PASHA: EFFICIENT HPO WITH PROGRESSIVE RESOURCE ALLOCATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Hyperparameter optimization (HPO) and neural architecture search (NAS) are methods of choice to obtain the best-in-class machine learning models, but in practice they can be costly to run. When models are trained on large datasets, tuning them with HPO or NAS rapidly becomes prohibitively expensive for practitioners, even when efficient multi-fidelity methods are employed. We propose an approach to tackle the challenge of tuning machine learning models trained on large datasets with limited computational resources. Our approach, named PASHA, is able to dynamically allocate maximum resources for the tuning procedure depending on the need. The experimental comparison shows that PASHA identifies well-performing hyperparameter configurations and architectures while consuming significantly fewer computational resources than solutions like ASHA.

1 INTRODUCTION

Hyperparameter optimization (HPO) and neural architecture search (NAS) yield state-of-the-art models, but often are a very costly endeavor, especially when working with large datasets and models. For example, using the results of (Sharir et al., 2020) we can estimate that evaluating 10 configurations for a 340-million-parameter BERT model (Devlin et al., 2019) on the 15GB Wikipedia and Book corpora would cost around $100,000.

To make HPO and NAS more efficient, researchers explored how we can learn from cheaper evaluations (e.g. on a subset of the data) to later allocate more resources only to promising configurations. This created a family of methods often described as multi-fidelity methods. Two well-known algorithms in this family are Successive Halving (SH) (Jamieson & Talwalkar, 2016; Karnin et al., 2013) and Hyperband (HB) (Li et al., 2018).

Multi-fidelity methods significantly lower the cost of the tuning. Li et al. (2018) reported speedups up to 30x compared to standard Bayesian Optimization (BO) and up to 70x compared to random search. Unfortunately, the cost of current multi-fidelity methods is still too high for many practitioners, also because of the large datasets used for training the model. As a workaround, they need to design heuristics which can select a set of hyperparameters or an architecture with a cost comparable to training a single configuration, for example, by training the model with multiple configurations for a single epoch and then selecting the best-performing candidate.

Such heuristics lack robustness and need to be adapted to the specific use-cases in order to provide good results. At the same time, they build on an extensive amount of practical experience suggesting that multi-fidelity methods are often not sufficiently aggressive in leveraging early performance measurements and that identifying the best performing set of hyperparameters (or the best architecture) does not require training a model until convergence. For example, Bornschein et al. (2020) show that it is possible to find the best hyperparameter – number of channels in ResNet-101 architecture (He et al., 2015) for ImageNet (Deng et al., 2009) – using only one tenth of the data.

The aim of our work is to design a method that consumes fewer resources than standard multi-fidelity algorithms such as Hyperband (Li et al., 2018) or ASHA (Li et al., 2020) and nonetheless is able to identify configurations that produce models with a similar predictive performance after being fully re-trained from scratch. In order to do this, we propose a variant of ASHA, called PASHA (Progressive ASHA), that starts with a small amount of initial maximum resources and gradually increases them as needed. ASHA in contrast has a fixed amount of maximum resources. Our

empirical evaluation shows that PASHA can save a significant amount of resources while finding similarly well-performing configurations as conventional ASHA.

To summarize, our contributions are as follows: 1) We introduce a new approach called PASHA that dynamically selects the amount of maximum resources to allocate for HPO or NAS (up to a certain budget), 2) Our empirical evaluation shows the approach significantly speeds up HPO and NAS without sacrificing the performance, and 3) We show the approach can be successfully combined with sample-efficient strategies based on Bayesian Optimization, highlighting the generality of our approach. Our implementation is based on the Syne Tune library (Salinas et al., 2022).

2 Related work

Machine learning systems used in real-world applications often rely on a large number of hyperparameters, and require testing many combinations of them in order to identify the optimal solution. This makes data-inefficient techniques such as Grid Search or Random Search (Bergstra & Bengio, 2012) very expensive in most practical scenarios. Various approaches have been proposed to find good parameters more quickly, and they can be classified into two main families: 1) Bayesian Optimization: evaluates the most promising configurations by modelling their performance. The methods are sample-efficient but they are often designed for environments with limited amount of parallelism; 2) Multi-fidelity: sequentially allocates more resources to configurations with better performance and allows high level of parallelism during the tuning. From these two families, multi-fidelity methods have typically been faster when run at scale and will be the focus of this work. It is also possible to combine ideas from the two families together, for example as done in BOHB by Falkner et al. (2018), and we will test a similar method in our experiments.

