Improving Model Training with Multi-fidelity Hyperparameter Evaluation

Yimin Huang [1] Yujun Li [1] Hanrong Ye [2] Zhenguo Li [1] Zhihua Zhang [3]

Abstract

The evaluation of hyperparameters, neural architectures, or data augmentation policies becomes a critical problem in advanced deep model training with a large hyperparameter search space. In this paper, we propose an efficient and robust bandit-based algorithm called Sub-Sampling (SS) in the scenario of hyperparameter search evaluation and its modified version for high GPU usage. It evaluates the potential of hyperparameters by the sub-samples of observations and is theoretically proved to be optimal under the criterion of cumulative regret. We further combine SS with Bayesian Optimization and develop a novel hyperparameter optimization algorithm called BOSS. Empirical studies validate our theoretical arguments of SS and demonstrate the superior performance of BOSS on a number of applications, including Neural Architecture Search (NAS), Data Augmentation (DA), Object Detection (OD), and Reinforcement Learning (RL).

1 Introduction

Hyperparameter optimization, a classical problem for model selection, receives increasing attention due to the rise of automated machine learning, where not only traditional hyperparameters such as learning rates (Zeiler, 2012), but also neural architectures (Liu et al., 2019), data augmentation policies (Cubuk et al., 2018) and other plausible variables are to be set automatically (Elsken et al., 2019; Zöller & Huber, 2019). This can be perceived as a search problem in a large hyperparameter search space – the goal is to find the hyperparameters that maximize the generalization capability of the model if trained with the searched hyperparameters. There are mainly three challenges: the initial design (Jones et al., 1998; Konen et al., 2011; Brockhoff et al., 2015; Zhang et al., 2019), the sampling method (Rasmussen, 2003; Hutter et al., 2011; Bergstra et al., 2011; Springenberg et al., 2016; Srinivas et al., 2010; Hennig & Schuler, 2012; Hernández-Lobato et al., 2014; Wang & Jegelka, 2017; Ru et al., 2018), and the evaluation method (Thornton et al., 2013; Karnin et al., 2013; Jamieson & Talwalkar, 2016; Li et al., 2016; Falkner et al., 2018).

In the literature, initial designs including Latin hypercube design (Jones et al., 1998) and orthogonal array (Zhang et al., 2019) are used to replace random initialization for improving experimental effect and reducing the time required for convergence. One straightforward idea of sampling methods is to use grid search (Wu & Hamada, 2011) or random search (Bergstra & Bengio, 2012). However, the required number of function evaluations, which involve model training and thus computationally expensive, grows exponentially with the number of hyperparameters. Population-based methods (Hansen, 2016; Jaderberg et al., 2017) such as genetic algorithms, evolutionary strategies, and particle swarm optimization use guided search and usually outperform random search. These algorithms are often time-consuming for the evolution of the population. To develop efficient sampling strategies, model-based methods are paid more attention in which a surrogate model of the objective function is built, allowing to choose the next hyperparameters to evaluate in an informed manner. Among them, Bayesian optimization (BO) becomes popular with different probabilistic surrogate models, such as Gaussian processes (GPs) (Snoek et al., 2012), random forests (Hutter et al., 2011), or Tree-structure Parzen Estimator (TPE) (Bergstra et al., 2011). These model-based methods are shown to outperform random search in terms of sample efficiency (Thornton et al., 2013; Snoek et al., 2015). In this paper we focus on the hyperparameter evaluation problem. In many machine learning problems such as Neural Architecture Search (NAS) and Data Augmentation (DA), neural architectures and augmentation policies can always be parameterized. In most scenarios, the performance of a hypothesis relies heavily on these hyperparameters. And each evaluation of hyperparameter configurations is expensive. Thus, an efficient evaluation method is a key point in hyperparameter optimization (HPO). A general solution for fast, accurately

[1]Huawei Noah's Ark Lab, China [2]Department of Computer Science and Engineering, HKUST, HongKong, China [3]School of Mathematical Sciences, Peking University, Beijing, China. Correspondence to: Yimin Huang <yimin.huang@huawei.com>.

Proceedings of the 5^{th} *MLSys Conference*, Santa Clara, CA, USA, 2022. Copyright 2022 by the author(s).

and robustly evaluating these hyperparameters, neural architectures or augmentation policies from a large search space is desirable.

While the many previous studies focus on sampling strategies, multi-fidelity methods concentrate on accelerating function evaluation which could be extremely expensive for big data and complex models (Krizhevsky et al., 2012). These methods use low-fidelity results derived with small amount of resources to select configurations quickly. Details refer to Feurer & Hutter (2018). The successive halving (SH) algorithm (Jamieson & Talwalkar, 2016) is proposed to identify the best configuration among K configurations. It evaluates all hyperparameter configurations, throws the worst half, doubles the budgets and repeats until one configuration left. Note that the number of configurations K is required as an input, but there is no guidance for choosing a proper K. HyperBand (HB) (Li et al., 2016) addresses this trade-off by implementing the SH algorithm with multiple values of K, with each such run of SH as a "bracket". Falkner et al. (2018) further combined Bayesian optimization (BO) and HyperBand (HB), termed BOHB, which extracted the information in the current bracket to fit a Bayesian surrogate model for sampling configurations in the next bracket.

However, it is worth noting that the HB algorithm was proposed for identifying the best configuration among the alternative set, not for collecting data to estimate the sampling model in BO. Hence, combining BO and HB directly is not appropriate, which will be addressed in our work. These highly relevant works will be explained in more detail in Appendix A. Recently, there are many interesting work about HPO. Paul et al. (2019) proposed an algorithm for tuning hyperparameters of the policy gradient methods in RL. Keshtkaran & Pandarinath (2019) tuned hyperparameters of a sequential autoencoder for spiking neural data. Law et al. (2019) learned a joint model on hyperparameters and data representations from many previous tasks. Li et al. (2020) considered hyperparameter selection when fine-tuning from one domain to another. Klein et al. (2019) presented a meta-surrogate model for task generation for HPO which facilitates developing and benchmarking of HPO algorithms.

In this paper we focus on the need for collecting high-quality data with multi-fidelity methods. We propose a new efficient and general-purpose algorithm for fast evaluation called Sub-Sampling (SS). Compared with the HB algorithm, SS evaluates the potential of the configurations based on the sub-samples of observations. It guarantees the perfect overall performance in the sence that the cumulative regret is asymptoticallly optimal. Appendix C provides theoretical analysis and Figure 1 compares SS and SH on one example. SS has a jump in the figure since its criterion is based on the potential rather than the current performance of the configurations as in SH.

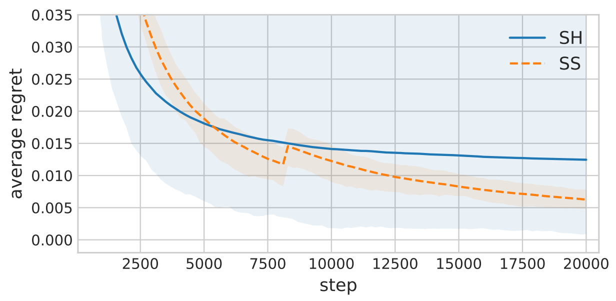

Figure 1. Using SS and SH as policies for Multi-Armed Bandit (MAB) problems (Section 5.1). The shaded part indicates the min/max ranges.

Further, we combine BO and SS to deal with many popular machine learning tasks in Section 5. The estimation of the surrogate model in the BO framework is more accurate based on these promising data obtained by SS. In the next bracket, we can sample better configurations from the model with higher probability. As a result, the trajectory of the search procedure is more reliable. Extensive experiments on various problems show the superior performance of BOSS, including Neural Architecture Search (NAS), Data Augmentation (DA), bounding box scaling in Object Detection (OD), and tuning hyperparameters of PPO (Schulman et al., 2017) in Reinforcement Learning (RL).

2 Proposed Sub-Sampling for Efficient Hyperparameter Evaluation

In this section, we propose a novel efficient nonparametric solution called Sub-Sampling (SS) for evaluating a pool of configurations $\mathcal{C} = \{c_1, \dots, c_K\}$ that minimizes the total regrets. The main idea is to assign more budgets to the configuration of more potential. Let $Y_t^{(k)}$ be the validation loss of c_k at the t-th evaluation, n_k be the number of evaluations of c_k so far, and $n = \sum_{i=1}^{K} n_k$ be the total number of evaluations for all the K configurations. To compare configurations for selecting next ones to evaluate with designed budgets, SS uses all available data $\{Y_t^{(k)}\}_{t=1}^{n_k}$ generated so far from different rounds with different initializations[1] to measure the potential of configurations while SH just uses the data $Y_{n_k}^{(k)}$ from the current round. Note that in the early stage of network training, the performance relies strongly on the initialization which has high randomness. SS uses data which has n_k samples of c_k for each k to reduce the impact

[1] $Y_t^{(k)}$ is the accuracy of the corresponding network trained with the certain budgets used in this round and the t-th initialization.

of randomness. This reduces probability of misjudging in SS.

Specifically, c_k is said to has more potential than $c_{k'}$, denoted by $c_k \preccurlyeq c_{k'}$, if $n_k < n_{k'}$ and (a) $n_k < q_n$ or (b) $q_n \leq n_k$ and $\bar{Y}^{(k)}_{1:n_k} \leq \bar{Y}^{(k')}_{j:(j+n_k-1)}$, for some $1 \leq j \leq n_{k'} - n_k + 1$, where $\bar{Y}^{(k)}_{l:u} = \sum_{v=l}^{u} Y^{(k)}_v/(u-l+1)$. In this definition, n_k sub-samples from $n_{k'}$ samples of $c_{k'}$ are used to compare c_k and $c_{k'}$, thus called sub-sampling. The parameter q_n is given to balance exploration and exploitation. It is a nonnegative increasing threshold for SS such that $q_n = o(\log n)$ and $q_n/\log\log n \to \infty$ as $n \to \infty$.

