^{(j)}(x), \tag{3}$$

where each "elite sampling distribution" $q_{e_t}^{(j)}$ was used to generate the $n_t^{(j)}$ elite samples in iteration t, such that $\sum_{j=1}^{k_t} n_t^{(j)} = N_{e_t}$. (Intuitively, $\bar{q}_t(x)$ represents a weighted combination of all the elite sampling distributions, where the weight is equal to the percentage of elite samples generated from each elite sampling distribution.) Our full estimate for θ_t^* is defined in terms of $q_{e_t}^{(j)}$:

$$\hat{\theta}_t^* = \underset{\theta}{\operatorname{argmax}} \sum_{j=1}^{N_{e_t}} \frac{\Theta_{\lambda_t}(x_{e_t}^{(j)})}{\bar{q}_t(x_{e_t}^{(j)})} \ln(q_\theta(x_{e_t}^{(j)})) = \underset{\theta}{\operatorname{argmax}} \sum_{j=1}^{N_{e_t}} \frac{1}{\bar{q}_t(x_{e_t}^{(j)})} \ln(q_\theta(x_{e_t}^{(j)})), \tag{4}$$

We also tried the naive approach of using only the particular sampling distribution from which the elite sample was generated (i.e., assigning a weight of 1 to the particular sampling distribution from which the elite sample was generated and 0 to the rest) but it yielded poor convergence results.

The solution of the optimization problem given by Equation 4 can be obtained in closed form if q belongs to a natural exponential family. In this work we assume q to be a Gaussian distribution. Then the convex optimization for hyperparameters θ, in this case the mean μ and standard deviation σ of the uncorrelated Gaussian distributions of the elite samples, can be solved by differentiating Equation 4 with respect to each parameter and setting them to zero to give

$$\mu_t^i = \frac{\sum_{j=1}^{N_{e_t}} \frac{x_{e_t}^{(j,i)}}{\bar{q}_t(x_{e_t}^{(j)})}}{\sum_{j=1}^{N_{e_t}} \frac{1}{\bar{q}_t(x_{e_t}^{(j)})}} \qquad \text{and} \qquad (\sigma_t^i)^2 = \frac{\sum_{j=1}^{N_{e_t}} \frac{(x_{e_t}^{(j,i)} - \mu_t^i)^2}{\bar{q}_t(x_{e_t}^{(j)})}}{\sum_{j=1}^{N_{e_t}} \frac{1}{\bar{q}_t(x_{e_t}^{(j)})}}, \tag{5}$$

where $i = 1, \ldots, d$ and $x^{(j,i)}$ refers to the i^{th} dimension of j^{th} sample.

Using this multiple importance sampling extension of the CE method improves performance over the original CE method for certain settings of the hyperparameter. But more importantly for current purposes, by reducing the variance, it "stabilizes" our use of cross-validation to estimate the optimal κ (described below), which proved crucial to having the cross-validation result in good performance.

3 Cross-validation for cross-entropy method

κ_t is the hyperparameter that specifies how to map the $t-1$ dataset to θ_t. In the original CE method, κ is pre-fixed to a single value that is used throughout the optimization run. If that κ is too large, it will slow down the convergence to an optimum, and if it is too small then the algorithm will either suffer premature convergence to a local optimum or not converge at all.

In ML, hyperparameters of an algorithm A are often optimized through cross-validation. This starts by many times forming a partition of the available data into a "held-in" dataset and a "held-out" dataset. For each such partition, A is trained on the held-in dataset to set parameters $\hat{\theta}^*$, which are then used to evaluate performance on the held-out dataset. That performance is then averaged over all partitions to get an overall estimate for the performance of A. Different values of the hyperparameter in A will result in different values for this average (estimated) held-out performance. Accordingly, we can set the hyperparameter to whatever value optimizes the associated average held-out performance, and then use that value to train A on the entire dataset.

