CMA-ES FOR HYPERPARAMETER OPTIMIZATION OF DEEP NEURAL NETWORKS

Ilya Loshchilov & Frank Hutter
Univesity of Freiburg
Freiburg, Germany,
{ilya,fh}@cs.uni-freiburg.de

ABSTRACT

Hyperparameters of deep neural networks are often optimized by grid search, random search or Bayesian optimization. As an alternative, we propose to use the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which is known for its state-of-the-art performance in derivative-free optimization. CMA-ES has some useful invariance properties and is friendly to parallel evaluations of solutions. We provide a toy usage example using CMA-ES to tune hyperparameters of a convolutional neural network for the MNIST dataset on 30 GPUs in parallel.

Hyperparameters of deep neural networks (DNNs) are often optimized by grid search, random search (Bergstra & Bengio, 2012) or Bayesian optimization (Snoek et al., 2012a; 2015), with the latter known as the most effective method. For the optimization of continuous hyperparameters, Bayesian optimization based on Gaussian processes (Rasmussen & Williams, 2006) is known as the most effective method. While for joint structure search and hyperparameter optimization, tree-based Bayesian optimization optimization methods (Hutter et al., 2011; Bergstra et al., 2011) are known to perform better (Bergstra et al.; Eggensperger et al., 2013; Domhan et al., 2015), here we focus on continuous optimization. We note that integer parameters with rather wide ranges (e.g., number of filters) can, in practice, be considered to behave like continuous hyperparameters.

As the evaluation of a DNN hyperparameter setting requires fitting a model and evaluating its performance on validation data, this process can be very expensive, which often renders sequential hyperparameter optimization on a single computing unit infeasible. Unfortunately, Bayesian optimization is sequential by nature: while a certain level of parallelization is easy to achieve by conditioning decisions on expectations over multiple hallucinated performance values for currently running hyperparameter evaluations (Snoek et al., 2012a) or by evaluating the optima of multiple acquisition functions concurrently (Hutter et al., 2012), perfect parallelization appears unattainable since the decisions in each step depend on all data points gathered so far. Here, we study the use of a different type of derivative-free continuous optimization method that allows for perfect parallelization.

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES (Hansen & Ostermeier, 2001)) is a state-of-the-art optimizer for continuous black-box functions. While Bayesian optimization methods often perform best for small function evaluation budgets (e.g., below 10 times the number of hyperparameters being optimized), CMA-ES tends to perform best for larger function evaluation budgets; for example, Loshchilov et al. (2013) showed that CMA-ES performed best among more than 100 classic and modern optimizers on a wide range of blackbox functions. In a nutshell, CMA-ES is an iterative algorithm, that, in each of its iterations, samples λ candidate solutions from a multivariate normal distribution, evaluates these and then adjusts the sampling distribution used for the next iteration to give higher probability to good samples. Usual values for the so-called population size λ are around 10 to 20; in the study we report here, we used a larger size $\lambda = 30$ to take full benefit of 30 GeForce GTX TITAN Black GPUs we had available. Larger values of λ are also known to be helpful for noisy and multi-modal problems. Since all variables are scaled to be in [0,1], we set the initial sampling distribution to $\mathcal{N}(0.5, 0.2^2)$. We didn't try to employ any noise reduction techniques (Hansen et al., 2009) or surrogate models (Loshchilov et al., 2012).

In the study we report here, we used AdaDelta (Zeiler, 2012) and Adam (Kingma & Ba, 2014) to train DNNs on the MNIST dataset (50k original training and 10k original validation examples). The 19 hyperparameters describing the network structure and the learning algorithms are given in

