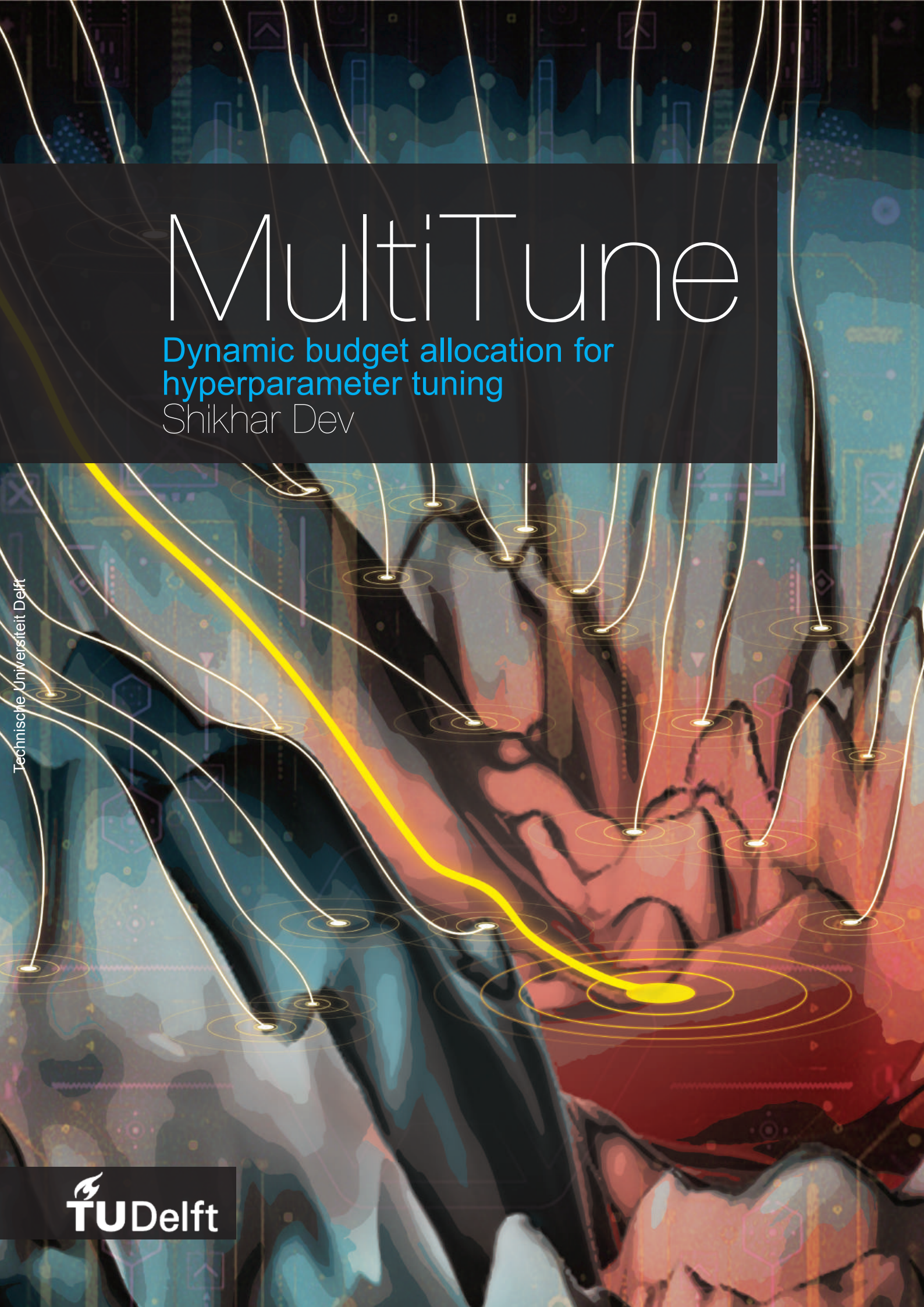

MultiTune
Dynamic budget allocation for hyperparameter tuning
Shikhar Dev
Technische Universiteit Delft
TU Delft

MultiTune

Dynamic budget allocation for hyperparameter tuning

by

Shikhar Dev

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday August 27, 2020 at 01:30 PM.

Student number:	4773071	
Project duration:	November 12, 2019 – August 27, 2020	
Thesis committee:	Dr. Lydia Chen,	TU Delft, supervisor
	Dr. Neil Yorke-Smith,	TU Delft
	Dr. Jan Rellermeyer,	TU Delft

This thesis is confidential and cannot be made public until August 31, 2020.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Hyperparameter optimization(HPO) forms a critical aspect for machine learning applications to attain superior performance. BOHB (Bayesian Optimization and HyperBand) is a state of the art HPO algorithm that approaches HPO in a multi-armed bandit strategy, augmented with Bayesian optimization to drive configuration sampling. However, BOHB requires predefined distribution of fidelities for each tuning task. The challenge in this is that it is impossible to define fidelities a priori, since each machine learning model is uniquely complex and requires different amount of compute resources for convergence. Furthermore, in our empirical analysis, we found that each HPO task rendered different performance trajectories on different fidelity (budget) types. Thus, the challenge of defining fidelities also extends to choosing an optimal budget type. To alleviate these challenges, we present MultiTune: a budget allocation scheme that builds on top of BOHB to dynamically define fidelities for optimization. MultiTune incorporates an algorithm to dynamically choose a preferred budget type for an HPO task, coupled with 2D gradient based budget constraint explorations to enable granular definition of fidelities. Through our empirical analysis, we show that MultiTune can consistently converge to a well performing configuration without significant computation overhead.

Shikhar Dev
Delft, August 2020

Preface

The thesis that lies before you summarises research entailing 9 months of hard work. The research builds upon a background of machine learning, deep learning and aspects of AutoML developed over last 2 years of my time here at TU Delft. The research began by attempting to improve state-of-art hyperparameter tuning algorithms by conjugating multi-fidelity approaches with search algorithms that haven't been studied together in the past. However, a conclusion drawn in the beginning phases triggered a pivot to approach a more fundamental challenge of defining fidelities. So, in this thesis we researched methodologies of defining fidelities for machine learning models so as to enable generalizability and scalability for the state-of-the-art hyperparameter tuning algorithms.

While the research pushed me to levels I was never exposed to, this has been a deeply enriching experience. So, to attribute acknowledgements, here's to the crazy ones. Those, far far away from their island of comfort. But more importantly, here's to the people who enable that journey. This research would simply not have been possible without the guidance of my supervisor Dr. Lydia Chen. Thank you very much Lydia, for pushing me to think and work far outside my comfort zone. Your inputs were invaluable, even beyond the scope of this thesis. Here's to a life time of co-passengers - friends who stuck through thick and thin. And above all, here's to the unrelenting love, support and belief from my family. No word could sufficiently express my gratitude towards you. So here's a mere thank you, to all of you who helped build the person that have become.

I would also like to express my gratitude towards Dr. Jan Rellermeyer and Dr. Neil Yorke-Smith, who graciously accepted to be a part of the defence committee.

Shikhar Dev
Delft, August 2020

Contents

1 Introduction

Machine learning and deep learning techniques have enabled numerous data driven intelligence in tasks such as classification, regression, clustering, translation, etc. The fundamental building block of machine learning applications revolve around generalizable structures that can represent the underlying pattern in data. Such structures include Deep Neural Network(DNN), Support Vector Machines(SVM), etc. While these structures have the capacity to learn their respective model parameters automatically by training on sample data, their hyperparameters require manual tuning. Hyperparameters in a DNN may include learning parameters such as the learning rate, regularization parameter, etc as well as network parameters such as the number of hidden layers, number of nodes in a hidden layer, etc. Values assigned to hyperparameters have a significant influence on model performance, and thus tuning hyperparameters is a key requirement for designing machine learning applications.
Given its significance, numerous approaches have been studied for tuning hyperparameters. Traditional methods treated hyperparameter tuning as a black-box optimization problem, and used search algorithms such as Bayesian Optimization Evolutionary Strategies, etc to determine good values hyperparameters. However, most modern machine learning applications require extensive training to converge to a final model performance. Training duration of hours to days render traditional black-box approaches prohibitively expensive. To address this challenge, numerous approaches have been studied, that leverage multiple fidelities of the machine learning model for the tuning process. One such method, BOHB is the state-of-the-art hyperparameter tuning algorithm that follows the multi-armed bandit strategy, augmented with Bayesian Optimization driven search algorithm. This approach significantly reduces the time required for the tuning process, with the caveat of requiring a priori specification of fidelities for the machine learning model.
In this thesis, we research approaches to dynamically build fidelities that would enable BOHB to tune hyperparameters without requiring the specification of fidelities. The guideline for research would be to minimize the amount of compute resource consumed required for the tuning process, while being generalizable to a variety of machine learning tasks.
We begin this chapter in section 1.1 by specifying the motivation for this research, followed by the research questions in section 1.2 along with the contributions of this thesis in section 1.3. Finally, we conclude this chapter in section 1.4 with an overview of this document.

1.1. Motivation

Numerous approaches have been studied to speed up the hyperparameter tuning process by using Bayesian Optimization in conjunction with a multi-fidelity strategy [12, 14, 18, 20, 21]. While these approaches did offer speedup in the tuning process by using cheap evaluations, each of these methods burden the users to define fidelities. Even though BOHB improved upon the performance of these approaches, the underlying problem of defining fidelities persisted. In our evaluation of BOHB, we observed that mis-specification the algorithm entailed sever repercussions in the tuning process.

- If the budgets allocated were too low, rendering cheap fidelities through out, the algorithm did not converge to the expected performance by the network.

- If the budgets allocated were too high, the algorithm wasted significant proportion of compute resources on configurations which converged to sub-par performance.

Furthermore, while iterations of training dataset seems like an intuitive budget type to be used for definition of fidelities, Li et al. also proposed other budget types for the tuning process. However, it is not very clear what budget type would render the most effective distribution of fidelities.
So, to alleviate the challenge of specifying budgetary constraints and types for specification of fidelities, we perform an empirical study in the course of this thesis to develop a budget distribution scheme.

1.2. Research Questions

Building on the motivations established for the thesis in section 1.1, let us specify the objective of this thesis in form of research questions. The thesis revolves around the primary overarching question:

How to best allocate compute resources for tuning hyperparameter via BOHB?

We divide this question into sub-questions to perform quantitative analysis via experiments. Each research question critically informs the subsequent questions that cumulatively help answer the overarching objective of defining an approach for definition of fidelities.

1. Does varying budget constraints influence the hyperparameter tuning process?
   (a) For a specified HPO task, is there a difference in performance of different budget types?
   (b) Given an HPO task with specified budget type, how much does varying the constraint parameters influence the tuning process?
2. Having specified a budget type, can we alleviate the need for setting budget constraints?
3. Without exhaustively evaluating each budget type, can we identify the preferred type for HPO?
4. How can we integrate budget type selection and dynamic fidelity definitions in a unified algorithm?
5. Does MultiTune consistently perform better than mis-specified vanilla BOHB for hyperparameter tuning tasks on
   (a) image based datasets?
   (b) text based datasets?

1.3. Contribution of thesis

Our primary contributions on the course of this thesis have been the following.

- Remove the requirement of a priori budget specification for tuning hyperparameters via BOHB
  - Developed 3 different budget allocation algorithms with the ability to specify fidelities in a granular fashion by leveraging 2 dimensional gradient based explorations. Each algorithm explored one of the three primary budget types in the form of epochs, training duration and subset of training dataset. Each of the primary budget type was paired with a secondary budget type namely epochs with training data subset, training duration with training data subsets and training data subset with epochs.
  - Defined a heuristic to identify a preferred primary budget type for a machine learning task, and developed mathematical models to reuse readings from this heuristic evaluation phase to augment the Bayesian model used during the hyperparameter tuning experiment.
  - Developed MultiTune, an algorithm that amalgamated both the above contributions in form of a budget allocation scheme which can choose the preferred budget type, define granular fidelities and converge to the desired final performance consuming minimal resource.
- Benchmarked each of the algorithms on image as well as text based dataset.

1.4. Report Outline

The rest of this thesis report has been structured according to the following overview.
In Chapter 2, we establish the motivation for hyperparameter optimization and provide sufficient background to continue the discussion towards the state-of-the-art algorithm. In Chapter 3, we continue our discussion diving deep into BOHB, the state of the art hyperparameter optimization algorithm. We provide an introduction to the two components of BOHB: the Bayesian Optimization and HyperBand. We specifically discuss tree-structured parzen estimator, a variant of Bayesian optimization that has been used in this algorithm. We conclude this chapter discussing the advantages and the key bottlenecks of this algorithm, establishing the need for our research. In Chapter 4, we propose numerous budget allocation schemes, attempting to alleviate the fidelity specification challenge of BOHB. We identify heuristics within these algorithms that contribute in building MultiTune: the primary contribution of this thesis. Chapter 5 begins by specifying the datasets and models considered for experimentation, in form of 5 hyperparameter tuning tasks. We then continue into describing experiments that drove design decisions for the algorithms, as well as experiments benchmarking their performance on the hyperparameter tuning problems. We conclude each experiment with results observed during the experiment, along with inferences derived from the results. Finally in Chapter 6, we summarise this thesis by reviewing the research questions established in this chapter. We also identify limitations of MultiTune and define avenues of research for future work.

2

Hyperparameters and their optimization

Hyperparameters have a significant impact on the performance of machine learning models: a well tuned hyperparameter configuration can make the difference between a mediocre performance and a state-of-the-art performance [10]. This chapter briefly introduces hyperparameters and their types, establishes the need for choosing the right configuration for each of the hyperparameters and the challenges involved in converging to the right configuration. Furthermore, the chapter formalises the hyperparameter optimization problem with a mathematical notation. We follow this through by introducing the landmark approaches for hyperparameter optimization and establishing their limitations. Finally, we conclude this chapter by defining some key metrics that can be used to benchmark and evaluate the efficacy of hyperparameter tuning algorithms.

