Optimization of Machine Learning Jobs in the Cloud

(extended abstract of the MSc dissertation)

João Pedro Neves Nogueira
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisors: Professor Paolo Romano

Abstract

Machine Learning and Cloud Computing have been two of the fastest growing areas in the the past few years. Recent developments have emerged regarding machine learning optimizations, enhancing their accuracy and training time. However, for optimization procedures that have very large datasets and many hyperparameters, most users turn to the cloud to offload the inherent computation that would otherwise be infeasible locally. In order to do so, users face the task of picking cloud parameters to deploy their machine learning jobs which can be difficult due to the wide range of possible configurations and whose misconfiguration translates into large, unnecessary costs for large scale models. Recent state-of-the-art systems have taken the approach of performing optimization of both cloud configurations and machine learning hyperparameters in a joint fashion, with the goal of minimizing cloud related expenses while reaching the best hyperparameter configuration for the specified machine learning algorithm. Nonetheless, the optimization procedure involved by these approaches can also require exploring a large number of expensive configurations and impose, in its turn, large economical costs. It is thus crucial that the optimization procedure is as time efficient as possible and converges rapidly towards the optima. This thesis proposes Hydra, a self-tuning system solution that performs optimization of machine learning algorithms improving some drawbacks of extended state-of-the-art systems by rapidly converging towards the optimum solution without wasting time on bootstrapping the model, using many low-budget evaluations of configurations while applying transfer-learning to enhance the models' performance, ultimately reducing overall costs by 35% of the extended work.

1 Introduction

Machine learning (ML) has emerged as a popular research area that aims at automating model building in order to develop self learning systems. ML branches from artificial intelligence and pursues the goal of extracting information from data for decision making, pattern identification, and many other possible applications requiring minimal human interaction. Some examples of ML applications are website recommendation services [19], satellite image recognition [18] and many more [7, 24]. For a ML algorithm to learn and achieve a good accuracy, the data used in it needs to have both quality - the training data should be representative of the target application scenario - and quantity - a sufficiently large volume of data should be available to provide an adequate characterization of the phenomenon to be modeled. Since end-users need ML models to be built as fast as possible, the training procedure also demands relatively high resource requirements that scale with the targeted accuracy and amount of training data.

As the amount of digital data grow, novel sophisticated ML algorithms are developed and larger applications for ML are frequently deployed, demanding an exponential amount of resources from users in large scale jobs. Associated to these models are hyperparameters, which is a type of parameter that controls the training process of ML algorithm. To enhance the accuracy of ML model, end users were accustomed to tune its parameters, but the complexity of testing and tuning the hyperparameters of a ML job has become prohibitive given the increasing complexity of the ML jobs being currently used. Therefore, researchers have investigated automated optimization techniques that address hyperparameter selection of machine learning jobs [8, 21, 28]. These optimization methods follow a black box approach, which requires testing the model multiple times in different configurations. Given the resource-intensive nature of training and optimizing complex ML jobs [9], users have naturally turned to the cloud to deploy this kind of jobs.

Cloud computing is one of the areas in technology that has bloomed more in recent years, allowing us to offload large workloads to large data centers that have the ability to process them in a relatively short time. Given the abundance and heterogeneity of available cloud resources, users are faced with a complex choice when they need to pick the right type and amount of resources for deploying their jobs. Thereupon, researchers developed systems such as [5, 24], to perform cloud optimization that enables users to reach a decision for what type of cloud configurations should it pick to perform a certain job.

Unfortunately, though, most of the existing literature looks at the optimization of the cloud configuration for a ML job and at the tuning of the hyper-parameters of a ML job as to two independent problems. Only very recently [5], the importance of jointly optimizing these two types of parameters has been recognized. In fact, the choice of hyper-parameters related to, e.g., the synchronization of the parallel/distributed training process can be strongly affected by the number and type of cloud resources employed to support the training process. As a consequence, optimizing the two set of parameter

independently, as done in most of the existing literature, can lead to identifying configurations that are up to 3.7× less efficient [5]. On the other hand, joint optimizing these two set of parameters leads to an exponential growth of the resulting search space, urging for novel solutions that can efficiently crawl this search space and minimize the cost and latency of the resulting optimization process.

1.1 Objectives

In the following, the state of the art in the area of optimization of ML training jobs in the cloud is critically analyzed. In the light of this analysis, two main research directions are identified and proposed for my MSc dissertation:

1. Investigating how to extend BOHB [8], a recently proposed method for hyperparameter tuning to optimize, in a joint fashion, both the model's hyper-parameter and the choice of the underlying cloud platform. The key idea at the basis of BOHB is to test configurations "partially", i.e., allocating an intentionally limited "budget" (e.g., time or cost) to each configuration test and timing out the testing once the allocated budget is depleted. This information is used to build a model of the application's efficiency over the set of untested configurations, which can then be consulted to drive the optimization process. The process is then repeated iteratively, invoking the model to select which configurations to test in the next iteration, which will test a number of configurations decreased by a factor β, allocating to each configuration test a budget increased by the same factor β. Unlike in conventional model-driven approaches [12, 14, 28], which do not explicitly control the cost of testing a configuration, the cost incurred to create a model can be significantly reduced.
2. BOHB predicts the quality of a configuration via models that are built considering a specific testing budget. As the optimization process progresses, the budget used for testing increases exponentially, and the number of configurations tested at each iteration also drops with an exponential rate. As a consequence, the models used by BOHB, as the optimization process evolves, are based on an exponentially decaying number of configurations, which, we argue, can limit their prediction accuracy significantly. To cope with this limitation, we plan to extend BOHB to incorporate transfer learning techniques aimed at extrapolating the predicted configuration quality across different budgets. Through the use of transfer learning techniques, the models used by BOHB to steer the optimization process will be able to retain and exploit the knowledge acquired when testing configurations with smaller budgets.