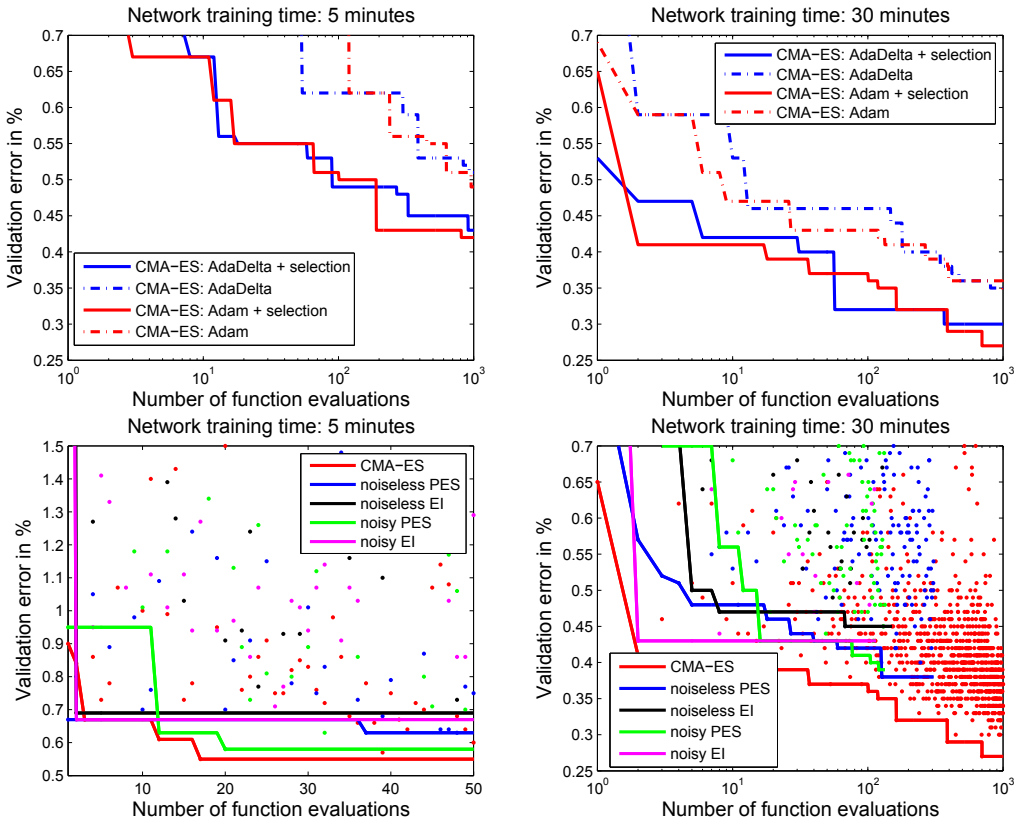

Figure 1: **Top**: Best validation errors found for AdaDelta and Adam with and without batch selection when hyperparameters are optimized by CMA-ES with training time budgets of 5 and 30 minutes. **Bottom**: Validation errors for Adam with batch selection when solutions are evaluated **Bottom-Left**: sequentially for 5 minutes each; **Bottom-Right** in parallel for 30 minutes each.

Table 1; the code is also available at `https://sites.google.com/site/cmaesfordnn/` (anonymous for the reviewers). We considered both the default (shuffling) and online loss-based batch selection of training examples (Loshchilov & Hutter, 2015). The objective function is the smallest validation error found in all epochs when the training time (including the time spent on model building) is limited.

The baseline we compare CMA-ES to is GP-based Bayesian optimization, as implemented by the widely known Spearmint system (Snoek et al., 2012a) (available at `https://github.com/HIPS/Spearmint`). In particular, we compared to Bayesian optimization with two different acquisition functions: (i) Expected Improvement (EI), as described by Snoek et al. (2012b) and implemented in the main branch of Spearmint; and (ii) Predictive Entropy Search (PES), as described by Hernández-Lobato et al. (2014) and implemented in a sub-branch of Spearmint (available at `https://github.com/HIPS/Spearmint/tree/PESC`). Experiments by Hernández-Lobato et al. (2014) demonstrated that PES is superior to EI; our own (unpublished) preliminary experiments on the black-box benchmarks used for the evaluation of CMA-ES by Loshchilov et al. (2013) also confirmed this. Both EI and PES have an option to notify the method about whether the problem at hand is noisy or noiseless. To avoid a poor choice on our side, we ran both algorithms in both regimes. Similarly to CMA-ES, to benefit from parallel evaluations in EI&PES, we set the maximum number of concurrent jobs in Spearmint to 30.

