Internship with the Argonne National Laboratory : a collaboration with DeepHyper

Joceran Gouneau

Prasanna Balaprakash, Romain Egelé (Supervisors)

***Abstract*—Bayesian optimization (BO) is a widely used approach for computationally expensive black-box optimization such as hyperparameter optimization (HPO) [1] [2] [3] [4] of deep learning methods. To this end, DeepHyper [5] [6] is a scalable, open-source software package for automated machine/deep learning (ML/DL) ; it enables scientists to rapidly develop ML/DL models using BO methods on leadership-class systems such as Theta. In order to improve the performances of our HPO framework, we propose a new method : Asynchronous Distributed Bayesian Optimization, and apply it to the case of plasma disruptions prediction in a tokamak fusion reactor. We show a clear improvement of scalability on large High Performance Calculus (HPC) systems with our new method and better predictions on the fusion problem with hyperparameters found by this algorithm. This work enables DeepHyper to take advantage of the computation power of even bigger systems and demonstrates its usefulness in a concrete use-case.**

***Index Terms*—Automated Machine Learning, Machine Learning, Deep Learning, Hyperparameter Optimization, Black-box Optimization, High Performance Calculus, Fusion, Disruption.**

I. Introduction

Automated machine learning (AutoML) has the goal to ease setting up machine learning methods for non experts. One of the applications of this research field is HPO ; most of ML methods require parameters to be fixed by humans, like certain thresholds or coefficients, these are called hyperparameters and are generally arbitrarily chosen : HPO aims at automatically searching for the best set of hyperparameters for a given model and problem. DeepHyper is an AutoML package that provides among other tools a framework to perform HPO. The way it works is the following : the user defines the hyperparameter space he wants to search through as well as the process parameterized by it : this process is called the black-box function, it takes as input a configuration of the previously defined hyperparameters and returns a single scalar value which acts as a score of how good the hyperparameter configuration was ; the user then only has to execute the search that will repeatedly run the black-box function with various configurations, aiming on the long run for the best hyperparameters.

II. Details of Work/Research

A. Documentation, Tests, and Tutorials

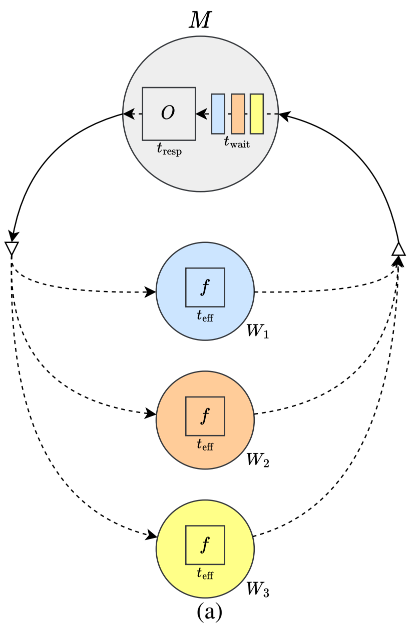
My work with the DeepHyper team began with helping cleaning and reorganizing their documentation, this helped me understanding the architecture of the project while already being able to work on it. Some of the code also needed to be refactored so a bit of work was done on the existing Pytest pipeline to verify that it was working correctly in my workspace and to refactor it along with the modifications done. Once my understanding of the package was strong enough I got in charge to update some of the tutorials and also made one on how to use this tool along with Pytorch (instead of Tensorflow like in most of their tutorials).

B. Development of the ADBO method

The next main goal was to improve the worker utilization of our method at scale, that is the parallelization of the search on HPC systems ; the main contribution of this work was the finding of a new method better at large scale : the Asynchronous Distributed Bayesian Optimization. The previous method (Fig. 1a) consisted of a single manager using a surrogate model to understand the mapping between configurations and scores in order to find the best one ; the role of the manager is to send untested configurations to the workers and gather the scores once the execution is finished, it then optimizes the surrogate model with these new configuration/score pairs to find better configurations to explore. While this works at low scale, this centralization (a single manager) becomes a bottleneck at large scale, hence the need to distribute even the surrogate model and optimization process by having an optimizer on each worker and only sharing the configurations/scores pairs through asynchronous communication between the workers (Fig. 1b). While I contributed a bit to the creation of this method, my main work was on the conception and execution of benchmarks to compare the performances of this new algorithm to the old one, depending on a number of parameters and on a number of machine configurations, with an increasing number of nodes and workers per nodes on the Theta system.