2.1. Hyperparameters

In context of a machine learning model, Hyperparameters refer to parameters that define the architecture of the model. Unlike model parameters which are learnt during the training phase via optimization algorithms such as Stochastic Gradient Descent (SGD) [19] or Adam [13], hyperparameters cannot be estimated from the training data. Tuning them often requires an expertise in machine learning and is even influenced by the developers' domain understanding of the problem. Here, developer refers to creator of the machine learning model and problem refers to the task for which the model is being designed. For example, consider the task of detecting Pneumonia via chest x-rays. This can be viewed as a classification task (Pneumonia positive or negative) with x-ray images as the input and the classification prediction as the output. Such classification models typically involve deep neural networks with varying architectures and design choices, governed by the model's hyperparameters.
Furthermore, as established via the no free lunch theorem, no one model can generalize to every problem [23]. So, for every unique machine learning task, the model architecture needs to be specially designed and tuned in order to attain the best performance. In general, the performance metric in machine learning tasks can be defined in terms of generalizing to never-before-seen dataset of distribution similar to the training dataset. Thus, approaching each pattern recognition task requires designing models with different set of hyperparameters of varying types and values. Mentioned below are some of the categories of hyperparameters that can be encountered in the model design process.

- **Continuous Hyperparameters**
  Such hyperparameters can take real values within the given range. Examples include learning rates or the value of C and γ in case of a support vector machine.
- **Integer Hyperparameters**
  Such hyperparameters can take integer values within the specified range. Examples of such hyperparameters include the number of nodes in the given layer of deep neural network (DNN), number of layers in the DNN, etc.
- **Categorical Hyperparameters** This type of hyperparameters entail categorical choices such as choosing the type of activation function for each of the nodes, or choosing the back-propogation algorithm to be followed, etc.

- **Conditional Hyperparameters** Conditional hyperparameters include parameters whose values are relevant only in case prior conditions are met. For example, choosing the value of regularization rate is required only if some regularization is being used in the neural network.

Having established the baseline understanding of hyperparameters along with an intuition of the importance to tune them let us dive deeper into understanding the impact of tuning hyperparameters via an example in section 2.2.

2.2. Why tune hyperparameters?

Consider the example of a classification model for detecting Pneumonia. On regard to the architecture of the deep learning model, if it is not sufficiently complex, the train accuracy will saturate at a low level and will never be able to accurately detect Pneumonia on unseen x-ray images. This phenomenon is called underfitting. On the other hand, if the model is overly complex, the model might result in very high training accuracy but low validation and test accuracy. This phenomenon is called overfitting and is also undesirable. So to attain justifiable accuracy on never-before-seen images or more generally never-before-seen data for any machine learning task, the model needs to be sufficiently complex and learn patterns that are the most representative of the distribution in training dataset. Consider the example demonstrated in figure 2.1. Here, the blue dots represent distribution of training samples, the red line represents the actual function and the blue line represents the predicted function by the machine learning model. As it can be observed, for the machine learning model to closely represent the actual distribution, the model should not overfit or underfit and be sufficiently complex to represent the underlying dataset. Since the complexity and the architecture of a model is largely defined by its hyperparameters, the hyperparameters have a significant impact on the model performance.

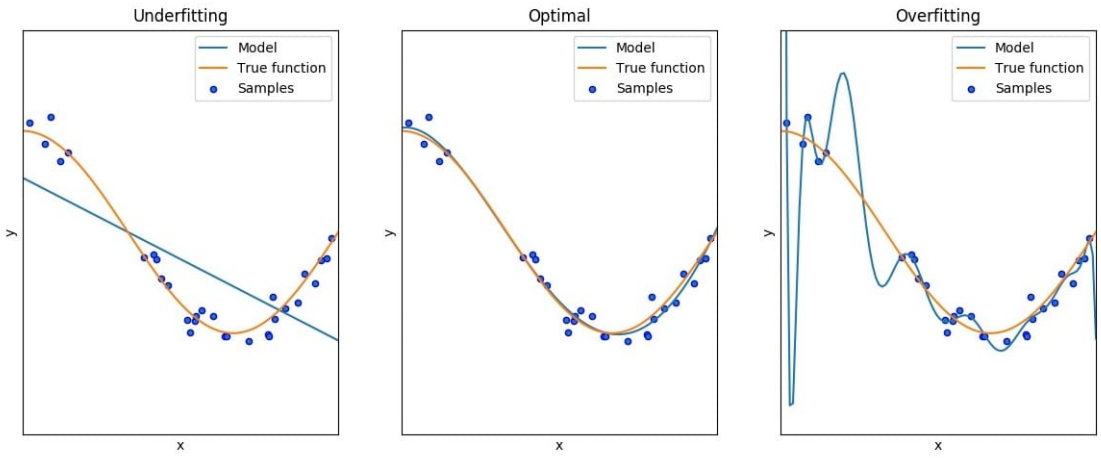


Figure 2.1: Depiction of the performance of underfitting, well tuned and an overfitting machine learning model.

It is also important to point out that the hyperparameters not only include parameters that define the architecture of the model being used, but also includes other parameters that influence the training process. Some of the examples include the learning rate, the optimization algorithm being used, etc. Let us look specifically into the learning rate and understand the influence it can have on a machine learning model. The learning rate varies between 0 and 1, and is tuned typically in a logarithmic fashion.

As it can be visualised in figure 2.2 if the learning rate is too low, the model would take significantly higher time to train for an acceptable training accuracy (Assuming the model is sufficiently complex and that the training accuracy does not saturate at a low value). This renders the training process extremely expensive and time consuming. On the other hand, if the learning rate is is very high, the back-propagation algorithm will assign unrealistic weights and biases to each of the nodes in every iteration, disallowing the model to ever converge to an acceptable accuracy. As shown in figure 2.2, the model will avoid the local loss minima all together making it impossible for the model to converge.

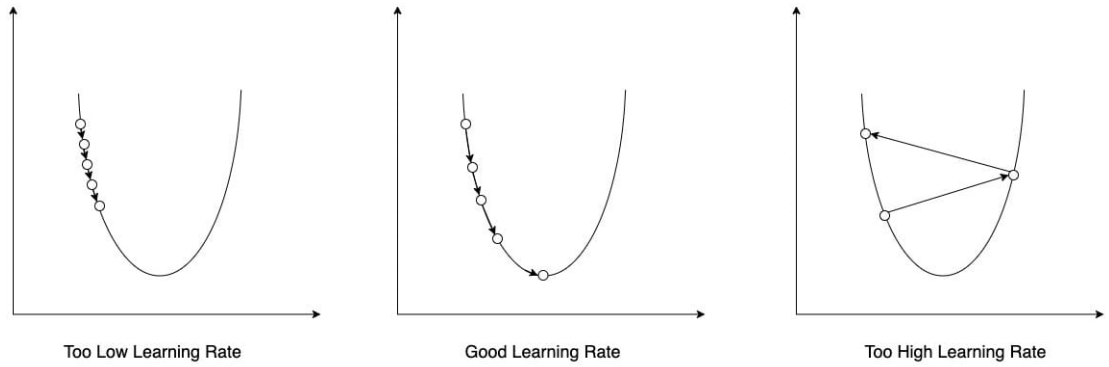


Figure 2.2: Depiction of the model optimization with different learning rates

While it is extremely important to tune the hyperparameters of machine learning models for optimal performance, tuning them is not a trivial task. Section 2.3 showcases the challenges involved in tuning hyperparameters.

2.3. Why is HPO difficult?

This section briefly explains the challenges involved in optimizing hyperparameters for a machine learning model.

- **Extremely expensive**: The traditional process of tuning hyperparameters include selecting a value for each of the hyperparameters being tuned, evaluating the model performance on the current configuration and choosing the next set of values, until exhaustion of budget or until acceptable validation accuracy has been attained. Evaluating the model performance entails building the model based on the current configuration, training it on the training set and computing the validation error. Given that modern machine learning models take days to even weeks for the training process, tuning hyperparameters manually becomes prohibitively expensive.

- **Curse of dimensionality**: Consider the process of optimizing hyperparameters of a machine learning model for a specified task. If we are tuning the learning rate for the model, the tuning range is typically limited from 10^{-6} through to 0.5, varying in logarithmic fashion. While a low resolution distribution would increase the possibility of encountering better configuration, covering the entire range of values with low resolution becomes extremely expensive. For simplicity, consider evaluating a total of 10 possibilities for the learning rate. If we need to tune one additional hyperparameter with a resolution of 10 values, the number of possible configurations increases to 100. Similarly, with increasing number of hyperparameters, the number of possible configurations increases in an exponential fashion. With this, exploring the entire search space becomes prohibitively expensive, making the tuning process extremely difficult.

- **Non-convexity**: Another complexity in the process of hyperparameter optimization is that the loss distribution of the model on regard to hyperparameter values is not convex in nature. Consider the hypothetical loss landscape showcased in figure 2.3. This figure represents the variation of validation loss of a machine learning model based on values of three hyperparameters represented in the X, Y and Z axis. Validation loss of the model can be visualised through the color at any given point. The end objective of the optimization process is to reach to the red region representing the global loss minima, as soon as possible. However, as it can be seen in the figure, the loss surface is not smooth and has numerous local minima, which is higher than the global minima. In absence of sufficient exploration, it could be very tempting to terminate the optimization process at local minima, not only for humans, but even for search algorithms that tends to balance an exploration-exploitation trade-off. Furthermore, as the number of hyperparameters being considered increases, the number of dimensions in the loss landscape increases linearly, making the optimization process extremely complex.

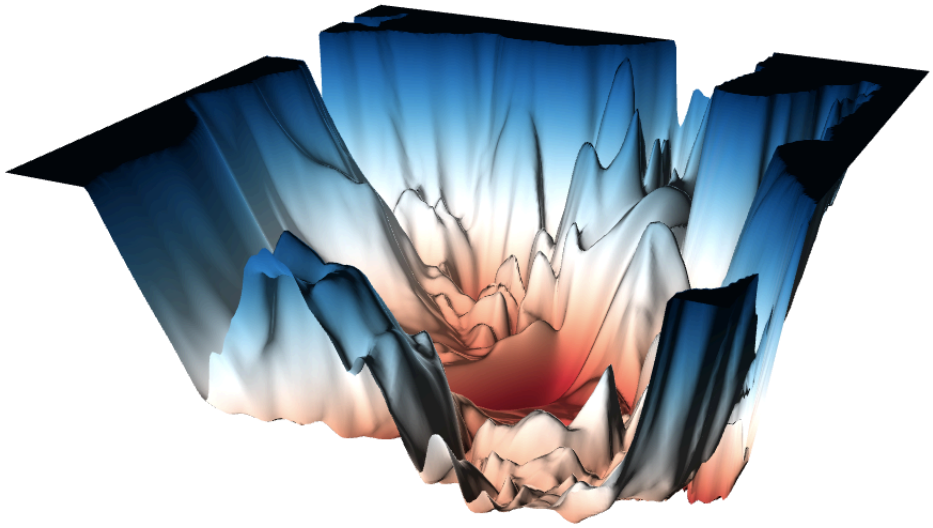


Figure 2.3: Visual representation of the loss landscape. Here, x, y and z axis represent the values of three different hyperparameters and the color distribution represents loss of the model for the given configuration of x-y-z values. This image has been generated using an online visualizing tool [1].

2.4. Terminologies used for HPO Algorithms

Having established ideal qualities for an HPO algorithm, let us begin understanding some of the landmark HPO algorithms that form the basis of this research. But before we proceed to the algorithms itself, we need to establish certain terminologies that are common across all algorithms.

- **Experiment**: The end-to-end optimization process including specifying the machine learning model, selecting hyperparameters, evaluating hyperparameters and in the end returning the best performing hyperparameter.
- **Trial**: Within an experiment, numerous hyperparameters are evaluated. The evaluation of hyperparameter include setting up the model with hyperparameters, training the model on the training set for specified amount of budget and evaluating the validation loss / accuracy. Each such evaluation cycle of a hyperparameter is referred to as a trial in this document.
- **Model**: This refers to the machine learning model whose hyperparameters are to be tuned. For each trial of a new hyperparameter, the model is re-built for evaluation.
- **Configuration Space**: The configuration space describes the domain from which hyperparameters are to be chosen for an experiment. It consists of a list of all the hyperprarameters that are to be tuned, and each of their type (Integer, Real, Categorical, etc), their range (bounds of possible values), their distribution (linear, logarithmic, discrete, etc) and in case of conditional hyperparameter, their conditional relationship with other hyperprarameters.
- **Budget**: This refers to the type and the bounds of budget that the optimization is to be performed on. It could be the maximum number of epochs per configuration in an experiment, the maximum sub-set of training samples per configuration, feature subsets for different configurations, etc. The idea of defining budget is to enable intelligent distribution of budget across different hyperparameters. The bounds of the budget is defined in terms of the amount of resources allocated per hyperparameter. For black-box optimization algorithms, the same amount of budget is allocated to each of the hyperparameters being evaluated. On the other hand, multi-fidelity algorithms require specification of the minimum and the maximum budget per configuration. Further discussion of intelligent budget allocation can be found in chapter 4, where we discuss multi-fidelity algorithms.
- **Objective Function** Objective functions provide quantification of the quality of a hyperparameter at the given budget. For example, consider optimizing one hyperparameter for a DNN with the

goal of finding the best configuration of the hyperparameter, with unrestricted amount of computational budget. In this scenario, the objective function could simply be the validation loss of the DNN for a value of the hyperparameter. The end goal of the optimization algorithm is to minimize or maximize this objective function. In our context, evaluating the objective function once means training the neural network with the specified budget and evaluating the validation loss.