Successive halving (Karnin et al., 2013; Jamieson & Talwalkar, 2016) is conceptually the simplest multi-fidelity method. Its key idea is to run all configurations using a small amount of resources, which depends on the minimum resources and the minimum early stopping rate, and then successively promote only a fraction of the most promising configurations to be trained using more resources. Another popular multi-fidelity method, called Hyperband (Li et al., 2018), performs successive halving with different early stopping rates. ASHA (Li et al., 2020) extends the simple and very efficient idea of successive halving by introducing asynchronous evaluation of different configurations, which leads to further practical speedups thanks to better resource allocation.

Related to the problem of efficiency in HPO, cost-aware HPO explicitly accounts for the cost of the evaluations of different configurations. Previous work on cost-aware HPO for multi-fidelity algorithms such as CAHB (Ivkin et al., 2021) keeps a tight control on the budget spent during the HPO process. This is different from our work, as we reduce the budget spent by terminating the HPO procedure early instead of allocating the compute budget in its entirety. Moreover, PASHA could be combined with CAHB to leverage the cost-based resources allocation.

Recently, researchers considered dataset subsampling to speedup the search of the best hyperparameters or architectures. Shim et al. (2021) have combined coresets with PC-DARTS (Xu et al., 2020) and showed that they can find well-performing architectures using only 10% of the data and 8.8x less search time. Similarly, Visalpara et al. (2021) have combined subset selection methods with the Tree-structured Parzen Estimator (TPE) for hyperparameter optimization (Bergstra et al., 2011). With a 5% subset they obtained between an 8x to 10x speedup compared to standard TPE. However, in both cases it is difficult to say in advance what subsampling ratio to use. For example, the 10% ratio in (Shim et al., 2021) incurs no decrease in accuracy, while reducing further to 2% leads to a substantial (2.6%) drop in accuracy.

In practice, it is difficult to find a trade-off between the time required for tuning (proportional to the subset size) and the loss of performance for the final model because these change, sometimes wildly, between datasets. We approach this issue in a principled way using our PASHA algorithm.

Further, Zhou et al. (2020) have observed that for a fixed number of iterations, rank consistency is better if we use more training samples and fewer epochs rather than fewer training samples and more epochs. This observation gives further motivation for using the whole dataset for HPO/NAS and using approaches like PASHA to save computational resources.

3 PROBLEM SETUP

The problem of selecting the best configuration of a machine learning algorithm to be trained is formalized in Jamieson & Talwalkar (2016) as a non-stochastic bandit problem. In this setting the learner (the hyperparameter optimizer) receives N hyperparameter configurations and it has to identify the best performing one with the constraint of not spending more than a fixed amount of resources R (e.g. total number of training epochs) on a specific configuration. R is considered given, but in practice users do not have a good way for selecting it, which can have undesirable consequences: if the value is too small, the model performance will be sub-optimal, while if the budget is too large, the user will incur a significant cost without any practical return. This leads users to overestimate R, setting it to a large amount of resources in order to guarantee the convergence of the model. We maintain the concept of maximum amount of resources in our algorithm but we prefer to interpret R as a "safety net", a cost not to be surpassed (e.g. in case an error prevents a normal behaviour of the algorithm), instead of the exact amount of resources spent for the optimization.

This setting could be extended with additional assumptions, based on empirical observation, removing some extreme cases and leading to a more practical setup. In particular, when working with large datasets we observe that the curve of the loss for configurations (called arms in the bandit literature) continuously decreases (in expectation). Moreover, "crossing points" between the curves are rare (excluding noise), and they are almost always in the initial part of the training procedure. For example, analysis from Domhan et al. (2015) suggests that crossings between curves at later stages of training are relatively rare. Further, Viering & Loog (2021); Mohr & van Rijn (2022) provide an analysis of learning curves and note that in practice most learning curves are well-behaved, with Bornschein et al. (2020) reporting similar findings.