In case (a), it reveals that the performance of c_k is observed too little, so we cannot judge its potential. Thus, c_k needs to be explored. In case (b), it indicates that c_k has the potential to exceed $c_{k'}$, although the current performance $Y^{(k)}_{n_k}$ may be worse than $Y^{(k')}_{n_{k'}}$. Hence, c_k needs to be exploited with more budgets.

Algorithm 1 Sub-Sampling

```
Input: The set of K configurations $\mathcal{C} = \{c_1, \dots, c_K\}$; maximum budget $R$; minimum budget $b$; ratio $\eta$ (default $\eta = 3$).
Output: $\{c_{\hat{\pi}_1}, \dots, c_{\hat{\pi}_N}\}$ with corresponding evaluations.
1: $r = 1$, evaluate all configurations with budget $b$.
2: for $r = 2, 3, \dots, [\log_\eta(R/b)]$ do
3:     Select the leader $c_\zeta$, which has the most observations.
4:     $\mathcal{I}' = \{k : c_k \in \mathcal{C}\backslash c_\zeta,\ c_k \preccurlyeq c_\zeta\}$.
5:     if $\mathcal{I}' == \varnothing$ then
6:         Evaluate $c_\zeta$ with budgets $\eta^r b$.
7:     else
8:         Evaluate $c_k$ with budgets $\eta^r b$ for each $k \in \mathcal{I}'$.
9:     end if
10: end for
```

The proposed SS method is described in Algorithm 1. The sequence of the configurations $\hat{\pi}_1, \dots, \hat{\pi}_N \in \mathcal{I} = \{1, 2, \dots, K\}$ is a sampling strategy for HPO, where $N = \sum_{k=1}^{K} N_k$ is the number of total observations. This strategy $\hat{\pi} = \{\hat{\pi}_t\}$ is a sequence of random variables $\hat{\pi}_t \in \{1, 2, \dots, K\}$ denoting that at each time $t = 1, 2, \dots, N$, the $\hat{\pi}_t$-th configuration is selected to evaluate. Let r denote the round number. In the first round, all configurations are evaluated with minimum budget b since there is no information about them. In round $r \geq 2$, we select the leader of configurations which is the one evaluated the most times, and the budgets increase as $\eta^r b$. If two configurations have the same number of observations n_k, we choose the one with lower $\bar{Y}^{(k)}_{1:n_k}$ as the leader. In each round $r \geq 2$, non-leaders will be evaluated, if they have more potential than the leader, otherwise, the leader will be evaluated.

SS vs SH

In HPO problems, the randomness of the early observations comes from the randomness of initialization, since the performance of an under-trained neural network with small budgets strongly depends on its initialization. Thus, the comparisons at the first stage of SH are unreliable and the abandoned configurations have no chance to be evaluated again. In contrast, SS uses data of all stages to weaken the impact of initial value and always gives a chance for each configuration to make the comparison of potential. Consequently, SS can get more reliable results. In Section 5.1, Table 1 shows that SS has higher accuracy of finding the optimal configuration than SH especially in the case of big randomness.

Both SS an SH are different from the UCB-based procedures (Burtini et al., 2015) which must know the underlying probability distributions to measure the potential of a configuration by an upper confidence bound of the observation value. However, only SS achieves asymptotically optimal efficiency. These theoretical results are discussed in Appendix C.

3 Bayesian Optimization via Sub-Sampling

In this section, we propose a novel algorithm called BOSS, which combines Bayesian Optimization (BO) and Sub-Sampling (SS), to search out the optimal configurations efficiently and reliably.

Bayesian optimization is a sequential design strategy for optimizing black-box functions. In hyperparameter optimization problems, the validation performance of a machine learning algorithm can be regarded as a function $f : \mathcal{X} \to \mathbb{R}$ of hyperparameters $x \in \mathcal{X}$ and the goal is to find the optimal $x_* \in \arg\min_{x\in\mathcal{X}} f(x)$. In most cases, $f(x)$ does not admit an analytic form, which is approximated by a surrogate model in BO, such as Gaussian processes, random forests, or TPE, based on the data collected on the fly $D_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ where $y_i = f(x_i) + \varepsilon(b)$ and the error term $\varepsilon(b)$ is dependent on the budget b and satisfies that $\lim_{b\to\infty} \varepsilon(b) = 0$. The BO methods differ in different surrogate models for the conditional probability $p(y|x, D_n)$. In this work, we adopt TPE (Bergstra et al., 2011) as the surrogate model, which utilizes a kernel density estimator to model the data densities to deal with both discrete and continuous hyperparameters simultaneously. The specific procedure of TPE with the expected improvement (EI) as the acquisition function is given as follows.

This strategy models $p(x|y, D_n)$ instead of $p(y|x, D_n)$ and uses two densities below to estimate:

$$p(x|y, D_n) = \begin{cases} \ell(x|D_n) & \text{if} \quad y < \alpha \\ g(x|D_n) & \text{if} \quad y \geq \alpha, \end{cases} \tag{1}$$

where α is a given demarcation point and its value will be discussed later. $\ell(x)$ is the density satisfying that the observation y was less than α and is assumed to be irrelevant with the specific value of y. It is estimated by a kernel density estimator with the observations $\{x_i\}$ such that corresponding observation y_i was less than α. And, $g(x)$ is the similar density formed by using the remaining observations.

According to the model, we set an acquisition function to sample next point x that is most likely to be optimal. A common acquisition function is the expected improvement (EI): $a_\alpha(x) = \mathbb{E}_{p(y|x,D_n)}(\alpha - y)\mathbb{I}(y \leq \alpha)$, where $\mathbb{I}$ is the indicator function. Bergstra et al. (2011) showed that maximizing the EI function is equivalent to maximizing the ratio $\ell(x|D_n)/g(x|D_n)$ of densities in Equation (1).

Note that the observations usually represent the value of a loss function in model training. Since lower losses mean better models, we pay more attention in the area of lower losses. Hence, the estimation of $\ell(x)$ is much more vital. So, we want smaller α to be better. Unfortunately, if α is too small, there is no enough data for estimation. In practice, the TPE algorithm chooses α to be some quantile γ of the observed values.

In the literature of BO, surrogate models (step 1) (Springenberg et al., 2016), acquisition functions (step 2) (Wang & Jegelka, 2017) and batch sampling methods (step 3) (González et al., 2016) are well studied, but the problem of hyperparameter configuration's evaluation (step 4) remains largely open. While one could evaluate a hyperparameter configuration by training the corresponding model until convergence, this is quite expensive for complex models and big data especially when there are many configurations to evaluate. In BOHB, this issue is addressed with SH. However, SH only guarantees the performance of the best configuration, implying that the total data used to estimate the surrogate model can be extremely poor. Measuring the quality of data for model training, total regrets are more desired than the regret of the best one, which is the key motivation in our work.

The proposed algorithm BOSS is described in Algorithm 2. The key in our design is a sub-sampling method for hyperparameter configurations evaluation, which we theoretically prove to be asymptotically optimal.

During initialization, we calculate the maximum stages through the maximum budget R allowed for a single configuration and the ratio η. Then, the number of configurations K and the minimum budget b in each bracket are obtained. For initialization, we sample the configurations $\mathcal{C}$ from a uniform distribution and call the SS algorithm to evaluate $\mathcal{C}$. The collected data helps to refit the model and update the acquisition function for sampling in the next bracket. This BO framework gives the final best configuration.

Algorithm 2 BOSS

```
Input: Maximum budget $R$; ratio $\eta$.
Output: The configuration with the best performance.
1: Initialize the surrogate model and the acquisition function with a uniform distribution.
2: $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$.
3: for $s = s_{\max}, s_{\max} - 1, \ldots, 0$ do
4:     $K = \lceil \frac{B\eta^s}{R(s+1)} \rceil$, $b = R\eta^{-s}$.
5:     Sample $K$ configurations $\mathcal{C}$ from the acquisition function.
6:     Call SS with $(\mathcal{C}, b, \eta)$.
7:     Use the output data from SS to refit the model and update the acquisition function.
8: end for
```

4 Asynchronous Parallelization

In the BO framework, parallelization cannot happen between different brackets since the model needs to be updated. Hence, parallel acceleration occurs within each bracket. Both BOHB and BOSS can be only accelerated in SH or SS. The difference is that in SH, we know the number of configurations which need to be evaluated in the next round, so we can choose K/η^r good configurations in advance. It is not necessary to wait until all configurations are evaluated. This asynchronous parallelization is a simple version of ASHA (Li et al., 2018) and implemented in BOHB. For SS, we do not know the number of configurations which need to be evaluated in the next round. This makes asynchronous parallelization like ASHA fails.

Algorithm 3 Modified Sub-Sampling

```
Input: The set of $K$ configurations $\mathcal{C} = \{c_1, \ldots, c_K\}$; minimum budget $b$; conserved factor $\beta$; ratio $\eta$.
Output: $\{c_{\widetilde{\pi}_1}, \ldots, c_{\widetilde{\pi}_N}\}$ with corresponding evaluations.
1: $s = \lfloor \log_\eta(K) \rfloor$, $V_k^{(-1)} = 0$ for any $k \in \mathcal{I}$.
2: for $r = 0, \ldots, s$ do
3:     $K_r = \lfloor K\eta^{-r} \rfloor$, $b_r = b\eta^r$
4:     Evaluate top $K_r$ configurations with budgets $b_r$ sorted by $V_k^{(r-1)}$ in ascending order.
5:     Select the leader $c_\zeta$.
6:     Compute the sorting criterion $V_k^{(r)}(\zeta)$ for $k \in \mathcal{I}$.
7: end for
```

For the same reason, the total budget B in BOHB is known while it is not clear in BOSS. It makes us only compare the two methods with the same maximum budget R, not with the same total budget. In order to give a fair comparison and for asynchronous parallelization, we propose a modified version.

