Late-Breaking Developments in the Field of Artificial Intelligence
Papers Presented at the Twenty-Seventh AAAI Conference on Artificial Intelligence

Using Machine Learning to Improve Stochastic Optimization

David H. Wolpert
Information Sciences Group
Los Alamos National Laboratory
Los Alamos, NM 87545

Dev Rajnarayan
Sensor Platforms, Inc.
2860 Zanker Road Ste. 210
San Jose CA 95134

Abstract

In many stochastic optimization algorithms there is a hyperparameter that controls how the next sampling distribution is determined from the current data set of samples of the objective function. This hyperparameter controls the exploration/exploitation trade-off of the next sample. Typically heuristic "rules of thumb" are used to set that hyperparameter, e.g., a pre-fixed annealing schedule. We show how machine learning provides more principled alternatives to (adaptively) set that hyperparameter, and demonstrate that these alternatives can substantially improve optimization performance.

1 Approach

Let $G : X \rightarrow \mathbb{R}$ be an objective function. We use the term **Parametric Stochastic Optimization** (PSO) to refer to a class of algorithms that search for the minimizer of G by iteratively stochastically sampling G. In PSO, at each step t the set of all earlier samples of G, written as $d^t \equiv \{[x^1, G(x^1)], \ldots [x^t, G(x^t)]\}$, is used to generate a value θ_t parameterizing a distribution $q_{\theta_t}(x)$. $q_{\theta_t}(x)$ is then sampled to generate x^{t+1}. Then the resultant pair $(x^{t+1}, G(x^{t+1}))$ is added to d^t to give d^{t+1}, and the process repeats.

Examples of PSO include PBIL (Li, Kwong, and Hong 2011), PBIL2 (Sebag and Ducoulombier 1998), Monte Carlo Optimization (MCO (Ermoliev and Norkin 1998; Robert and Casella 2004))), EDA's (Lozano et al.) and the related techniques of MIMIC (de Bonet, Isbell, and Viola 1997), Cross Entropy method (CE (Rubinstein and Kroese 2004; Rubinstein 2001; Rubenstein 2005; Kroese, Porotsky, and Rubinstein 2006a)), and delayed sampling Probability Collectives (PC (Wolpert, Strauss, and Rajnarayan 2006)).

Other algorithms not normally viewed as PSO can be re-expressed as PSO algorithms. For example, many Genetic Algorithms (GA's (Mitchell 1996)) can be cast this way, by identifying θ_t with $\{[x^{t-N}, G(x^{t-N})], \ldots, [x^t, G(x^t)]\}$, the time t population of N individuals and associated G values, which is used to stochastically generate the time $t + N$ population.

The central issue in PSO algorithms is how to map the current data set d^t to the next sampling distribution $q_{\theta_{t+1}}$. Often this map can be cast as a function $\Gamma_{\gamma_t} : d_t \rightarrow \theta_{t+1}$ governed by a hyperparameter γ_t that can vary in time. For example, in a GA, γ_t would be mutation rates, crossover rates, etc., all of which in general are varied in time. As another example, in the CE method applied to Euclidean-valued x, q_θ can be a mixture of Gaussians with parameter vector θ. That θ would be updated to minimize a cross-entropy based on the dataset: $\Gamma_{\gamma_t}(d^t) \equiv \text{argmin}_\theta[\sum_{t' \leq t} I(G(x^{t'}) \leq \gamma_t) \ln(q_\theta(x^{t'}))]$, where $I(.)$ equals 1/0 depending on whether its argument is true/false. (More precisely, θ_{t+1} is given by running an algorithm like EM to approximate the minimizer of the sum.)

In general γ_t determines the "width" of $q_{\theta_{t+1}}$, and so controls the exploration / exploitation trade-off in the generation of the sample x^{t+1}. So a major factor in how well a PSO algorithm performs is how the hyperparameter γ_t is updated from one step t to the next. Often this is done using a heuristic "rule of thumb". For example, in the CE method, γ_t is typically set to the G value of the pair in d^t whose G value is the κ'th percentile of those pairs in d^t. So κ is like an annealing rate. (Strictly speaking, κ is the hyperparameter, since ultimately it is what specifies how d^t gets mapped to θ_{t+1}.) In the conventional CE method, κ is simply pre-fixed to an arbitrary value that is used throughout the optimization.

How to set hyperparameters based on a current dataset is also a major issue in machine learning (ML), which has developed many techniques for doing it that far outperform heuristics. Many of these techniques can be "translated" to apply to PSO, thereby providing more principled alternatives to the heuristics conventionally used in PSO.