More formally, let us define R as the total number of resources needed to train an ML algorithm to convergence. Given $\Sigma_m(i)$ the ranking of configuration i after using m resources for training, $\exists R^* \ll R : \forall i \in [n], \forall r > R^*, \Sigma_{R^*}(i) = \Sigma_r(i)$. The existence of such a quantity, limited to the best performing configuration, is also assumed by Jamieson & Talwalkar (2016), and it is leveraged to quantify the budget required to identify the best performing configuration. If we knew R^*, it would be sufficient to run all configurations with exactly that amount of resources to identify the best one and then just train the model from scratch with all the data using that configuration. Unfortunately that quantity is unknown and can only be estimated during the optimization procedure.

4 METHOD

Our approach, named PASHA, is an extension of ASHA (Li et al., 2020) inspired by the "doubling trick" (Auer et al., 1995). PASHA targets improvements for hyperparameter tuning on large datasets by hinging on the assumptions made about the crossing points of the learning curves made in Section 3. The algorithm starts by allowing a small initial amount of resources and progressively increases them if the ranking of the configurations in the top two *rungs* (rounds of promotion) has not stabilized. The ability of our approach to stop early automatically is the key benefit. We illustrate the approach in Figure 1, showing how we stop evaluating configurations for additional rungs if their ranking is stable.

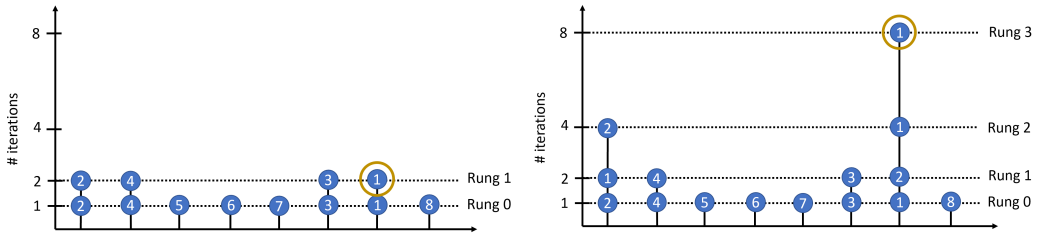

Figure 1: Illustration of how PASHA stops early if the ranking of configurations has stabilized. Left: the ranking of the configurations (displayed inside the circles) has stabilized, so we can select the best configuration and stop the search. Right: the ranking has not stabilized, so we continue.

Algorithm 1 `PASHA()`

input minimum resource r, reduction factor η
$t = 0, R_0 = \eta^2 r$, $K_0 = \lfloor \log_\eta(R_0/r) \rfloor$
while desired **do**
  for each free worker **do**
    $(\theta, k) =$ `get_job()`
    `run_then_return_val_loss`$(\theta, r\eta^k)$
  end for
  for completed job (θ, k) with loss l **do**
    Update configuration θ in rung k with loss l.
    if $k \geq K_t - 2$ **then**
      $\Sigma_k =$ `configuration_ranking`(k)
    end if
    if $k = K_t - 1$ and $\Sigma_k \not\equiv \Sigma_{k-1}$ **then**
      $t = t + 1$
      $R_t = \eta^t R_0$
      $K_t = \lfloor \log_\eta(R_t/r) \rfloor$
    end if
  end for
end while

Algorithm 2 `get_job()`

```
// Check if there is a promotable config.
```
for $k = K_t - 1, \ldots, 1, 0$ **do**
  candidates = `top_k`(rung k, |rung k|$/\eta$)
  promotable = $\{c \in$ candidates : c not promoted$\}$
  if |promotable| > 0 **then**
    return promotable[0], $k + 1$
  end if
  `// If not, grow bottom rung.`
  Draw random configuration θ.
  return $\theta, 0$
end for

We describe the details of PASHA in Algorithm 1 and 2. Given η, a hyperparameter used both in ASHA and PASHA to control how many configurations to prune, PASHA sets the maximum resources to be used for evaluating a configuration using the reduction factor η and the minimum amount of resources r to be used. PASHA increases the maximum number of resources allocated to promising configurations each time the ranking of configurations in the top two rungs becomes inconsistent. For example, if we can currently train configurations up to rung 2 and the ranking of configurations in rung 1 and rung 2 is not consistent, then we allow training part of the configurations up to rung 3, i.e. one additional rung.