We modify Algorithm 1 to a new version by defining a criterion $V_k^{(r)}(\zeta)$ to sort these configurations in round r, where $V_k^{(r)}(\zeta) = \bar{Y}_{1:n_k}^{(k)} - \max_{1 \le j \le n_\zeta - n_k + 1} \bar{Y}_{j:(j+n_k-1)}^{(\zeta)} - \beta \cdot \max(0, q_n - n_k)$. If the conserved factor β is larger, case (a) in Section 2 will have a higher priority than case (b). The modified sub-sampling (MSS) algorithm is described in Algorithm 3.

The procedure of MSS is similar to SH except that the sorting criterion is changed from $Y_{n_k}^{(k)}$ to $V_k^{(r)}$. Therefore, it is easy to utilize parallel resources like ASHA. Finally, BOSS can be accelerated in parallel by replacing SS with MSS in step 6 in Algorithm 2.

4.1 Aggressive Parallelization

For the large-scale hyperparameter optimization problem for deep learning with GPU devices, the GPU computation resource is highly valuable. However, the difference of configuration numbers in the adjoining iterations and brackets results in idle GPUs from time to time, which reduces the actual searching time significantly and sometimes leads to inferior performance.

To address this issue, we put forward an aggressive asynchronous parallel version of BOSS ("parallel BOSS") as shown in Algorithm 7 and Algorithm 8 in Appendix D. The design philosophy of this parallel version is to minimize the GPU idle time. In this algorithm, when there is available GPU and returned evaluation results in the current bracket, it returns the top one configuration c_k of the current SS iteration based on the sorting of $V_k^{(r)}(\zeta)$. When there is no returned evaluation results in this bracket, the algorithm will randomly pick one configuration to evaluate. In this way, the algorithm makes the best use of GPU resources. On the other hand, the exploration of this algorithm is strengthened while the exploitation is weaken.

Moreover, as the batch BO algorithms like BOSS and BOHB sample a batch of configurations without returned evaluation results when generating new bracket, these configurations will usually be centralized in the hyperparameter space. This phenomenon is not good for exploration. To address this issue, the Constant Liar algorithm (Ginsbourger et al., 2007) uses a greedy approach to iteratively construct data points (the "liar") and updates the Gaussian Process model with this "liar" value. In this way, the diversity of the sampled configurations will be increased. This technique is used to update the Gaussian process model of parallel BOSS.

5 Experimental Results

This section illustrates the benefits of SS and AMSS over SH and ASHA respectively by synthetic experiments. Then, we apply the proposed BOSS to a variety of machine learning problems ranging from Neural Architecture Search (NAS), Data Augmentation (DA), Object Detection (OD) to Reinforcement Learning (RL) which can all be cast as HPO problems. Their common point is that it takes a long time to evaluate a single hyperparameter configuration. Due to limitations of space, experiments on RL are put in Appendix E. From the results, we can see that BOSS is consistently better than BOHB, but the performance difference is little. Thus, a paired t-test for BOSS and BOHB with all the data of NAS, data augmentation, and objective detection is given. The alternative hypothesis is that BOSS is greater than BOHB. The p-value of this test is 0.03729 which tells us the advantage of BOSS is statistically significant at 95% confidence level. Further, since the utility of GPUs is not effective in both BOSS and BOHB, we compare the difference of GPUs' usage between BOSS and Parallel BOSS.

5.1 Synthetic Experiments

In this subsection, we first compare the proposed SS and SH in synthetic experiments from two aspects, the cumulative regret and the probability of finding the true optimal configuration. Suppose there are K configurations, and the response from the k-th configuration, follows the normal distribution $N(\mu_k, \sigma)$, where $\mu_k = k/K$, σ is its standard deviation and $k \in \{0, 1, ..., K-1\}$. The t-th evaluation of the k-th configuration with budget b means randomly sampling b samples from $N(\mu_k, \sigma)$ and returning their mean denoted by $Y_t^{(k)}$. A sequence of configurations and the number of budgets used to evaluate these configurations are designed by the HPO algorithms.

Figure 2 demonstrates that the average regrets, defined by $\frac{1}{N}\sum_{i=1}^{N}(y_i - \min_k \mu_k)$, of these algorithms are large in the beginning, since they try all configurations in the early iterations. After that, they use collected information to determine exploration or exploitation. For configurations with small variances, the difference between them can be easily distinguished with a small amount of budgets. Hence, SH quickly converges while SS as a more conservative policy converges slower. However, the performance of SH is greatly affected by the responses' instability. For the responses with large variances, SH fails to find the optimal configuration since it wrongly evaluates the configurations with small budgets and has no chance to fix it. On the contrary, SS is more stable to find the optimal one. For different choices of η, this conclusion still holds. These figures also reveal that SS has more robust results than SH with different η.

Moreover, Table 1 demonstrates the proportion of selecting the optimal configuration by SH and SS with $\eta = 3$. Under different settings, SS performs better than SH. The deviation of responses has a great influence on SH. SH gets a high accuracy in the circumstance of small deviations, but the accuracy decreases quickly as the deviation gets

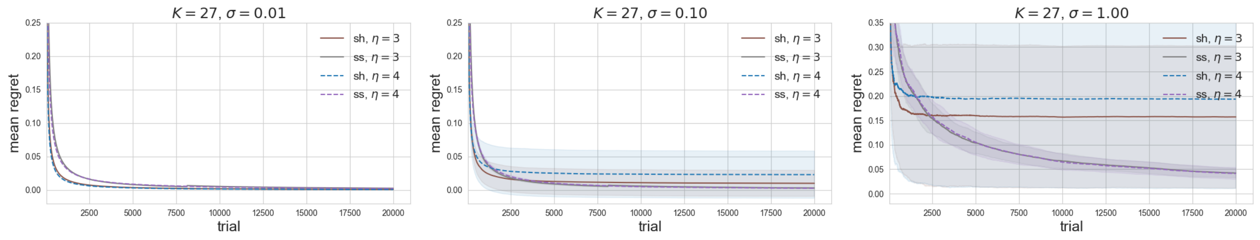

Figure 2. The comparison of the average regrets of SH and SS with different η. For each setting, we conduct the experiment 50 runs.The curves present the mean of the average regrets. The shaded part indicates the min/max ranges.

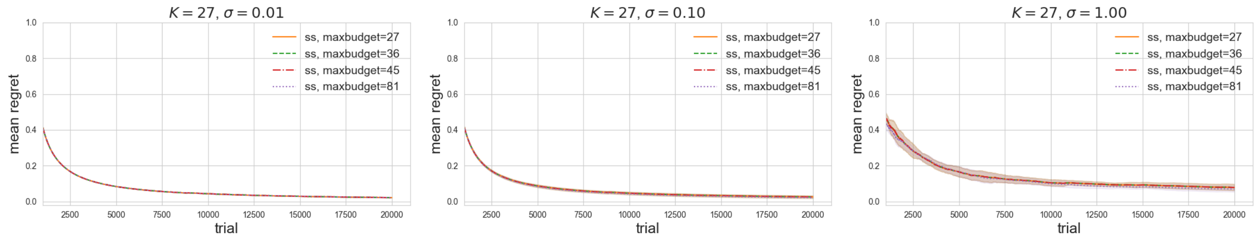

Figure 3. The comparison of the average regrets of SH and SS with different maximum budgets. For each setting, we conduct the experiment 50 runs.The curves present the mean of the average regrets. The shaded part indicates the min/max ranges.

Table 1. The accuracy (%) of selecting the optimal configuration.

Method	$K = 27$			$K = 54$		
	$\sigma = 0.01$	0.10	1.00	0.01	0.10	1.00
SH	100	76	24	100	62	18
SS	100	100	100	100	100	88

larger. When $K = 27$, SS can always find the optimal configuration under different deviations even with a large deviation of 1.00. Note that we have compressed the mean value between 0 and 1, and the deviation of 1.00 can make it difficult to distinguish between configurations. When $K = 54$, it sometimes fails to get the optimal configuration since it needs more steps to do exploration and exploitation. We also consider the effect of maximum budgets. Figure 3 shows the results with $\eta = 3$ and different maximum budgets. The performances of the algorithms are highly similar. The parameter q_n equals $\sqrt{\log n}$.

Figure 4 illustrates the comparison of SH, ASHA, SS, MSS and AMSS. For the new parameter β, we also make a comparison to find its effect.

We can see that MSS and AMSS outperform SH and ASHA in all circumstances. In the cases with small deviations, these algorithms converge to the same point. In the cases with large deviations, the advantages of MSS and AMSS over SH and ASHA are more obviously. As for the effect of the parameter β, it reveals that β can control the degree of conservation. The algorithms with large β take conservative behaviors to explore more configurations in the initial stages. This phenomenon is especially obvious in the context of small deviations. When $\sigma = 0.01$, the performances of $\beta = 100$ are the worst at the beginning. The reason is that large β considers the number of observations only. They ignores the current performance of the configurations. However, their final performances are highly similar. Note that AMSS has the same result with the same trials, and asynchronous parallelization can reduce running time which makes AMSS more efficient.

5.2 Data Augmentation Search

Data augmentation is an effective technique to generate more samples from data by rotating, inverting or other operations for improving the accuracy of image classifiers. However, most implementations are manually designed with a few exceptions. Cubuk et al. (2018) proposed a simple procedure called AutoAugment to automatically search for improved data augmentation policies. Unfortunately, it is very time-consuming, e.g., it takes 5000 GPU hours in searching procedure for CIFAR100 (Krizhevsky et al., 2009). More recently, Ho et al. (2019) and Lim et al. (2019) designed more efficient algorithms for this particular task.

