Parallelizing Hyperband for Large-Scale Tuning

Lisha Li[†], Kevin Jamieson[*],
Afshin Rostamizadeh[**], and Ameet Talwalkar[†]
[*]UC Berkeley, [†]Carnegie Mellon University, and [**]Google

1 INTRODUCTION

Although machine learning models have recently achieved dramatic successes in a variety of practical applications, these models are highly sensitive to internal parameters, i.e., *hyperparameters*. In this modern era of machine learning, three trends motivate a new approach to hyperparameter tuning:

(1) **High-dimensional hyperparameter spaces**. Machine learning models are becoming increasingly complex, as evidenced by modern neural networks with dozens of hyperparameters. For such complex models with hyperparameters that interact in unknown ways, a practitioner is forced to evaluate potentially thousands of different hyperparameter settings.
(2) **Increasing training times**. As datasets grow larger and models grow more complex, training a model is becoming dramatically more expensive, often taking days or weeks on specialized high-performance hardware. This trend is particularly onerous in the context of hyperparameter tuning, as a new model must be trained to evaluate each putative hyperparameter configuration.
(3) **Rise of parallel computing**. The combination of a growing number of hyperparameters and longer training time per model preclude the use of sequential hyperparameter tuning methods; we simply cannot wait months or years to find a suitable hyperparameter setting. Thankfully, the expansion of cloud computing resources has made specialized hardware like GPUs and TPUs [1] widely accessible. Leveraging these parallel and distributed computational resources presents an attractive path forward to combat the increasingly challenging problem of hyperparameter optimization.

Our goal is to design a hyperparameter tuning algorithm that can effectively leverage parallel resources. This goal seems trivial with standard methods like random search, where we could train different configurations in an embarrasingly parallel fashion. However, in practice, the number of putative configurations often dwarfs the number of available parallel resources. Hence, we aim to tackle the following problem, which we refer to as the **large-scale regime** for hyperparameter optimization:

Evaluate orders of magnitude more hyperparameter configurations than available parallel workers in a small multiple of the wall-clock time needed to train a single model.

Our main contribution in this work is a practical hyperparameter tuning algorithm for the large-scale regime that exploits parallelism and aggressive early-stopping techniques. We build upon the *Hyperband* algorithm [6], adapting it for the parallel setting through asynchronous promotions. Furthermore, we conduct a thorough empirical study of our algorithm on multiple benchmarks, demonstrating success in the large-scale regime on tasks with up to 500 workers.

2 ALGORITHM

```
Input: $r$, $\eta$ (default $\eta = 3$), $s$
Algorithm async_SHA()
    repeat
        for free worker do
            $(\theta, k)$ = get_job()
            worker performs run_then_return_val_loss($\theta, r\eta^{s+k}$)
        end
        for completed job $(\theta, k)$ with loss $l$ do
            Update configuration $\theta$ in rung $k$ with loss $l$.
        end

Procedure get_job()
    // A configuration in a given rung is
    ‘‘promotable’’ if its validation loss places it in
    the top $1/\eta$ fraction of completed configurations in
    its rung and it has not already been promoted.
    Let $\theta_k$ be the furthest trained promotable configuration
      $\theta$, with $k$ indicating its rung.
    if $\theta_k$ exists then
        Promote $\theta_k$ to rung $k + 1$.
        return $\theta_k, k + 1$
    else
        Add a new configuration $\theta_0$ to bottom rung.
        return $\theta_0, 0$
    end
```

Algorithm 1: Asynchronous Successive Halving Algorithm.

Hyperband [6] calls the Successive Halving algorithm [4, 5], with different early-stopping rates as a subroutine. Li et al. [6] showed that Successive Halving with aggressive early-stopping matches or outperforms Hyperband for a wide array of hyperparameter optimization tasks. Additionally, for modern day hyperparameter tuning problems with high-dimensional search spaces and models with high training cost, aggressive early-stopping is *necessary* for the problem to be tractable. Hence, we focus on adapting the synchronous Successive Halving algorithm (SHA) for the parallel setting. Note that parallelizing Hyperband is trivia after parallelizing SHA—simply take the best performing configuration across multiple brackets of SHA with different early-stopping rates.

Algorithm 1 requires as input a minimum resource r, the reduction factor $\eta > 1$, and the minimum early-stopping rate s. We will refer to trials of SHA with different values of s as *brackets* and, within a bracket, we will refer to each round of promotion as a *rung* with the base rung numbered 0 and increasing. For a given η, only the top $1/\eta$ fraction of configurations are promoted to the next rung. For a given s, a minimum resource of $r\eta^s$ will be allocated to each configuration. Hence, lower s corresponds to more aggressive early-stopping, with $s = 0$ prescribing a minimum resource of r. Another component of the algorithm is the `run_then_return_val_loss`(θ, r) subroutine, which returns the validation loss after training the model with the hyperparameter setting θ and resource allocation r. The subroutine is asynchronous and the code execution of `async_SHA` continues after the job is passed to the worker.

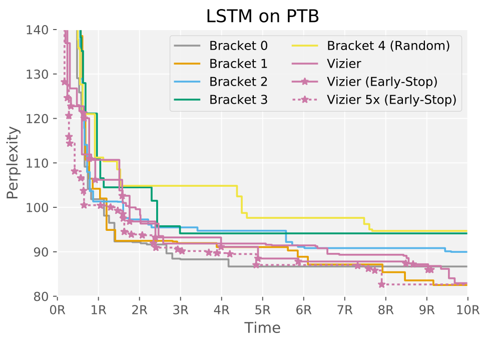
Asynchronous SHA promotes configurations to the next rung whenever possible instead of waiting for a rung to complete before proceeding to the next rung. Additionally, if no promotions are possible, the asynchronous algorithm simply adds a configuration to the base rung, so that more configurations can be promoted to the upper rungs.