The minimum amount of resources r is a hyperparameter to be set by the user. It is significantly easier to set compared to R as r is the minimum amount of resources required to see a meaningful difference in the performance of the models, and it can be easily estimated empirically by running a few small-scale experiments.

Given that deep learning algorithms typically rely on stochastic gradient descent, ranking inconsistencies can occur between similarly performing configurations. Hence, we need some benevolence in estimating the ranking. We propose to use a soft ranking approach as detailed shortly where we group configurations based on their validation performance metric (e.g., accuracy). For practical purposes we also set a maximum amount of resources R so that PASHA can default to ASHA if needed and avoid increasing the resources indefinitely. While it is not generally reached, it provides a safety net in case some components are misconfigured or the configurations do not reflect the expectations of the user.

SOFT RANKING

In soft ranking, configurations are still sorted by predictive performance but they are considered equivalent if the performance difference is smaller than a value ϵ (or equal to it). Instead of producing a sorted list of configuration, this provides a list of lists where for every position of the ranking there is a list of equivalent configurations. The concept is explained graphically in Figure 2, and we also provide a formal definition. For a set of n configurations $c_1, c_2, \cdots, c_i, \cdots, c_n$ and performance metric f (e.g. accuracy) with $f(c_1) \leq f(c_2) \leq \cdots \leq f(c_i) \leq \cdots \leq f(c_n)$, soft rank at position i is defined as

$$\text{soft rank}_i = \{c \in \text{configurations} : |f(c_i) - f(c)| \leq \epsilon\}\,.$$

When deciding on if to increase the resources, we go through the ranked list of configurations in the top rung and check if the current configuration at the given rank was in the list of configurations for that rank in the previous rung. If there is a configuration which does not satisfy the condition, we increase resources.

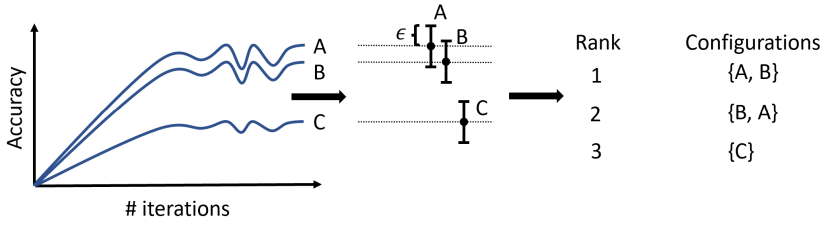

Figure 2: Illustration of soft ranking. There are three lists with the first two containing two items because the scores of the two configurations are closer to each other than ϵ.

AUTOMATIC ESTIMATION OF ϵ BY MEASURING NOISE IN RANKINGS

We provide an intuitive yet principled method to automatically estimate a suitable value of ϵ based on the noise in rankings of configurations. Our intuition is that if two configurations repeatedly swap their rankings, then they perform similarly well and the performance difference in the current epoch or rung is simply due to noise. We want to measure this noise and use it to automatically estimate the value of ϵ.

Formally we can define a set of pairs of configurations that perform similarly well by the following:

$$\begin{aligned} S : \{(c, c') : &(\Sigma_{r_1}(c) > \Sigma_{r_1}(c') \wedge \Sigma_{r_2}(c) < \Sigma_{r_2}(c') \wedge \Sigma_{r_3}(c) > \Sigma_{r_3}(c')) \\ &\vee (\Sigma_{r_1}(c) < \Sigma_{r_1}(c') \wedge \Sigma_{r_2}(c) > \Sigma_{r_2}(c') \wedge \Sigma_{r_3}(c) < \Sigma_{r_3}(c'))\} \end{aligned} \tag{1}$$

for resources $r_1 > r_2 > r_3$ (in practice epochs). We only consider configurations that made it to the latest rung, so $r\eta^{K_t-1} \geq r_1 > r\eta^{K_t-2}$. The value of ϵ can then be calculated as the N-th percentile (we use $N = 90$) of distances between the performances of configurations in S:

$$\epsilon = \text{P}_{N,(c,c')\in S}|f_{r_1}(c) - f_{r_1}(c')|.$$

To uniquely define f_{r_1}, we take the maximum r_1 currently available. The value of ϵ is recalculated every time we receive new information about the performances of configurations.