1.2 Contributions

This thesis focuses on the analysis of machine learning optimization systems, providing insights and developing a system that covers underlying drawbacks of recent state-of-the-art systems that are shown to have good performances. The main contributions are:

- Overview analysis and comparison of state-of-the-art systems.
- Hydra, a self-tuning system solution that performs optimization of machine learning algorithms improving some drawbacks of previous systems by rapidly converging towards the optimum solution without wasting time on bootstrapping the model, using many low-budget evaluations of configurations while applying transfer-learning to enhance the models' performance, ultimately reducing overall costs.

2 Related work

In this section we will start by introducing state-of-the-art systems in the context of hyperparameter optimization and then we will transition to cloud optimization systems.Finally, a brief summary of the presented state-of-the-art techniques is discussed.

2.1 Hyperparameter Optimization

As previously discussed, in machine learning many algorithms require the user to set some *hyperparameters*. This type of parameters are called hyper because they influence how the algorithm will learn. Examples of hyperparameters are the synchronization method and batch size used by different workers in a distributed training process [22]. Unfortunately, guessing a good value for a model's hyperparameters beforehand is far from being a trivial task, as their correct tuning is affected by a large number of factors, such as the shape of the function that the ML model is learning or the number/type of computational resources being harnessed in the learning process. Machine learning algorithms tend to have large training times and require a large number of resources such as powerful CPUs and in some cases even one or more GPUs. Given this, it is imperative that the optimization task minimizes both cost and time while providing a set of hyperparameters that ensure optimal (or close to optimum) performance. In this section we will review some state-of-the-art optimization techniques in the scenario of hyperparameter optimization that approach the problem in different ways.

Bayesian Optimization The existing approaches that use Bayesian Optimization for hyperparameter optimization build a model, often based on Gaussian Processes, that predicts, for each possible hyperparameter value, the corresponding accuracy achievable by the model. A great advantage of this model is that it accumulates all data from previous evaluations of the objective function, which typically leads to producing more accurate predictions and, consequently, to enhance the speed of convergence towards optimal solutions.

This technique proves to be very efficient in providing highly accurate configurations but there are some intrinsic weaknesses associated to it, such as: *i)* it needs to have some samples before building a model; *ii)* each individual sample has a high cost; *iii)* GPs are very slow to train, especially if a large number of configurations have been tested. Consequently the initial phase is costly and slow. Besides this, in large datasets or in scenarios with a substantial amount of hyperparameters, BO will scale poorly because it will need to

train the algorithm on the whole dataset and build increasingly complex models for each hyperparameter it is added.

In the following, we will discuss some algorithms that mitigate this disadvantage by adopting techniques such as transfer learning and others that provide cheaper costs and meet the same or better results in the process.

Hyperband Hyperband [21] (HB), is characterized as a model-free technique of hyperparameter optimization that originated from pure exploitation bandit problems, that have the goal of minimizing *regret*, that is the distance from the optimal solution as fast as possible, in any setting. This algorithm extends the Successive Halving [15] (SH) algorithm that performs in the following way: Given a growth factor *N*, a minimum and maximum *budget* (e.g. wall clock time), a fixed number of randomly sampled configurations will be evaluated with the minimum budget, then they will be compared. The top *1/N*, multiplied by the number of tested configurations will pass to the next phase, which is similar to the previous phase, with the difference that it only evaluates the passed configurations and having the budget multiplied by N. This algorithm stops when the maximum budget value is reached.

As described in [21], low budgets will produce noisy evaluations that can be misleading in SH, so HB tackles this concern by doing multiple runs of SH and increasing, at each run, the minimum budget. As such, HB mitigates the risk of being biased, hence staying in the Successive Halving iteration. However, it may not scale well when the budget increases. Ultimately, HB recommends the configuration that performed best across every run of SH.

HB has the advantage of being very fast regarding proposing good configurations in early stages. By comparing it with BO, it proposes configurations and converges faster in the early stages. However, due to its stochastic nature, HB suffers from the same issue of Random Search [3]. For instance, HB does not leverage the information of previously done evaluations, since it only maintains a record of the best performing configuration, making it converge slowly and most likely not reach the global optimum.

Fabolas Fabolas [17] is a state-of-the-art technique for hyperparameter optimization that tries to increase the efficiency of BO when used to optimize machine learning jobs that need to digest large datasets. The idea at the basis of Fabolas is to infer optimal configurations for training using the full dataset, based only on observations performed using a subsampled dataset. This leads to speeding up the initial model building phase and provides faster results when compared to traditional BO-based hyperparameter optimization techniques [12, 14, 28].

Producing a model while using a subsampled dataset will result in cheaper function evaluations, however it will also produce a worse approximation of the objective function which in turn will provide worse samples. To tackle this issue, Fabolas models accuracy and training time as a function not only of the hyperparameters' configuration, but also of the dataset size. Based on this model, Fabolas seeks the best trade-off global optimum. Finally, Fabolas extrapolates the knowledge to the original dataset by predicting what configurations will achieve the best result.

This approach of constraining the resources needed to build the model in order to lower the cost of function evaluations is similar to the HB approach of doing *budget* runs to minimize cost. However Fabolas is not as fast as HB in the initial phase but it does converge faster than BO in general.

BOHB BOHB [8] is a state-of-the-art technique that combines two techniques that were previously discussed, namely Hyperband [21] and BO [4]. It does so in order to leverage the advantages that both algorithms bring, while minimizing their disadvantages.