In their search space, a policy consists of 5 sub-policies with each sub-policy consisting of two image operations to be applied in sequence. Additionally, each operation is also associated with two hyperparameters: (1) the probability of applying the operation, and (2) the magnitude of

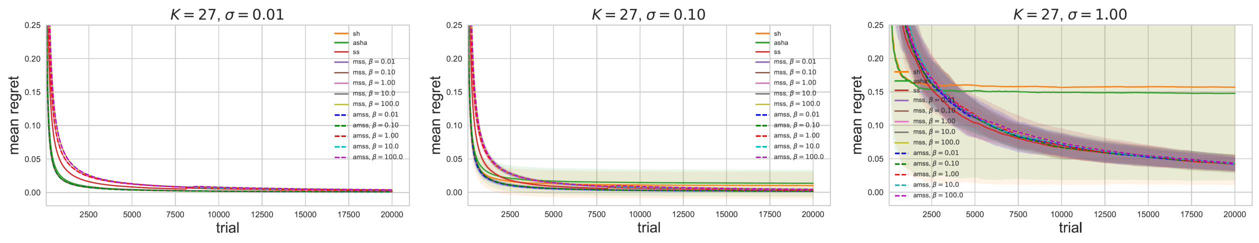

Figure 4. The comparison of the average regrets of SH, ASHA, SS, MSS and AMSS. For each setting, we conduct the experiment 50 runs.The curves present the mean of the average regrets. The shaded part indicates the min/max ranges.

the operation. In total, there are 16 operations in the search space. Each operation also comes with a default range of magnitudes. These settings are described in Cubuk et al. (2018). For this problem, we need to tune two hyperparameters of each sub-policy and choose the best five sub-policies to form a policy. This is a natural HPO problem. BOSS can be adopted directly without any modification. As for the setting of the parameters η and R in Algorithms 2, we set the ratio $\eta = 3$ refers to the same setting as SH and BOHB. The maximum budget R equals to one third of the number of epochs for convergence. In our experience given in Figure 3, small changes of R have no effect on the performance. The following applications all use this setting.

We search the data augmentation policy in the image classification tasks of CIFAR-10 and CIFAR-100. We follow the setting in AutoAugment (Cubuk et al., 2018) to search for the best policy on a smaller data set, which consists of $4,000$ randomly chosen examples, to save time for training child models during the augmentation search process. For the child model architecture, we use WideResNet-28-10 (28 layers - widening factor of 10) (Zagoruyko & Komodakis, 2016). The augmentation policy is combined with standard data pre-processing: on one image, we normalize the data in the following order, use the horizontal flips with 50% probability, zero-padding and random crops, augmentation policy, and finally Cutout with 16×16 pixels (DeVries & Taylor, 2017). We run BOSS, BOHB and HB with budgets of $1, 5, 16, 50, 150$ epochs using 32 parallel workers for 10 iterations. We run BO, SH and Random Search with budgets of 150 epochs using 32 parallel workers for 10 iterations. Every two workers use one NVIDIA V100 GPU for parallel training. For each sub-task, we use a SGD optimizer with a weight decay of 0.0005, momentum of 0.9, learning rate of 0.1.

We use the found policies to train final models on CIFAR-10, CIFAR-100 with 200 epochs. All the results of the baselines are replicated in our experiments and match the previously reported results (Falkner et al., 2018; Cubuk et al., 2018). Instead of searching the augmentation policy in a large amount of GPU days, we use the searched policy reported in Cubuk et al. (2018) to evaluate its performance.

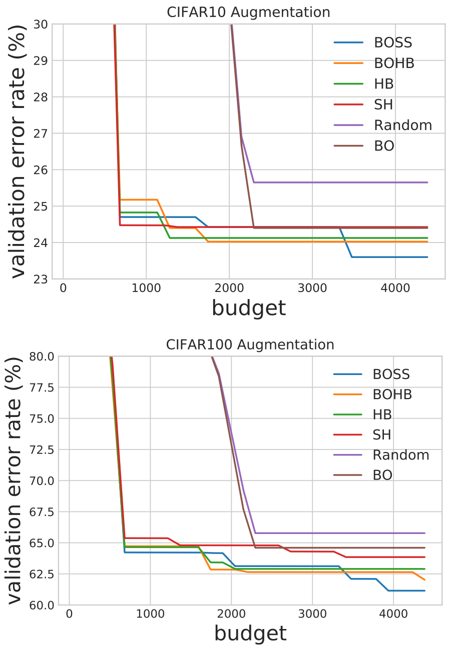

Figure 5. Data Augmentation (DA) as a hyperparameter optimization (HPO) problem.

Figure 5 demonstrates the performance during the searching procedure. The budget means the total epochs used in the searching process. The results show that BOHB performs better than BOSS in the beginning, but BOSS converges to better configurations in the end. This is caused by the conservative and the asymptotic optimality of BOSS. The performances of BO, HB, SH and Random Search are weaker than BOHB and BOSS, since they either sample configurations uniformly without considering the information brought by previous trials or evaluate configurations slowly. Note that the error rates of these methods are high especially on CIFAR100. This is because in the searching procedure, used budgets are much smaller than the training procedure for the searched policy.

The results listed in Table 2 show that the naive HPO methods including Random Search, SH, HB and BO have similar results as the original RL-based DA method, and the proposed efficient search scheme BOSS has the best accuracy and BOHB is close behind.

Table 2. Top-1 accuracy (%) of data augmentation on the test dataset of CIFAR-10 and CIFAR-100. All of the reported results are averaged over 5 runs.

Method	Test Accuracy (std)	
	CIFAR10	CIFAR100
AA	97.09 (0.14)	82.42 (0.17)
Random	97.01 (0.11)	82.41 (0.23)
SH	96.93 (0.14)	81.49 (0.23)
HB	97.00 (0.10)	82.07 (0.11)
BO	97.11 (0.16)	82.19 (0.25)
BOHB	97.25 (0.13)	82.52 (0.15)
BOSS	**97.32** (0.12)	**83.03** (0.22)

5.3 Neural Architecture Search

One crucial aspect of the deep learning development is novel neural architectures. Designing architectures manually is a time-consuming and error-prone process. Because of this, there is a growing interest in automated neural architecture search. Elsken et al. (2019) provided an overview of existing work in this field of research. We use the search space of DARTS (Liu et al., 2019) as an example to illustrate HPO methods on NAS. Particularly, their goal is to search for a cell as a basic unit. In each cell, there are N nodes forming a fixed directed acyclic graph (DAG). Each edge of the DAG represents an operation, such as skip-connection, convolution, max pooling, etc., weighted by the architecture parameter α. For the search procedure, the training loss and validation loss are denoted by L_{train} and L_{val} respectively. Then the architecture parameters are learned with the following bi-level optimization problem:

$$\begin{aligned} &\min_{\alpha} \quad L_{val}(\omega^*(\alpha), \alpha), \\ &\text{s.t. } \omega^*(\alpha) = \arg\min_{\omega} L_{train}(\omega, \alpha). \end{aligned}$$

Here, α are hyperparameters in the HPO framework. For evaluating α, we need to optimize its network parameters ω. It is usually time-consuming. Note that BOSS is exactly proposed to make a fast and efficient evaluation for this problem.

We search the neural architectures in the image classification tasks of CIFAR10 and CIFAR100. We follow the settings of DARTS (Liu et al., 2019). The architecture parameter α determines two kinds of basic units: normal cell and reduction cell. We run BOHB, BOSS and HB with maximum budgets of 50 epochs using 32 parallel workers for 10 iterations. We run BO, *SH* and Random Search with total budgets as the same as BOSS using 32 parallel workers for 10 iterations. For a sampled architecture parameter, we fix it in the training process of updating model parameters. A SGD optimizer is used with learning rate of 0.025, momentum of 0.9, weight decay of 0.0003, and a cosine learning decay with an annealing cycle.

After searching out the optimal configuration, we evaluate it with 200 epochs. Figure 6 shows the highly similar results of that in data augmentation.

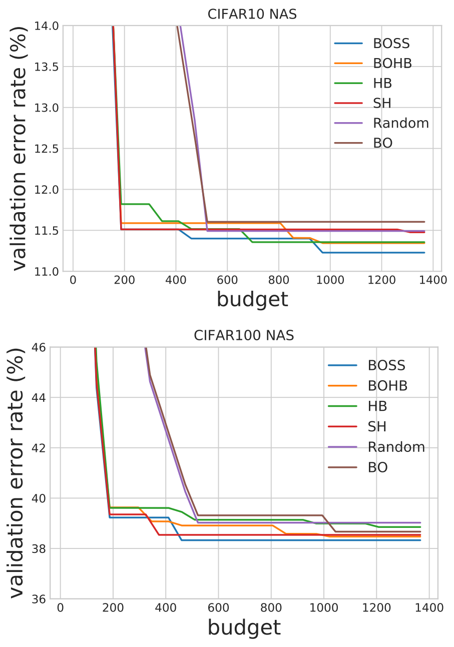

Figure 6. Neural architecture search (NAS) as a hyperparameter optimization (HPO) problem.

The results are listed in Table 3. Note that the search space and search procedure of DARTS are designed on CIFAR10. The naive HPO methods including Random Search, SH, HB and BO are worse than DARTS while BOSS and BOHB have comparable accuracy. On CIFAR100, the advantage of these HPO methods increases, and BOSS is significantly better than other search procedures.

Table 3. Test accuracy (%) of NAS on CIFAR-10 and CIFAR-100. All of the reported results are averaged over 5 runs.