- **Performance Metric**: Performance metric reffers to the metric used for gauging the quality of the hyperprameter configuration. In typical hyperparameter tuning cases, we recommend using the validation accuracy as the performance metric. This could however be changed to precision, recall, the F1 score, etc. Do note however, that these metrics are to be determined out of the validation set. Monitoring accuracy of the training dataset would not be useful: the hyperparameters achieved would overfit the model on training dataset, causing very high variance and disallowing generalizability. Similarly, using the test dataset for tuning hyperparameters is also not recommended, since the performance observed is not truly representative of the model's ability to generalize to never before seen data. Since the hyperparameters would have been tuned based on the performance of the model on test data set in this situation, the model would in fact never be subjected to never before seen data. Thus, it is recommended that we always split the available data into training, validation and test datasets and use the performance metric obtained on validation dataset to drive hyperparameter tuning.

2.5. HPO Algorithms

We have now established the importance of tuning hyperparameters for obtaining the optimal performance from machine learning models. We have also discussed some of the challenges involved in tuning hyperparameters. With this, it is also established that manually tuning hyperparameters, or traversing the entire search space is not a realistic approach. Thus, the need for algorithms to intelligently tune hyperparameters of a model has become apparent. This section begins by specifying the hyperparameter optimization problem with a mathematical notation in section 2.5.1 followed by introducing the two fundamental approaches followed by hyperparameter optimization algorithms showcased in figure 2.4.

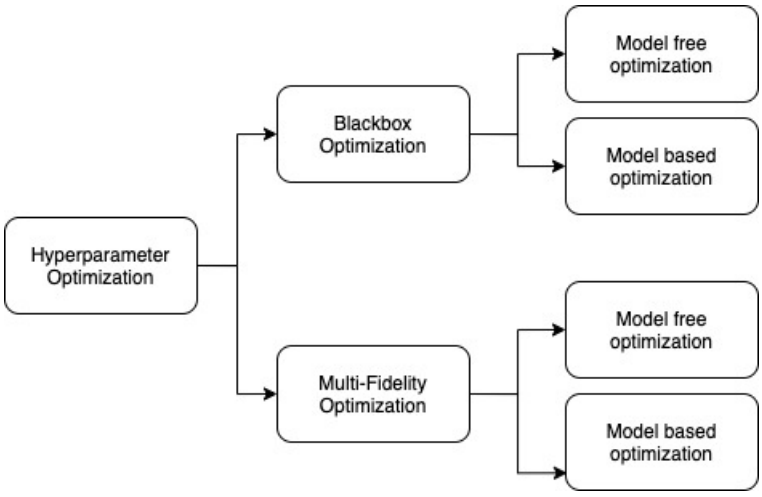


Figure 2.4: Overview of the different hyperparameter approaches

2.5.1. Mathematical Formulation of HPO

This section specifies the automatic hyperparameter optimization problem. The content of this section has been adapted from the article on hyperparameter optimization by Feurer et al. [7]. Let us first specify the following notations.

$\mathcal{A}$ **:** The machine learning model whose hyperparameters are to be tuned

N **:** The number of hyperparameters in the configuration space of the considered model

Λ : $\Lambda_1 \times \Lambda_2 \times \Lambda_3 \times ... \Lambda_N$ is the overall hyperparameter configuration space of the model, where Λ_n is the domain of n^{th} hyperparameter

λ : A vector of hyperparameters consisting of a legal value for each of the N hyperparameters of $\mathcal{A}$ from their respective domains

$\mathcal{A}_\lambda$: Algorithm $\mathcal{A}$ initialised with hyperparameters λ

The objective of the hyperparameter optimization is to essentially find the hyperparameter configuration that minimizes the validation loss for the given machine learning model, in-turn improving the test accuracy and generalizability of the model. So in this context, the loss of a model $\mathcal{A}$, for a given configuration of hyperparameters λ can be defined as $\mathcal{L}$ ($\mathcal{A}_\lambda$, D_{train}, D_{valid}), where D_{train} is the training dataset and D_{valid} is the validation dataset. The end objective is to find the hyperparameter configuration (λ^*) that minimizes this loss function. Equivalently the optimal hyperparameter configuration that minimizes the validation loss can be formulated as,

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \mathcal{L}(\mathcal{A}_\lambda, D_{train}, D_{valid}) \tag{2.1}$$

where, λ^* represents the optimal hyperparameter configuration for the given model and the configuration space.
This understanding of hyperparameter optimization can be abstracted to a higher level to also include evaluating n different models $\mathcal{A} = \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, ..., \mathcal{A}^{(n)}\}$. The fundamental idea of searching for the combination of algorithm and the respective hyperparameter configuration, that minimizes the validation loss still remains the same, and can be represented as

$\Lambda^{(i)}$ denotes the hyperparameter space of $\mathcal{A}^{(i)}$, for i = 1, 2...n

$\mathcal{L}(\mathcal{A}_\lambda{}^{(i)}, D_{train}, D_{valid})$ denotes the loss of $\mathcal{A}^{(i)}$ using $\lambda \in \Lambda^{(i)}$ trained on D_{train} and validated on D_{valid}.

This abstracted version of hyperparameter optimization problem is often coined as the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem. Mathematically, this can be represented as

$$\mathcal{A}^*{}_{\lambda^*} \in \arg\min_{\mathcal{A}^{(i)} \in \mathcal{A}, \lambda \in \Lambda^{(i)}} \mathcal{L}(\mathcal{A}_\lambda{}^{(i)}, D_{train}, D_{valid}) \tag{2.2}$$

where, $\mathcal{A}^*$ represents the optimal algorithm with λ^* as the optimal hyperparameter configuration.

2.5.2. Black-box optimization algorithms

Black-box optimization can be defined as the process of optimizing complex functions for which no gradient or Hessian information is available, and it is much more feasible to experiment with the function than to understand and model the function. Hyperparameter optimization perfectly aligns with this description of black-box optimization since the relationship between different hyperparameters' values and the performance of the machine learning model, referred to as the objective function is extremely complex to define. Furthermore, since we have no information about this relationship between hyperparameters and the model performance, we do not have the gradient or the Hessian information either. So, the only way to optimize for model performance is to treat the HPO problem as a black-box optimization problem, evaluating the validation loss for each of the hyperparameter configurations. This process can be visualised via figure 2.5.
Within the black-box approach, the total budget is required as an input, and this budget is equally distributed among each of the configurations selected for evaluation. Depending on how hyperparameter configurations are populated for experiment, there are two fundamental techniques within the black-box approach.

1. **Model free optimization technique**: Model-free approach to black-box HPO includes grid search and random search algorithms to populate hyperparameter configurations for evaluating the model performance. Here, the grid search attempts to evaluate the complete Cartesian product of the search space, given sufficient budget. Specifically, the algorithm does not distinguish hyperparameters based on their sensitivity, and draws equally distributed configurations. In the example

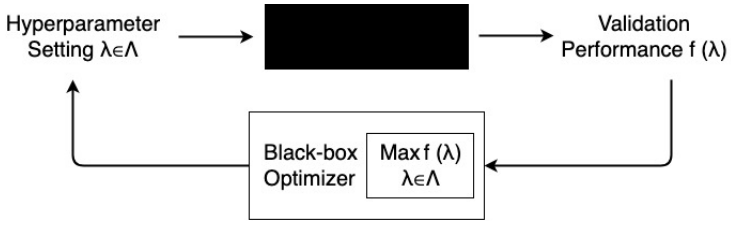
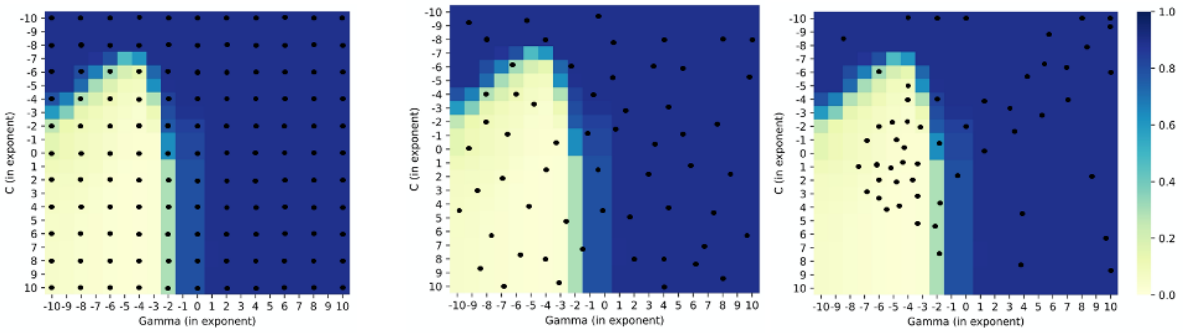


Figure 2.5: Depiction of hyperparameter optimization as a blackbox optimization technique. The figure has been adapted from NIPS 2018 tutorial of Frank Hutter and Joaquin Vanschoren on AutoML [9]

demonstrated in figure 2.6, grid search algorithm ends up choosing only 3 unique configurations each for the important as well as un-important hyperparameters. In case of random search, even though the number of configurations drawn is the same as grid search, 9 unique configuration would be selected owing to the random nature of draw. Due to this diversity of configurations, the probability of converging to a better performing configuration in random search is higher than grid search, making the it consistently perform as good or better than grid search [3]. However, both these algorithms discard all information about prior population of configurations and every configuration chosen for the experiment is independently drawn from the search space. Because of this, the final performance of the optimization experiment might be sub-optimal, paving the path for model based optimization algorithms.



(a) Grid Search (b) Random search (c) Model based search

Figure 2.6: Demonstration of grid search, random search and model search based hyperparameter opimization experiments.

2. **Model based search**: Model-based algorithms are another approach to black-box optimization for HPO, where in we evaluate a number of hyperparameter configurations with equal amount of budget allocated for each of the configuration evaluation trials. However, unlike the model-free approach, the choice of hyperparameter configurations for the experiment in the model-based algorithms is based on prior trials of the experiment. In other words, after evaluating a threshold number of randomly chosen hyperparameter configurations, the new configurations for trials are chosen based on the performance of the already evaluated configurations. The idea is to focus the search in spaces with higher possibility of better convergence performance. Furthermore, most HPO algorithms of model-based black-box optimization approach incorporate a version of exploration-exploitation trade-off while choosing configurations. This trade-off is important to ensure that the algorithm does not converge on local optima in fact finds the global optima.

2.5.3. Multi-fidelity optimization algorithms

On larger search space, black-box optimization becomes unreasonably expensive for the tuning process. As the number of hyperparameters and their search space increases, the number of hyperparameter configurations that need to be evaluated increases exponentially to accommodate for reasonable exploration. Without exploration, the probability of missing out on the ideal hyperparameter configuration increases, rendering the tuning experiment useless. Since evaluating each configuration entails training the model completely and then evaluating the validation loss, black-box optimization becomes extremely expensive.

In multi-fidelity optimization, we approach HPO by defining multiple fidelities of the objective function with increasing degree of accuracies. That is, we create multiple cheap-to-evaluate approximations of the objective function that enables evaluating numerous configurations at very low cost. Once promising configurations have been identified in these cheaper fidelities, they are evaluated on more expensive fidelities that are closer to the actual objective function and offer a more reliable approximation.

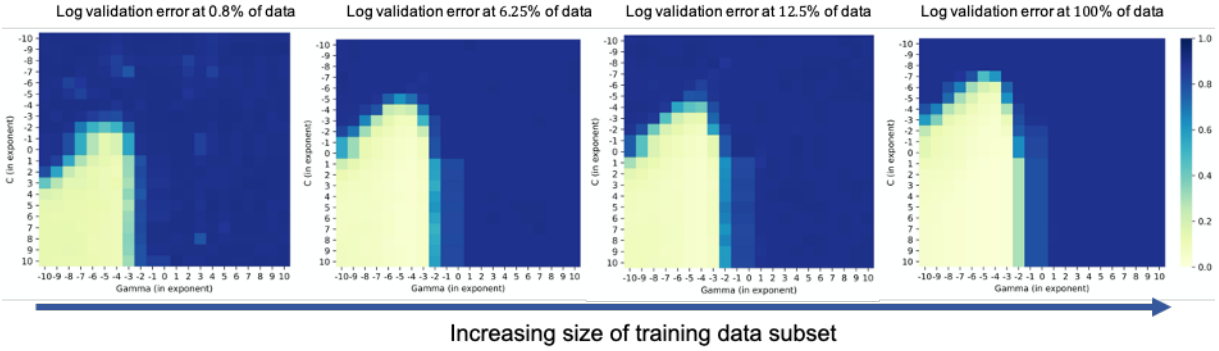


Figure 2.7: Demonstration of the use of multiple fidelities for tuning C and γ for SVM on MNIST dataset. The x axis denotes γ in a logarithmic scale, and y axis denotes C in logarithmic scale. The validation loss has been visualised via color distribution in logarithmic scale, with blue denoting validation loss of the order of 10^0 and yellow showing the validation loss of the order of $10^{-4.5}$.

Consider the example shown in figure 2.7. The figure depicts the correlation between hyperparameters C and γ with the validation loss of an SVM used for classifying digits from MNIST dataset. The x axis shows variation of hyperparameter γ in a logarithmic scale and y axis shows C in logarithmic scale. The validation loss can be visualised by the color of the region, varying in logarithmic scale, starting from 10^0 depicted by red to $10^{-4.5}$ depicted by blue. Sub-plot (i) shows the loss surface when trained with only 400 samples of the total of 50,000 samples in the training dataset. Similarly, sub-plot (ii) shows the loss surface with 3,125 samples, (iii) with 12,500 and finally (iv) with all of 50,000 samples. Here, evaluating the validation loss for a hyperparameter configuration with 400 training samples is 1000 times faster than that with 50,000 training samples [14]. As you can notice, the loss surface does not vary significantly with increasing number of samples, and the region with promising configurations of C and γ in lower fidelity (evaluating with 400 training samples) is the same as that in higher fidelity (evaluating with 50,000 samples). This knowledge allows us to limit our exploration in the hyperparameter search space and to distribute more resources in evaluating promising hyperparameter configurations. Thus, using multiple fidelities of the objective function allows us to find optimal hyperparameter configuration much faster than simply treating the objective function as a black box. Apart from using sub-set of training samples, lower fidelity functions can also be derived by the following methods.