Note there is no bound on the number of rungs in a bracket and, therefore, no bound on the maximum resource that can be allocated to a given configuration. However, an upper bound on the maximum resource is often desired (i.e. limit the number of epochs of stochastic gradient descent when training a neural network to prevent overfitting [3]). Algorithm 1 can be easily adapted to enforce a maximum resource per configuration by limiting the number of rungs. Specifically, for a given maximum resource R and bracket s, limit the number of rungs to $\log_\eta(R/r) - s + 1$.

3 EMPIRICAL EVALUATION

We conduct empirical studies on two large-scale benchmarks that take on the order of weeks to run. Both benchmarks show that asynchronous SHA is well suited for the large-scale regime and can successfully find good hyperparameter settings under resource and time constraints. We consider brackets of asynchronous SHA with $\eta = 4$ and a bounded resource per configuration R, with $time(R)$ representing the average time to train a single model. For both tasks, the resources allocated to different configurations is the number of training records, which translates into the number of training iterations after accounting for certain hyperparameters. Finally, due to the computational burden of these experiments, we perform only a single trial of each tuner.

For the first task, we tune a one layer LSTM language model for next word prediction on the Penn Treebank (PTB) dataset [7]. The benchmark has 10 hyperparameters that control both the model architecture as well as the optimization routine. We also evaluate Vizier with and without the performance curve early-stopping rule [2] on this benchmark to provide a baseline for Bayesian optimization methods. Each tuner is given 500 workers and $10 \times time(R)$. We set the minimum resource to $r = R\eta^{-4}$, leading to 5 possible brackets of SHA. The results in Figure 1 show that the brackets with more aggressive early-stopping rates outperform random search. Bracket 0 found a good configuration for this task in $3 \times time(R)$. By that time, random search (Bracket 4) had evaluated 1.5k configurations, compared to 1.8k for Vizier, 2.3k for Vizier with early-stopping, and 52k for Bracket 0, the bracket with the most aggressive early-stopping rate. Our results show bracket 0 and bracket 1 are competitive with Vizier, despite being much simpler and easier to implement. Additionally, whereas Vizier (Early-Stop) uses the heuristic performance curve early-stopping method introduced by Golovin et al. [2], SHA offers a way to perform principled early-stopping. Lastly, for a fair comparison of parallel Hyperband to Vizier, we also show



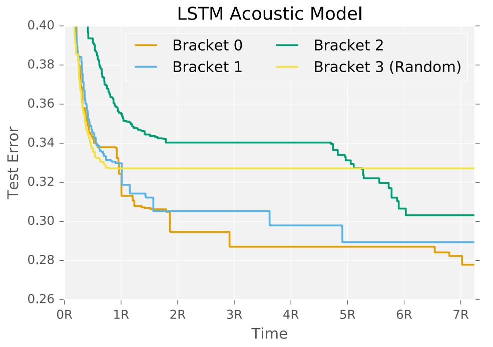


Figure 1: Large-scale experiments that take on the order of weeks to run. The x-axis is measured in units of average time to train a configuration, i.e. $4R$ indicates $4\times$ the time to train an average configuration. Due to the high computational cost, progress for a single trial is shown in each chart.

the performance of Vizier 5× (Early-Stop), which has 2.5k workers, i.e. 5× the resources as that of a single bracket. Remarkably, parallel Hyperband, via bracket 1, matches the performance of Vizier 5× (Early-Stop) without any of the optimization overhead associated with Bayesian methods, using simple random sampling and adaptive resource allocation.

The acoustic modeling task trains an LSTM on a small collection of 250 thousand spoken recordings of anonymized and aggregated typed search queries. The search space we consider has 8 hyperparameters and includes models with up to 5 LSTM layers. Our model uses the LSTM cell introduced in Sak et al. [8] with a recurrent projection layer. Each tuning method is given 20 workers and allowed to run for $7 \times time(R)$. Note that the search space includes models that take nearly the entire time allowance to train. The minimum resource for SHA is set to $r = R\eta^{-3}$ for a total of 4 possible brackets. The results in Figure 1 show a similar relative ordering of the brackets of SHA. Bracket 0 found a configuration with error rate below 30% in $2 \times time(R)$ and evaluated over one thousand configurations in that time with just 20 workers.

REFERENCES

[1] Jeff Dean and Urs Hölzle. Build and train machine learning models on our new google cloud tpus, 2017. URL https://www.blog.google/topics/google-cloud/google-cloud-offer-tpus-machine-learning/.
[2] Daniel Golovin, Benjamin Sonik, Subhodeep Moitra, Greg Kochanski, John Karro, and D.Sculley. Google vizier: A service for black-box optimization. In *KDD*, 2017.

[3] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *ICML*, 2016.
[4] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*, 2015.
[5] Z. Karnin, T. Koren, and O. Somekh. Almost optimal exploration in multi-armed bandits. In *ICML*, 2013.
[6] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv:1603.06560*, 2016.
[7] Mitchell Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.
[8] Haşim Sak, Andrew Senior, and Beaufays Françoise. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *ISCA*, 2014.