C. Application to the Fusion problem

Once this work was done a collaboration started with another scientific team developing a neural network to predict plasma disruptions in a tokamak fusion reactor [7] ; the goal of this collaboration was to improve their model using our software to find better hyperparameters (within the hyperparameter space defined in TABLE I). It was first necessary to understand their model, how it works and how it is implemented, but their code needed a clean-up, so my first task was to re-implement their pipeline with Tensorflow ; I also made some optimizations, resulting in an even faster pipeline (around 4 times faster). Our algorithm was then put to the test, we managed to improve the AUC score of the model from 0.875 to 0.913 (TABLE II) and show the ability of this hyperparameter



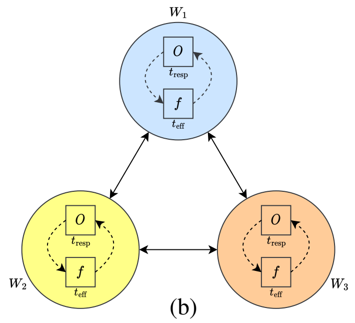


Fig. 1: Centralized (1a) and distributed (1b) search models. A circle represents a process with W for a worker and M for a manager, an arrow represents a communication, O represents the optimizer, and f represents the computation of the black-box function. t_{wait} is the time for which a worker waits before being processed by the optimizer. t_{resp} is the time taken by the optimizer to suggest a new configuration. t_{eff} is the time it takes to compute the black-box function.

Algorithm 1: Greedy Caruana algorithm for ensemble construction

```
Inputs : $\mathcal{C}$: models that can be used to build the ensemble,
         $M$: maximum number of different models to put in
         the ensemble, $X, y$: data and labels on which
         evaluate the models
Output: $\mathcal{E}$ the best ensemble found
/* Initialization for ensemble
   construction                                    */
1  $\mathcal{E} \leftarrow \{\}$
2  $max\_auc \leftarrow 0$
/* Model selection                                 */
3  while $|\mathcal{E}$.unique()$| \leq M$ do
4      $\theta* \leftarrow_{\theta \in \mathcal{C}}$ AUC($\mathcal{E} \cup \{\theta\}, X, y$)
5      if AUC($\mathcal{E} \cup \{\theta*\}, X, y$) $\leq max\_auc$ then
6          $\mathcal{E} \leftarrow \mathcal{E} \cup \{\theta*\}$
7          $max\_auc \leftarrow$ AUC($\mathcal{E}, X, y$)
8      else
9          return$\mathcal{E}$
10     end
11 end
12 return$\mathcal{E}$
```

search to generate diverse efficient configurations well suited for ensemble prediction. Ensemble prediction consists of using the prediction of an ensemble $\mathcal{E} = \{m_{\theta_i} \in [1, \ldots, n]\}$ of multiple models m_{θ_i} in order to make a better prediction ; it also allows to separate the uncertainty of a prediction into its epistemic and aleatoric components [8], giving more information on its origin : whether it comes from the ignorance of the model or the quality of the data. So the next task was to study the uncertainty quantification (UQ) of the models and ensembles of models in order to see if ensemble prediction indeed gave better insights ; different algorithms were used for ensemble calibration : basic top-k models, greedy Caruana (Algorithm 1), and a gradient-based algorithm I came up with (Eq. 3). In order to have a better UQ the models needed to be calibrated, so that the prediction probability they output would represent more accurately their level of confidence in their predictions ; the calibration process consists of finding the best α and β to minimize the cross-entropy of the model when its output y is passed through the following sigmoid :

$$\sigma_{\alpha,\beta}(y) := \frac{1}{1 + \exp(-(\alpha \cdot y + \beta))} \tag{1}$$

While this calibration is done model by model and fixed before ensemble constitution for top-k and greedy Caruana, it is an integral part of the gradient-based algorithm which goal is to calibrate the whole ensemble directly :

$$\begin{aligned} m_{\mathcal{E}}(x; \omega) &:= \sum_{i=1}^{K} \Gamma_i \cdot \sigma_{\alpha_i,\beta_i}(m_{\theta_i}(x)) \\ \Gamma_i &:= \frac{\exp(\gamma_i)}{\sum_{j=1}^{K} \exp(\gamma_j)} \end{aligned} \tag{2}$$