- By using fewer epochs of the training algorithms.
- By using fewer features from the full feature set.
- By down-sampling images in image based objectives.

Similar to the black-box techniques, populating hyperparameter configurations for multi-fidelity approach can also be done in a model based as well as model free fashion. While model free algorithms such as Successive Halving [11] and Hyperband [17] established the use of multi-fidelity approaches for hyperparameter optimization, they relied on random search for populating configurations for evaluation. Given the extremely large exploration these algorithms can undertake, the possibility of attaining a well performing hyperparameter configuration early on increases significantly. However, since the population in these algorithms is not guided via a search algorithm, converging to the best configuration cannot be guaranteed. To alleviate this challenge, numerous approaches have been studied that augment the multi-fidelity capabilities of Hyperband with search strategies primarily based on Bayesian optimization [2, 6, 14, 22]. Given the advantage of using multiple fidelities along with a complex search algorithm, these model based multi-fidelity algorithms can deliver better final result than vanilla Hyperband. Here, better final performance refers to delivering the configuration with lower validation error.

BOHB is one such model based multi-fidelity algorithms that has been empirically shown to perform better than several state-of-the-art hyperparameter optimization methods [6] in numerous machine learning tasks.

2.6. Expectations from ideal an HPO algorithm

Finally, having established some of the landmark approaches for hyperparameter optimization, let us now take a step back and consider the larger picture of end goals that HPO techniques are designed for. Listed below are some of the expectations from an ideal HPO algorithm. The list has been adapted from the article on BOHB by Falkner et al. [6].

1. **Strong anytime performance**
   Consider a large neural network for classifying objects from 1920 * 1080 images. Training such a neural could take significant time, on the order of days or even weeks. Even evaluating a limited number of hyperparameter configuration for the full training budget becomes unfeasible. So, in such scenarios, it is desirable for the optimization algorithm to already yield good configurations within a limited budget.

2. **Strong final performance**
   On the other hand, the end objective could also be to find the best performing model to establish state-of-the accuracy for a problem, even at the expense of compute resources. In such scenarios, the HPO algorithm should converge to the best performing hyperparameter configuration given sufficient budget.

3. **Effective use of parallel resources**
   The efficiency of parallel processing in form of GPUs for machine learning applications has been realised, and has led to development of specialised processors designed for machine learning applications. Furthermore, the availability of cloud compute platforms has enabled researchers and businesses to easily access such parallel resources allowing wide spread adaptation of such compute platforms for deploying machine learning applications. The HPO algorithm should leverage the availability of parallel resources so as to scale justifiably with increasing compute capabilities in form of hardware infrastructure.

4. **Scalability**
   Given the wide range of domains and applications that machine learning models are being used for, models could require tuning a multitude of hyperparameters with varying domains, ranges, distributions and even conditionalities. The HPO algorithm should support tuning all the types of hyperparameters with reasonable exploration-exploitation across the search space to ensure good model performance given sufficient budget.

5. **Robustness and Flexibility**
   Continuing from the previous point, consider the wide range of applications for which machine learning models are developed. Here, the performance of a model might be hugely impacted by slight variation on one of the hyperparameters, while the performance of a model in another application might not be effected from changes in the same hyperparameter. Similarly, some models could report more noisy validation losses than others. The HPO algorithm should handle such variations effectively and yield an optimal configuration irrespective of the application or the domain. Furthermore, it is highly unlikely for the objective functions (see section 2.4 for definition) to be convex. So it is important for the HPO algorithms to not get stuck at local optima and to in fact congerge to global optima.

6. **Simplicity**
   The algorithm itself should be simple to use in terms of not requiring the users of the algorithm to fully understand the algorithm before using it and not requiring numerous parameters to be set before the algorithm can be used optimally. Furthermore, the algorithm should be reproducible, allowing users to reproduce optimization experiments. Lastly, the algorithm should not involve significant development overhead before a model can be tuned. It should allow for a ”plug-and-play” experience to tune machine learning models of any domain or application.

7. **Computational Efficiency**
   While evaluating the model configuration is computationally expensive, the optimization algorithm itself should not have extreme computational requirements. For example, the algorithms based on standard Gaussian Process would have cubic computational complexity on the hyperparameter search space [6]. Such algorithms create a new bottleneck in the optimization process, rendering the overall optimization algorithm ineffective.

2.7. Conclusion

This chapter forms the stepping stone for the rest of the thesis providing an in-depth discussion of the relevant topics. We start by understanding what hyperparameters are, followed by establishing the need for tuning hyperparameters and why tuning them is an extremely challenging task. Next, we formulate a representation of the hyperparameter tuning problem setting the stone for automatic tuning algorithms. We begin by briefly exploring the landmark approaches of hyperparameter tuning and finally establish BOHB [6] as the start of the art approach. Next, in chapter 3, we will dive deeper into understanding BOHB before we can get into the crux of the research work done in this thesis.

3

BOHB: the state of the art

BOHB is a state-of-the-art hyperparameter tuning algorithm that follows multi-armed bandit strategy amalgamated with Bayesian Optimization(BO) to efficiently derive optimal configurations for the hyperparameters. This chapter begins by providing an introduction to BOHB in section 3.1, followed by a discussion on the two critical components that BOHB builds on, in sections 3.2 and 3.3. We then summarise the advantages and shortcomings of BOHB, establishing the need for dynamic budget allocation.

3.1. Introduction

In chapter 2, we established the need for tuning hyperparameters on machine learning models and provided an introduction to two fundamental hyperparameter tuning approaches: the black-box optimization approach and the multi-fidelity approach. State-of-the-art black-box optimization methods leveraged search algorithms to derive optimal configurations to minimize the number of black-box evaluations. On the other hand, state-of-the-art multi-fidelity approaches utilized forms of early-terminations and multiple fidelities to make cheaper evaluations, allowing a wider range of exploration for configurations. BOHB kneads both of these fundamental approaches in a unified HPO algorithm that can deliver superior performance on each of the metrics listed in section 2.6.

The search algorithm in BOHB utilizes a variant of Bayesian optimization, wherein the objective function is modeled based on Tree-structured Parzen Estimator (TPE). While Gaussian processes are predominantly used in Bayesian Optimization tasks, they entail computations that grow in cubic complexity with increasing dimensions. This renders Gaussian Processes ineffective for hyperparameter tuning tasks, which require tuning numerous hyperparameters, each of which add a dimension to the objective function. TPE approach on the other hand scales linearly on the number of dimensions and allows modelling complex hyperparameter specifications, owing to the non-parametric modeling approach. This choice of Bayesian Optimization distinguished BOHB from other studies that tried to deliver Bayesian optimization based multi-fidelity hyperparameter tuning and helped BOHB deliver superior performance.

The multi-fidelity aspect of BOHB is realized via Hyperband [17], an approach for tuning hyperparameters that builds on the early stopping principles of Successive Halving [11]. The underlying idea is that the promising range of values for hyperparameters can be identified in approximations of machine learning models that are cheap to evaluate. Cheap approximations can be realized in form of using only a small part of dataset while training the machine learning model,or training for fewer epochs than what is required for convergence. So, we perform numerous cheap evaluations of the objective function, allowing for sufficient exploration and then evaluate the promising configurations on increasingly accurate approximations objective function. This could mean utilizing the complete training dataset, or increasing the number of epochs. The key disadvantage in using Hyperband for tuning hyperparameters is that all the information gathered about the objective function is discarded, and each configuration sampled is drawn at random. By augmenting Hyperband with Bayesian optimization, Falkner et al. were

able to address this challenge and offer fast and search based hyperparameter tuning in BOHB [6]. Let us now dive deeper into Hyperband and follow it through with how Bayesian optimization is integrated with it in BOHB.

3.2. Hyperband

Hyperband is an optimization algorithm proposed by Li et al., that enables the multi-fidelity aspect of BOHB. Let us first build the background of Successive Halving [11], on which the Hyperband algorithm builds upon.

3.2.1. Successive Halving

In Successive Halving (SH), we begin by uniformly distributing a minimum budget across all the hyperparameters. Here, budget could be thought of, as the number of epochs for which the model is trained, or the amount of training dataset used for training the model among other possibilities. Once the minimum-budget has been lapsed, we monitor the performance of each of the hyperparameters and continue only half of the best performing configurations for the next termination cycle. So, the worst performing half of configurations are simply not evaluated any further, and the amount of budget allocated to the better performing half is doubled. This cycle is repeated, until we end up with a single best performing configuration. This has been intuitively illustrated as an example in table 3.1. Consider tuning 3 hyperparameters of a convolutional neural network for an MNIST classification task. Here, we start off by evaluating 16 unique configurations for each of the 3 hyperparameters and evaluate each configuration for 1 epoch. At the end of 1 epoch for each of the 16 configurations, we select 8 configurations which had the highest validation accuracy and double the the budget of each configurations to 2 epochs. This process is repeated until a single configuration remains, which will be evaluated for the maximum budget.

Number of configurations	Number of epochs per configuration
16	1
8	2
4	4
2	8
1	16

Table 3.1: Illustration of successive halving algorithm.

3.2.2. Challenge with Successive Halving

While the algorithm is extremely intuitive, the configuration that remains at the end of the successive halving execution is not necessarily the best configuration. Consider the hypothetical but realistic scenario demonstrated in figure 3.1. Here, let the red and the blue lines represent the performance of two different configurations. At the end of the first termination cycle, it seems like the red line performs better than the blue line, owing to the lower validation error. However, when trained to convergence, it could be observed that the configuration represented by the green line was able to reach a lower validation error than the red line. In successive halving however, the green line would be terminated early on, due to the initial slow rate of decrement of validation error. More generally, for a given termination cycle, consider n hyperparameter configurations being evaluated.

Let B denote the total budget that is to be spent in this termination cycle. Successive halving uniformly distributes the B resources across n configurations. However, as observed in the above example, certain promising configurations may converge slower than other configurations, causing the algorithm to miss out on such configurations. So, a trade-off is to be made to allocate sufficiently large B to accurately identify well performing configurations, while minimising the budget spent on poor performing configurations. This is referred to as the ”n vs B/n” challenge in the article on Hyperband [17].

3.2.3. Hyperband

While the successive halving algorithm suffered from ”n vs B/n” challenge, it did provide a very intuitive mean to define multiple fidelities and leverage early terminations to accelerate the hyperparameter

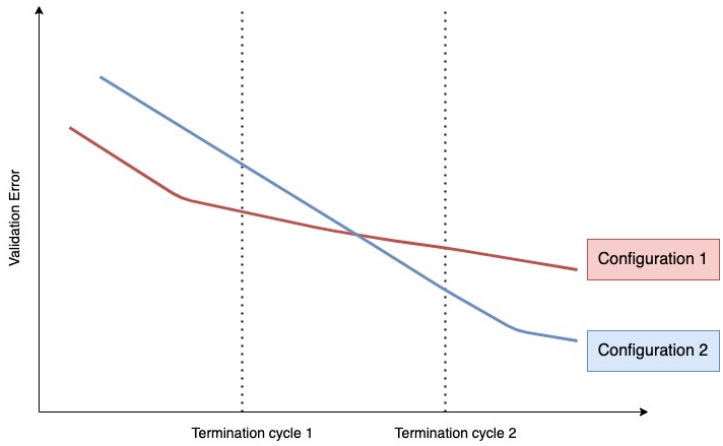


Figure 3.1: Demonstration of ”n vs B/n” challenge of successive halving.

optimization process. To that end, Li et al. proposed Hyperband [17], an algorithm that runs multiple iterations of successive halving with variations in the starting number of configurations and the minimum budget allocated per budget. This hedges the risk of missing out on a well-performing configuration with a slow rate of convergence. Table 3.2 showcases how Hyperband hedges the successive halving algorithm with the same minimum budget, maximum budget and eta to enable n vs B/n trade-offs, reducing the risk of terminating potentially well performing configuration early on.

i	**S = 4**		**S = 3**		**S = 2**		**S = 1**		**S = 0**	
	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i
0	16	1	8	2	4	4	4	8	5	16
1	8	2	4	4	2	8	2	16		
2	4	4	2	8	1	16				
3	2	8	1	16						
4	1	16								

Table 3.2: Hyperband algorithm with the same minimum budget, maximum budget and η as that in table 3.1. Here, s denotes the different successive halving brackets with $s \in [0, S_{max}]$, N_i represents the number of configurations considered in each termination cycle, and R_i represents the amount of resources (Epochs in this example) allocated per configuration.

Algorithm 1 specifies the Hyperband approach to perform hyperparameter tuning. Hyperband requires specification of the following inputs by users for its execution.

- Minimum budget: The minimum budget refers to the minimum amount of budget that each configuration is to be evaluated for.

- Maximum budget: The maximum budget refers to the maximum budget for which the best performing configuration is to be evaluated for.

- η: Referred to as the **reduction factor**, η defines the fraction of hyperparameters being terminated at the end of each termination cycle. It also defines the factor by which the budget per configuration increases after each termination cycle.

Algorithm 1 Overview of the Hyperband algorithm

Input: min_budget, max_budget, η
S_{max} = $\lfloor \log_{\eta} \frac{max\_budget}{min\_budget} \rfloor$

for $s \in \{S_{max}, S_{max} - 1, ..., 0\}$ **do**
  Sample $n = \lceil \frac{S_{max}+1}{s+1} * \eta^s \rceil$ configurations
  Execute successive halving starting with n configurations and min_budget = $\eta^s * max\_budget$
end
return *the best performing configuration*

By performing multiple iterations of successive halving with varying starting budgets and starting number of configurations, Hyperband was able to reduce the possibility of wrongful termination of configurations. However, it still sampled configurations in random fashion for each of the successive halving iteration, discarding learning opportunity from all prior evaluations of configurations. This severely limit the possibility of convergence to a global optima of the best performing configuration.