Figure 1 (**top**) shows the results of running CMA-ES on 30 GPUs on eight different hyperparameter optimization problems: all combinations of using (1) AdaDelta (Zeiler, 2012) or Adam (Kingma & Ba, 2014); (2) standard shuffling batch selection or batch selection based on the latest known loss (Loshchilov & Hutter, 2015); and (3) allowing 5 minutes or 30 minutes of network training time. We

note that in all cases CMA-ES steadily improved the best validation error over time and in the best case yielded validation errors below 0.3% in a network trained for only 30 minutes (and 0.42% for a network trained for only 5 minutes). We also note that batch selection based on the latest known loss performed better than shuffling batch selection and that the results of AdaDelta and Adam were almost indistinguishable. Therefore, the rest of the figure shows only the case of Adam with batch selection based on the latest known loss.

Figure 1 (**bottom**) compares the results of CMA-ES vs. Bayesian optimization with EI&PES. In this figure, to illustrate the actual function evaluations, each evaluation within the range of the y-axis is depicted by a dot. Figure 1 (**bottom left**) shows the results of all tested algorithms when solutions are evaluated sequentially with a relatively small network training time of 5 minutes each. Note that we use CMA-ES with $\lambda = 30$ and thus the first 30 solutions are sampled from the prior isotropic (not yet adapted) Gaussian with a mean of 0.5 and standard deviation of 0.2. Apparently, the results of this sampling are as good as the ones produced by EI&PES. This might be because of a bias towards the middle of the range, or because EI&PES do not work well on this noisy high-dimensional problem, or because of both. Quite in line with the conclusion of Bergstra & Bengio (2012), it seems that the presence of noise and rather wide search ranges of hyperparameters make sequential optimization *with small budgets* rather inefficient, i.e., as efficient as random sampling. One way to combat this would be to support prior distributions over good parameter ranges in GP-based Bayesian optimization, but to date no system implements this.

Figure 1 (**bottom right**) shows the results of all tested algorithms when solutions are evaluated *in parallel* on 30 GPUs. Each DNN now trains for 30 minutes, meaning that, for each optimizer, running this experiment sequentially would take 30 000 minutes (or close to 21 days) on one GPU; in parallel on 30 GPUs, it only required 17 hours. Compared to the sequential 5-minute setting, the greater budget of the parallel setting allowed CMA-ES to improve results such that most of its latest solutions had validation error below 0.4%. The internal cost of CMA-ES was virtually zero, but it was a substantial factor for EI&PES due to the cubic complexity of standard GP-based Bayesian optimization: after having evaluated 100 configurations, it took roughly 30 minutes to generate 30 new configurations to evaluate, and as a consequence 300 evaluations by EI&PES took more wall-clock time than 1000 evaluations by CMA-ES. This problem could be addressed by using approximate GPs Rasmussen & Williams (2006) or another efficient multi-core implementation of Bayesian Optimization, such as the one by Snoek et al. (2015). However, the performance of EI&PES in terms of the validation error was also inferior to the one of CMA-ES. One reason might be that this benchmark was too noisy and high-dimensional for EI&PES.

In conclusion, we propose to consider CMA-ES as one alternative in the mix of methods for hyperparameter optimization of DNNs. It is powerful, computationally cheap and natively supports parallel evaluations. Our preliminary results suggest that CMA-ES can be competitive especially in the regime of parallel evaluations. However, we still need to carry out a much broader and more detailed comparison, involving more test problems and the tree-based Bayesian optimization algorithms TPE (Bergstra et al., 2011) and SMAC (Hutter et al., 2011).

REFERENCES

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *JMLR*, 13:281–305, 2012.

Bergstra, J., Yamins, D., and Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. pp. 115–123.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Proc. of NIPS'11*, pp. 2546–2554, 2011.

Domhan, Tobias, Springenberg, Jost Tobias, and Hutter, Frank. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proc. of IJCAI'15*, pp. 3460–3468, 2015.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., and Leyton-Brown, K. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *Proc. of BayesOpt'13*, 2013.