Cross-validation can also be used with any stochastic optimizer that has a hyperparameter to estimate the best value of that hyperparameter, due to a formal identity equating stochastic optimization and supervised machine learning [13]. Here we exploit this and use cross-validation to pick the best κ_t for iteration t in the CE method. In order to measure performance on the held-out datasets, we use $\mathbb{E}_{q_\theta}[G(x)]$. Concretely, we use several values of κ_t in the CE method with held-in datasets to produce associated estimates for the optimal θ, $\hat{\theta}^*_{train}$, and then evaluate performance on the associated held-out dataset, as given by

$$\sum_{j=1}^{n_{test}} \frac{q_{\hat{\theta}^*_{train}}(x^{(j)}_{test})}{\bar{q}_t(x^{(j)}_{test})} G(x^{(j)}_{test}), \tag{6}$$

where n_{test} is the number of test points. In this work we used k-fold cross-validation, where data is divided equally into k partitions, and $k-1$ of these are used for training and the remaining one for testing. Crucially, since the CE method is an estimation of distribution algorithm, multiple importance sampling can be used to reuse the existing samples and no new samples of G are required during this use of cross-validation. (Note that we use the combined sample distribution described in Section 2 for the multiple importance sampling process.) The final choice for κ_t produced by the cross-validation is

$$\kappa^*_t = \underset{\kappa_t}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{j=1}^{n_{test}} \frac{q^i_{\hat{\theta}^*_{train}}(x^{(j)}_{test_i})}{\bar{q}_t(x^{(j)}_{test_i})} G(x^{(j)}_{test_i}) \tag{7}$$

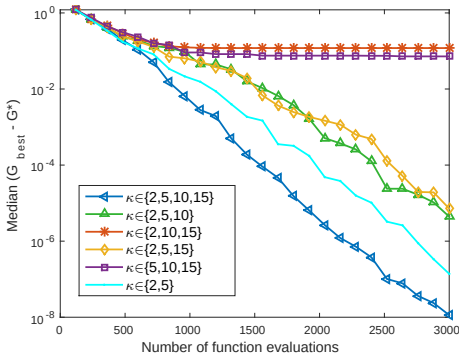
We refer to this algorithm where the hyperparameter of the (extended) CE method is adaptively set through cross-validation as XVCE.

4 Results

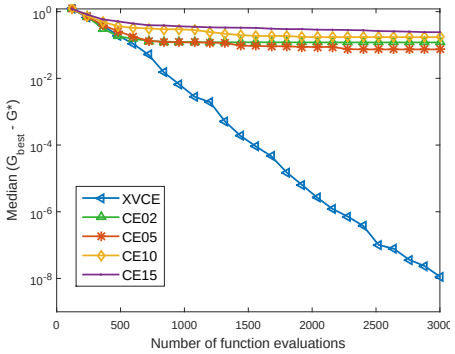
We looked at elite samples in the range of 2-15% of the entire available history at each iteration. For the original CE method implementation we ran 4 values of κ at 2, 5, 10 and 15%. For the XVCE implementation, the best κ_t for each iteration t is picked from the same four options ($\kappa_t \in \{2\%, 5\%, 10\%, 15\%\}$) using cross-validation. In the plots, CExx represents the original CE algorithm with xx representing value of κ. For each of the tested algorithms, we used a single Gaussian as the sampling distribution. The bounds on design variables are implemented by using truncated Gaussian distributions. 5-fold cross-validation is used in all the cases. We compare the performance of each algorithm by repeating for 100 trials with randomly picked initial parameters, θ_0, to get the performance statistics. For a given trial in each test problem, the same set of initial parameters for the Gaussian distribution and same initial population were used across all the CE algorithms. The metric used to analyze the performance of the algorithms is the distance of the median best solution to the known global optimum: semilog plot with $G_{best} - G^*$ as a median of 100 repetitions on a log scale against the number of function evaluations. G_{best} refers to the best solution obtained by the algorithm after certain number of function evaluations and G^* is the known true optimum of the analytic test problem.