3.3. Bayesian Optimization

Let us now look at the Bayesian optimization algorithm used in BOHB and how it has been incorporated to work together with multi-fidelity readings. BOHB utilizes Tree Parzen Estimator (TPE) based Bayesian optimization [4] method for modeling the objective function.
Consider hyperparameter tuning as an optimization problem for a multi-dimensional objective function, $f(x)$ where $x \in X$, representing all the hyperparameters. Here, the end goal is to find a value for each of the dimensions for which the objective function attains the global minimum value. In other words, we try to find $x_* \in \arg\min_{x \in X} f(x)$. Multiple dimensions represent multiple hyperparameters being tuned for the model, and the performance of the objective function correlates to the performance of the model under consideration for the given configuration of hyperparameters.
The objective function is unknown a priori, and Bayesian optimization functions by modeling the objective function based on observations for each of the dimensions being optimized. Bayesian optimization approaches are built up on two building blocks: the modeling function and the acquisition function. The modeling function represents the model that has been built of the unknown objective function, and the acquisition function defines the strategy to choose points at which objective function is to be evaluated. The primary purpose of acquisition function is to balance an exploration-exploitation trade-off with the aim of converging to a global optima with as few evaluations as possible. The generic Bayesian Optimization algorithm can be summarised in following algorithm. The steps are repeated till the budget for optimization has been exhausted or the observed gain is saturated.

- Select point that maximizes the acquisition function $a(x)$. The dimension of x is the same as that of the objective function.
  $x_{new} = \arg\max_x a(x)$
- Evaluate objective function $f(x)$ for input x_{new}
  $y_{new} = f(x_{new}) + \epsilon$, where ϵ represents the inherent noise in each observation
- Augment this new reading to the observation repository D and refit the probabilistic model
  $D \leftarrow D \cup (x_{new}, y_{new})$

The most commonly used modelling strategy used is the Gaussian process, and the most commonly used acquisition function is the Expected improvement [5]. However, since Gaussian processes scale in cubic fashion with increasing dimensions, it would not be an effective choice for hyperparameter tuning. So, BOHB leverages TPE based modeling strategy and acquisition function to enable optimal performance for multiple dimensions and modeling objective function with varying distributions and relationships. Having established the general idea of Bayesian optimization, let us now look at how TPE incorporates these general principles to converge to an optimal value.

3.3.1. Tree-structured Parzen Estimator

Unlike Gaussian process where we directly model $p(y|x)$, TPE models $p(x|y)$ and $p(y)$ by using kernel density estimators to develop models. The prediction model in TPE, p(x|y) is defined using 2 densi-

ties: $l(x)$ and gx, representing separate models for good and bad performing configurations respectively. The performance of a configuration x is gauged by the corresponding loss $f(x)$ observed on the objective function. If the loss observed was was less than α, it is categorized as good performing configuration, else a poor performing configuration. Here,

$$\begin{aligned} l(x) &= p(y < \alpha|D), \textit{ representing model of good performing configurations} \\ g(x) &= p(y \geq \alpha|D), \textit{ representing model of poor performing configurations} \end{aligned} \tag{3.1}$$

To select the next configuration (x_{new}) to evaluate, we maximize the ratio $l(x)/g(x)$. Intuitively, this corresponds to maximizing the possibility of selecting configurations similar to those represented in $l(x)$, the region with good performing configurations. Now that we have established a baseline for Bayesian optimization with TPE, let us understand how BOHB incorporates TPE and enables multi-fidelity readings into its models.

3.3.2. Multi-fidelity Bayesian Optimization in BOHB

Now that we have understood the Hyperband algorithm and walked through TPE based Bayesian optimization in a blackbox setting, let us dive deeper into how Falkner et al. incorporated the two algorithms together to establish the state of the art in BOHB.
The underlying idea is to maintain a TPE based Bayesian model for each budget (fidelity) and to use the model of the largest available budget to draw configurations. We use the model from the largest budget, since it most closely represents the actual objective function. A single multi-dimensional model is maintained for each budget, so as to better incorporate the complex relationships between hyperparameters. Each of the dimensions in the predicted model represent a hyperparameter.
Before a useful model can be developed for a budget, we require a minimum number of evaluations of the model with the corresponding budget. So, we only begin relying on the model to sample configurations after a certain number of observations have been made for the corresponding budget. On this regard, we sample configurations at random until sufficient observations have been made available. Each observation is recorded in the form of $g(x, b) + \epsilon$, where x is a d dimensional vector, representing d hyperparameters and b is the budget for which the observation was made. ϵ represents the inherent noise present in the observation owing to the stochastic aspects of the model. Furthermore, even when sufficient observations have been made to develop models, we still draw a fraction of configurations at random to incorporate a level of stochastic exploration. With this, consider the following terminologies to summarise the use of Bayesian Optimization in BOHB.

- N_{min}: The minimum number of observations required to build the model. By default, this has been set to d + 2, where d is the number of hyperparameters being optimized.
- D: A record of all the readings observed so far represented in the format (x_i, y_i, b), where x_i is a configuration, y_i is the corresponding validation error observed by training the machine learning model for a budget of b.
- ρ: Fraction of random samples to draw.
- q: Percentile to differentiate between good and bad readings.

The use of Bayesian model in BOHB can then be specified via two key components discussed below.

1. **Drawing Samples**
    - If rand() < ρ or if no model is already available, return a random configuration
    - Arrange the observations available for the corresponding budget into two separate tables, split based on the percentile factor, q. Table l represents the top q percent observations made so far, and table g represents the rest of the observations with relatively poor performance.
    - Fit corresponding kernels on the model over observations available so far, arranged in $l(x)$ and $g(x)$ based on equation 3.1.
    - Return the configuration that maximizes $\frac{l(x)}{g(x)}$.

2. **Updating model**

   - For a given observation (x_i, y_i, b), avoid updating the model if
     - a model is already available for a higher budget
     - observations corresponding to current budget b < N_{min}
   - If neither of the conditions are true, update kernels on each dimension of the current model

3.4. Advantages and shortcomings of BOHB

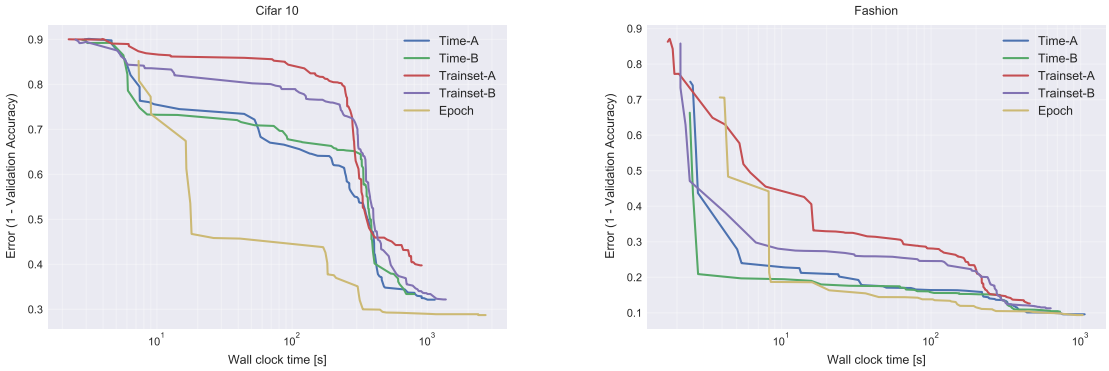
Falkner et al. successfully demonstrated the efficacy of BOHB against state-of-the-art hyperparameter tuning approaches, including black-box approaches like SMBO, as well as other multi-fidelity approaches like Hyperband. In fact, BOHB delivered superior performance as compared to other techniques that utilized model based multi-fidelity approach, owing to the choice of tree-based parzen estimators for Bayesian optimization. Falkner et al. also demonstrated BOHB's scalability to parallel resources, allowing significant speedup in the tuning process.

However, BOHB relies on its users to specify parameters for defining fidelities. Reliably specifying fidelities forms a critical component of the algorithm. Ranks of hyperparameter configurations on the lower fidelities should tentatively correlate to ranks of the configurations in higher fidelities as well as the actual objective function. If the cheaper fidelities do not attain some correlation to the actual performance of configurations, BOHB would in fact perform worse than black-box optimization techniques [6].

Furthermore, before defining compute constraints for fidelities, we need to specify the type of budget used for defining the fidelities. In our study, we experimented with three different budget types: epochs or the number of training iterations, using proportions of training dataset for evaluating a configuration, and specifying the time duration, for which models are trained to evaluate the configuration. To establish a baseline for defining fidelities, we performed the following experiments. Consider hyperparameter tuning tasks on CIFAR 10 and Fashion MNIST datasets, as described in section 5.1. We used vanilla BOHB with the 5 different configurations specified in table 3.3, for the hyperparameter tuning task.

	Minimum Budget	**Maximum Budget**	**Eta**	**Other parameters**
Epoch based	1 epoch	16 epochs	2	
Trainset A	0.41% training dataset	100% training dataset	3	Epochs per configuration (epc) = 2
Trainset B	0.41% training dataset	100% Training Dataset	3	Epochs per configuration (epc) = 4
Time A	370 ms	30 seconds	3	
Time B	185 ms	15 seconds	3	

Table 3.3: Configurations considered for evaluating budget constraints and budget type.



(a) CIFAR 10

(b) Fashion MNIST

Figure 3.2: Hyperparameter tuning experiments using BOHB with multiple configurations for defining fidelities, mentioned in table 3.3.

In figure 3.2a representing performance on CIFAR 10, it can be observed that different configurations converged to different levels of final error. We see that configuration Trainset-A performed the worst, converging to the maximum validation error i.e. minimum validation accuracy. Varying a parameter in Trainset-B improved its final performance significantly, as compared to Trainset-A. However, even after improvement, Time based budgets (Time-A and Time-B) performed better than either of the trainset based experiments, since the trajectories of time based budget were consistently lower than trainset based budget, offering better any-time performance. However, the Epoch trajectory outperformed every other budget types, with better final performance, as well as better any-time performance. However, if you look at the final part of the Epoch trajectory, the experiment evaluations for over 2000 seconds even after convergence and saturation in gain. 2000 seconds is a significant overhead, considering such a small network and a training dataset size of 50000 32*32 images. In real world tuning tasks, such overhead could mean unreasonably high wasted resources.

On the other hand for hyperparameter tuning task on Fashion MNIST, represented in figure 3.2b, we can observe variations in performance of budget types as compared to that from CIFAR 10. Here, we see that even though the final performance attained by epoch based and time based budgets were similar, the any-time performance highlighted stark difference in their performance. Time based budget were significantly faster than epoch based budget in converging to approximately 80% accuracy.
Thus, definition of fidelities, in terms of the budget type as well as budget constraints play a critical role in the performance of BOHB for hyperparameter tuning tasks. Ideally we would want to allocate just enough resources to reliably specify fidelities and in turn to minimize the total resources consumed in the optimization process while still obtaining the best configuration from the search space. BOHB's reliance on users to specify these parameters significantly hinders the usability of the algorithm, making the need for dynamic budget allocation algorithms critically apparent.

3.5. Conclusion

This chapter establishes the inner working BOHB, the state of the art hyperparameter optimization algorithm. We begin by identifying the two primary components of BOHB, in form of Hyperband and Bayesian optimization (BO), and describing their inner workings. Specifically, we focused on finite horizon Hyperband, which is the version utilized in BOHB. In the BO front, we provide a general introduction to the workings of BO, followed by a discussion on Tree-structured Parzen Estimator based BO. We then continue into how the two algorithms are integrated into BOHB. Finally, we discuss the key advantages that distinguishes BOHB from the rest of the HPO methods, followed by identifying its shortcomings, that opens the door to our research. In chapter 4, we will dive deeper into the challenges of BOHB and discuss our approach to overcome these challenges.

4

Budget allocation schemes

While we have discussed the consequences of mis-specifying budgets on BOHB performance, this chapter begins by first widening the scope and exploring three different types of budget that could be used to formulate different fidelities. Next, we discuss fixed horizon budget allocation schemes for each of the budget types to set up the necessary background for further study. We then study the implications of different budget types on the optimization process, and discover the need for a pivot. Next, we explore inter playing two different budget types for the optimization process. Finally, we introduce *Multi-Tune*: the algorithm that explores different budget types to derive the best performing budget type and delivers an infinite budget allocation scheme.

4.1. Budget Types

One of the fundamental ideas that establishes BOHB as the state of the art in hyperparameter optimization is its use of multiple fidelities in the optimization process. Having cheaper, less accurate approximations of the objective function allows BOHB to explore a wide search space without any significant repercussions, and using more expensive approximations at the later stage allows BOHB to accurately evaluate promising configurations derived from the cheaper fidelities. But how do we define these fidelities that approximate the objective function with varying degree of accuracy? This can be accomplished by allocating different training budgets for evaluations. Evaluating objective functions with lower training budget would require significantly less compute as compared to evaluating objective function with higher training budget. However, performance of the objective function at lower budget is not fully representative of the performance of the configuration being evaluated. In other word, given more training budget, the configuration is likely to attain better performance on the objective task. In this thesis, we explored three different types of budget with which fidelities can be defined. Mentioned below is a brief introduction to each of the budget type explored. Note that these are the types of budget which are being used for defining fidelities, and not the budget value itself. The values are defined uniquely by the budget allocation schemes discussed further in this chapter.