Hansen, Nikolaus and Ostermeier, Andreas. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Table 1: Hyperparameters descriptions, pseudocode transformations and ranges

name	description	transformation	range
x_1	selection pressure at e_0	$10^{-2+10^{2x_1}}$	$[10^{-2}, 10^{98}]$
x_2	selection pressure at e_{end}	$10^{-2+10^{2x_2}}$	$[10^{-2}, 10^{98}]$
x_3	batch size at e_0	2^{4+4x_3}	$[2^4, 2^8]$
x_4	batch size at e_{end}	2^{4+4x_4}	$[2^4, 2^8]$
x_5	frequency of loss recomputation r_{freq}	$2x_5$	$[0, 2]$
x_6	alpha for batch normalization	$0.01 + 0.2x_6$	$[0.01, 0.21]$
x_7	epsilon for batch normalization	10^{-8+5x_7}	$[10^{-8}, 10^{-3}]$
x_8	dropout rate after the first Max-Pooling layer	$0.8x_8$	$[0, 0.8]$
x_9	dropout rate after the second Max-Pooling layer	$0.8x_9$	$[0, 0.8]$
x_{10}	dropout rate before the output layer	$0.8x_{10}$	$[0, 0.8]$
x_{11}	number of filters in the first convolution layer	$2^{3+5x_{11}}$	$[2^3, 2^8]$
x_{12}	number of filters in the second convolution layer	$2^{3+5x_{12}}$	$[2^3, 2^8]$
x_{13}	number of units in the fully-connected layer	$2^{4+5x_{13}}$	$[2^4, 2^9]$
x_{14}	Adadelta: learning rate at e_0	$10^{0.5-2x_{14}}$	$[10^{-1.5}, 10^{0.5}]$
x_{15}	Adadelta: learning rate at e_{end}	$10^{0.5-2x_{15}}$	$[10^{-1.5}, 10^{0.5}]$
x_{16}	Adadelta: ρ	$0.8 + 0.199x_{16}$	$[0.8, 0.999]$
x_{17}	Adadelta: ϵ	$10^{-3-6x_{17}}$	$[10^{-9}, 10^{-3}]$
x_{14}	Adam: learning rate at e_0	$10^{-1-3x_{14}}$	$[10^{-4}, 10^{-1}]$
x_{15}	Adam: learning rate at e_{end}	$10^{-3-3x_{15}}$	$[10^{-6}, 10^{-3}]$
x_{16}	Adam: β_1	$0.8 + 0.199x_{16}$	$[0.8, 0.999]$
x_{17}	Adam: ϵ	$10^{-3-6x_{17}}$	$[10^{-9}, 10^{-3}]$
x_{18}	Adam: β_2	$1 - 10^{-2-2x_{18}}$	$[0.99, 0.9999]$
x_{19}	adaptation end epoch index e_{end}	$20 + 200x_{19}$	$[20, 220]$

Hansen, Nikolaus, Niederberger, André SP, Guzzella, Lino, and Koumoutsakos, Petros. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *Evolutionary Computation, IEEE Transactions on*, 13(1):180–197, 2009.

Hernández-Lobato, José Miguel, Hoffman, Matthew W, and Ghahramani, Zoubin. Predictive entropy search for efficient global optimization of black-box functions. In *Proc. of NIPS'14*, pp. 918–926, 2014.

Hutter, F., Hoos, H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION'11*, pp. 507–523, 2011.

Hutter, F., Hoos, H., and Leyton-Brown, K. Parallel algorithm configuration. In *Proc. of LION'12*, pp. 55–70, 2012.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Loshchilov, Ilya and Hutter, Frank. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*, 2015.

Loshchilov, Ilya, Schoenauer, Marc, and Sebag, Michele. Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In *Proc. of GECCO'12*, pp. 321–328. ACM, 2012.

Loshchilov, Ilya, Schoenauer, Marc, and Sebag, Michèle. Bi-population cma-es agorithms with surrogate models and line searches. In *Proc. of GECCO'13*, pp. 1177–1184. ACM, 2013.

Rasmussen, C. and Williams, C. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Proc. of NIPS'12*, pp. 2960–2968, 2012a.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012b.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Md, Ali, Mostofa, Adams, Ryan P, et al. Scalable bayesian optimization using deep neural networks. *arXiv preprint arXiv:1502.05700*, 2015.

Zeiler, Matthew D. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.