Method	Test Accuracy (std)	
	CIFAR10	CIFAR100
DARTS	97.26 (0.13)	81.71 (0.31)
Random	96.86 (0.19)	81.65 (0.34)
SH	96.55 (0.20)	81.19 (0.18)
HB	96.99 (0.18)	81.63 (0.37)
BO	96.87 (0.15)	82.37 (0.24)
BOHB	97.23 (0.17)	82.67 (0.27)
BOSS	**97.29** (0.15)	**83.10** (0.23)

Moreover, when transferred to ImageNet in the mobile setting, the same architectures searched from CIFAR10 and CIFAR100 by BOSS achieve high accuracy of 74.46% and 74.82% respectively while DARTS reported the accuracy of 73.3%. This phenomenon reveals the weakness of DARTS in that it is prone to search architectures with many skip-connect operations which is not preferred. (Zela et al., 2020; Liang et al., 2019) reduced the number of skip-connect operations by early stopping. However, this issue disappears in BOSS naturally which is depicted in Figures 7 and 8, because DARTS changes architecture parameters and network parameters in turn while BOSS does not change the architecture during the training of network.

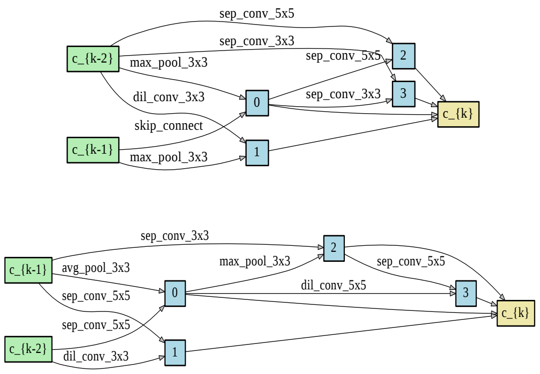

Figure 7. The architectures of normal cell (left) and reduction cell (right) learned by BOSS on CIFAR10.

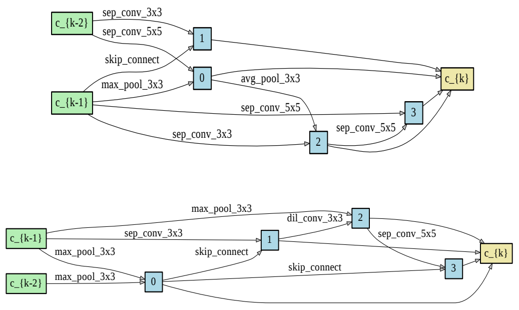

Figure 8. The architectures of normal cell (left) and reduction cell (right) learned by BOSS on CIFAR100.

5.4 Object Detection

Object detection is to use an anchor box to cover the target object. Most state-of-the-art OD systems follow an anchor-based diagram. Anchor boxes are densely proposed over the images and the network is trained to predict the boxes' position offset as well as the classification confidence. Existing systems use ad-hoc heuristic adjustments to define the anchor configurations based on predefined anchor box shapes and size. However, this might be sub-optimal or even wrong when a new dataset or a new model is adopted. Hence, the parameters of anchor boxes for object detection need to be automatically optimized including number, scales, and ratios of anchor boxes. Refer to Ma et al. (2020) for more details. Ma et al. (2020) discussed HPO problems in object detection, proposed a formulation and analyzed several automatic HPO methods including BOSS.

In the literature, the anchor shapes are typically determined by manual selection (Dai et al., 2016; Liu et al., 2016; Ren et al., 2015) or naive clustering methods (Redmon & Farhadi, 2017). Different from the traditional methods, there are several works focusing on utilizing anchors more effectively and efficiently (Yang et al., 2018; Zhong et al., 2018).

Table 4 compares BOSS with several existing anchor initialization methods as follows.

(a) Set anchor scales and ratios like most detection systems by manually searching.
(b) Use K-means method proposed in YOLOv2 (Redmon & Farhadi, 2017) to obtain clusters and treat them as initial anchors.
(c) Use random search to determine anchors.
(d) Use BOHB to determine anchors.
(e) Use BOSS to determine anchors.

The standard criteria of object detection, mean Average Precision (mAP) and Average Recall (AR), are used to measure the performance. The subscript of $AP_{50,75}$ means different IoU thresholds, and the subscript of $AR_{1,10,100}$ represents different numbers of given objects per image. S, M and L refer to small, medium and large size of an object respectively. For fairness of comparing these different search schemes, we use the same training procedure which is Faster-RCNN (Ren et al., 2015) combined with FPN (Lin et al., 2017) as detector, ResNet-50 (He et al., 2016) as backbone on MSCOCO (Lin et al., 2014).

The results reveal that all three HPO methods outperform two classical ones. Moreover, BOSS has uniformly better performance than other two HPO methods which shows the usefulness and effectiveness of BOSS. In addition, since the training procedures of MetaAnchor (Yang et al., 2018) and Zhong's method (Zhong et al., 2018) are different from ours, we just compare the difference before and after using hyperparameter searching. BOSS brings about 2.4% mAP improvement while MetaAnchor and Zhong's method increase mAP by 1.0% and 1.1%, respectively. This advantage of BOSS is that it considers both sampling strategy and efficient evaluation while the other two methods just consider the former.

Table 4. Comparisons of different hyperparameter searching schemes with the same training procedure on MSCOCO.

Method	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AR_1	AR_{10}	AR_{100}	AR_S	AR_M	AR_L
Manual Search	36.4	58.2	39.1	21.3	40.1	46.5	30.3	48.8	51.3	32.1	55.6	64.6
K-Means	37.0	58.9	39.8	21.9	40.5	48.5	31.0	49.3	51.9	32.9	57.5	66.3
Random Search	37.2	58.8	39.9	21.7	40.6	48.1	31.1	49.5	52.1	33.8	57.3	66.5
BOHB	37.8	58.9	40.4	22.7	41.3	49.9	31.8	50.2	52.5	34.2	**57.5**	65.2
BOSS	**38.8**	**60.7**	**41.6**	**23.7**	**42.5**	**51.5**	**32.3**	**51.2**	**54.0**	**35.5**	57.3	**68.2**

5.5 Parallel BOSS

The parallel BOSS (Algorithm 7) is an aggresive gpu-efficient version of BOSS. To evaluate its performance, we implement it for a challenging industrial feature classification task with deep neural networks. The training set and testing set contain acoustic feature of around 75000 and 15000 samples. The feature is effectively augmented, resulting in 2.2TB training data and 11GB testing data. The performance metric is the classification precision (higher better). Here we compare BOSS with BOHB and TPE (parallel version implemented with constant liar algorithm) algorithms under two settings provided in Table 5.

Table 5. Experiment settings.

Setting	min budget	max budget	η	max duration
No. 1	1*2	27*2	3	60h
No. 2	1*5	27*5	3	90h

Table 6. Comparisons of different HPO algorithms on a feature classification task under setting 1 and setting 2. Due to the limited computation resource, we only compare TPE and parallel BOSS for setting 2.

Method	Setting 1		Setting 2	
	Avg top 10	Best	Avg top 10	Best
TPE	0.8871	0.8886	0.8924	0.8936
BOHB	0.8803	0.8868	-	-
BOSS	0.8866	0.8890	-	-
parallel BOSS	**0.8908**	**0.8918**	**0.8935**	**0.8944**

The results in Table 6 show that the standard TPE also has a comparable performance since its usage of GPUs is efficient. We compare the theoretical and practical GPU usage of BOSS and parallel BOSS under setting 1, shown in Figure 9. The curves of practical GPU memory useage fit well to the theoretical GPU usage. The parallel BOSS fully utilizes the GPU resources and yields stronger results in these tasks.

6 Conclusions

In this paper we have proposed BOSS which combines BO and SS for HPO problems. The major contribution is to develop SS as an efficient and effective evaluation procedure and its asynchronously parallel version MSS. The proposed methods are for fast hyperparameter search evaluation by measuring the potential of hyperparameter configurations. It promises to improve robustness compared to algorithms based on successive halving. The main result is the asymptotic optimality of the cumulative regret of the proposed SS algorithm. This advantage is suitable for the BO iteration since it can collect high-quality data to estimate the surrogate model more efficiently. It can get the evaluation with less budgets which is a kind of early-stop methods. Experiments show that SS can find the best configuration quickly and correctly and BOSS works for many popular applications. Future work to improve BOSS may further learn the search space (Perrone et al., 2019) or hyperparameter importance (Hoos & Leyton-Brown, 2014). On the other hand, the ensuing problem is that when we need the early-stop techniques. Obviously, if there is no relationship between the performance with different budgets, these techniques will not work. Thus, how to judge whether a task is suitable for early stopping becomes an important prerequisite.

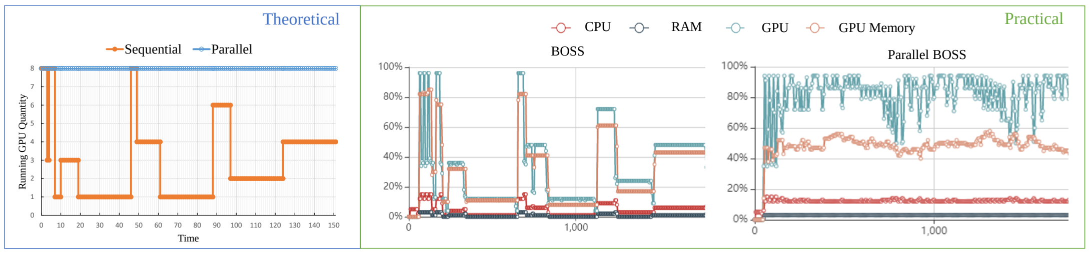

Figure 9. The usage of GPU in a complete BOSS period.

REFERENCES

Agarwal, A., Bartlett, P. L., and Duchi, J. C. Oracle inequalities for computationally adaptive model selection. *arXiv preprint arXiv:1208.0129*, 2012.

Agrawal, R. Sample mean based index policies with $O(\log n)$ regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.