5 EXPERIMENTS

In this section we quantify the advantage provided by PASHA. Its goal is not to provide a model with a higher accuracy, but to identify the best configuration in a shorter amount of time so that we can then re-train the model from scratch. Overall, we target a significantly faster tuning time and on-par predictive performance when comparing with the models identified by state-of-the-art optimizers like ASHA. Re-training after HPO or NAS is important because HPO and NAS in general require to

reserve a significant part of the data (often around 20 or 30%) to be used as a validation set. Training with fewer data is not desirable because in practice it is observed that training a model on the union of training and validation sets provides better results.

We tested our method on two different sets of experiments. The first set evaluates the algorithm on NAS problems and employs NASBench201 (Dong & Yang, 2020), while the second set focuses on hyperparameter optimization and is ran over two large-scale datasets from PD1 benchmark (Wang et al., 2021).

5.1 SETUP

Our experimental setup consists of two phases: 1) run the hyperparameter optimizer until $N = 256$ candidate configurations are evaluated; and 2) use the best configuration identified in the first phase to re-train the model from scratch. For the purpose of these experiments we re-train all the models using only the training set. This avoids introducing an arbitrary choice on the validation set size and allows us to leverage standard benchmarks such as NASBench201. In real-world applications the model can be trained on both training and validation sets. All our results report only the time invested in identifying the best configuration since the re-training time is comparable for all optimizers. All results are averaged over multiple repetitions, with the details specified for each set of experiments separately.

We use 4 workers to perform evaluations in parallel and asynchronously. The choice of R is sensitive for ASHA since it can make the optimizer consume too many resources and penalize the performance of the algorithm. To have a fair comparison, we make R dataset-dependent adopting the maximum amount of resources in the considered benchmarks. r is also dataset dependent and η, the halving factor, is set to 3 unless otherwise specified. The same values are used for both ASHA and PASHA.

We compare PASHA with ASHA (Li et al., 2020), a recent state-of-the-art approach for hyperparameter optimization, and other relevant baselines. In particular, we consider "one-epoch baseline" that trains all configurations for one epoch (the minimum available resources) and then selects the most promising configuration, and "random baseline" that randomly selects the configuration. For our one-epoch baseline we sample $N = 256$ configurations, using the same scheduler and seeds as for PASHA and ASHA. For our random baseline we sample $N = 2560$ configurations to obtain a better performance estimate of a model with random configuration.

5.2 NAS EXPERIMENTS

For our NAS experiments we leverage the well-known NASBench201 (Dong & Yang, 2020) benchmark. The task is to identify the network structure providing the best accuracy on three different datasets (CIFAR-10, CIFAR-100 and ImageNet16-120) independently. We use $r = 1$ epoch and $R = 200$ epochs. We repeat the experiments using 5 random seeds for the scheduler and 3 random seeds for NASBench201 (all that are available), resulting in 15 repetitions. Some configurations in NASBench201 do not have all seeds available, so we impute them by averaging over the available seeds. To measure the predictive performance we report the best accuracy on the combined validation and test set provided by the creators of the benchmark.

The results in Table 1 suggest that PASHA consistently leads to strong improvements in runtime, while achieving similar accuracy values as the baseline algorithm ASHA. The one-epoch baseline has noticeably worse accuracies than ASHA or PASHA, suggesting that PASHA does a good job of deciding when to continue increasing the resources – it does not stop too early. Random baseline is a lot worse than the one-epoch baseline, so there is value in performing NAS. We also report the maximum resources used to find how early the ranking becomes stable in PASHA.

It is important to observe that the time required to train a model is about 1.3h for CIFAR-10 and CIFAR-100, and about 4.1h for ImageNet16-120, making the total tuning time required by PASHA comparable or faster than the one required for training.

We also ran additional experiments testing PASHA with alternative ranking functions, a reduction factor of $\eta = 2$ and $\eta = 4$ instead of $\eta = 3$, and the usage of PASHA as a scheduler in MOBSTER (Klein et al., 2020). These experiments provided similar findings as the above and are described next.

Table 1: NASBench201 results. PASHA leads to large improvements in runtime, while achieving similar accuracy as ASHA.