BOHB performs BO with a different modeling scheme, instead of using GP to model the objective function, it uses a Tree Parzen Estimator [4] (TPE). TPE uses a kernel density estimator which instead of modeling the objective function directly as GP does, models two different densities over the input configuration space. These densities are represented by $l(x)$ and $g(x)$; where the first density captures configurations that performed significantly well, that are above a certain α threshold, and the second has configurations that have undesirable results, that are below the α threshold. This change of model came due to the fact that TPE scales better than GP, while maintaining the support for mixed discrete and continuous configuration spaces. After declaring the budgets boundaries, the algorithm will start by doing the Hyperband method with a relevant difference: unlike HB, BOHB does not sample configurations randomly. Instead, it does not always randomly sample configurations; conversely, it relies on the TPE-based models, constructed in previous HB runs, to determine which configurations to test in the next HB run. This way, the knowledge acquired by testing configurations in previous HB runs is retained and exploited to drive the future HB runs and enhance convergence speed.

This method has the speed advantage of Hyperband, i.e., it is able to reduce the cost of evaluating the quality of configurations by controlling the computational budget allocated over time to function evaluation. Additionally, BOHB selects, with a small, user-tunable probability, configurations in a purely random way (i.e., without consulting the model). This design choice improves the robustness of BOHB in presence of inaccurate/flawed models, which, in pure model-driven approaches (e.g., based on BO [28]) are likely to hinder the efficiency of the optimization process.

2.2 Optimization in the Cloud

This section reviews a set of state-of-the-art approaches that tackled the problem of optimizing, according to different metrics, the efficiency of complex applications to be deployed on the cloud. As it will discussed, most of the solution in this area of the literature treat the application as a black-box and focus solely on the identification of the right amount and type of cloud resources to be allocated to the application to meet user-defined constraints on QoS.

Lynceus Lynceus [5] is a recent approach for the optimization of cloud-based jobs. It adopts model-driven optimization as [2] but it refines its model in a different manner. Lynceus is a budget-aware and long-sighted self-tuning system of cloud resources that has the goal of discovering the configurations that minimize the execution cost of data analytic jobs by ensuring that the maximum execution time constraint is followed and the evaluation of a configuration does not exceed a given budget, where a configuration in this scenario is composed by cloud parameters (e.g. virtual machine type and number of instances) as well as hyperparameters of machine learning jobs.

To achieve its goals, this system has the following strategy: At the beginning of the exploration phase, where it strives to find a good configuration and there is a large uncertainty in the cost model, Lynceus allows for a larger budget and presents a more explorative behavior. As the system explores more configurations and the cost model becomes more accurate, the budget will decrease. In this phase, Lynceus adopts a more careful and exploitative approach where it only selects configurations that will not compromise the given budget, while leveraging the cost model to achieve the maximum shorter reward.

Lynceus does consider both cloud and application's configuration parameters jointly. However, due to its reliance on BO, it suffers of the same problems already discussed when introducing CherryPick, which are reacquiring an initial bootstrapping phase that, may lead to testing very expensive configurations.

Overall by the analysis of the table we get that:

- Only Lynceus aims at optimizing both cloud and application parameters. The authors of that solution have also reported experimental data that confirms the relevance of optimizing these parameters in a joint fashion, with gains (in terms of cost reduction for the users) that can extend up to a factor 3.7× when compared to solutions that optimize the two set of parameters independently.
- Unfortunately, the reliance of Lynceus on BO [28] exposes it to a number of shortcomings that have been highlighted by the recent literature on hyperparameter optimization.
- There are techniques that do not require to build a model, hence, in that short time window, they gain some benefit for those resources, however they are quickly outclassed in terms of convergence to the global optimum after some time.

3 The Hydra Optimizer

In this section we propose and cover the design/implementation of Hydra, a system that build on BOHB and extends it to address its main shortcomings, such as the inability to extrapolate how the quality of configurations vary across budgets. We also present variants of this system that try to balance the economic cost of the optimization process by taking it into account throughout the run.

3.1 Overview

Performing optimization of machine learning algorithm is an expensive procedure. As a matter of fact, even with BO-based techniques [28], which strive to minimize the number of evaluations needed to reach a global optimum, each evaluation can still be very demanding in terms of resources and time. Other techniques, such as Fabolas [17], have addressed this problem by using sub-sampling in the training dataset so as to reduce cost of evaluating the quality of configurations during the optimization process.

These techniques require a model to be built firstly in order to produce results that can lead them to the global optima and since this task is done by randomly sampling some configurations there is always a fixed amount of resources that is spent and do not contribute to the end goal in that time window. This said, BOHB [8] leverages both Hyperband [21] and Bayesian Optimization to produce results before the model has been built and after, provide more information to the model with the configurations that have been evaluated. This allows for achieving convergence rates that are faster than BO, while constraining evaluations to a budget, so as to reduce the cost of the optimization process. BOHB then effectively counters the presented issue with traditional BO. Furthermore it inherits some beneficial features of Hyperband in the sampling phase, since it has the possibility of exploring random configurations, while reducing the penalty of it being sub-optimal via the use of the Successive Halving [15] technique.

Extending BOHB to jointly optimize both the ML applications and the cloud configuration is not trivial. The first challenge with BOHB is understanding what is the most efficient way to employ BOHB for this purpose. One key question that arises is whether the cloud configurations should be treated in an opaque way i.e. similar to additional hyperparameters in a hyperparameter configuration. The risk of such a simplistic approach is that it exposes to the risk of sampling very expensive configurations that require a large amount of computational resources unnecessarily, e.g., in the initial phases of the optimization process where no or very little knowledge is available on the job being optimized.

Another shortcoming of BOHB that we intend to address is its inability to exploit information gathered when testing configurations with small budgets. In order to overcome this limitation, we plan to use transfer-learning. By recording the performance of various configurations evaluated using diverse budget levels, Hydra can leverage that information to find a trend between budgets and use it to predict the performance of configurations on larger budgets.

3.2 Design Details

Hydrais a solution that extends BOHB [8], which itself its an extension of Hyperband [21]. Since BOHB proved to possess the speed of Hyperband while being more likely to select high quality configurations via model-based techniques, we argue that by having a similar system with a richer model and a more effective way to extract the model's knowledge (via

the use of alternative acquisition functions), one can further enhance the efficiency of the optimization process.