As an example, cross-validation for PSO proceeds as follows: At each t, multiple times partition d^t into a d^t_{in} and a d^t_{out} (each such partition is called a "fold"); For each candidate value of γ_t, run $\Gamma_{\gamma_t}(d^t_{in})$ to produce a value $q_{\theta_{t;\gamma_t}}$; Use d^t_{out} to estimate the performance we would get if we formed x^{t+1} by sampling $q_{\theta_{t;\gamma_t}}$; Choose that candidate γ_t which results in the best average of these estimated performances over all folds; Run $\Gamma_{\gamma_t}(d^t)$ with that chosen γ_t to construct θ^{t+1}.

With each pair $(x^i, G(x^i))$ in the data set, we have the associated (value of θ specifying the) distribution that generated x^i. So we can use importance sampling (Robert and Casella 2004) to combine the G pairs in each data set d^t_{out}. This is how we "estimate performance we would get ... by sampling $q_{\theta_{t;\gamma_t}}$". For example, in a greedy approach, the performance measure for γ_t is $\mathbb{E}_{q_{\theta_{t;\gamma_t}}}(G) = \int dx\, G(x) q_{\theta_{t;\gamma_t}}(x)$. We estimate this integral as $\sum_j q_{\theta_{t;in}}(x^j_{out}) G(x^j_{out}) \;/\; h_j(x^j_{out})$, where $h_j(.)$

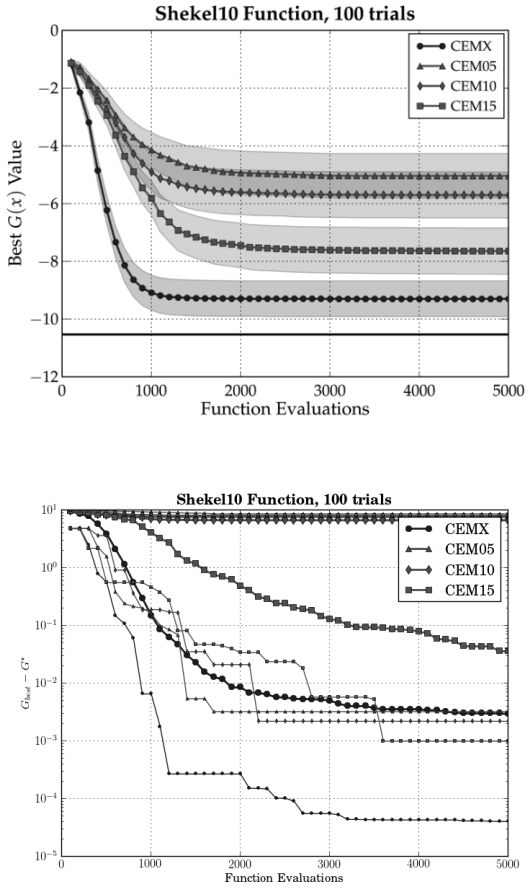

Figure 1: The top figure plots mean performance and error in the mean against dataset size, using 100 trials. The bottom figure plots median performance (large symbols) against dataset size along with best and worst performances (small performances) out of all 100 trials. (Only best performances are visible.) Graphs whose name ends in a number are for the conventional CE method (that number is the value of κ). The CEMX graphs are for cross-validation.

is defined as the distribution $q_\theta(.)$ that was sampled at the earlier stage of the PSO algorithm to generate x^j.

No new samples of G are required in this use of cross-validation; it is not simply a re-running of the entire PSO algorithm. Note also that γ_t is updated dynamically, in response to new data. So these updates will vary from one run of the PSO algorithm to the next. This contrasts with many conventional approaches where γ_t is updated by a pre-fixed, "one size fits all" schedule. Note as well that in our procedure all old data is used, in a principled fashion, weighted by the sampling distributions h_j. There is no need for *ad hoc* data-aging to weight more recent data more heavily.

2 Experiments

We used a set of nine objective functions defined over Euclidean x's to compare the performance of the CE method when cross-validation is used to set its hyperparameter κ to performance of the CE method when the conventional CE heuristic (described above) is used instead. The Euclidean spaces range from 4 to 8 dimensional, and each has properties that make it difficult for local optimizers to find the global optima, e.g., multiple local minima with the worst minima having the largest basin of attraction. (A few of these problems have previously been used to test the CE method (Kroese, Porotsky, and Rubinstein 2006b).)

