IE613 PROJECT
HYPERBAND: A NOVEL BANDIT-BASED APPROACH TO HYPERPARAMETER OPTIMIZATION

Saptarshi Majumder Abhishek Narayan Chaudhury Subhadeep Chaudhuri

Supervisor: Prof. Manjesh K. Hanawal

ABSTRACT

Applying machine learning (ML) and complex predictive systems requires users to set various knobs and parameter configurations, which are broadly referred to as hyperparameters (HPs). While good performance hinges on a careful tuning of those, this process is time-consuming and crucially depends on the level of expertise of the practitioner. ML examples of HPs are the learning rate of stochastic gradient descent algorithms or the architectural choices of deep neural networks. In order to automate the tuning of HPs, various approaches such as random search or Bayesian optimization (BO) have successfully cast this problem as a global black-box optimization problem. In that formulation, querying the black-box function of interest, say f, typically corresponds to a full training of the underlying ML model together with its evaluation on some validation data. Here we discuss a specific technique of hyperparameter optimization as a non-stochastic infinite-armed bandit problem known as the Hyperband. Hyperband relies on a successive halving procedure, where a pool containing randomly sampled HPs is progressively trimmed according to a theoretically-justified resource schedule. Despite its simplicity, Hyperband was proven to outperform BO on many ML-related tuning tasks in regimes where solutions with moderate precision are acceptable.

1 Introduction

Hyperparameters are an important ingredient of machine learning algorithms. The choice of hyperparameters hugely affect the accuracy of the algorithm and also affect the computation time to reach the accepted minimization of the loss functions considered in the problem. In light of this importance of the parameters, researchers in recent years have largely worked on methods to efficiently find the optimal parameters.

Generally, hyperparameter optimization problem is categorized into two different types:

- The Hyperparameter Selection Algorithms: That attempts to improve upon the grid and random search algorithms. Researchers have found using rank plot aggregate statistics that the state-of-the-art Bayesian optimization techniques surpass the random search but they fail to show any notable difference across the benchmark data sets made by [4].
- The Hyperparameter Evaluation Algorithms and select configurations randomly.

The research on hyperparameter optimization is mainly focused on the first type of the study i.e. the selection algorithms, and is mainly studied based on Bayesian optimization. Bayesian optimization (BO) is a well-established methodology to optimize expensive black box functions. It relies on a probabilistic model of an unknown target $f(x)$ one wishes to optimize and which is repeatedly queried until one runs out of budget (e.g., time). Queries consist in evaluations of f at different HP configurations $x_1, \ldots, x_n$ selected according to an explore-exploit trade-off criterion or acquisition function. The HP configuration corresponding to the best query is then returned. One popular approach is to impose a Gaussian Process (GP) prior over f and, in light of the observed queries

$f(x_1), \ldots, f(x_n)$ compute the posterior GP. The GP model maintains posterior mean and variance functions that are required when evaluating the acquisition function for each new query of f. When n is large and scalability matters, the GP surrogate model can be replaced by other models such as random forest, or (Bayesian) neural networks.

For the sake of completeness, we now consider another modeling strategy called Tree-structured Parzen estimator (TPE)[1].The Gaussian-process(GP) based approach modeled $p(y|x)$ directly, whereas this strategy models $p(x|y)$ and $p(y)$.To that end, $p(y|x)$ is modeled by transforming the generative process of hyperparameters, replacing the distributions of the configuration prior with non-parametric densities. For the configuation space $\mathcal{X}$ described using uniform, log-uniform, quantized log-uniform, and categorical variables, the TPE algorithm makes the following replacements: uniform $\rightarrow$ truncated Gaussian mixture, log-uniform $\rightarrow$ exponentiated truncated Gaussian mixture, categorical $\rightarrow$ re-weighted categorical.Using different observations $\{x(1), ..., x(k)\}$ in the non-parametric densities, these substitutions represent a learning algorithm that can produce a variety of densities over the configuration space $\mathcal{X}$.The TPE defines $p(x|y)$ using two such densities:

$$p(x|y) = \begin{cases} l(x) & \text{if } y \leq y* \\ g(x) & \text{if } y \geq y* \end{cases}$$

where $l(x)$ is the density formed by using the observations $\{x(i)\}$ such that corresponding loss $f(x(i))$ was less than $y*$ and $g(x)$ is the density formed by using the remaining observations. Whereas the GP-based approach favoured quite an aggressive $y*$ (typically less than the best observed loss), the TPE algorithm depends on a $y*$ that is larger than the best observed $f(x)$ so that some points can be used to form $l(x)$. The TPE algorithm chooses $y*$ to be some quantile γ of the observed y values, so that $p(y \geq y*) = \gamma$, but no specific model for $p(y)$ is necessary.