In Hydra, we have selected Gaussian Process [25] as base ML technique, since they are the most frequently used models in Bayesian Optimization due to their ability to provide smooth and accurate uncertainty estimates. Specifically, we use Gaussian Processes with Matérn $5/2$ Kernels [11], which is also a common choice in the Bayesian optimization literature given that it produces less restrictive smoothness assumptions [17] — an important feature, given that we plan to add another dimension to the model's feature space, namely the Hyperband budget. Hydra supports various EI-based acquisition functions, including novel ones defined for being used in the context of the HB optimization method.

BOHB only trains configurations using a single budget, i.e. the largest budget for which a minimum pre-determined number of configurations has been gathered. As a result, only a subset of the available info is exploited whenever the model is used/queried. This problem is solved in Hydra by considering budget as an extra feature and training a single model with data gathered using diverse budgets, in order to enable the construction of models that can extrapolate the trends that arise when the budget varies. Recall also that the objective is to find a configurations that has maximizes accuracy using the **full** budget, so the models should be used to identify configurations that will excel at full budget, but that will be at least initially evaluated with lower budget, based on the SH algorithm.

In Hydra, there are some parts of the system that we chose not to change, having the similar behavior as BOHB, such as instead of always using the model to predict a configuration, we still use a probability of sampling instead a random configuration, thereby maintaining Hyperband theoretical guarantees. When producing a prediction, Hydrauses the Hyperband algorithm to determine how many configurations will be generated. If at least $(d + 1)$ configurations (where d is set to the number of dimension in the configuration space) have been evaluated, the model is used to make a prediction on a configuration. Otherwise, we sample configurations randomly according to a uniform distribution. Even if there are enough results, there is always a probability (which as in BOHB we set to 33.3%) that we chose to sample randomly.

The process of producing a prediction with a model implies: (i) to train the model with all the collected data from the evaluated configurations, and (ii) identify the *incumbent* configuration, i.e., the one that the model predicts to yield the best result (e.g. highest accuracy, lowest loss) when deployed using the full budget. The main reason behind focusing on choosing incumbents that have the highest budget is to guide the model to focus on achieving the best performing configurations on the maximum budget, where there is a higher probability in sampling better configurations. However, by doing so, in scenarios where there are no sampled configurations that have as high budget value, such as the beginning of the first Hyperband bracket, we drop that constraint and allow the incumbents budget to take the value of the highest sampled budget from the gathered data. To collect more information about Hydrabehavior, we have also implemented multiple variants that have a different take on how the predictions are made, and in the following sections we will explain what are the main differences and their purpose.

3.2.1 Budget Sampling

Analogously to Hyperband and BOHB, in Hydrathe notion of budget can mapped to different metrics that constraint the amount of resources consumed when training a ML model, e.g. wall-clock time, iteration, algorithm epochs, cloud cost.

In both these systems, the algorithm starts by sampling a number of configurations randomly, and evaluates them using the lowest possible budget. In Hydrawe use the same procedure, but, as in BOHB, at some point, when enough configurations are gathered to build a model, we can start choosing which configurations to evaluate by leveraging its knowledge. To get a prediction from the model, we go through the search space and for each different configuration (or for a set of randomly chosen configurations if the configuration space is too large to be exhaustively sampled) we compute the Expected Improvement [10] (EI). Finally, we select the configuration that has the highest EI value.

Differently from BOHB, though, in Hydrawe treat the budget as a model's feature. When querying the model, we need therefore to establish what value of the budget to specify. The Hyperband algorithm only determines which configurations to evaluated at the beginning of a bracket. From that moment on, the top configurations will pass to the next stage, to be evaluated with a higher budget. This Successive Halving [15] will happen until the maximum budget is reached and the highest quality configuration *using full budget* is returned. Thus, one may argue that the model should be queried to identify the configuration that will achieve maximum quality (e.g., accuracy/loss) in the maximum or final budget. However it is also arguable that what matters in a stage of a bracket is the performance of the configuration on the current budget of that stage. Indeed, a poor performance in early stages would reduce the odds of that configuration to be among the top ones that proceed to the following stages with higher budgets. Consequently we consider two variants: one that selects configurations according to their EI value considering full/maximum budget, which we call Full-Budget Sampling (FBS), and another variant that similarly selects configurations according to their EI value, but considering the current stage budget. We name this variant Current Budget Sampling (CBS).

The main advantage of the FBS strategy is that it will favor configurations that excel on higher budgets. However, if a configuration does fall short in its performance in lower budget values, it may not be tested in higher budget levels, since Successive Halving may discard that configuration. Conversely, CBS will focus on picking configurations that excel on the lowest budget of the current bracket, thus improving the odds that a model chosen configuration passes through the initial Successive Halving pruning of configurations. Clearly, these

two variants will have the same behavior when predicting configurations on a bracket that has as initial budget value the maximum budget value of the experiment.