In all our experiments $\kappa \in \{5, 10, 15\}$. We tested performance both when q_θ is a single Gaussian and when it is a mixture of Gaussians. For the latter case we used cross-validation to (adaptively) set the number of mixing components (1, 2, or 3) as well as set κ. (This can be thought of as using to determine the number of "species" in a population dynamically, to speed up the overall optimization.) As is conventional, when measuring performance we treat the cost of deciding where next to sample as negligible compared to the cost of evaluating that sample.

As described in detail in (Rajnarayan and Wolpert 2008), our results show that adaptively changing κ and/or the number of mixing components, to dynamically control the exploration / exploitation tradeoff, can substantially improve optimization performance. In particular setting those hyperparameters via cross-validation can work well. Concretely, in all our experiments the variant of the CE method that uses cross-validation to set κ resulted in at least as good values of the objective function as the best performing (over all fixed values of κ and numbers of mixing components) instances of the conventional CE method, over the entire range of up to 10000 function evaluations. In many cases there was a (very) statistically significant gain in performance compared to the best alternative. As an illustration, in Fig. 1 we present results for the Shekel10 (4-dimensional) objective function for the experiments where q_θ was a mixture of Gaussians.

3 Extensions

We have argued that for general PSO algorithms, the ML technique of setting hyperparameters via subsampling the data set can substantially improve performance, and demonstrated this explicitly for the CE method. However for the *particular* PSO algorithm of MCO, the connection with ML goes far deeper; in the associated paper (Wolpert, Rajnarayan, and Bieniawski 2013), we show that MCO and ML are in fact formally identical. This identity is extremely powerful. By exploiting it we can translate *all* of the techniques that have demonstrated great power in the domain of ML to apply directly to stochastic optimization.

Preliminary experiments have confirmed that exploiting ML this way substantially improves MCO performance (Wolpert, Rajnarayan, and Bieniawski 2013). In particular, we found that in addition to the use of subsampling to set hyperparameters, the ML techniques of adding a regularizer to the objective function, bagging (Breiman 1996b), and stacking (Breiman 1996a; Sill et al. 2009; Clarke 2003; Wolpert 1992) all improve optimization performance, sometimes dramatically. Future work involves exploiting this formal identity further, by investigating the use of other ML techniques to improve stochastic optimization.

References

Breiman, L. 1996a. Stacked regression. *Machine Learning* 24(1):49–64.

Breiman, L. 1996b. Bagging predictors. *Machine Learning* 24(2):123–140.

Clarke, B. 2003. Bayes model averaging and stacking when model approximation error cannot be ignored. *Journal of Machine Learning Research* 683–712.

de Bonet, J. S.; Isbell, Jr., C. L.; and Viola, P. 1997. MIMIC: Finding optima by estimating probability densities. In Mozer, M. C.; Jordan, M. I.; and Petsche, T., eds., *Advances in Neural Information Processing Systems*, volume 9, 424. The MIT Press.

Ermoliev, Y. M., and Norkin, V. I. 1998. Monte carlo optimization and path dependent nonstationary laws of large numbers. Technical Report IR-98-009, International Institute for Applied Systems Analysis.

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

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

Li, H.; Kwong, S.; and Hong, Y. 2011. The convergence analysis and specification of the population-based incremental learning algorithm. *Neurocomputing*.

Lozano, J.; Larraaga, P.; Inza, I.; and Bengoetxa, E. *Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithms*. Springer Verlag.

Mitchell, M. 1996. *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.

Rajnarayan, D., and Wolpert, D. 2008. Bias-variance techniques for monte carlo optimization: Cross-validation for the ce method. *arXiv preprint arXiv:0810.0877*.

Robert, C. P., and Casella, G. 2004. *Monte Carlo Statistical Methods*. New York: Springer-Verlag.

Rubenstein, R. 2005. The stochastic minimum cross-entropy method for combinatorial optimization and rare-event estimation. unpublished.

Rubinstein, R., and Kroese, D. 2004. *The Cross-Entropy Method*. Springer.

Rubinstein, R. Y. 2001. Combinatorial optimization via cross-entropy. In Gass, S., and Harris, C., eds., *Encyclopedia of Operations Research and Management Sciences*. Kluwer.

Sebag, M., and Ducoulombier, A. 1998. Extending population-based incremental learning to continuous search spaces. *Lecture Notes in Computer Science* 1498:418–??

Sill, J.; Takacs, G.; L., M.; and D., L. 2009. Feature-weighted linear stacking. unpublished: arXiv:0911.0460.

Wolpert, D. H.; Rajnarayan, D.; and Bieniawski, S. 2013. Probability collectives in optimization. In *Handbook of Stastistics*. Cambridge University Press.

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

Wolpert, D. H. 1992. Stacked generalization. *Neural networks* 5(2):241–259.