In recent studies, the use of non-stochastic multi-arm bandit algorithm suggested initially in Successive Halving [7] and further modified in Hyperband, is studied extensively to get a fast, simple, general purpose algorithm with very less number of inputs. The algorithm Hyperband addresses the problem of HP tuning by incrementally allocating more resources to the best-performing candidates initially taken from a pool of randomly sampled candidates. Supported by theoretical guarantees for correctness and sample complexity, Hyperband follows a resource allocation schedule determined from two parameters, namely the fraction $\frac{1}{\eta}$ of candidates to incrementally discard (by default, η = 3) and the maximum resources to be allocated to a given candidate (e.g., a maximum number of epochs or a certain maximum time budget). This schedule is structured in several independent brackets that represent different resource-allocation trade-offs, for instance, having many initial candidates run for the smallest amount of resources or starting with a few candidates immediately benefiting from the maximum resources. As a result, an execution of Hyperband produces the data $D^{HB} = [\bigcup_{b \in brackets} D^{HB,b}]$ where $D^{HB,b}$ is formed of all the N_b triplets $f(x^n, y^n, r^n)_{n=1}^b$ evaluated in the bracket b, recording the HP candidates, their evaluations and the corresponding resources used.

The project report proceeds as follows: In Section 2, we lay the foundation of the ideas behind Hyperband, followed by discussions on Hyperband in Section 3, and finally we conclude with some proposed future extensions of work in Section 4.

2 Literature Survey

The hyperparameter optimization problem can be formulated as follows:

Let $\theta_1, ..., \theta_n$ denote the hyperparameters of a machine learning algorithm, and let $\Theta_1, ..., \Theta_n$ denote their respective domains. The algorithm's hyperparameter space is then defined as $\Theta = \Theta_1 \times \ldots \times \Theta_n$. When trained with $\theta \in \Theta$ on data D_{train}, we denote the algorithm's validation error on data D_{valid} as $V(\theta, D_{train}, D_{valid})$. Using k-fold cross-validation, the hyperparameter optimization problem for a given dataset D is to minimize[5]:

$$f^D(\theta) = \frac{1}{k} \sum_{i=1}^{k} V(\theta, D^i_{train}, D^i_{valid})$$

Hyper parameters θ_i can be numerical (real or integer, as, e.g., the strength of a regularizer) or categorical (unordered,with finite domain, as, e.g., the choice between different kernels). Furthermore, there can be conditional hyper parameters, which are only active if another hyper-parameter takes a certain value; for example, setting the "number of principal components" is conditioned on the usage of PCA as a pre-processing method. The most frequently used hyper-parameter optimization method is grid search, which often performs poorly and does not scale to high dimensions. Therefore, a large body of recent work has focused on better-performing methods, in particular Bayesian Optimization.

More recently, researchers have considered the hyperparameter optimization technique as a non-stochastic multi-armed bandit problem in Successive Halving [7], and as a infinitely armed pure exploration bandit problem in the Hyperband algorithm [8]. In the Hyperband Algorithm the optimisation algorithm can be considered as

$\mathcal{X}$ – Space of valid hyperparameter configurations
$l_k : \mathcal{X} \to [0, 1]$ – Sequence of loss functions defined over $\mathcal{X}$
$l_k(x)$ – Validation error of model trained with k units of resources , $x \in \mathcal{X}$

We define,

$$l_* = \lim_{k \to R} l_k$$
$$\nu_* = \inf_{x \in \mathcal{X}} l_*(x)$$

Objective: Identify hyperparameter configuration $x \in \mathcal{X}$ s.t. $l_*(x) - \nu_*$ is minimized by drawing as many random configurations as desired while using as few total resources as possible.

2.1 Successive Halving

Successive Halving [7] optimizes on the resource allocations. Successive Halving improves on Random Search by dividing and selecting randomly generated hyperparameter configurations more efficiently than Random Search, without making more assumptions about the nature of the configurations space.

Successive Halving is a robust, general-purpose solution to the Non-stochastic best arm identification problem. Let us first state some of the issues encountered in the non-stochastic setting as follows:

- No information about the rate of convergence of each arm's loss sequence. We only know that the sequence of loss for each arm would converge eventually.
- The sequence of loss might exhibit a high degree of non-smoothness and non-monotonicity.
- Cost of computing validation loss for a particular hyperparameter setting midway is often more expensive than performing a single training iteration.