Auer, P., Cesabianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305, 2012.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2011.

Bertrand, H., Ardon, R., Perrot, M., and Bloch, I. Hyperparameter optimization of deep neural networks: Combining hyperband with Bayesian model selection. In *Conférence sur l'Apprentissage Automatique*, 2017.

Brockhoff, D., Bischl, B., and Wagner, T. The impact of initial designs on the performance of matsumoto on the noiseless bbob-2015 testbed: A preliminary study. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 1159–1166, 2015.

Burtini, G., Loeppky, J., and Lawrence, R. A survey of online experiment design with the stochastic multi-armed bandit. *arXiv preprint arXiv:1510.00757*, 2015.

Chan, H. P. The multi-armed bandit problem: An efficient non-parametric solution. *Annals of Statistics*, pp. To appear, 2019.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

Dai, J., Li, Y., He, K., and Sun, J. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pp. 379–387, 2016.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Falkner, S., Klein, A., and Hutter, F. BOHB: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

Feurer, M. and Hutter, F. Hyperparameter optimization. In Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.), *AutoML: Methods, Sytems, Challenges*, chapter 1, pp. 3–37. Springer, December 2018. To appear.

Ginsbourger, D., Le Riche, R., and Carraro, L. A multi-points criterion for deterministic parallel global optimization based on gaussian processes. *NCP07*, 2007.

González, J., Dai, Z., Hennig, P., and Lawrence, N. Batch bayesian optimization via local penalization. In *Artificial intelligence and statistics*, pp. 648–657, 2016.

Hansen, N. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hennig, P. and Schuler, C. J. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012.

Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pp. 918–926, 2014.

Ho, D., Liang, E., Stoica, I., Abbeel, P., and Chen, X. Population based augmentation: Efficient learning of augmentation policy schedules. *arXiv preprint arXiv:1905.05393*, 2019.

Hoos, H. and Leyton-Brown, K. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pp. 754–762, 2014.

Hutter, F., Hoos, H. H., and Leyton-brown, K. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523, 2011.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Chrisantha, F., and Kavukcuoglu, K. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Jamieson, K. and Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pp. 240–248, 2016.

Jones, D. R., Schonlau, M., and Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

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

Keshtkaran, M. R. and Pandarinath, C. Enabling hyperparameter optimization in sequential autoencoders for spiking neural data. In *Advances in Neural Information Processing Systems*, pp. 15911–15921, 2019.

Klein, A., Dai, Z., Hutter, F., Lawrence, N., and Gonzalez, J. Meta-surrogate benchmarking for hyperparameter optimization. *arXiv preprint arXiv:1905.12982*, 2019.

Konen, W., Koch, P., Flasch, O., Bartz-Beielstein, T., Friese, M., and Naujoks, B. Tuned data mining: a benchmark study on different tuners. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 1995–2002, 2011.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lai, T. L. Adaptive treatment allocation and the multi-armed bandit problem. *Annals of Statistics*, 15(3):1091–1114, 1987.

Lai, T. L. and Robbins, H. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1): 4–22, 1985.

Law, H. C., Zhao, P., Chan, L. S., Huang, J., and Sejdinovic, D. Hyperparameter learning via distributional transfer. In *Advances in Neural Information Processing Systems*, pp. 6801–6812, 2019.

Li, H., Chaudhari, P., Yang, H., Lam, M., Ravichandran, A., Bhotik, R., and Soatto, S. Rethinking the hyperparameters for fine-tuning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=B1g8VkHFPH`.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., and Talwalkar, A. Massively parallel hyperparameter tuning. *arXiv preprint arXiv:1810.05934*, 2018.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., and Li, Z. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

Lim, S., Kim, I., Kim, T., Kim, C., and Kim, S. Fast autoaugment. *arXiv preprint arXiv:1905.00397*, 2019.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=S1eYHoC5FX`.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Ma, W., Tian, T., Xu, H., Huang, Y., and Li, Z. Aabo: Adaptive anchor box optimization for object detection via bayesian sub-sampling. In *European Conference on Computer Vision*, pp. 560–575. Springer, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Paul, S., Kurin, V., and Whiteson, S. Fast efficient hyperparameter tuning for policy gradient methods. In *Advances in Neural Information Processing Systems*, pp. 4618–4628, 2019.

Perchet, V. and Rigollet, P. The multi-armed bandit problem with covariates. *Annals of Statistics*, 41(2):693–721, 2013.

Perrone, V., Shen, H., Seeger, M. W., Archambeau, C., and Jenatton, R. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, pp. 12751–12761, 2019.

Rasmussen, C. E. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pp. 63–71. Springer, 2003.

Redmon, J. and Farhadi, A. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Ru, B., Osborne, M. A., Mcleod, M., and Granziol, D. Fast information-theoretic bayesian optimisation. In *International Conference on Machine Learning*, pp. 4384–4392, 2018.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. Scalable Bayesian optimization using deep neural networks. In *International conference on machine learning*, pp. 2171–2180, 2015.

Sparks, E. R., Talwalkar, A., Franklin, M. J., Jordan, M. I., and Kraska, T. Tupaq: An efficient planner for large-scale predictive analytic queries. *arXiv preprint arXiv:1502.00068*, 2015.

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. Bayesian optimization with robust bayesian neural networks. In *Advances in neural information processing systems*, pp. 4134–4142, 2016.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. Gaussian process optimization in the bandit setting: no regret and experimental design. In *International Conference on Machine Learning*, pp. 1015–1022, 2010.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855, 2013.

Wang, J., Xu, J., and Wang, X. Combination of hyperband and Bayesian optimization for hyperparameter optimization in deep learning. *arXiv preprint arXiv:1801.01596*, 2018.

Wang, Z. and Jegelka, S. Max-value entropy search for efficient bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3627–3635. JMLR. org, 2017.

Wu, C. J. and Hamada, M. S. *Experiments: planning, analysis, and optimization*, volume 552. John Wiley & Sons, 2011.

Yakowitz, S. and Lowe, W. Nonparametric bandit methods. *Annals of Operations Research*, 28(1):297–312, 1991.

Yang, T., Zhang, X., Li, Z., Zhang, W., and Sun, J. MetaAnchor: Learning to detect objects with customized anchors. In *Advances in Neural Information Processing Systems*, pp. 320–330, 2018.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gDNyrKDS.

Zhang, X., Chen, X., Yao, L., Ge, C., and Dong, M. Deep neural network hyperparameter optimization with orthogonal array tuning. In *International Conference on Neural Information Processing*, pp. 287–295. Springer, 2019.

Zhong, Y., Wang, J., Peng, J., and Zhang, L. Anchor box optimization for object detection. *arXiv preprint arXiv:1812.00469*, 2018.

Zöller, M.-A. and Huber, M. F. Survey on automated machine learning. *arXiv preprint arXiv:1904.12054*, 9, 2019.

A RELATED WORK

In this section, we will provide a slice of details regarding the multi-fidelity methods. The shortcomings of historical methods will be revealed.

A.1 Successive Halving

The basic objective of multi-fidelity methods is to identify the best configuration out of a given finite set of configurations based on low-fidelity approximations of their performance. In order to achieve this objective, it is possible to drop hyperparameter configurations if they perform badly with small computing resources. Based on this idea, Jamieson & Talwalkar (2016) proposed the Successive Halving algorithm originally introduced by Karnin et al. (2013) for HPO. Query all K configurations with a given initial budget for each one; then, remove the half that performed worst, double the budget and successively repeat until only a single configuration is left, which is described in Algorithm 4. The main theorem in Jamieson & Talwalkar (2016) indicates that SH can return the true best configuration when the total used budget B is larger than a certain value. SH is an extremely simple, yet powerful, and therefore popular policy for the selection of multi-fidelity methods. Li et al. (2018) presented Asynchronous Successive Halving Algorithm (ASHA), a practically parallelized version of SH for dealing that configurations are typically orders of magnitude more than available parallel workers.

Algorithm 4 Successive Halving

Input: The set of K configurations $\mathcal{C} = \{c_1, \dots, c_K\}$; minimum budget b; ratio η.
Output: Evaluation results of configurations.

```
1: $s = \lfloor \log_\eta(K) \rfloor$, total budget $B = Kbs$
2: for $r = 0,..., s$ do
3:     $K_r = \lfloor K\eta^{-r} \rfloor$
4:     $b_r = b\eta^r$
5:     Evaluate current $K_r$ configurations with budgets $b_r$
6:     Keep top $K_r/\eta$ configurations
7: end for
```

As discussed in Section 1, SH suffers from the trade-off between the budget and the number of configurations. Given a total budget of $B = Kbs$, users need to decide the number of configurations K. Trying larger K and assigning a small budget to each can result in prematurely terminating good configurations while trying only a few and assigning them a larger budget can result in wasting resources on evaluating a poor configuration.

A.2 HyperBand

Hyperband (Li et al., 2016) is presented to combat the trade-off problem in SH. It uses different values of K and calls the SH algorithm as a subroutine. There are two components to HB shown in Algorithm 5; (1) the inner loop invokes SH for fixed values of K and b; (2) the outer loop iterates over different values of K and b.

HyperBand begins with the most aggressive bracket $s = s_{max}$, which sets K to maximize exploration, subject to the constraint that at least one configuration is allocated R resources. Each subsequent bracket reduces K by a factor of approximately η until the final bracket, $s = 0$, in which every configuration is allocated R resources (this bracket simply performs the classical random search).

Algorithm 5 HyperBand

Input: Maximum budget R; ratio η
Output: The configuration with the best performance

```
1: $s_{max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{max} + 1)R$
2: for $s = s_{max}, s_{max} - 1, \dots, 0$ do
3:     $K = \lceil \frac{B\eta^s}{R(s+1)} \rceil$, $b = R\eta^{-s}$
4:     Sample $K$ configurations randomly
5:     Call SH with $(K, b)$
6: end for
```

In practice, HB works very well and typically outperforms random search and Bayesian optimization methods for small total budgets. However, the brackets in HB are independent of each other. And, the configurations in the next bracket are still sampled randomly without any guidance which means the information of previous brackets wastes.