1. **Epochs**: In this type of budget, a cheap less accurate fidelity would correspond to running fewer epochs of the training set across the machine learning model. Similarly a more accurate but expensive fidelity could be defined by allocating more training epochs for each configuration of hyperparameters. Table 4.1 shows an example of using epochs as a budget for HPO process. As expected, the cost of evaluation for each of the configuration is directly correlated to the number of epochs allocated by the allocation scheme: fewer the epochs, lower the cost.

2. **Portion of training dataset**: In this type of budget, we divide the training dataset in varying sizes and use corresponding parts of the training dataset to evaluate configurations. For example, assume we want to evaluate 20 different configurations using 1/10 of the whole training dataset. This would entail independently training the model for each of the configurations only using 1/10 of the dataset. Here, the 1/10 part of the dataset being used is consistent across all the configurations to ensure that the performance of the objective function is in fact representative of the

configuration. If the portion of training set being used is uniquely sampled for each of the configurations, the performance received could be biased, because of different training set being used. Furthermore, for evaluation of each configuration, it is not sufficient to pass the portion of training set through the model for one iteration: the model would simply not learn its model parameters in this scenario, and the performance could again be biased because of random allocation of model parameters. So, usage of training dataset as a budget type also entails defining the number of iterations .
Conclusively, fidelities with varying accuracies can be defined using varying sizes of training dataset that is used in evaluating the objective function. Table 4.2 shows an example of using training dataset as a budget for the HPO process.

3. **Time**: For using time as a budget type, we divide fidelities based on the amount of time spent for training. So, a cheap less accurate fidelity could mean training the model with the set hyperparameter configuration for a fraction of time that would be required for training the model to saturation. Similarly, defining a more accurate approximation of the objective function would entail allocating more time for training the model. Analogous to epochs as a budget, we use the full training dataset in this budget type, but instead of limiting training by the number of epochs, we limit it via time spent. Table 4.3 portrays an example of using time as a budget type.

4.2. Fixed horizon budgets

As discussed in section 3.2, vanilla BOHB leverages the budget allocation principles from Hyperband. More specifically, BOHB leverages the finite horizon Hyperband principles for defining the experiment. This requires users of BOHB to mandatorily specify the following parameters.

1. **Minimum Budget**: Minimum amount of budget to be spent per configuration. This value is independent of the type of budget being used, and the distribution can simply be noted numerically.
Minimum budget define the minimum amount of training resource, such that the if the model with a specified configuration of one or more hyperparameters is trained for the given amount of resource, the performance metric obtained is representative of the configuration itself, and is not biased from randomness of the initial model parameters.

2. **Maximum Budget**: Maximum budget to be spent per configuration. Similar to the minimum budget, this maximum budget is simply a numerical representation.
Maximum budget defines the maximum amount of training resource that needs to be spent on training the model with a specified configuration of one or more hyperparameters such that the performance metric obtained is the best performance that can be obtained with the specified configuration. In other words, adding further training resource for the configuration would not result in a better performance.

3. η **(Reduction Factor)**: This factor determines the fraction of configurations that are to be terminated at each termination cycle (Rung) of the successive halving iteration. For example, if η is 3, $1/3$ of the best performing configurations from current rung will be promoted to the next rung.
On the other hand, η also determines the factor by which the amount of budget allocated to each of the configurations being promoted to next rung of a successive halving iterations increases. For example, if η is 3, each of the $1/3$ configurations being promoted to the next rung will have 3 times more budget allocated for the next rung.

Having established the mandatory parameters required for vanilla BOHB, let us now discuss how different budget types could be incorporated to use the BOHB algorithm. It should be noted that the underlying Bayesian modeling approach used for BOHB in the vanilla fixed horizon tuning process is in accordance to the discussion section 3.3.

4.2.1. Epochs based

Consider the following inputs for using epoch based budget type for BOHB:

- Minimum Budget: 2 (epoch)

- Maximum Budget: 32 (epochs)

- η: 2

These inputs entail S_{max} = $\log_{\eta}(max\_budget/min\_budget)$ to be 4, causing 5 iterations of successive halving. As discussed in section 3.2, the starting budget and the starting number of configurations for each of the iterations of successive halving is unique. At the beginning of each of the successive halving iterations, configurations are either sampled from the Bayesian model developed so far, or randomly to introduce some exploratory noise. With this, the budget allocation scheme for a fixed horizon epoch based BOHB can be realised in form of table 4.1.

i	S = 4		S = 3		S = 2		S = 1		S = 0	
	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i
0	16	2	8	4	4	8	4	16	5	32
1	8	4	4	8	2	16	2	32		
2	4	8	2	16	1	32				
3	2	16	1	32						
4	1	32								

Table 4.1: Budget distribution example for fixed horizon epoch based BOHB.

Here, N_i represents the number of configurations that are to be evaluated at each rung of the successive halving iteration. Similarly, R_i represents the number of epochs, each of the configurations in the current rung is to be evaluated. Variable i denotes progress of rungs in each of the successive halving iterations. Since S_{max} is 4 in our example, i varies from 0 to 4 in the most aggressive termination iteration (S = 4). Across iterations, the starting budget for each of the configurations varies, hedging the risk associated with the ”B vs n/B” challenge discussed in section 3.2.

4.2.2. Trainset portions based

Before approaching BOHB with trainset based budget, let us first visualise how a single iteration of successive halving is undertaken with this budget type. Consider the iteration with S = 4 in table 4.2, where in we start with 16 configurations evaluated on 6.25% of training data set and end with a single configuration that has been trained in 100% of the training data set.

Having established the progress of a single successive halving iteration, let us now delve into BOHB with training data subsets as a budget. Consider the following inputs for using training data subsets as a budget type for BOHB:

- Minimum Budget: 6.25% of dataset

- Maximum Budget: 100% of dataset

- η: 2

Similar to our epoch based budget example, S_{max} here would equate to 4, causing 5 successive halving iterations. Apart from the minimum and maximum trainset budgets, the user would also need to specify the number of times the model is to be trained with each of the training data subsets. This is analogous to epochs in the standard machine learning context, but with training data subsets rather than the whole training dataset. This number has been referred to as **Epochs Per Configuration (EPC)** in this thesis. In our example, we could consider EPC as 5, meaning irrespective of what percentage of training dataset is being used for evaluation of the configuration, this data subset would be trained on the model for 5 iterations. If we are using training dataset as a budget type in a fixed horizon setting, we would also need to mandatorily define EPC as an input to the experiment.

i	S = 4		S = 3		S = 2		S = 1		S = 0	
	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i
0	16	6.25	8	12.5	4	25	4	50	5	100
1	8	12.5	4	25	2	50	2	100		
2	4	25	2	50	1	100				
3	2	50	1	100						
4	1	100								

Table 4.2: Budget distribution example for fixed horizon training data subset based BOHB.

Table 4.2 showcases the distribution of budgets and the number of configurations considered for each of the budgets for the example input considered above. Here, the flow of the experiment is such that 16 configurations are sampled at the beginning and each of these are trained with the same 6.25% of dataset for 5 EPCs each. Out of the 16, best performing 8 configurations are chosen for the next rung, and the allocated data is also doubled to 12.5%. A point to note here is that this is not a new 12.5% of randomly sampled data: the 6.25% is continued from the prior rung and an extra 6.25% of data is sampled from the rest of the training dataset and the model is trained using this 12.5% of data for 5 iterations. By the end of the first successive halving iteration and in fact consistently at the end of each of the successive halving iteration, the 100% of the training dataset is used. This implies that at the beginning of each successive halving iteration, we reset the sample source to 100% of the training data and start the iteration. Since in the last iteration, we use 100% of dataset for each of the configurations, the data being used remains consistent across each of the configurations sampled in this iteration.

4.2.3. Training duration based

Consider the following inputs for using time based budget type for BOHB:

- Minimum Budget: 15 seconds
- Maximum Budget: 4 minutes (240 seconds)
- η: 2

The minimum budget, maximum budget and η have consistently been chosen across all budget types such that S_{max} always has value 4, causing 5 successive termination cycles and 5 successive halving iterations. This design choice is primarily based on recommendations from Li et al. [17].

i	S = 4		S = 3		S = 2		S = 1		S = 0	
	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i
0	16	15	8	30	4	60	4	120	5	240
1	8	30	4	60	2	120	2	240		
2	4	60	2	120	1	240				
3	2	120	1	240						
4	1	240								

Table 4.3: Budget distribution example for fixed horizon training duration based BOHB.

Table 4.3 represents the distribution of budgets while using training duration as a budget in terms of seconds. For example when S = 4 and i = 0, we sequentially sample 16 configurations either randomly or based on the Bayesian model and train the model for 15 seconds for each of the configurations and evaluate their performance metrics. On the next termination cycle, we promote 8 of the best performing configurations to the next rung and train them further for another 15 seconds, causing the total training duration to equal 30 seconds. This cycle is repeated until we have only one of the best performing configuration of the successive halving iteration which would have been trained for a total duration of 240 seconds. This process is repeated with correspondingly allocated budget for all of the 5 iterations of successive halving and the best performing configuration is returned to terminate the experiment.

4.3. Challenges with the fixed horizon budgets

While the fixed horizon budget scheme demonstrated an excellent avenue for efficiently converging to a well performing configuration for machine learning models, it requires specifying the minimum budget, the maximum budget as well as the budget type for each of the problems.

Consider optimizing the learning rate, the batch size and 3 other network hyperparameters for a CIFAR 10 model described in section 5.1.1. Figure 5.1a represents a study of fixed horizon BOHB with 3 different budget types optimizing hyperparameters for the CIFAR 10 model. We can see significant difference in the performance of each budget type: epoch based BOHB yielded the lowest validation error as compared to the remaining budget types. Furthermore, the superiority of epoch as a budget type here can also be observed in form of its any-time performance (lower validation error than others at any given time). So post experiment, we could derive here that epoch is the preferred budget for this hyperparameter tuning task.

Furthermore, in figure 5.2, we can see the impact of trying out different budgetary values for the time based as well as the training data subset based optimizers. We notice that varying budget parameters significantly improved the performance of the time and trainset based optimizers. In the new budget constraints for both the optimizers, we were able to attain a final performance similar to that of epoch based optimizer. So, the budget parameters do have a significant impact in the tuning process. However, we were only able to derive the better budgetary parameters after experimentation, and it is not possible to set good budgetary constraints a priori.

While we came across numerous attempts to augment Hyperband with search algorithms, we did not find any study that establishes a blueprint for choosing the budget type or the budget constraints for hyperparameter tuning tasks. Given the criticality of defining budget type and budget parameters, and given lack of any research in this direction, it seeds a very interesting avenue of research .

4.4. Infinite horizon two dimensional budget exploration

To combat each of the challenges discussed in section 4.3, we define three different infinite horizon budget allocation schemes. The fundamental idea behind each of the three allocation schemes is to run multiple BOHB experiments of a primary budget type while varying the auxiliary budget after each BOHB experiment. This process is repeated until the termination condition is reached, which include saturation of gain in performance between BOHB experiments among others. The termination conditions have been studied further in section 4.4.1. Variations in auxiliary budget type is incorporated to leverage a versatile and granular ability of defining approximations of the objective function. We explore Training duration, Epochs and Training data subsets as the primary budget. While we explored each of the three budget types as a primary budget resource, we only limit the to a single auxiliary budget type leaving three or more dimensional exploration as future work. In the following subsections, we first establish the termination conditions for the budget allocation schemes in 4.4.1, followed by a general guideline for forming primary budget blueprints in section 4.4.2 and finally we discussion on each of the two dimensional budget explorations have been incorporated, as well as the rational behind the design choices. The standard approach followed in each of the infinite horizon schemes is to first form a blue print BOHB with the primary budget type with a starting value for the auxiliary budget type. Once the BOHB experiment is completed, the auxiliary budget is increased based on the gradient of gain in accuracy from the previous BOHB experiment. This process is repeated till the terminal condition has been met.

4.4.1. Termination conditions

Each of the infinite horizon allocation schemes incorporate the following termination conditions, listed in the order of priority, 1 being the highest priority. The priority defines the sequence of checks being performed at the end of every successive halving iteration. By default, saturation of accuracy gain between BOHB experiments is chosen as the terminal condition. Each of the infinite budget allocation schemes have been designed such that a single termination condition has been specified.

1. **Time Deadline**: In order to activate time deadline as a terminal condition, the users would need

to specify the maximum time duration they want to spend for tuning hyperparameters. The total time elapsed in the tuning process is continuously tracked, and is compared with the terminal time deadline at the end of every successive halving iteration. If the time deadline has been exceeded, the tuning process will terminate irrespective of the possibility of gaining better performance by tuning for longer duration.

2. **Target Accuracy**: If the users have specified a target accuracy they want to achieve out of the tuning experiment, as soon as the target accuracy has been achieved by the model, the tuning process is terminated. Even if the budgets for the tuning experiment has not been completely exhausted, indicating the possibility of the model to gain better accuracy, the tuning process will end if the target accuracy has been met. Here, the accuracy is referred to the performance metric that has been evaluated in the validation dataset. Performance metric could be the validation accuracy, precision, recall, F1 score, etc.

3. **Accuracy gain saturation**: Accuracy gain saturation is the default termination condition, and does not require any user input. The performance metric of each of the configurations evaluated in each of the BOHB experiment is tracked through out the tuning process. If the difference in performance metrics obtained via two consecutive BOHB experiments is less than 3%, the tuning process is considered complete.

4.4.2. Guidelines for creating blueprints

Each of the infinite horizon budget allocation schemes build on a fixed horizon budget scheme of the primary budget type and continually increasing the auxiliary budget that has a multiplier effect on the fixed budget blueprint. The underlying guideline for defining the fixed budget blueprint is to choose a minimum budget, a maximum budget and η such that

- $S_{max} + 1 = 5$ so as to get 5 iterations of successive halving and 5 termination rungs in each of the successive halving experiments, where

$$S_{max} = \log_{\eta} \frac{maximum\_budget}{minimum\_budget} \tag{4.1}$$

- these parameters provide provide sufficient exploration, while performing as cheap evaluations as possible. The reasoning behind choosing the least expensive setting is that, early on, we want to explore cheaper fidelities. The fidelities become more expensive as an when the auxiliary budgets increase.