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-10	ASHA	93.85 ± 0.25	3.0h ± 0.6h	1.0x	200.0 ± 0.0
	PASHA	93.57 ± 0.75	1.3h ± 0.6h	2.3x	36.1 ± 50.0
	One-epoch baseline	93.30 ± 0.61	0.3h ± 0.0h	8.5x	1.0 ± 0.0
	Random baseline	72.93 ± 19.55	0.0h ± 0.0h	NA	0.0 ± 0.0
CIFAR-100	ASHA	71.69 ± 1.05	3.2h ± 0.9h	1.0x	200.0 ± 0.0
	PASHA	71.84 ± 1.41	0.9h ± 0.4h	3.4x	20.5 ± 48.3
	One-epoch baseline	65.57 ± 5.53	0.3h ± 0.0h	9.2x	1.0 ± 0.0
	Random baseline	42.98 ± 18.34	0.0h ± 0.0h	NA	0.0 ± 0.0
ImageNet16-120	ASHA	45.63 ± 0.81	8.8h ± 2.2h	1.0x	200.0 ± 0.0
	PASHA	45.13 ± 1.51	2.9h ± 1.7h	3.1x	21.3 ± 48.1
	One-epoch baseline	41.42 ± 4.98	1.0h ± 0.0h	8.8x	1.0 ± 0.0
	Random baseline	20.97 ± 10.01	0.0h ± 0.0h	NA	0.0 ± 0.0

5.2.1 Reduction factor

An important parameter for the performance of multi-fidelity algorithms like ASHA is the reduction factor used in the algorithm. This hyperparameter controls the fraction of pruned candidates at every rung. The optimal theoretical value is e and the most common values used in practice are 2 and 3. In Table 2 we report the results of the different algorithms ran with $\eta = 2$ and $\eta = 4$. The results show that the gains provided by PASHA are consistent also for $\eta = 2$ and $\eta = 4$.

Table 2: NASBench201 results with various reduction factors η.

Dataset	Reduction factor	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-10	$\eta = 2$	ASHA	93.88 ± 0.27	3.6h ± 1.1h	1.0x	200.0 ± 0.0
		PASHA	93.53 ± 0.76	1.0h ± 0.3h	3.5x	9.1 ± 8.1
	$\eta = 4$	ASHA	93.75 ± 0.28	2.4h ± 0.6h	1.0x	200.0 ± 0.0
		PASHA	93.65 ± 0.65	1.1h ± 0.5h	2.3x	32.3 ± 50.2
CIFAR-100	$\eta = 2$	ASHA	71.67 ± 0.84	3.8h ± 1.0h	1.0x	200.0 ± 0.0
		PASHA	71.65 ± 1.42	0.9h ± 0.1h	4.2x	5.9 ± 2.0
	$\eta = 4$	ASHA	71.43 ± 1.13	2.7h ± 0.9h	1.0x	200.0 ± 0.0
		PASHA	72.09 ± 1.22	1.0h ± 0.4h	2.8x	25.1 ± 49.0
ImageNet16-120	$\eta = 2$	ASHA	46.09 ± 0.68	11.9h ± 4.0h	1.0x	200.0 ± 0.0
		PASHA	45.35 ± 1.52	2.8h ± 0.6h	4.2x	9.3 ± 7.1
	$\eta = 4$	ASHA	45.43 ± 0.98	7.9h ± 3.0h	1.0x	200.0 ± 0.0
		PASHA	45.52 ± 1.30	2.9h ± 1.1h	2.8x	18.4 ± 18.7

5.2.2 Bayesian Optimization

Bayesian Optimization combined with multi-fidelity methods such as Successive Halving can improve the predictive performance of the final model (Klein et al., 2020). In this set of experiments, we verify PASHA can speedup also these kinds of methods. Our results are reported in Table 3, where we can clearly see PASHA obtains a similar accuracy result as ASHA with significant speedup.

Table 3: NASBench201 results for ASHA with Bayesian Optimization searcher – MOBSTER (Klein et al., 2020) and similarly extended version of PASHA. The results show PASHA can be successfully combined with a smarter configuration selection strategy.