A.3 Bayesian Optimization and HyperBand

To overcome the limitation that HB does not adapt the configuration sampling methods to the evaluations, the recent approach BOHB (Falkner et al., 2018) combines Bayesian optimization and HyperBand. Its idea is to build the relation between brackets by the Bayesian model and to replace HB's random search by BO. Particularly, as shown in Algorithm 6, BOHB relies on HB to determine how many resources used to evaluate configurations, but it replaces the random selection of configurations in each bracket by a model-based search. Once the number of configurations in a bracket is determined, the standard SH algorithm runs with these configurations. Then, we collect these configurations and their performance to fit a Bayesian surrogate model. For the next bracket, the configurations will be sampled from the model. Compared to the original HB, BOHB guides the search. There are two other works that also attempted to combine Bayesian optimization with HyperBand. Bertrand et al. (2017) used a Gaussian process with a squared exponential kernel as the Bayesian surrogate model instead of the TPE method of BOHB and dealt with model selection tasks. Wang et al. (2018) sampled trial points one by one using BO building the relation of configurations in the bracket, not between brackets.

Unfortunately, HB is designed to identify the best configuration. There is no guarantee for the evaluation of other configurations while BO needs that all configurations not only the best one used to estimate the model have high-level

Algorithm 6 BOHB

Input: R, the maximum amount of resources that can be allocated to a single configuration; ratio η

Output: The configuration with the best performance

```
1: $s_{max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{max}+1)R$
2: for $s = s_{max}, s_{max}-1, \dots, 0$ do
3:     $K = \lceil \frac{B\eta^s}{R(s+1)} \rceil$, $b = R\eta^{-s}$
4:     Sample $K$ configurations from the Bayesian model
5:     Call SH with $(\mathcal{C}, b)$
6:     Use the data from SH to refit the model
7: end for
```

performance. In other words, it will lead to a wrong estimation of the surrogate model. Therefore, it is necessary to propose another criterion for fast evaluation.

A.4 Bandit-based Methods concerning Cumulative Regret

In the literature, there are many bandit policies proposed to get optimal cumulative regret, which means they pursue an optimal sequence of configurations instead of the final return configuration only. Lai & Robbins (1985) gave an asymptotic lower bound for the regret in the multi-armed bandit problem and proposed an index strategy that achieved this bound. Lai (1987) showed that when probability distributions belong to a specified exponential family, a policy that pull the arm of the largest upper confidence bound (UCB) is optimal. The UCB policy is constructed from Kullback-Leibler (KL) information between estimated observation distributions of the arms. Agrawal (1995) modified the UCB policy without knowing the total sample size. To better describe applications, the first work to venture outside the realm of parametric modeling assumptions appeared in Yakowitz & Lowe (1991). As opposed to the traditional multi-armed bandit problem, they proposed non-parametric policies not based on KL-information but under some moment conditions. Auer et al. (2002) provided the policy named UCB1 which can achieve logarithmic regret if observation distributions are supported on $[0, 1]$. Chan (2019) proposed an efficient non-parametric solution and proved optimal efficiency of the policy. However, their observation distribution must belong to an one-parameter exponential family which would be extended in our work. Moreover, they only handled the standard multi-armed bandit problems while HPO is the ultimate goal in our work.

B Multi-Armed Bandit Problem for HPO

In this section the notation and the standard multi-armed bandit problem with K configurations (arms) will be introduced for discussing another criterion, cumulative regret.

Recall the traditional setup for the classic multi-armed bandit problem. Let $\mathcal{I} = \{1, 2, \dots, K\}$ be a given set of $K \geq 2$ configurations. Consider a sequential procedure based on past observations $Y_t^{(k)}$, where t represents the observation time from the k-th configuration. Let N_k be the number of observations from the k-th configuration with different budgets, and $N = \sum_{k=1}^K N_k$ is the number of total observations. For each configuration, the observations $\{Y_t^{(k)}\}_{t\geq 1}$ are assumed to be independent and identically distributed with expectation given by $\mathbb{E}(Y_t^{(k)}) = \mu_k$ and $\mu_* = \min_{1\leq k\leq K} \mu_k$. In HPO problems, the randomness of the observations comes from the randomness of initialization, because the performance of an under-trained neural network with small budgets strongly depends on its initialization. For simplicity, assume without loss of generality that the best configuration is unique which is also assumed in Perchet & Rigollet (2013) and Chan (2019).

A policy $\pi = \{\pi_t\}$ is a sequence of random variables $\pi_t \in \{1, 2, \dots, K\}$ denoting that at each time $t = 1, 2, \dots, N$, the configuration π_t is selected to evaluate. Note that π_t depends only on previous $t-1$ observations. The objective of a good policy π is to minimize the cumulative regret

$$R_N(\pi) \triangleq \sum_{k=1}^{K} (\mu_k - \mu_*)\mathbb{E}N_k = \sum_{t=1}^{N} (\mu_{\pi_t} - \mu_*).$$

Note that for a data-driven policy $\hat{\pi}$, the regret monotonically increases with respect to N. Hence, minimizing the growth rate of R_N is an important criterion which is considered in the later section. The successive elimination method (Perchet & Rigollet, 2013) is like SH to eliminate bad configurations successively and is given an upper bound of the cumulative regret for any N. However, the asymptotic property that we need in big data is not optimal.

An arm allocation policy π is said to be uniformly good if

$$R_N(\pi) = o(N^\varepsilon) \text{ for all } \varepsilon > 0.$$

Moreover, if π is uniformly good and some additional regularity conditions are satisfied, Lai & Robbins (1985) provided a lower bound of the growth rate:

$$\liminf_{N\to\infty} \frac{R_N(\pi)}{\log N} \geq \sum_{k:\mu_* < \mu_k} \frac{\mu_k - \mu_*}{KL(f_k, f_*)}. \qquad (2)$$

$KL(f, g)$ is the Kullback-Leibler divergence between density functions f and g, that is,

$$KL(f, g) = \mathbb{E}_f \Big[\log \frac{f(Y)}{g(Y)} \Big],$$

where $\mathbb{E}_f$ denotes expectation with respect to $Y \sim f$. Chan (2019) proposed an arm allocation policy in a strong assumption on the distribution of reward Y that made regrets achieve the lower bound in Equation (2). We say that it has the optimal rate. And, if the growth of the cumulative regret has the order of $\log N$, it is called a nearly optimal policy.

C Theoretical Results

In this section, we prove that the proposed SS method is asymptotically optimal using the tools from multi-armed bandit (MAB) (Agarwal et al., 2012; Sparks et al., 2015; Jamieson & Talwalkar, 2016). Each arm corresponds to a fixed hyperparameter setting, the arm collection $\mathcal{I}$ corresponds to the set of configurations $\mathcal{C}$, pulling an arm corresponds to a fixed number of training iterations, budget corresponds to the number of samples in one pull and the loss corresponds to the validation loss.

Given an arm allocation policy π, consider the cumulative regret $R_N(\pi) = \sum_{k=1}^{K}(\mu_k - \mu_*)\mathbb{E}N_k$, where $\mu_k = \mathbb{E}(Y_t^{(k)})$ and $\mu_* = \min_{1\leq k\leq K}\mu_k$. Lai & Robbins (1985) provided an asymptotic lower bound of $R_N(\pi)/\log N$ with $\sum_{k:\mu_*<\mu_k}(\mu_k-\mu_*)/KL(f_k,f_*)$, where $f_k(y)$ is a density function of $Y_t^{(k)}$, $f_* = f_{\arg\min_k \mu_k}$, and the function $KL(\cdot,\cdot)$ is the Kullback-Leibler divergence. The cumulative regret is called near optimality if $R_N(\pi) = O(\log N)$, or optimality if it achieves this optimal growth rate specified by the lower bound.

In the literature, most researchers only considered a one-parameter exponential family (Perchet & Rigollet, 2013; Chan, 2019). For wider applications, we study a general exponential family defined by

$$f(y,\theta,\phi) = e^{\frac{\theta y - g(\theta)}{\phi}} h(y,\phi),\ \phi > 0, \tag{3}$$

where functions g and h are decided by a specific probability distribution. When $h(y,\phi) = 1/\phi$, it is a standard exponential distribution. The normal distributions $N(\mu,\sigma^2)$ are included with $\theta = \mu, \phi = \sigma^2, g(\theta) = \theta^2/2, h(y,\phi) = y^2/(2\phi) + 0.5\log(2\pi\sigma^2)$. Let $f_k(y) = f(y,\theta_k,\phi_k)$ for $k = 1,\ldots,K$. Let $f_* = f(\cdot,\theta_*,\phi_*)$, where $\mu_* = \min_{1\leq k\leq K}\mu_k$ and θ_*, ϕ_* are the corresponding parameters. Note that $\mu = \mathbb{E}Y = g'(\theta)$ and $Var(Y) = \phi g''(\theta) > 0$. Let $b(\mu)$ be the inverse function of $g'(\theta)$. The following theorem gives the property of the proposed sub-sampling method (Algorithm 1) for the exponential family. The near optimality is obtained by bounding the tails of distributions. The proof is given in Appendix.

Theorem 1 *For the exponential family (3), the SS policy $\hat{\pi}$ given in Algorithm 1 satisfies*

$$\limsup_{N\to\infty}\frac{R(\hat{\pi})}{\log N} \leq \sum_{k:\mu_*<\mu_k}\frac{(\mu_k-\mu_*)\phi_*}{(b(\mu_k)-b(\mu_*))\mu_k - (g(b(\mu_k)) - g(b(\mu_*)))},$$

and is thus nearly optimal.