4.4.3. Epochs with increasing trainset

Here, we use epochs as the primary budget type and training dataset as the auxiliary budget type. Training dataset is used as the auxiliary budget, since it provides a easy and logical segmentation of fidelities. Following the standard approach described in the introduction of section 4.4, we start off forming a blueprint fixed horizon with epoch as the primary budget, and consecutively increase the accuracy of approximation of the objective function by increasing portions of training dataset used. Once the training dataset has been exhausted, if further experiments are required, we introduce a multiplier term to the epochs from the fixed budget blueprint. Let us dive deeper into this allocation scheme in the following sub-sections.

Blueprint for epoch based BOHB

For definition of fixed budget BOHB, we would need to set three parameters: minimum_budget, maximum_budget and η. To enable the cheapest possible fidelity, we can set minimum_budget as 2. We cannot set it lower than 2, since performing 1 epoch of training would mean that the performance metric received will not be entirely representative of the hyperparameter configuration, but that of the initial model parameters. Setting 2 epochs as the minimum budget allows the model parameters to be learnt, albeit not entirely accurately. So, even though the performance received is not entirely representative of the configuration, it is a good approximation. Now, having set **minimum_budget = 2**, we need to define η and the maximum_budget. Apart from determining the aggressiveness of termination in successive halving, η also influences the degree of exploration we want to consider. Given that we have

an infinite horizon, the number of configurations selected per BOHB experiment can be kept low, since the number of configurations would also increase as we increase the auxiliary budget. Furthermore, the minimum value η can take is 2. So, let us allocate the set η **= 2** Finally, now that we have the minimum budget and η, we can determine the maximum budget based on the idea that $S_{max} + 1 = 5$. Consider the following calculations.

$$S_{max} + 1 = 5$$
$$S_{max} = 4$$

From equation 4.1, we have

$$\log_{\eta} \frac{maximum\_budget}{minimum\_budget} = 4$$
$$\log_{2} \frac{maximum\_budget}{2} = 4$$
$$maximum\_budget = 2 * 2^4$$
$$maximum\_budget = 32$$

Consecutively, we have minimum_budget = 2, maximum_budget = 32 and η = 2, forming the blueprint same as what has been described in table 4.1.

Allocation Scheme

In this budget allocation scheme, we first establish a fixed horizon BOHB budget allocation scheme with epochs as a budget. However, unlike the standard fixed budget epoch based BOHB described in section 4.2.1, we start with only **6.25% of the training dataset**. So, we perform the first BOHB experiment with budgets defined in table 4.1 with 6.25% of data. For the next iteration, since we only have one data point so far, we cannot compute gradient. So, we double the amount of training dataset to 12.5% of the actual training data. Now, consider the following terminologies.

$best\_acc\_1$: *Best validation accuracy received in BOHB evaluation with 6.25% data.*
$best\_acc\_2$: *Best validation accuracy received in BOHB evaluation with 12.5% data.*
$trainset\_1 : 6.25\%$*(Amount of training dataset considered for BOHB experiment in first trial)*
$trainset\_2 : 12.5\%$*(Amount of training dataset considered for BOHB experiment in second trial)*

The gradient can then be calculated as

$$g_{(2,1)} = \frac{best\_acc\_2 - best\_acc\_1}{trainset\_2 - trainset\_1}$$

Finally, the next, and similarly the consecutive training dataset budget can be calculated as

$$trainset\_budget_n = max(100, \lceil trainset\_budget_{n-1} * (1 + g_{((n-1),(n-2))}) \rceil) \quad (4.2)$$

A key point to note here is that the maximum amount of training dataset that can be allocated is 100% of the training data available. In the condition that the gain of accuracy has not saturated, we introduce the **epoch_multiplier**. While the training dataset budget is less than 100%, the epoch_multiplier remains 1. Once the training dataset budget crosses 100%, the epoch_multiplier is increased by 0.5 for every BOHB experiment. In terms of the budget, it would mean that each of the epoch budget values will be multiplied by 1.5 for the first BOHB experiment after training dataset budget of 100% has been crossed. This can be better visualised from table 4.4.

i	S = 4		S = 3		S = 2		S = 1		S = 0	
	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i
0	16	3	8	6	4	12	4	24	5	48
1	8	6	4	12	2	24	2	48		
2	4	12	2	24	1	48				
3	2	24	1	48						
4	1	48								

Table 4.4: Epoch based budget blueprint for infinite horizon epoch with increasing training dataset allocation scheme for the first BOHB experiment once the training dataset budget has crossed 100%.

4.4.4. Trainset with increasing epochs

In this budget allocation scheme, we use portions of training dataset as the primary budget type and epochs as the auxiliary budget type. Similar to the guideline discussed in the introduction of section 4.4, we start by defining a fixed horizon BOHB with budget distribution based on the primary budget type. The auxiliary budget type is consequently increased based on the gradient of gain in performance metrics across iterations BOHB experiments. Let us start by establishing the blueprint and then continue to the gradient based increment scheme.

Blueprint for training dataset based BOHB

For training dataset based portion based budget type, we would need to define the starting epochs per configuration (EPC) in addition to the minimum_budget, the maximum_budget and η. Section 4.2.2 provides an introduction to how each of these parameters influence the amount of compute resources allocated for the BOHB experiment. Let us begin by defining a starting point for EPC. Briefly, this value represents the number of times each portion of the budgeted training dataset will iterate through the model in the training phase. The minimum value EPC could be set to is 2, since if the model parameters aren't tuned at all, the performance metric received would not be representative of the quality of the configuration, but that of the model parameters. So, to enable the cheapest possible fidelity, we set **EPC = 2**.

Furthermore, the maximum possible budget in case of a trainig dataset based BOHB is 100, representing training the model with 100% of the training dataset available. So, we can set the **maximum_budget = 100**. Similar to the rational discussed in the blueprint definition of section 4.4.3, we set η **= 2**. Finally, to ensure $S_{max} + 1 = 5$, consider the following calculations.

$$S_{max} + 1 = 5$$
$$S_{max} = 4$$

From equation 4.1, we have

$$\log_{\eta} \frac{maximum\_budget}{minimum\_budget} = 4$$
$$\log_{2} \frac{100}{minimum\_budget} = 4$$
$$minimum\_budget = \frac{100}{2^4}$$
$$minimum\_budget = 6.25$$

So to summarise, we have starting EPC = 2, maximum_budget = 100, minimum_budget = 6.25 and η = 2. With this setting, we get the budget distribution same as what has been described in table 4.2.

Allocation Scheme

In this budget allocation scheme, we begin by defining a blueprint for training dataset based BOHB as described in the section above. In the first BOHB experiment, we set EPC = 2. On the second BOHB experiment, since we do not have the gradient of gain available yet, we simply double the EPC, setting it to 4 and repeat the BOHB blueprint with training dataset budget. For all subsequent BOHB experiments until one of the terminal conditions defined in section 4.4.1 have been met, we follow the following approach to increase the EPC. With this established, consider the following terminologies.

$best\_acc\_1$: *Best validation accuracy received in BOHB evaluation with EPC = 2.*

$best\_acc\_2$: *Best validation accuracy received in BOHB evaluation with EPC = 4.*

$EPC\_1 : 10\%$*(The epochs per configuration defined for the first iteration of BOHB experiment.)*

$EPC\_2 : 20\%$*(The epochs per configuration defined for the second iteration of BOHB experiment.)*

The gradient can then be calculated as

$$g_{(2,1)} = \frac{best\_acc\_2 - best\_acc\_1}{EPC\_2 - EPC\_1}$$

Finally, the next, and similarly the consecutive training dataset budget can be calculated as

$$EPC_n = \lceil EPC_{n-1} * (1 + g_{((n-1),(n-2))}) \rceil) \tag{4.3}$$

So to conclude, the infinite aspect of using training dataset portions as a budget is achieved by continually increasing the number of iterations each of the training dataset budget is passed thorough the model. The higher the EPC, the more accurate the representation of the objective function and the more expensive the evaluation.

4.4.5. Training duration with increasing trainset

In this budget allocation scheme, we use the time duration spent on training the model as a primary budget and percentage of training dataset as the auxiliary budget. Following the allocation schemes described in sub-sections 4.4.3 and 4.4.4, we start by defining a blueprint for the fixed horizon budget allocation scheme and leverage a continuously increasing auxiliary budget as the means to achieve infinite horizon, along with granular control on accuracy of fidelities (expensiveness of fidelities). We begin by defining the blueprint and follow it through with the end-to-end allocation scheme.

Blueprint for training duration as a budget

For defining a fixed horizon BOHB with training duration as a budget, we require three inputs: the minimum_budget, the maximum_budget and η. In line to the rational discussed in the introductory part of subsection 4.4.3, we set η **= 2**. Furthermore, to select the minimum_budget, we heuristically choose 15 seconds as the minimum budget. Here, a training duration of 15 seconds indicate that irrespective of the number of training samples being considered, the model will be trained for 15 seconds. Unlike typical machine learning pipelines where we gauge the training progress in terms of the epochs or the number of iterations for which the training dataset has been subjected to the model, we ignore the iteration count in this budget type and instead track the amount of time spent for training. Even though modern machine learning models can take hours or even days to complete the training process, we could limit the starting minimum_budget to 15 seconds, since the auxiliary budget and the additional multiplier factor will adapt to the model performance and gains in accuracy. So far, we have established η = 2 and minimum_budget = 15. To calculate the maximum_budget, we follow calculations similar to what has been presented in sections 4.4.3 as well as 4.4.4. We have established that $\log_\eta \frac{maximum\_budget}{minimum\_budget} = 4$. Filling in the appropriate values, we get

$$\log_2 \frac{maximum\_budget}{15} = 4$$
$$maximum\_budget = 15 * 2^4$$
$$maximum\_budget = 240$$

Consecutively, with η = 2, minimum_budget = 15 and maximum_budget = 240, we get the blueprint as described in table 4.3.

Allocation Scheme

Consistent with the rest of the infinite horizon budget allocation schemes discussed so far, we start out by defining the blueprint budget allocation scheme of a fixed horizon setting. We begin the tuning

experiment with setting the auxiliary budget of training dataset portion to 10% of the data. In other words, we perform the blueprint BOHB described above in this subsection with 10% of data for the first BOHB experiment. For the second experiment, we simply double the amount of training dataset considered to 20% of the total training dataset. For each of the consecutive datasets, we follow a gradient based approach. Consider the following terminologies.

$best\_acc\_1$: *Best validation accuracy received in BOHB evaluation with 10% data.*

$best\_acc\_2$: *Best validation accuracy received in BOHB evaluation with 20% data.*

$trainset\_1 : 10\%$*(Amount of training dataset considered for BOHB experiment in first trial)*

$trainset\_2 : 20\%$*(Amount of training dataset considered for BOHB experiment in second trial)*

The gradient can then be calculated as

$$g_{(2,1)} = \frac{best\_acc\_2 - best\_acc\_1}{trainset\_2 - trainset\_1}$$

Finally, the next, and similarly the consecutive training dataset budget can be calculated as

$$trainset\_budget_n = max(100, \lceil trainset\_budget_{n-1} * (1 + g_{((n-1),(n-2))}) \rceil) \tag{4.4}$$

Similar to the condition discussed in sub-section 4.4.3, the auxiliary budget in this allocation scheme also has an upper limit of 100%. Till the auxiliary budget reaches 100%, we set the time_multiplier = 1. Once we have exhausted defining fidelities by taking parts of dataset, we start to increase the time_multiplier by 0.5 for each iteration of BOHB. So, as soon as the auxiliary budget crosses 100%, the time_multiplier would become 1.5, modifying the blueprint BOHB by increasing each of the time budget. This change with time_multiplier = 1.5 can be visualised in the table

i	S = 4		S = 3		S = 2		S = 1		S = 0	
	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i	N_i	R_i
0	16	22	8	45	4	90	4	180	5	360
1	8	45	4	90	2	180	2	360		
2	4	90	2	180	1	360				
3	2	180	1	360						
4	1	360								

Table 4.5: Budget distribution example for fixed horizon training duration based BOHB with time_multiplier set to 1.5.

4.5. Multi-Tune

While the budget allocation schemes discussed in section 4.4 introduced three different infinite horizon budgets, along with granular ability to control the fidelity, it still required defining the primary budget type. However, this is not a trivial choice, since different budget types are better suited for different machine learning problems. So, to enable the algorithm to take an informed decision on the primary budget type without requiring human intervention, this budget allocation scheme first explores a miniature version of each of the three allocation schemes discussed in section 4.4. Based on the performance obtained in each of the three budget types, the primary budget type is chosen. The primary budget type is then paired with the corresponding auxiliary budget type from section 4.4 and the tuning process continues as per the corresponding primary and auxiliary budget allocation scheme. Let us begin by first defining each of the miniature types, followed by describing a mechanism to leverage all the information obtained during miniature evaluation in terms of transferring learnings to the Bayesian Model.

4.5.1. Miniature evaluations

To evaluate the best performing budget type for tuning the hyperparameters of a given machine learning problem, we start off by considering the blueprint budget allocation scheme for each of the three budget type as described in tables 4.1, 4.2 and 4.3. For the initial version of this exploration scheme, we evaluated 1 successive halving iteration (S = 4) from each of the budget types. However, initial experiments revealed that simply observing two termination rungs was sufficient to determine the best

performing budget type. So, the miniature evaluations for each of the budget types can be defined as follows. A point to note is that each of the budget type has its own Bayesian Model and is uninfluenced by the performance observed via other budget types. While drawing the configurations, it might be drawn at random or based on the Bayesian model built for that budget type. A detailed discussion on sampling configurations can be found in section 3.3.