Proposed Algorithm

The central idea for the proposed algorithm is as follows: Given an input budget (for eg. training time),

- Uniformly allocate the budget to a set of arms for a predetermined amount of iterations.
- Evaluate performance.
- Throw out the worst half
- Repeat until just one arm remains

The exact algorithm is being given below [1]

Algorithm 1 Successive Halving

Input: Budget B, n arms where $l_{i,k}$ denotes the k^{th} loss from the i^{th} arm
Initialize: $S_0 = [n]$
for $k = 0, 1, \ldots, \lceil log_2(n) \rceil - 1$ **do**
    Pull each arm in S_k for $r_k = \lfloor \frac{B}{|S_k| \lceil log_2(n) \rceil} \rfloor$ additional times and set $R_k = \sum_{j=0}^{k} r_j$
Output: Singleton element of $S_{\lceil log_2(n) \rceil}$

We might also remove the "budget" input by what authors call the "doubling trick", as follows:

- Set $B \leftarrow n$ where n is the number of arms
- Run the algo to completion with B
- Set $B \leftarrow 2B$
- Repeat to infinity

Analysis of Successive Halving

We define $\nu_i = lim_{\tau\to\infty} l_{i,\tau}$ which exists by assumption. WLG, we assume $\nu_1 < \nu_2 \leq \cdots \leq \nu_n$. For each $i \in [n]$, let $\gamma_i(t)$ be the point-wise smallest, non-increasing function of t with $|l_{i,t} - \nu_i| \leq \gamma_i(t) \ \ \forall t$. We also define $\gamma_i^{-1}(\alpha) = min\{t \in \mathbb{N} : \gamma_i(t) \leq \alpha\}$ for all $i \in [n]$. We state the three main theorems stated in the paper, which also compares and exposes the gap between the uniform allocation of resources and the Successive Halving algorithm.

Theorem 1 *Let* $\nu_i = lim_{\tau\to\infty} l_{i,\tau}, \bar{\gamma}(t) = max_{i=1,\ldots,n}\gamma_i(t)$ *and* $Z_{SH} = 2\lceil log_2(n)\rceil max_{i=2,\ldots,n}\ i(1 + \bar{\gamma}^{-1}\left(\frac{\nu_i-\nu_1}{2}\right)) \leq 2\lceil log_2(n)\rceil\left(n + \sum_{i=2,\ldots,n}\bar{\gamma}^{-1}\left(\frac{\nu_i-\nu_1}{2}\right)\right)$.

If Successive Halving run with a budget $B > Z_{SH}$*, then the best arm is guaranteed to be returned. Moreover, if the Successive Halving algorithm is bootstrapped by the "doubling trick" that takes no arguments as input, then this procedure returns the best arm once the total number of iterations taken exceeds just* $2Z_{SH}$.

Theorem 2 *(Uniform strategy-sufficiency) Let* $\nu_i = \lim_{\tau\to\infty} l_{i,\tau}$, $\bar{\gamma}(t) = max_{i=1,\ldots,n}\gamma_i(t)$ *and* $z_U = max_{i=2,\ldots,n}\ n\ \bar{\gamma}^{-1}\left(\frac{\nu_i-\nu_1}{2}\right)$.

The uniform strategy takes no input arguments and returns the best arm at timestep t *for all* $t > z_U$

Theorem 3 *(Uniform strategy-necessity) For any timestep* t *and final values* $\nu_1 < \nu_2 \leq \ldots \leq \nu_n$*, there exists a sequence of losses* $\{l_{i,t}\}_{t=1}^{\infty}, i = 1, 2, ..., n$ *such that if* $t < max_{i=2,\ldots,n}\ n\bar{\gamma}^{-1}\left(\frac{\nu_i-\nu_1}{2}\right)$ *then the uniform strategy does not return the best arm at timestep* t.

However, the Successive Halving algorithm also suffers from drawbacks, the primary one being that in Successive Halving, there is a trade-off between how many configurations we need to select at start and how many cuts we need (known as the "n vs. B/n" trade-off). Hyperband, which is the main focus of this project, solves this issue. But before diving into it, we first look at some other bandit settings of interest.

2.2 Infinitely Armed Bandit

The main challenges of the infinitely many armed bandits with respect to the multi-armed bandits come from two sources. First, we need to find a good arm among the sampled ones. Second, we need to sample (at least once) enough arms in order to have (at least once) a reasonably good one. These two difficulties ask for a while which we call the arm selection trade-off. It is different from the known exploration/exploitation trade-off and more linked to model selection principles: On one hand, we want to sample only from a small subsample of arms so that we can decide, with enough accuracy, which one is the best one among them. On the other hand, we want to sample as many arms as possible in order to have a higher chance to sample a good arm at least once. This trade-off makes the problem of infinitely many armed bandits significantly different from the classical bandit problem.In the paper [3] they have discussed an algorithm for Infinitely many armed bandits named Simple Regret for Infinitely many arms (SiRI) and its analysis which is given in [3].In the theorem below we show a bound on the regular regret obtained by [3] for the infintely armed bandit case