Figure 1a shows the performance of the XVCE algorithm when the choice of κ is made from subsets of available values of $\{2\%, 5\%, 10\%, 15\%\}$. Even when only a subset of the possible κ values were

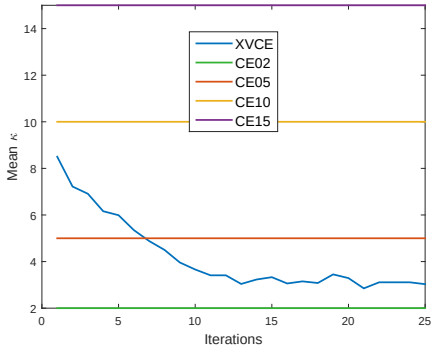
provided to the cross-validation, XVCE still performs as well or outperforms the best CE algorithm for any single one of those κ's, as shown in Figure 1b. Indeed, median performance of the XVCE algorithm is ~ 7 orders of magnitude better than any of the CE algorithms run with a fixed κ. In addition, as seen in Figure 1a, providing all four options for κ to the cross-validation algorithm results in better performance than when the choice was restricted to a subset of those four values. This is true even when one of the particularly poorly performing values of κ is removed from the set of possible values. Figure 1c illustrates how XVCE changes κ as the optimization progresses to match the changing need to trade off exploration and exploitation.



(a) Different sets of choices for κ in XVCE



(b) Comparison of XVCE to CE



(c) Dynamically changing κ

Figure 1: Algorithm performance comparison for Hartmann 6 function.

5 Discussion

We demonstrate how to use ML techniques to dynamically update hyperparameters of a stochastic optimization algorithm dynamically as the optimization progresses. Specifically, we show how to use cross-validation to update the exploration / exploitation parameter of the CE method, instead of setting it with a fixed *ad hoc* heuristic. In addition to the conventional single Gaussian version of the CE method investigated here, Gaussian mixture models could also be used, in which case the number of mixture components becomes another hyperparameter that could be dynamically updated using cross-validation. (The parameters of a mixture distribution in the CE method are typically set via EM.) Further improvements should also be possible by exploiting other ML techniques like bagging and regularization. In general, the techniques outlined in this work are applicable to any estimation of distribution algorithms for global optimization and this the focus of ongoing research. Preliminary experiments on applying cross-validation to Univariate Marginal Distribution Algorithm (UMDA) [5] confirm its effectiveness.

Acknowledgement

This work was supported in part by the AFOSR MURI on managing multiple information sources of multi-physics systems, Program Manager Jean-Luc Cambier, Award Number FA9550-15-1-0038. DHW also acknowledges the support of the Santa Fe Institute.

References

[1] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[2] J. S. De Bonet, C. L. Isbell, P. Viola, et al. Mimic: Finding optima by estimating probability densities. *Advances in neural information processing systems*, pages 424–430, 1997.

[3] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[4] D. P. Kroese, S. Porotsky, and R. Y. Rubinstein. The cross-entropy method for continuous multi-extremal optimization. *Methodology and Computing in Applied Probability*, 8(3):383–407, 2006.

[5] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization in continuous domains by learning and simulation of gaussian networks. In *Conference on Genetic and Evolutionary Computation (GECCO'00) Workshop Program*. Morgan Kaufmann, 2000.

[6] J. A. Lozano. *Towards a new evolutionary computation: advances on estimation of distribution algorithms*, volume 192. Springer Science & Business Media, 2006.

[7] M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[8] M. Pelikan, D. E. Goldberg, and F. G. Lobo. A survey of optimization by building and using probabilistic models. *Computational optimization and applications*, 21(1):5–20, 2002.

[9] M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In *Advances in Soft Computing*, pages 521–535. Springer, 1999.

[10] R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[11] M. Sebag and A. Ducoulombier. Extending population-based incremental learning to continuous search spaces. In *International Conference on Parallel Problem Solving from Nature*, pages 418–427. Springer, 1998.

[12] E. Veach and L. J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 419–428. ACM, 1995.

[13] D. Wolpert and D. Rajnarayan. Using machine learning to improve stochastic optimization. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[14] D. H. Wolpert, C. E. Strauss, and D. Rajnarayan. Advances in distributed optimization using probability collectives. *Advances in Complex Systems*, 9(4):383–436, 2006.