3.2.2 Cost of evaluating configurations

In the cloud different choices of type and numbers of virtual machines yield different costs. Hydrakeeps the cost factor into account by incorporating several cost-aware acquisition functions, which will be evaluated in the following chapter. Expected Improvement per dollar is a classic technique to keep into account costs in BO. However, Hydraintroduces a new cost-aware acquisition function tailored for operating in a successive halving scheme. By Expected Improvement per dollar with the Budget Sampling variants discussed in section 3.2.1, with FBS we can sample configurations that have the potential of achieving a high accuracy in higher budget values while possessing a low economic cost. This may hamstring FBS capacity of providing good configurations in low budget value scenarios even more, however we can ensure that whenever a configuration that was sampled through the model, if it reaches the highest budget value stage, it will have a reduced cost. On the other hand, if we consider CBS with this variant, it will sample very economic and well performing configurations in the initial budget value of a bracket, but on the higher budget value stages the configuration might achieve higher than expected costs. This economic-cost reducing variant may have a cost-reducing prospect, however we need to take into consideration that duplicating a model that is already considered slow compared to Tree-structered Parzen estimators as shown in BOHB [8] may deter the algorithm from being fast and by consequence proving to have a higher economic cost. When we are combining the economic cost variant Expected Improvement per Dollar with budget sample variants FBS and CBS, we are restricting both base model and cost model to have the same *target* budget, e.i. sample according to maximum or current bracket budget. In order to have a prediction that can leverage the performance of configurations on higher budget values while reducing the cost of configurations on current budget values, we developed another variant called Hybrid Sampling. Essentially it uses FBS to retrieve the Expected Improvement of a configuration the highest budget possible, while dividing the cost of the same configuration but on the initial budget a bracket. Since Hyperband proposes a large quantity of configurations in lower budget values, we expect this to minimize the economic costs greatly in early stages while providing the ability to outperform others in the last.

3.2.3 Cost of identifying the next configuration to be evaluated

Hyperparameter optimization of large machine learning models can have a high economic cost [29], especially in scenarios where there is a need to rent computational power. In situations such as this, we want the optimization to be as efficient as possible so that the optimization economic cost is as low as possible and produces the best result. Unfortunately, without querying the model for all possible configurations, e.g., using a grid-based approach, one cannot guarantee to have correctly identified the configuration that the model predicts to be the optimum. In BOHB, there are always a fraction of configurations that are randomly sampled, and in scenarios like we have described previously where there are associated economic costs to each configuration that is sampled, having an under performing result can have an even more negative impact. These situations can't be avoided in Hydraas well since we need initial results to build a model, and we need Hyperband's theoretical properties. However, we can greatly reduce the economic cost of the optimization when using the model to predict a configuration. This is done by replicating the Gaussian Process model we use to predict the accuracy/loss of a given configuration, but instead of feeding performance-related information, we use economic-cost related information and finally, when calculating the Expected Improvement of a configuration, we divide it by the predicted economic cost given by this new model. When the search space is too big to compute exhaustively the acquisition function on all configurations, Hydra supports a simple heuristic that was already used in Fabolas [17], namely a mix of uniform random sampling and sampling via a gaussian centered on the currrent incumbent. As an alternative, one could have used other black-box optimizers such as Direct [16] or CMAES [13].

4 Evaluation

This section evaluates Hydravia five different experiments, where we perform 10 iterations of the standard Hyperband algorithm, forming 2 identical configuration samples as in table 2, with varying budgets for each experiment. We first present the settings of each experiment. Then, we compare how each variant performs in each different environment measuring the loss of configurations achieved and accumulated cost ($) spent performing the optimization and the time taken. After this, we will select two of the best performing variants and compare them in the same experiments against state of the art algorithms that are related to Hydra, namely Hyperband and BOHB. We chose those algorithms because we want to establish experimentally if Hydracan outperform them in different scenarios. We include among the baselines also a variant of BOHB, which, instead of using the Tree-structured Parzen estimator, relies on the same modeling techniques used in Hydra. This allows for discriminating the effects of using different modeling techniques (TPE vs Hydra's GP-based acquisition functions) and of different input data sets (including or not the budget in the set of features fed to the models).

4.1 Test Environment

In this chapter we present five different experiments. Three of the five experiments only have a single difference, which is the machine learning model used to train the MNIST [20] dataset. This dataset is composed of 70000 28x28 pixel images of size-normalized handwritten digits from zero to nine

and has training set of 60000 and a test set of 10000 image examples. These three experiments have the budget associated to the dataset size (number of images) used to train different neural networks. The minimum budget used was 3750 images, and the maximum was 60000 images. These three experiments were conducted using different neural networks, namely CNN [1] (convolutional neural network), RNN (recurrent neural network) [27] and a multilayer neural network [6]. These neural network models were trained in a public cloud, namely Amazon Web Services (AWS), over a large number of configurations (288) and the corresponding cost and execution time were made publicly available [24] [5]. Table 1 has information about the hyperparameters used in these datasets. The configuration space considered in these experiments includes both parameters describing the type and amount of virtual machines to provision from the cloud and three models' hyperparameters, namely learning rate, batch size and synchronization type. From here on we will mention theses experiments as CNN, RNN and Multilayer. In CNN, RNN and Multilayer experiments we use as the economic cost measure the cost in dollars ($) of training a model with a given hyper-parameter configuration in AWS using the picked virtual machine type (which has an associated cost per second). Some other parameters of the optimization process in these experiments are equal, such as the intermediate budget values, as detailed in table 4.

CNN, RNN and Multilayer Experiment Search Space	
Hyperparameter	Values
Batch Size	[16, 256]
Learning Rate	[0.00001, 0.0001, 0.001]
Number of Workers	[8, 16, 32, 48, 64, 80]
Synchronization type	[asynchronous, synchronous]
Virtual machine Flavor	[t2.small, t2.medium, t2.xlarge, t2.2xlarge]

Table 1. Description of CNN, RNN and Multilayer hyperparameter values.

The fourth experiment was selected as it was previously used in the evaluation of BOHB. As such this experiment focuses solely on the problem of hyper-parameter optimization, i.e., it does not include the type/amount of cloud resources in the configuration space. Analogously to the previous experiments, it also uses MNIST and a CNN. However, it considers a set of seven hyper-parameters, see table 2, yielding a total of 135000 possible different combinations.