Theorem 4 *Let* $\delta > 0$*. Assume that* n *is larger than a constant that depends on* $\beta, \bar{E}, \bar{E}', \bar{B}, C$*. Depending on the value of* β*, we have the following results, where* E *is a large enough constant.*

- *Case* $\beta < 2$: *With probability larger than* $1 - \delta$*,*

$$r_n \leq E_n^{-1/2} log(1/\delta)(log(log(1/\delta)))^{96} \sim n^{-1/2}$$

- *Case* $\beta > 2$: *With probability larger than* $1 - \delta$*,*

$$r_n \leq E(n\ log(n))^{-1/\beta}(log(log(log(n)/\delta)))^{96} \times log(log(n)/\delta) \sim (n\ logn)^{-1/\beta} polyloglog\ n$$

- *Case* $\beta = 2$: *With probability larger than* $1 - \delta$*,*

$$r_n \leq E\ log(n)n^{-1/2}(log(log(log(n)/\delta)))^{96} \times log(log(n)/\delta) \sim n^{-1/2} log\ n\ polyloglog\ n$$

From this algorithm they have concluded the bound they obtained (Theorem 4) is minimax optimal for $\beta < 2$ without additional log n factors. We emphasize it since the previous results on infinitely many armed bandits give results which are optimal up to a polylog n factor for the cumulative regret. For $\beta \geq 2$, their result is optimal up to a polylog n factor. They conjecture that the lower bound for $\beta \geq 2$ can be improved to $(\log(n) = n)1 = \beta$ and

that SiRI is actually optimal up to a $polyloglog(n)$ factor for $\beta > 2$. Moreover for an infinite time horizon using use a doubling trick to double the size of the sample in each period and throw away the preliminary samples that were used in the previous period. Using these modifications it is straightforward to transform SiRI into an anytime algorithm. The simple regret in this anytime setting will only be worsened by a $polylogn$, where n is the unknown horizon. Specifically, in the anytime setting, the regret of SiRI modified either using the doubling trick has a simple regret that satisfies with high probability

$$r_n = O(polylog(n)max(n^{\frac{-1}{2}}; n^{\frac{-1}{\beta}} polylog(n))) \tag{1}$$

2.3 Pure Exploration

In the paper [2] researchers have discussed about Pure exploration in multi armed bandit which has inspired the infinitely armed bandit which is studied to use in the hyperband technique.In the article the researchers have addressed pure exploration problem as the design of strategies making the best possible use of available numerical resources (e.g., as CPU time) in order to optimize the performance of some decision-making task. That is, it occurs in situations with a preliminary exploration phase in which costs are not measured in terms of rewards but rather in terms of resources, that come in limited budget. A motivating example concerns recent works on computer-go. A given time, i.e., a given amount of CPU times is given to the player to explore the possible outcome of a sequences of plays and output a final decision. An efficient exploration of the search space is obtained by considering a hierarchy of forecasters minimizing some cumulative regret.Below we demonstarted the algorithm for pure exploration as demonstrated in the research paper[2].

Algorithm 2 Pure Exploration in Multi-Armed Bandits

1: **Input:** K probability distributions for the rewards of the arms, $\nu_1, \ldots, \nu_K$
2: **for** each round t= 1,2,. . . **do**
3: The forecaster chooses $\phi_t \in P1, ..., K$ and pulls It at random according to ϕ_t;
4: The environment draws the reward Y_t for that action
5: The forecaster outputs a recommendation $\psi_t \in P1, ..., K$;
6: If the environment sends a stopping signal, then the game takes an end; otherwise, the next round starts.

3 Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization

Having set the stage with the previous discussions, we now dive into the main algorithm of focus, the Hyperband algorithm [8]. Hyperband is parameter-free and has strong theoretical guarantees for correctness and sample complexity. The approach relies on an early-stopping strategy for iterative algorithms of machine learning. The rate of convergence does not need to be known in advance and, in fact, the algorithm adapts to it so that if we replace our iterative algorithm with one that converges faster, the overall hyperparameter selection process is that much faster.

The underlying principle of the procedure exploits the intuition that if a hyperparameter configuration is destined to be the best after a large number of iterations, it is more likely than not to perform in the top half of configurations after a small number of iterations. That is, even if performance after a small number of iterations is very unrepresentative of the configurations absolute performance, its relative performance compared with many alternatives trained with the same number of iterations is roughly maintained.