Since the large deviation rate function is not KL divergence under the exponential family, this upper bound is not optimal. But, it has the nearly optimal order of $\log N$. Consider the classical case of the one-parameter exponential family, i.e., $\phi_k = 1$ for any k, the large deviation rate function $I_*(\mu_k)$ turns out to be KL divergence by direct calculation. It means that the sub-sampling method is optimal under the one-parameter exponential family (Chan, 2019). Corollary 1 reveals that the proposed policy is optimal for the one-parameter exponential family. The upper bound of the regret given in Perchet & Rigollet (2013) does not have this optimality.

Corollary 1 *For the one-parameter exponential family, the SS policy $\hat{\pi}$ given in Algorithm 1 satisfies*

$$\limsup_{N\to\infty}\frac{R(\hat{\pi})}{\log N} \leq \sum_{k:\mu_*<\mu_k}\frac{\mu_k-\mu_*}{KL(f_k,f_*)},$$

and is thus optimal.

From these two theoretical results, it reveals that when the distributions of different arms are more different, the convergence rate is faster since we can identify the best arm quickly. And, smaller $\mu_k - \mu*$ will naturally lead to smaller regret.

Further, we can establish a theory for BOSS that if the leading configuration is inferior, a better configuration will be chosen with probability $1 - o(n^{-1})$ as follows,

Theorem 2 *Let c_ζ be the leader in Algorithm 2, and $S_{n_k}^{(\zeta)} = \min_{1\leq j\leq n_\zeta - n_k+1}\bar{Y}_{j:(j+n_k-1)}^{(\zeta)}$. If $\mu_k < \mu_\zeta$, then $\mathbb{P}(\bar{Y}_{n_k}^{(k)} \leq S_{n_k}^{(\zeta)}) = 1 - o(\frac{1}{n})$.*

D Aggressive Algorithms

Algorithm 7 parallel BOSS

Input: Maximum budget R; minimum budget r; ratio η; maximum duration T.

Output: The configuration with the best performance.

```
1: Initialize BOSS with $R$, $r$ and $\eta$.
2: **while** there is idle GPU **do**
3:     **if** duration $t \geq T$ **then**
4:         Break
5:     **end if**
6:     Find the next configuration $c_k$ and budget $b$ to evaluate based on Algorithm 8.
7:     Evaluate configuration $c_k$ with budget $b$ and update the Gaussian Process models.
8: **end while**
```

Algorithm 8 parallel BOSS runtime

Input: Current bracket s, current SS iteration r, maximum budget R.
Output: The next configuration c_k and budget b to evaluate.

```
1:  if all configurations of current SS iteration $r$ scheduled then
2:      if all configurations of current bracket $s$ scheduled then
3:          Sample a new bracket $s_{next}$, return one configuration $c_k$ of the first SS iteration, and $b = R\eta^{-s_{next}}$.
4:      else
5:          if there is performace record of this bracket $s$ then
6:              Select the leader $c_\zeta$, which has the most observations.
7:              Return the top one configuration $c_k$ of the next SS iteration $r+1$ based on the sorting of $V_k^{(r)}(\zeta)$, and budget $b = R\eta^{-s+r+1}$.
8:          else
9:              Randomly pick one configuration $c_k$ from this bracket, with budget $b = R\eta^{-s+r+1}$.
10:         end if
11:     end if
12: else
13:     if there is performace record of this bracket then
14:         Select the leader $c_\zeta$, which has the most observations.
15:         Return the top one configuration $c_k$ of the current SS iteration $r$ based on the sorting of $V_k^{(r)}(\zeta)$, and budget $b = R\eta^{-s+r}$.
16:     else
17:         Randomly pick one configuration $c_k$ from this bracket, with budget $b = R\eta^{-s+r}$.
18:     end if
19: end if
```

E EXPERIMENTS ON REINFORCEMENT LEARNING

In last few years, several different approaches have been proposed for reinforcement learning with neural network function approximators, e.g., deep Q-learning (Mnih et al., 2015), "vanilla" policy gradient methods (Mnih et al., 2016), trust region policy gradient methods (Schulman et al., 2015), and proximal policy optimization (PPO) (Schulman et al., 2017). In these methods, there are a few hyperparameters that need to be determined. As an example, we tune the hyperparameters for the PPO Cartpole task (Falkner et al., 2018) including the numbers of units in layers 1 and 2, batch size, learning rate, discount, likelihood ratio clipping, and entropy regularization. Different from the previous three applications, the budget becomes the number of trials used by an agent.We run each configuration for nine trials and report the average number of episodes until the PPO has converged, which means that the learning agent achieves the highest possible reward for five consecutive episodes. For each configuration, we stop training after the agent has either converged or ran for a maximum of 1000 episodes. For each hyperparameter optimization method, we implement five independent runs.

Table 7. The epochs needed to learn a game with policies obtained by different HPO methods.

Method	Random	BOHB	BOSS
Min terminated epoch	102.55	80.33	**72.77**

Table 7 shows that the agent using the policy obtained by BOSS is the fastest learner. The agent in RL needs to learn a stable policy which matches the conservation of BOSS.

F PROOF OF THEOREM 1

First, we prove a lemma which is an extension of Theorem 1 in Chan (2019),

$$\limsup_{r\to\infty} \mathbb{E}n_k^{(r)}/\log r \le \sum_{k:\mu_*>\mu_k} \phi_*/[(b(\mu_k)-b(\mu_*))\mu_k - (g(b(\mu_k))-g(b(\mu_*)))].$$

For this purpose, the next main process is to develop a new Chernoff bound for the exponential family like Lemma 3 in Chan (2019). Now we claim that the following two inequalities hold for the exponential family.

$$P(\bar{Y}_{1:j}^{(k)} \ge a) \le \exp\{-jI_k(a)\} \qquad if\ a > \mu_k,$$

$$P(\bar{Y}_{1:j}^{(k)} \le a) \le \exp\{-jI_k(a)\} \qquad if\ a < \mu_k,$$

where the function $I_k(a) = \sup_{t\in\mathbb{R}}(ta - \ln\mathbb{E}\exp\{tY^{(k)}\})$ is the large deviation rate function.

The generic Chernoff bound for a random variable $\bar{Y}$ with $t > 0$ is

$$P(\bar{Y} \ge a) = P(\exp\{t\bar{Y}\} \ge exp\{ta\}) \le \frac{\mathbb{E}\exp\{t\bar{Y}\}}{\exp\{ta\}}.$$

When $\bar{Y} = \bar{Y}_{1:j}^{(k)}$ is the mean of j i.i.d. random variables $Y_1^{(k)}, \ldots, Y_j^{(k)}$, optimizing over t, we get

$$\begin{aligned}P(\bar{Y}_{1:j}^{(k)} \geq a) &\leq \inf_{t>0} \exp\{-ta\}\Pi_{i=1}^{j}\mathbb{E}\exp\{tY_i^{(k)}/j\} \\ &= \inf_{t>0} \exp\{-ta + \sum_{i=1}^{j} \ln \mathbb{E}\exp\{tY_i^{(k)}/j\}\} \\ &= \exp\{-j \sup_{t>0}(ta - \ln \mathbb{E}\exp\{tY_1^{(k)}\})\} \\ &= \exp\{-j \sup_{t>\theta_k}((t-\theta_k)a \\ &- \ln \mathbb{E}\exp\{(t-\theta_k)Y_1^{(k)}\})\}.\end{aligned}$$

Let $J_k(t) = (t-\theta_k)a - \ln \mathbb{E}\exp\{(t-\theta_k)Y_1^{(k)}\}$, we can simplify $J_k(t)$ by direct calculation that

$$J_k(t) = [(t-\theta_k)a - (g(t) - g(\theta_k))]/\phi_k.$$

Then,

$$J_k^{'}(t) = [a - g'(t)]/\phi_k,$$

and

$$J_k^{''}(t) = -g''(t)/\phi_k < 0.$$

Hence, when $a > \mu_k = g'(\theta_k)$, we have

$$\sup_{t>\theta_k} J_k(t) = J_k(b(a)) = I_k(a),$$

which means the first inequality holds.

Similarly, when $a < \mu_k = g'(\theta_k)$, we have

$$\sup_{t<\theta_k} J_k(t) = J_k(b(a)) = I_k(a).$$

The second inequality is obtained together with

$$\begin{aligned}P(\bar{Y}_{1:j}^{(k)} \leq a) &\leq \inf_{t<0} \exp\{-ta\}\Pi_{i=1}^{j}\mathbb{E}\exp\{tY_i^{(k)}/j\} \\ &= \exp\{-j \sup_{t<\theta_k}((t-\theta_k)a \\ &- \ln \mathbb{E}\exp\{(t-\theta_k)Y_1^{(k)}\})\}.\end{aligned}$$

Finally, let $\xi_k = 1/I_*(\mu_k)$ in Lemma 1 of Chan (2019), we get an extension of Theorem 1 in Chan (2019). Consequently, the near optimality can be obtained as follows,

$$\begin{aligned}&\limsup_{N\to\infty} R(\hat{\pi})/\log N \\ =&\limsup_{r\to\infty} \sum_{k:\mu_*<\mu_k} (\mu_k - \mu_*)\mathbb{E}n_k^{(r)}/\log n^{(r)} \\ \leq&\limsup_{r\to\infty} \sum_{k:\mu_*<\mu_k} (\mu_k - \mu_*)\mathbb{E}n_k^{(r)}/\log r \\ \leq& \sum_{k:\mu_*<\mu_k} (\mu_k - \mu_*)\phi_*/[(b(\mu_k) - b(\mu_*))\mu_k \\ &- (g(b(\mu_k)) - g(b(\mu_*))).\end{aligned}$$

□