CNN, RNN and Multilayer Experiment Search Space	
Hyperparameter	Values
Stochastic Gradient Descent Momentum	[0.0, 0.2, 0.4, 0.6, 0.8]
Learning Rate	[0.000001, 0.00001, 0.0001, 0.001, 0.01]
Number of Filter in Layer 1	[4, 8, 16, 32, 64]
Number of Filter in Layer 2	[0, 4, 8, 16, 32, 64]
Number of Filter in Layer 3	[0, 4, 8, 16, 32, 64]
Number of Hidden Units in the fully connected layer	[0, 4, 8, 16, 32, 64]
Dropout Rate	[0.0, 0.2, 0.4, 0.6, 0.8]

Table 2. MNIST hyperparameter values.

Given that exhaustively evaluating the acquisition function on all possible configurations is infeasible in this case, given the vastness of the configuration space, we only compute the acquisition function for 8000 configurations selected at random, where 70% are uniformly distributed throughout the whole search space and 30% sampled by centering a Gaussian on the current incumbent. We will be referencing this experiment as MNIST in the future. As cost metric we use time here, since a single machine was used in this experiment.

The final experiment involved a UNet [26] neural network adaptation in the context of Satellite Image Segmentation. This network uses Feature Pyramid Network [23] that has a size of 256*256*512 neurons with 1.2 gigabytes of training data. The loss function is soft-max cross-entropy, and the hyperparameters were machine type, batch size, learning rate, momentum, and synchronization type. In this experiment we performed the training in GPUs and used two different machines, wall-clock time was used as budget and we have the maximum budget as 5 hours and the minimum budget as 18 minutes and 45 seconds.

In the following section we will explain the workflow of the optimizer in our experiments in order to clarify the analysis of the results.

SH#	Initial budgets				
	3750	7500	15000	30000	60000
	Bracket 1	Bracket 2	Bracket 3	Bracket 4	Bracket 5
0	16	8	4	4	5
1	8	4	2	2	
2	4	2	1		
3	2	1			
4	1				

Table 3. Hyperband bracket decomposition with maximum budget = 60000, minimum budget = 3750.

4.1.1 Plotting details

To simplify the visualization of the plots, after the first bracket (31 explorations) we start plotting the incumbent of each optimizer. This is in fact the first point in which a full budget configuration is evaluated, thus allowing to establish the notion of currently known optimum. When an optimizer fails to find a better incumbent, we plot a black circle with the current incumbent loss, however when it upgrades an incumbent we plot the point similarly to the points in the first bracket. For each plot we mark with a red square the best achieved loss value for each optimizer. We also plot the variance of the plotted values with the same color as the optimizer. The results are based in 300 runs of each optimizer (10 iterations per run) for the CNN, RNN and Multilayer experiment, 50 for UNet dataset and 20 for MNIST dataset. All the runs use deterministic seeds to initialize the random number generators in each different run, making it possible to replicate the same results for every different run. In the next section we will start by presenting the results of every experiment with respect to the performance of the various variants of Hydrathat were developed in this dissertation.

4.2 Comparison with state of the art optimizers

In this section we present the results gathered using the CNN, RNN and multilayer neural network datasets and aim at evaluate the performance of the FBS and CBS with EI per $ variants of Hydraagainst two state of the art optimizers, namely Hyperband (HB) and BOHB (BOHB-TPE). As previously mentioned, we have also included a BOHB variant that uses Expected Improvement as the acquisition function (and the same Gaussian Process models as in Hydra) to isolate the gains deriving from incorporating information on configurations using different budgets in the model (as Hydradoes). We start by analyzing the accumulated economic cost and loss. In addition, we will provide a table which will contain the optimization overhead and additional information about and finally will sum up the analysis.

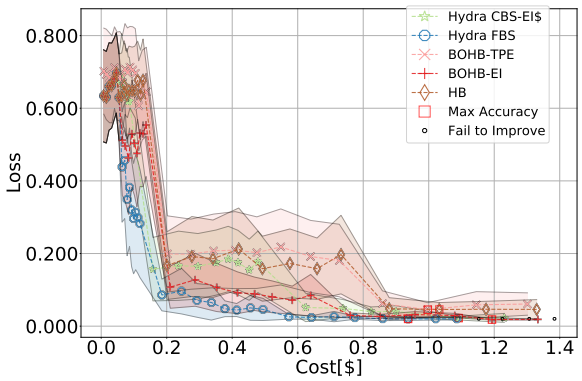


Figure 1. Accumulated Cost ($) and Loss in CNN, scaled in the first iteration.

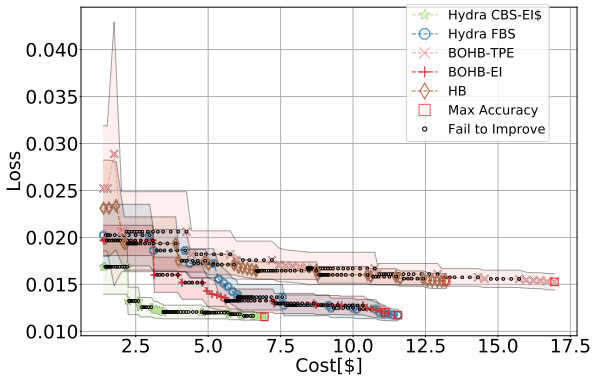


Figure 2. Accumulated Cost ($) and Loss in CNN, scaled in the 2nd to 10th iterations.

4.2.1 Cost of the optimization process

CNN In Figure 1 we can see the cost accumulation progression with respect to our elected variants of Hydra, namely FBS and CBS with EI per $, when compared with Hyperband, BOHB and BOHB with EI. This figure is focused on the first iteration, and we can clearly see that the FBS variant has a significant advantage over BOHB-EI, and CBS with EI per $ variant. This advantage is even more apparent with respect to BOHB-TPE and Hyperband. By analyzing, the remaining 9 iterations, see Figure 2, we see that the performance of FBS is closely matched with BOHB-EI. Both of them are shadowed by HydraCBS with EI per $, which is able to reach better configurations with almost half the cost. Comparing CBS with EI per $ with BOHB-TPE, we can even see a clearer advantage, where with just only 2$ it is able to match its minimum loss value throughout the hole experiment, spending on average 88% less.