3.1 Proposed Algorithm

The proposed Hyperband algorithm requires the following -

- The ability to sample a hyperparameter configuration *(get_random_hyperparameter_configuration()*)
- The ability to train a particular hyperparameter configuration until it has reached a given number of iterations (perhaps resuming from a previous checkpoint) and get back the loss on a validation set (*run_then_return_val_loss(num_iters = r,hyperparameters = t)*).

Uniform random sampling is an obvious choice of hyperparameter selection as it is typically trivial to implement over categorical, continuous, and highly structured spaces. However, the procedure under consideration is actually indifferent to the sampling distribution: the procedure performs better when the distribution is more well defined.

Almost all iterative algorithms (e.g. gradient methods) expose the ability to specify a maximum iteration, thereby fulfilling the general applicability of the two main requirements of this proposed algorithm. The python code snippet given below (Fig. 1)[6] may be considered to be the entirety of the codebase of Hyperband.

```
################################################################################
## We need to write the following hooks for our custom problem
################################################################################
from problem import get_random_hyperparameter_configuration,run_then_return_val_loss

max_iter = 81  ## Maximum iterations/epochs per configuration
eta = 3        ## Defines downsampling rate (default=3)
logeta = lambda x: log(x)/log(eta)
s_max = int(logeta(max_iter))  ## Number of unique executions of Successive Halving (minus one)
B = (s_max+1)*max_iter         ## Total number of iterations (without reuse) per execution of Succesive Halving (n,r)

#####################################################################
## Begin Finite Horizon Hyperband outlerloop. Repeat indefinetely.
#####################################################################

for s in reversed(range(s_max+1)):
    n = int(ceil(int(B/max_iter/(s+1))*eta**s)) ## Initial number of configurations
    r = max_iter*eta**(-s)                      ## Initial number of iterations to run configurations for

    #############################################################
    #### Begin Finite Horizon Successive Halving with (n,r)
    #############################################################
    T = [ get_random_hyperparameter_configuration() for i in range(n) ]
    for i in range(s+1):
        ## Run each of the n_i configs for r_i iterations and keep best n_i/eta
        n_i = n*eta**(-i)
        r_i = r*eta**(i)
        val_losses = [ run_then_return_val_loss(num_iters=r_i,hyperparameters=t) for t in T ]
        T = [ T[i] for i in argsort(val_losses)[0:int( n_i/eta )] ]

    ########  End Finite Horizon Successive Halving with (n,r)  ########
```

Figure 1: Hyperband pseudocode in Python

The Hyperband algorithm builds upon the Successive Halving algorithm (Algo. 1) with the outer loop of the algorithm passing over varying parameter choices, looping over varying degrees of breadth vs depth based search. The Successive Halving algorithm forms the inner loop, with the subroutine being called for a particular value of chosen hyperparameter inputs to the subroutine.

3.2 Analysis of Hyperband algorithm

As stated in the research paper [8] HyperBand technique is a modification of the Successive Halving Algorithm so using the Theorems [1],[2],[3] we can easily conclude that the HyperBand works better than the General setting .Moreover the HyperBand being and infinitely armed pure exploration problem the reserchers in the paper [8] have proved a similar bound on the simple regret just as discussed in the Equation[1] abouve . The bound on simple regret obtained in the paper is given by,

$$r_n = O(polylog(\frac{B}{\delta})max(B^{\frac{-1}{2}}; B^{\frac{-1}{\beta}})) \tag{2}$$

where polylogarithm is defined by a power series in z, which is also a Dirichlet series in s.

In the above, B is demonstrated as the budget allocation or we can define it as the number of rounds we will train with an hyper parameter.

3.3 Experiments

We perform hyperparameter optimization using the Hyperband algorithm on the MNIST dataset. The MNIST dataset is split into 60000 train samples and 10000 test samples, with each image of shape $(28, 28)$. The entire work was done on Google Colab, with each file consuming $\sim$6.2GB of the assigned RAM. The work was done using GPU hardware accelerator, whose exact configuration varies over time and include Nvidia K80s, T4s, P4s and P100s.