Where the α_i and β_i are used to calibrate the models, and the γ_i are used to weight the importance of each model in the output of the ensemble ; these parameters are computed through gradient descent with respect to the cross-entropy loss L of the corresponding ensemble :

$$\alpha*, \beta*, \gamma*, := \underset{\alpha,\beta,\gamma}{\arg\min}\, L\left(y, m_{\mathcal{E}}(x; \Omega)\right) \tag{3}$$

The final step was to show which method was the best one and study the improvements possible with ensemble prediction, unfortunately a better understanding of the input data

Name	Type	Range	Distribution	Default
batch_size	real	$[32, 256]$	log-uniform	128
dense_regularization	real	$[0, 1]$	uniform	0.001
dense_size	real	$[32, 256]$	log-uniform	128
dropout_prob	real	$[0, 0.5]$	uniform	0.1
length	real	$[32, 256]$	log-uniform	128
loss	categorical	[balanced_cross, balanced_focal, balanced_hinge, cross, focal, hinge]	uniform	hinge
lr	real	$[10^{-7}, 10^{-2}]$	log-uniform	2.10^{-5}
lr_decay	real	$[0.9, 1]$	uniform	0.97
momentum	real	$[0.9, 1]$	uniform	0.9
num_conv_filters	real	$[32, 256]$	log-uniform	128
num_conv_layers	discrete	$[1, 4]$	uniform	3
num_epochs	discrete	$[1, 32]$	uniform	32
regularization	real	$[0, 1]$	uniform	0.001
rnn_layers	discrete	$[1, 4]$	uniform	2
rnn_size	real	$[32, 256]$	log-uniform	200

TABLE I: Table of hyperparameters

was necessary to evaluate the performances of our methods and the contract went to an end before we could conclude with the help of an expert.

	Baseline	Best Model	Top-k	Caruana	Gradient
Test AUC	0.875	0.913	0.913	**0.917**	0.915
Balanced CE	0.285	0.165	0.162	0.158	**0.157**
Inference Time	34.68	21.7	/	/	/
#Parameters	600,873	332,742	/	/	/
#Models	1	1	80	7	10
Training Time	/	/	2.20	274.7	64.16

TABLE II: Comparison of the baseline, best model, and different ensembles regarding AUC as well as inference time (on the whole test set in s.) and number of parameters of the predictor.

III. Conclusion

During this internship, a new HPO method, which was found to be better at large scale, was developed along with the base code for implementing benchmarks and analysing performance, enabling better insights of our results and a more consistent improvement of DeepHyper. A successful application of our HPO algorithms was done on the fusion problem, showing a clear improvement in performances of the model, but still unclear improvements on UQ. Further work includes understanding these results on UQ in order to publish a paper, as well as improve even more the performances and ease of use of the software.

References

[1] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, ser. NIPS'11. Red Hook, NY, USA: Curran Associates Inc., 2011, p. 2546–2554.

[2] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International conference on machine learning*. PMLR, 2013, pp. 115–123.

[3] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian optimization of machine learning hyperparameters on large datasets," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 528–536.

[4] A. Alvi, B. Ru, J.-P. Calliess, S. Roberts, and M. A. Osborne, "Asynchronous batch Bayesian optimisation with improved local penalisation," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 253–262. [Online]. Available: https://proceedings.mlr.press/v97/alvi19a.html

[5] P. Balaprakash, R. Egele, M. Salim, V. Vishwanath, and S. M. Wild, "DeepHyper: Scalable Asynchronous Neural Architecture and Hyperparameter Search for Deep Neural Networks," 2018–2022. [Online]. Available: https://github.com/deephyper/deephyper

[6] P. Balaprakash, M. Salim, T. D. Uram, V. Vishwanath, and S. M. Wild, "DeepHyper: Asynchronous hyperparameter search for deep neural networks," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, 2018, pp. 42–51.

[7] S. A. . T. W. Kates-Harbeck, J., "Predicting disruptive instabilities in controlled fusion plasmas through deep learning." *Nature*, vol. 568, 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1116-4

[8] R. Egele, R. Maulik, K. Raghavan, P. Balaprakash, and B. Lusch, "AutoDEUQ: Automated deep ensemble with uncertainty quantification," *CoRR*, vol. abs/2110.13511, 2021. [Online]. Available: https://arxiv.org/abs/2110.13511