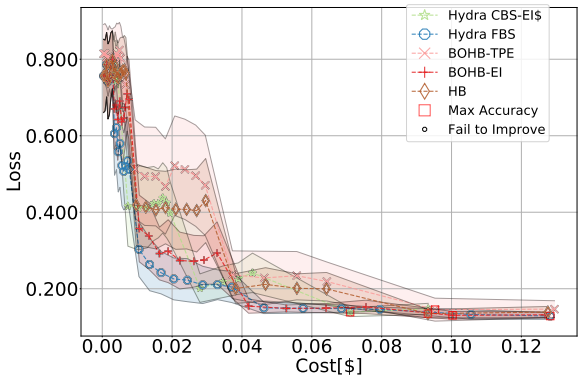
RNN and Multilayer for FBS and CBS with EI per $, the best performing variants would be by contrast CBS and CBS per $. We show the gains of using CBS in these experiment in figure 3. In these particular experiments, CBS is better than FBS and it also proves to be better than BOHB-EI. Interestingly CBS shows the exact same behavior as BOHB-EI in figure 3. This is because CBS samples always according to the same budget and it does not know any other result outside of the initial budget. Since the training set's configurations contain the same budget value the model wants to predict to, it will treat the configuration as if it does not have a budget hyperparameter, because it has no knowledge of any other configuration with a different value and it does not change the predicted value until it finishes the bracket. After all, if CBS ignores the budget dimension it essentially becomes like BOHB-EI in iteration 1. We can verify the advantage of inter-budget knowledge by the gains showed in figure 3. In these plots we can view a great advantage of using Hydracomparing it with Hyperband and BOHB-TPE, and even with BOHB-EI, which is indirectly a "enhanced" version of BOHB.

MNIST Since MNIST has its economic budget value equal to wall-clock time, we will show the comparison on section.

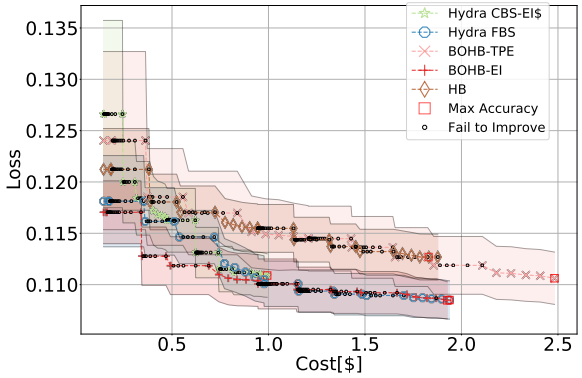
UNET In figure 4 (a) we can see that CBS with EI per $ manages to achieve better configurations with lower cost values than any other systems. We can see that in this experiment FBS does not distinguish itself from other variants, closely matching their loss values with the same cost. In figure 4 (b), we can see that FBS has in general better performance than other variants. Hyperband has great results too, which can indicate that this experiment is very hard to model. We can also see that CBS with EI per $ has in general, better performance than BOHB-EI and especially BOHB-TPE, and it also has lower costs. BOHB-TPE has on average the largest cost value.

4.2.2 Hydra Overhead

In this section we compare the overhead values obtained during the experiment. As demonstrated in table 4 we can view

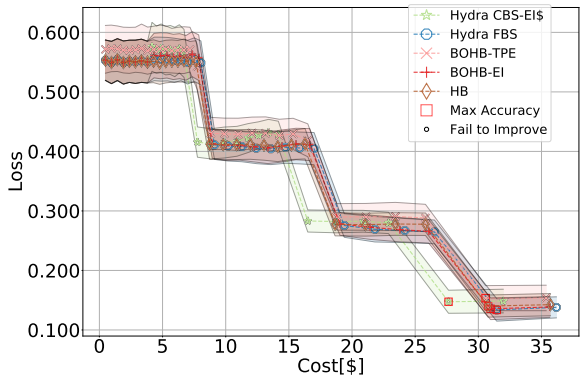


(a)

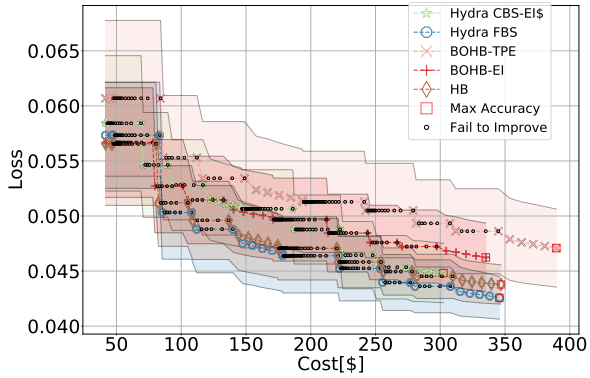


(b)

Figure 3. Accumulated Cost and Loss with Multilayer in the first iteration (a), and the second iteration (b)



(a)



(b)

Figure 4. Accumulated Cost in dollars [$] in UNET experiment focused in 1st Iteration (a) and focused on the remaining Iterations (b)

the accumulated overhead of each system in each experiment. Hyperband always has an almost non existing overhead and is because it always randomly samples the configurations that are to be evaluated. Across all experiments we see that BOHB-TPE and Hyperband have the lowest total overhead value, and especially in MNIST experiment, this difference is very noticeable. This happens because the Tree-structured Parzen Estimator used in BOHB-TPE is much faster than the Gaussian Process models used in Hydra, and this is exacerbated in UNET experiment because the search space is almost $500x$ larger than CNN, RNN and Multilayer experiment, and three orders of magnitude larger than UNET. This is minimized by only computing the EI of 8000 configurations in maximum, but it still has a sizable difference.