As demonstration, the search space is taken as follows:

```
Search space summary
Default search space size: 11
num_filters_1 (Choice)
{'default': 32, 'conditions': [], 'values': [16, 32, 64], 'ordered': True}
kernel_1 (Choice)
{'default': 3, 'conditions': [], 'values': [2, 3, 4, 5], 'ordered': True}
stride_1 (Choice)
{'default': 1, 'conditions': [], 'values': [1, 2, 3], 'ordered': True}
num_filters_2 (Choice)
{'default': 32, 'conditions': [], 'values': [16, 32, 64], 'ordered': True}
kernel_2 (Choice)
{'default': 3, 'conditions': [], 'values': [2, 3, 4, 5], 'ordered': True}
stride_2 (Choice)
{'default': 1, 'conditions': [], 'values': [1, 2, 3], 'ordered': True}
dropout_1 (Float)
{'default': 0.3, 'conditions': [], 'min_value': 0.0, 'max_value': 0.5, 'step': 0.1, 'sampling': None}
dense_units (Int)
{'default': 128, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
dense_activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh', 'sigmoid'], 'ordered': False}
dropout_2 (Float)
{'default': 0.5, 'conditions': [], 'min_value': 0.0, 'max_value': 0.5, 'step': 0.1, 'sampling': None}
learning_rate (Float)
{'default': 0.001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.01, 'step': None, 'sampling': 'log'}
```