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-10	ASHA BO	94.21 ± 0.28	5.0h $\pm$ 1.1h	1.0x	200.0 ± 0.0
	PASHA BO	94.00 ± 0.20	2.6h $\pm$ 1.8h	2.0x	70.7 ± 81.6
CIFAR-100	ASHA BO	72.79 ± 0.68	5.7h $\pm$ 1.4h	1.0x	200.0 ± 0.0
	PASHA BO	72.16 ± 1.07	1.6h $\pm$ 0.5h	3.7x	13.0 ± 8.7
ImageNet16-120	ASHA BO	46.21 ± 0.70	15.1h $\pm$ 4.0h	1.0x	200.0 ± 0.0
	PASHA BO	45.36 ± 1.06	3.9h $\pm$ 1.2h	3.9x	11.8 ± 7.9

5.3 HPO EXPERIMENTS

We further utilize the PD1 HPO benchmark (Wang et al., 2021) to show the usefulness of PASHA in large-scale settings. In particular, we take WMT15 German-English (Bojar et al., 2015) and ImageNet (Deng et al., 2009) datasets that use xformer (Lefaudeux et al., 2021) and ResNet50 (He et al., 2015) models. Both of them are datasets with a large amount of training examples, in particular WMT15 German-English has about 4.5M examples, while ImageNet has about 1.3M examples.

The task in PD1 is to optimize four hyperparameters that influence the training of the models: base learning rate η, momentum β, polynomial learning rate decay schedule power p and decay steps fraction λ. The details of the search space are described in Table 4. The minibatch size used for WMT experiments is 64, while the minibatch size for ImageNet experiments is 512. There are 1414 epochs available for WMT and 512 for ImageNet. Note that there are also other datasets available in the PD1 benchmark, but these only have a small number of epochs. As a result there would not be enough rungs to see benefits of the early stopping provided by PASHA.

PD1 is a surrogate benchmark, so we use KNN regression to obtain the performance of sampled configurations. We use the performance of the closest neighbour configuration ($N = 1$). We repeat our experiments using 5 random seeds (same seeds as for NASBench201) and there is only one dataset seed available.

Table 4: Search space for HPO on PD1 benchmark (Wang et al., 2021).

Hyperparameter	Range	Scaling
η	$[10^{-5}, 10.0]$	Log
$1 - \beta$	$[10^{-3}, 1.0]$	Log
p	$[0.1, 2.0]$	Linear
λ	$[0.01, 0.99]$	Linear

The results in Table 5 show that PASHA leads to large speedups on both WMT and ImageNet datasets. The speedup is particularly impressive for WMT where it is about 15.5x, highlighting how PASHA can significantly accelerate the HPO search on datasets with millions of training examples (WMT has about 4.5M training examples). The one-epoch baseline obtains similar accuracy as ASHA and PASHA for WMT, but performs significantly worse on ImageNet dataset. This result suggests that simple approaches such as the one-epoch baseline are not robust and solutions such as PASHA are needed (which we also saw on NASBench201). Selecting the hyperparameters randomly leads to significantly worse performance than any of the other approaches.

Table 5: Results of the hyperparameter optimization experiments on WMT and ImageNet tasks from PD1 benchmark. Mean and std. of the best validation accuracy (or its equivalent as specified in the PD1 benchmark) across 5 random seeds.

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
WMT	ASHA	62.72 ± 1.41	43.7h ± 37.2h	1.0x	1357.4 ± 80.4
	PASHA	62.04 ± 2.05	2.8h ± 0.6h	15.5x	37.8 ± 21.6
	One-epoch baseline	62.36 ± 1.40	0.6h ± 0.0h	67.3x	1.0 ± 0.0
	Random baseline	33.51 ± 21.54	0.0h ± 0.0h	NA	0.0 ± 0.0
ImageNet	ASHA	75.10 ± 2.03	7.3h ± 1.2h	1.0x	251.0 ± 0.0
	PASHA	73.37 ± 2.71	3.8h ± 1.0h	1.9x	45.0 ± 30.1
	One-epoch baseline	63.40 ± 9.91	1.1h ± 0.0h	6.7x	1.0 ± 0.0
	Random baseline	35.18 ± 31.31	0.0h ± 0.0h	NA	0.0 ± 0.0

6 Discussion

PASHA is an algorithm that is designed to make HPO and NAS more accessible to machine learning practitioners as it significantly lowers the time and cost required to find well-performing configurations. Its limitation is similar to that of ASHA and other multi-fidelity methods: it may early prune configurations that would perform well if they were trained for longer before making a stopping decision.