	Accumulated Overhead[s] over 258 explorations				
Experiment	Hydra FBS	Hydra CBS-EI/$	BOHB-EI	BOHB-TPE	Hyperband
CNN	21.5	26.9	17.6	5.4	0.2
RNN	20.3	26.3	16.4	5.3	0.1
Multilayer	20.0	25.8	16.8	5.4	0.1
MNIST	596.4	803.9	504.6	172.8	0.2
UNET	9.7	13.3	8.5	3.8	0.1

Table 4. Overhead value for each system in each experiment

5 Conclusions

Hyperparameter optimization of machine learning is an essential area of artificial intelligence that focuses on enhancing the performance of machine learning models. Unfortunately, though, this process is notorious for being costly and time consuming. Novel state-of-the-art systems regarding this topic have been significantly improving their performance and reducing the associated costs. However, since the majority of largest optimization tasks are performed in the cloud, it is crucial that systems are as fast and efficient as possible, and some, as covered in this report, present shortcomings and miss out on leveraging some techniques that would otherwise improve its performance and efficiency.

This thesis proposes Hydra, a self-tuning system solution that performs optimization of machine learning algorithms

improving some drawbacks of previous systems by rapidly converging towards the optimum solution without wasting time on bootstrapping the model, using many low-budget evaluations of configurations while applying transfer-learning to enhance the models' performance, ultimately reducing overall costs.

Hydra achieves consistently higher optimum convergence rates than the extended systems in its full-budget sampling variant in spite of having a slower and more complex model, and lower economic-cost. Comparing with BOHB, Hydra achieves 35% cost reduction while still maintain its speed due to the Hyperband structure, and still outperforms BOHB.

References

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi. 2017. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*. 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

[2] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. Cherrypick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, USA) *(NSDI'17)*. 469–482.

[3] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res.* 13 (Feb. 2012), 281–305. http://dl.acm.org/citation.cfm?id=2188385.2188395

[4] Eric Brochu, Vlad Cora, and Nando Freitas. 2010. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *CoRR* abs/1012.2599 (12 2010).

[5] Maria Casimiro, Diego Didona, Paolo Romano, Luís Rodrigues, and Willy Zwanepoel. 2019. Lynceus: Tuning and Provisioning Data Analytic Jobs on a Budget. arXiv:1905.02119 [cs.DC]

[6] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. 2012. *Deep Big Multilayer Perceptrons for Digit Recognition*. Springer Berlin Heidelberg, Berlin, Heidelberg, 581–598. https://doi.org/10.1007/978-3-642-35289-8_31

[7] Diego Didona and Paolo Romano. 2015. Using Analytical Models to Bootstrap Machine Learning Performance Predictors. https://doi.org/10.1109/ICPADS.2015.58

[8] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. *ArXiv* abs/1807.01774 (2018).

[9] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. 2019. Estimation of energy consumption in machine learning. *J. Parallel and Distrib. Comput.* 134 (2019), 75 – 88. https://doi.org/10.1016/j.jpdc.2019.07.007

[10] Jacob R. Gardner, Matt J. Kusner, Zhixiang Xu, Kilian Q. Weinberger, and John P. Cunningham. 2014. Bayesian Optimization with Inequality Constraints. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32* (Beijing, China) *(ICML'14)*. II–937–II–945.

[11] Marc G. Genton. 2001. Classes of Kernels for Machine Learning: A Statistics Perspective. *Journal of Machine Learning Research* 2 (2001), 299–312. http://www.jmlr.org/papers/v2/genton01a.html

[12] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. 2017. Google Vizier: A Service for Black-Box Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. 1487–1495. https://doi.org/10.1145/3097983.3098043

[13] Nikolaus Hansen. 2007. *The CMA Evolution Strategy: A Comparing Review*. Vol. 192. 75–102. https://doi.org/10.1007/3-540-32494-1_4

[14] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-based Optimization for General Algorithm Configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization* (Rome, Italy) *(LION'05)*. 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

[15] Kevin Jamieson and Ameet Talwalkar. 2015. Non-stochastic Best Arm Identification and Hyperparameter Optimization. (02 2015).

[16] Donald Jones. 2001. A Taxonomy of Global Optimization Methods Based on Response Surfaces. *J. of Global Optimization* 21 (12 2001), 345–383. https://doi.org/10.1023/A:1012771025575

[17] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. 2017. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017) (Proceedings of Machine Learning Research, Vol. 54)*. 528–536. http://proceedings.mlr.press/v54/klein17a.html

[18] Anil KumarGoswami, Shalini Gakhar, and Harneet Kaur. 2014. Automatic Object Recognition from Satellite Images using Artificial Neural Network. *International Journal of Computer Applications* 95 (06 2014), 33–39. https://doi.org/10.5120/16633-6502

[19] T. Lai and X. Zheng. 2015. Machine learning based social media recommendation. In *2015 2nd IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM)*. 28–32. https://doi.org/10.1109/ICSDM.2015.7298020

[20] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[21] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 18, 1 (Jan. 2017), 6765–6816. http://dl.acm.org/citation.cfm?id=3122009.3242042

[22] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (Broomfield, CO) *(OSDI'14)*. 583–598.

[23] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature Pyramid Networks for Object Detection. arXiv:1612.03144 [cs.CV]

[24] Pedro Mendes, Maria Casimiro, Paolo Romano, and David Garlan. 2020. TrimTuner: Efficient Optimization of Machine Learning Jobs in the Cloud via Sub-Sampling. arXiv:2011.04726 [cs.LG]

[25] Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*.

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597 [cs.CV]

[27] Alex Sherstinsky. 2018. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. *CoRR* abs/1808.03314 (2018). arXiv:1808.03314 http://arxiv.org/abs/1808.03314

[28] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'12)*. 2951–2959. http://dl.acm.org/citation.cfm?id=2999325.2999464

[29] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 3645–3650. https://doi.org/10.18653/v1/P19-1355