Figure 2: Hyperparameter search space under consideration

On completion of the algorithm execution, the model gives the following hyperparameter configurations as the optimal set:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 32)        832
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 64)        32832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 10, 10, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 10, 10, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 352)               2253152
_________________________________________________________________
dropout_1 (Dropout)          (None, 352)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                3530
=================================================================
Total params: 2,290,346
Trainable params: 2,290,346
Non-trainable params: 0
```

Figure 3: Optimal Hyperparameter Configuration

The evaluation of accuracy of the test dataset with the optimal hyperparameter configuration as shown in Fig.3 yields $\sim 99.27\%$ accuracy.

We now state some of the key results demonstrated by the authors on MNIST. Fig.4 shows an empirical comparison of the average test error across 70 trials of the individual brackets of Hyperband run separately as well as standard Hyperband. In practice, we do not know a priori which bracket $s \in \{0, \ldots, 4\}$ will be most effective in identifying good hyperparameters, and in this case neither the most (s = 4) nor least aggressive $(s = 0)$ setting is optimal. However, we note that Hyperband does nearly as well as the optimal bracket $(s = 3)$ and outperforms the baseline uniform allocation (i.e., random search), which is equivalent to bracket $s = 0$.

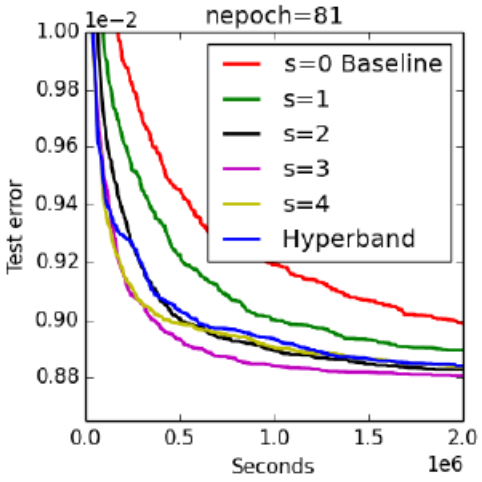

Figure 4: Performance of individual brackets in SH and Hyperband

4 Future Work

We now state some of the key follow-up works that have been taken up by researchers post introduction of the Hyperband algorithm for hyperparameter optimization. We also state the key observations from those works in comparison to the Hyperband algorithm, and provide a bird's eye view in this report.

4.1 HyperUCB

Hyperband[8], as discussed above, is an amended version of random search algorithm in which there is no learning to guide the search. Despite the fact that Hyperband is highly efficient for finding a good configuration, it does not find an optimum fast enough. Hence, modeling the hyperparameter optimization as a learning problem is more reliable than a search algorithm (full exploration problem). Therefore, instead of only sampling i.i.d. configurations of hyperparameters as Hyperband does, the authors propose to leverage the information of previous batches in order to pre-evaluate sampled configurations and to discard unpromising ones(introduces or teases the idea of context into the picture with contextual bandits). This is done by a UCB bandit strategy in a contextual setting.

The authors introduce HyperUCB [9], a model-based bandit framework, to accommodate exploitation into the purely exploratory algorithm of Hyperband. In HyperUCB, the arm selection is carried out by incorporating an Upper Confidence Bound (UCB) strategy to guide the search within the iterations in order to balance exploration vs. exploitation. They further model the arms in a contextual setting which generalizes the model for unseen arms (i.e., unseen hyperparameter configurations). Therefore, they employ a modified version of LinUCB (in which the outcome of every arm is modeled as a linear function of the context) in their approach to achieve a model-based Hyperband for the task of hyperparameter optimization. Empirically, they show that their proposed approach either outperforms Hyperband or performs on par on optimizing the hyperparameters of a deep learning model.

4.1.1 Problem

Let $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$ be a data set and M be a learning algorithm.The data is usually split into a training set for optimizing the parameters of the model, a validation set for optimizing the hyperparameters and a test set for evaluating the overall performance of the model. Assume that $\mathcal{H}$ is the set of all possible hyperparameter configurations, denoted by $\mathcal{L}(\lambda)$ the loss of M using $\lambda \in \mathcal{H}$ on the validation set. The goal is to find the best hyperparameter configurations $\lambda^* = argmin_\lambda \mathcal{L}(\lambda)$ which minimizes the validation loss for a given budget.

4.1.2 Contextual Bandits setting

In the contextual setting, reward is given as $r_t = R(x_t, I_t) + \eta_t$ where $x_t \in C \subset \mathbb{R}$ is the context vector, $I_t \in [k]$ is the action chosen based on x_t, $R : C \times [k] \to \mathbb{R}$ is the original reward function and η_t is conditionally sub-gaussian.

The policy $\pi : C \rightarrow [k]$ with $R_T(\pi)$ is given as,

$$R_T(\pi) = \mathbb{E}[\sum_{t=1}^{T} R(x_t, a_t^*) - \sum_{t=1}^{T} R(x_t, I_t)]$$

where $a_t^* = argmax_{a \in [k]} R(x_t, a)$

4.1.3 Contextual HyperUCB

In this section, we present the approach by the authors to upgrade Hyperband to a contextual bandit method using a UCB strategy. Let $\mathcal{H}$ be the space of all possible hyperparameter configurations for a machine learning approach. We are interested in finding $\lambda \in \mathcal{H}$ that gives the best performance y^* in terms of the validation loss $\mathcal{L}$ of the model,

$$\lambda^* = argmin_\lambda \mathcal{L}(\lambda)$$

We assume that a hyperparameter configuration can be represented by a d-dimensional vector λ and model the contextual bandit as a linear function of the configurations. After learning the parameters θ of the linear model, a new configuration λ can be evaluated as $\bar{y} = \theta^T \lambda$. The optimization problem in above equation suggests a lower confidence bound strategy since we aim to minimize $\mathcal{L}$.

4.1.4 Comparisons with Hyperband

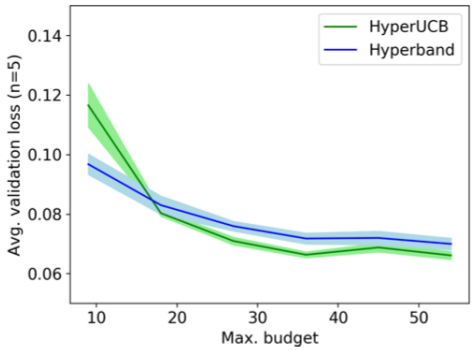