While we used PASHA on benchmarks that define resources in terms of epochs, it is not needed to set the resources on the level of epochs. In fact, in many large datasets it will be sufficient to train the model only for a small number of epochs. In these cases it can be useful to specify the rung levels in terms of iterations or data points processed so that we can still evaluate if to increase the resources multiple times. PASHA could then find the best hyperparameters even without seeing all of the training examples.

7 Conclusions

In this work we introduced a new variant of Successive Halving, called PASHA, that progressively increases the resources allocated to promising configurations. PASHA considers the rankings of configurations at successive rung levels and if the rankings have stabilized, stops the HPO procedure, returning the best configuration found. The approach is easy to use as it automatizes the choice of its internal parameters in an intuitive yet principled way based on noise in rankings of configurations.

Despite its simplicity, PASHA has led to strong improvements in the tuning time. For example, in many cases it has reduced the time needed to about one third compared to ASHA without a noticeable impact on the quality of the found configuration. In the case of the largest dataset with millions of training examples it has reduced the tuning time by a factor of about 15x. We have also demonstrated it is possible to successfully combine PASHA with more advanced search strategies based on Bayesian Optimization to obtain improvements in accuracy at a fraction of the time.

8 Reproducibility Statement

We release the code for our approach, including details for how to run the experiments. We use pre-computed benchmarks that make it possible to run the NAS and HPO experiments even without large computational resources.

REFERENCES

Peter Auer, Nicoló Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, pp. 322–331, 1995.

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

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, 2011.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Barry Haddow, Matthias Huck, Chris Hokamp, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Carolina Scarton, Lucia Specia, and Marco Turchi. Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, 2015. doi: 10.18653/v1/W15-3001.

Jorg Bornschein, Francesco Visin Visin, and Simon Osindero. Small data, big decisions: Model selection in the small-data regime. In *ICML*, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-jia Li, Kai Li, and Li Fei-fei. Imagenet: A large-scale hierarchical image database, 2009. URL `http://www.image-net.org/`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *ACL*, 2019. ISBN 1810.04805v2. URL `https://github.com/tensorflow/tensor2tensor`.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *ICML*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2015.

Nikita Ivkin, Zohar Karnin, Valerio Perrone, and Giovanni Zappella. Cost-aware adversarial best arm identification. In *ICLR NAS workshop*, 2021.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*, 2016.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pp. 1238–1246, 2013.

Aaron Klein, Louis C. Tiao, Thibaut Lienart, Cedric Archambeau, and Matthias Seeger. Model-based asynchronous hyperparameter and neural architecture search. In *arXiv*, 2020.

Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, and Susan Zhang. xformers: A modular and hackable transformer modelling library. `https://github.com/facebookresearch/xformers`, 2021.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *MLSys*, 2020.

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 2018. URL `http://arxiv.org/abs/1603.06560`.

Felix Mohr and Jan N. van Rijn. Learning curves for decision making in supervised machine learning - a survey. In *arXiv*, 2022.

David Salinas, Matthias Seeger, Aaron Klein, Valerio Perrone, Martin Wistuba, and Cedric Archambeau. Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *First Conference on Automated Machine Learning (Main Track)*, 2022. URL https://openreview.net/forum?id=BVeGJ-THIg9.

Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. In *arXiv*, 2020.

Jae-hun Shim, Kyeongbo Kong, and Suk-Ju Kang. Core-set sampling for efficient neural architecture search. In *ICML Workshops*, 2021.

Tom Viering and Marco Loog. The shape of learning curves: a review. In *arXiv*, 2021.

Savan Visalpara, Krishnateja Killamsetty, and Rishabh Iyer. A data subset selection framework for efficient hyper-parameter tuning and automatic machine learning. In *ICML Workshops*, 2021.

Zi Wang, George E. Dahl, Kevin Swersky, Chansoo Lee, Zelda Mariet, Zachary Nado, Justin Gilmer, Jasper Snoek, and Zoubin Ghahramani. Pre-trained Gaussian processes for Bayesian optimization. In *arXiv*, 2021.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *ICLR*, 2020.

Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. EcoNAS: Finding proxies for economical neural architecture search. In *CVPR*, 2020.