Fig. 1. Performance w.r.t. the budget.

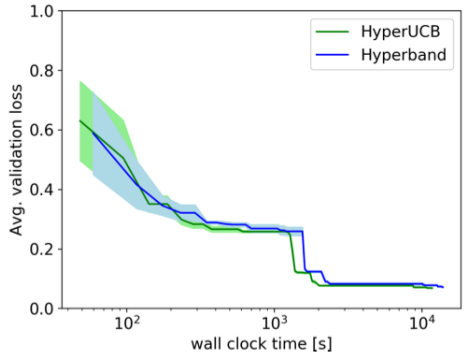

Fig. 2. Performance w.r.t. the time.

Figure 5: Comaprison of HyperUCB and HyperBand

Observations:

- Fig. 1 shows the validation loss averaged over five independent runs for various maximum budgets including standard errors. HyperUCB outperforms Hyperband as it consistently yields lower validation errors. This finding is based on the fact that using a higher budget, more rounds are conducted on which the bandit model can learn to discriminate promising from unpromising hyperparameter configurations. This can be hardly done with lower maximum budgets due to the lack of training data.
- Fig. 2 depicts the average validation loss in dependence of computational time, measured in seconds, for a budget of 45. It can be seen that HyperUCB performs at par with Hyperband, meaning it is as fast or faster than Hyperband.

4.2 A simple transfer-learning extension of Hyperband

In the research paper [10], the authors have discussed a transfer learning extension to the HyperBand idea. The main contributions of their paper are as follows

- They provide a scalable, model-based extension of Hyperband using the adaptive Bayesian linear regression (ABLR) model. It is done to take advantage of its established scalability and transfer learning capabilities.
- Their method supports transfer learning at two levels: within a single run of Hyperband, across different Hyperband runs (on related HP optimization tasks). Following the philosophy that emphasizes simplicity as a core feature, it is argued that this approach is even simpler, discarding any additional heuristics.
- They demonstrate the benefits of their methodology on synthetic simulations to tune the learning rates for the stochastic optimization of linear models and on real-world binary classification tasks with XGBoost. The experimental results indicate that ABLR+Hyperband consistently outperforms the approaches.

In the paper[10], the researchers have disscussed about the Bayesian optimization technique and also introduced Adaptive Bayesian linear regression (ABLR) and combined the idea with Hyperband. ABLR considers a joint model for the responses y_t made of two parts. First a shared feature map $\phi_z(x) : R^P \rightarrow R^D$ corresponding to a feedforward NN with D output units, where z collects all its weights and biases. Second, separate Bayesian linear regression surrogates that share the feature map $\phi_z(x))$ to model the black-box functions (noting $\Phi_z(X_t) = [\phi_z(x_t^n)]_n \in R^{N_t \times D}$)

$$P(y_t|w_t, z, \beta_t) = N(\Phi_z(X_t)w_t, \beta_t^{-1} I_{Nt}), P(w_t|\alpha_t) = N(0, \alpha_t^{-1} I_D)$$

The proposed combination of ABLR with Hyperband straightforwardly extends to the transfer learning setting. Assuming there are T tasks with data $[D_t^{HB}]_{t=1}^T$, corresponding for instance to the application of Hyperband to a given ML model over T different datasets. They adopt the methodology considering one Bayesian linear regression per HP optimization task and sharing the same feature map $\phi_z(x, r)$ across them. As a result, they obtained a transfer learning variant of Hyperband that can simultaneously exploit the data collected during previous Hyperband-based tuning tasks and within the current Hyperband run.

Comparison With HyperBand: In Fig. 6 [10] below, we demonstrate the comparison with the Hyperband technique obtained by the researchers. Their experimental protocol follows a leave-one-task-out procedure, where a BO problem is solved for a given held-out task using the data from the T - 1 remaining tasks. They thereafter report results averaged over both the T leave-one-task-out folds and the 30 random replications of the experiments. They aggregate results across tasks by first normalizing according to the results.

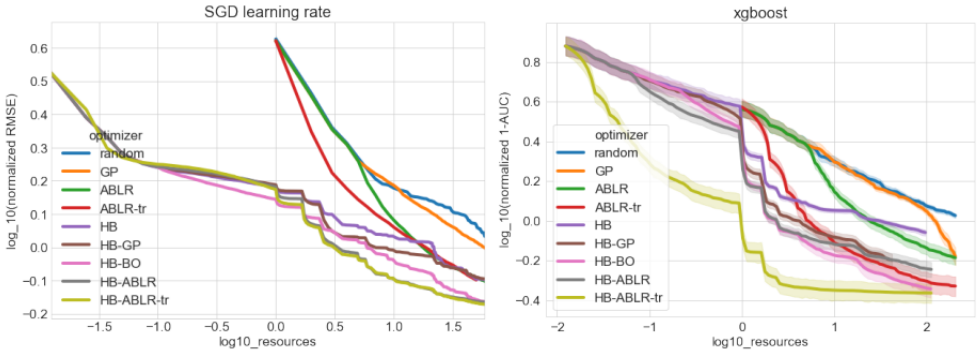

Figure 6: Left Diagram shows the comparison in a linear regression problem and the right diagram depicts comparison over the binary classification problem

5 Conclusion

In this report, we have briefly discussed about the Hyperband algorithm, its requisite theoretical blocks (such as infinite armed bandits, pure exploration) and some past works that acted as a precursor to its ideation (such as Successive Halving algorithm). We also touched upon the important theorems stated in some of the papers that provide a theoretical basis of the stated algorithms' validity.

Lastly, as part of the future work (follow-up work on Hyperband) we plan on taking up, we discussed briefly about the idea of HyperUCB and the transfer-learning extension of the Hyperband algorithm. We showed some of the empirical comparisons shown by the researchers in these future works as well, but chose not to include the graphs from the main Hyperband paper because we plan on simulating those results ourselves and include it in the main report.

References

[1] Aloïs Bissuel. Hyper-parameter optimization algorithms: a short review.

[2] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration for multi-armed bandit problems, 2010.

[3] Alexandra Carpentier and Michal Valko. Simple regret for infinitely many armed bandits, 2015.

[4] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[5] Matthias Feurer, Jost Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Feb. 2015.

[6] Kevin Jamieson. Hyperband: A novel bandit-based approach to hyperparameter optimization.

[7] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 240–248, 2016.

[8] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[9] Maryam Tavakol, Sebastian Mair, and Katharina Morik. Hyperucb: Hyperparameter optimization using contextual bandits. In Peggy Cellier and Kurt Driessens, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 44–50, Cham, 2020. Springer International Publishing.

[10] L. Valkov, Rodolphe Jenatton, Fela Winkelmolen, and C. Archambeau. A simple transfer-learning extension of hyperband. 2018.