- **Epochs with increasing training dataset**: With 6.25% of training dataset as the auxiliary budget, we first evaluate 16 configurations for 2 epochs, and promote 8 of the better performing configurations to be evaluated to 4 epochs.
- **Training dataset portions with increasing epochs**: With the EPC set to 2 as the auxiliary budget, we first evaluate 16 configurations with 6.25% of the training dataset and promote 8 of the better performing configurations to be evaluated for 12.5% of training data.
- **Training duration with training dataset**: We begin by evaluating 16 configurations by training the model with 6.25% of the training dataset for 15 seconds. Then, the better performing 8 configurations are promoted to be evaluated for 30 seconds over the same dataset.

For each of the budget types, the best performance metric observed is noted, along with the time duration taken to execute the evaluation for the budget type. The algorithm for selecting the primary budget type has been defined in algorithm 2.

Consider the following terminologies for the process.

acc_1 : *The best accuracy received among each of the three budget type.*

$budget_1$: *The primary budget type of the miniature which resulted in finding the best accuracy.*

dur_1 : *The amount of time spent in by miniature of type* $budget_1$ *in reaching the accuracy* acc_1

acc_2 : *The second best accuracy received among each of the three budget type.*

$budget_2$: *The primary budget type of the miniature which resulted in finding the second best accuracy.*

dur_2 : *The amount of time spent in by miniature of type* $budget_2$ *in reaching the accuracy* acc_2

Algorithm 2 Algorithm for choosing the primary budget type.

Result: Primary budget type to be used for the rest of the evaluations in infinite horizon allocation scheme with budget exploration.

$acc_{delta} = |acc_1 - acc_2|$
$dur_{delta} = |dur_1 - dur_2|$
ACC_DELTA_THRESH = 5%
DUR_DELTA_THRESH = 30 (Seconds)
if acc_{delta} > *ACC_DELTA_THRESH* **then**
  Return $budget_1$
else
  if dur_{delta} > *DUR_DELTA_THRESH* **then**
    if $dur_1 < dur_2$ **then**
      Return $budget_1$
    else
      Return $budget_2$
    end
  else
    Return $budget_1$
  end
end

4.5.2. Bayesian model

While we were able to find the best performing primary budget type using the miniatures defined in section 4.5.1, we have still spent significant time and compute resources in evaluating each of the miniatures that are not continued. Even though we optimize the evaluation of primary budget type by

Epochs		Training dataset		Training duration	
Epochs	**Trainset %**	**Trainset %**	**EPC**	**Duration**	**Trainset %**
2	6.25%	6.25%	2	15 Seconds	6.25%
4	6.25%	12.5%	2	30 Seconds	6.25%

Table 4.6: Miniature evaluations for each of the three primary budget types: epochs, training dataset and training duration.

simply continuing the evaluations made during the miniature evaluation phase, along with its Bayesian model, we would ideally like to use the Bayesian models from miniatures that are not continued as well. The underlying principle that enables reusing the evaluations from each of the primary budget types is the ability to translate readings from one budget type to another by leveraging deliberately assigned heuristics to each of the miniature evaluations.

To get started, consider the Epochs and the Training dataset columns in table 4.6. In each row, it can be observed that the quantity of data that has been assigned per configuration is the same. For example, in the first row with epochs as the primary budget, we train each model with 6.25% of data for 2 iterations. Even in training dataset as the primary budget, we train each model with 6.25% of data for 2 iterations. Similarly, on the second row, we train the model for 4 iterations using 6.25% of training data in epochs as the primary budget and for 2 iterations using 12.5% of training data in training dataset as the primary budget. While the second row is not directly comparable, it can be observed that the number of data samples each of the models have been through remain the same.
Following the same pattern, we can quantify the amount of compute each budget type spends at every stage of the miniature evaluations. And this parameter: the amount of compute forms the baseline that enables translation from one budget type to another. Before we dive deeper into how we could translate each of the primary budget types to one another, let us establish some terminologies and assign values where possible.

- e_{min}: This corresponds to the minimum budget in epoch based miniature budget. Similarly, the EPC in training dataset based miniature budget also has the same value as e_min. From table 4.6, e_min can be initialised with value 2.

- t_{min}: This variable corresponds to the minimum budget in training dataset based miniature budget. Similarly, the training dataset budget in epoch based as well as training duration based primary budgets also have access to the t_{min} percentage dataset. From table 4.6, t_{min} can be initialised with value 6.25.

- $epochs\_obtained$: This variable is valid only in case of training duration as a primary budget and corresponds to the number of iterations the allocated amount of training datasets have been iterated through in the specified budget of duration.

- $epochs\_per\_minute$: Similarly, this variable is valid only in case training duration as a primary budget and takes value $\frac{epochs\_obtained}{duration\_budget}$, where $duration\_budget$ refers to the budget of training duration allocated as a part of the miniature evaluation.

Let us now describe the translation algorithm from a source budget type of each of the primary budgets to both the remaining target budget types. Each translation essentially entails translating the budget variable from a source type to a target type. Once the source type budget has been translated to the target type budget, the performance obtained from each configuration can be modelled into the target Bayesian model by simply treating each evaluation of the source type as an evaluation of the destination type with translated budget. More specific description of how Bayesian models are built for each BOHB experiment can be found in section 3.3.

- **Base budget type: epochs**
  In each of the cases under this bullet, the source_budget refers to the number of epochs corresponding to the current budget.

  - Target budget type: training dataset
    $target\_budget = \frac{source\_budget}{e_{min}} * t_{min}$

Where, target_budget is the translated budget of trainset type.

– Target budget type: training duration
$target\_budget = \frac{source\_budget}{epochs\_per\_minute}$
Where, target_budget is the translated budget of training duration type.

- **Base budget type: training dataset portions**
In each of the cases under this bullet, the source_budget refers to the percentage of training dataset corresponding to the current budget.

  – Target budget type: epochs
  $target\_budget = \frac{source\_budget}{t_{min}} * e_{min}$
  Where, target_budget is the translated budget of epoch type.

  – Target budget type: training duration
  $target\_budget = \frac{source\_budget}{t_{min}} * (e_{min} * epochs\_per\_minute)$
  Where, target_budget is the translated budget of training duration type.

- **Base budget type: training duration**
In each of the cases under this bullet, the source_budget refers to the time duration in seconds spent on training the model, corresponding to the current budget.

  – Target budget type: epochs
  $target\_budget = epochs\_obtained$ *from the* $source\_budget$
  Where, target_budget is the translated budget of epoch type.

  – Target budget type: training dataset
  $target\_budget = \frac{epochs\_obtained}{e_{min}} * t_{min}$
  Where, target_budget is the translated budget of training dataset type.

The utility of these translation metrics have been validated by comparing hyperparameter tuning experiments with and without reuse of Bayesian models via translation. These results have been presented in section 5.4.

4.6. Conclusion

This chapter introduces the three primary budget types considered in this thesis for hyperparameter tuning. Then, having looked at the fixed horizon budget allocation schemes from the vanilla BOHB implementation, we established the scope of improvement encompassing two fundamental pain points: it is impossible to know a priori, the appropriate minimum and maximum bounds for budgets being used, and furthermore, it is non-trivial to decide on the best budget type for different machine learning tasks. So, to tackle both of these pain points, we start first by addressing the lower and upper bounds concerns by defining three different infinite horizon budget allocation schemes. Finally, we address the second pain-point of defining a way to allocate the best budget type for hyperparameter tuning purposes via an exploratory algorithm that encompasses each of the three infinite horizon algorithms. Next, in chapter 5, we will experiment with each of these budget allocation schemes and evaluate their utility.

5 Experiments

This chapter describes the experiments that have been performed in the course of this thesis. The experiments have been primarily designed to gauge the performance of different versions of the MultiTune algorithm, and to establish motivations and justifications to the design choices for the algorithm. Cumulatively the experiments also attempt to answer the research questions proposed in section 1.2. We begin in section 5.1 by briefly describing the 5 datasets, models and their hyperparameters that have been considered for the experiments that follow. Next, in sections 5.2, 5.3, 5.4, 5.5 and 5.6, we discuss the experiments that have been undertaken. For each of the experiments, we begin by defining the hypothesis and the motivation for the experiment. We run 3 iterations for each benchmarking experiments and plot the average results. Here, the average results indicate mean of the lowest validation error (highest validation accuracy) observed so far for a given wall-clock-time. Additionally, we also show the standard deviation for each of the average plots as a shaded region along the plot. We have chosen wall-clock-time as a metric to benchmark against, since we perform all the experiments on a common compute platform, allowing for reasonable comparison in each benchmark. Finally, each of the experiments also includes a brief discussion on the results and its inferences.

5.1. Data and Models

In this thesis, each of the experiments that have been performed use one or more of the models described in this section for tuning hyperparameter. We use BOHB as the primary optimizer for hyperparameter tuning, with varying budget allocation setups. Each of the following subsections briefly describe the different datasets being used, respective models for each of the datasets along with their hyperparameter search-space.

5.1.1. CIFAR 10

CIFAR 10 dataset [15] is a 10 class classification dataset, which is made up of 32 * 32 color images, implying 3 channels for each of the image samples. The dataset consists of 50,000 training samples and 10,000 test samples. For each of the experiments, we have reserved 10,000 images out of the 50,000 training samples as the validation dataset.

For the experiments, we use a simple model to limit the cost of computation. Even though the expected validation and test accuracy from this model is in the proximity of 72-74% which is far from the state of the art, this example is sufficiently complex to analyse hyperparameter tuning. The model consists of 2 convolution layers and 3 fully connected layers, each with RELU as the activation function. We use SGD optimizer for learning the weights and biases of the network. Table 5.1 enlists the hyperparameters considered for this model, their ranges along with the distribution type for each of the hyperparameters.

5.1.2. MNIST

MNIST [16] dataset is again a 10 class classification dataset which is made up of 28 * 28 grey scale images. Here, the 10 classes represent each of the 10 digits in natural number. The dataset consists of 60,000 training samples and 10,000 test samples. Similar to that in CIFAR 10, we reserve 10,000 images from the training dataset as validation set for our model. Similar to CIFAR 10, we use use

Hyperparameter	Value Type	Range	Distribution
Learning Rate	Float	[1e-6, 1e-2]	Logarithmic
Hidden Nodes	Ingeger	[20,100]	Discrete
Batch Size	Categorical	{64, 128, 256, 512}	Choice
Convolution Channel 1	Categorical	{32, 64, 128}	Choice
Convolution Channel 2	Categorical	{64, 128, 256, 512}	Choice

Table 5.1: Hyperparameters considered for experiments with CIFAR 10 dataset.

CNNs for classifying MNIST dataset. Even though the model is fairly simple, we expect above 98% validation and test accuracy consistently. The model used here consists of 2 convolution layers and 2 fully connected layer, along with dropout. We use SGD optimizer for learning the weights and biases of the network. The hyperparameters considered for this neural network along with their ranges and distribution have been showcased in table 5.2.

Hyperparameter	Value Type	Range	Distribution
Learning Rate	Float	[1e-6, 1e-2]	Logarithmic
Optimizer	Categorical	{Adam, SGD}	Discrete
Batch Size	Categorical	{128, 256, 512}	Discrete
SGD Momentum	Float	[0, 0.99]	Conditional: Only active if optimizer == SGD
Number of conv layers	Integer	[1,3]	Discrete
Number of filters in first conv layer	Integer	[4, 64]	Logarithmically varying int values
Number of filters in second conv layer	Integer	[4, 64]	Logarithmically varying int values Conditional: Only valid if number of layers >= 2
Number of filters in third conv layer	Integer	[4, 64]	Logarithmically varying int values Conditional: Only valid if number of layers >= 3
Dropout Rate	Float	[0,0.9]	Continous
Number of hidden units in fully connected layer	Integer	[8, 256]	Logarithmically varying int values

Table 5.2: Hyperparameters considered for experiments with MNIST dataset.

5.1.3. Fashion MNIST

Fashion MNIST dataset was created as an alternate to the MNIST, given its overuse in the machine learning community and the strong urge to seek alternatives for it [24]. Similar to the MNIST dataset, Fashion-MNIST is a 10 class classification dataset, consists of 60,000 training samples and 10,000 test sample. Each of the sample is a 28 * 28 grey scale image. Consistent with CIFAR 10 and MNIST, we reserve 10,000 images from the training dataset as the validation set, making the train-validate-test split as 50,000 - 10,000 - 10,000.

For this dataset, we use a fairly simple CNN, consisting of 2 convolution layers and 1 fully connected layer, along with dropout. Here, we use Adam optimizer to train the weights and biases, and expect the validation and test accuracy close to 92 %. Table 5.3 represents the hyperparameters considered for this classification task, their ranges and their distributions.

Hyperparameter	Value Type	Range	Distribution
Learning Rate	Float	[1e-5, 1e-1]	Logarithmic
Dropout	Float	[0.1, 0.8]	Uniform
Batch Size	Categorical	{64, 128, 256, 512, 1024}	Choice
Convolution Channel 1	Categorical	{10, 12, 14, 16, 18, 22}	Choice
Convolution Channel 2	Categorical	{24, 28, 32, 36, 40, 44}	Choice

Table 5.3: Hyperparameters considered for experiments with Fashion MNIST dataset.

5.1.4. Twitter dataset

This dataset is used for part of speech classification of tweets []. The training dataset consists of 1,827 tweets, which have been annotated with 25 different part-of-speech tags, making it a 25 class classification problem. Similar to the rest of the experiments, we have reserved a part of the training dataset as the validation set, which in this case consists of 327 tweets, making the total training dataset size of 1,500 tweets. The test dataset consists of 500 tweets.
The model has been taken from the research article of Hendrycks et al. [8] where they study different approaches to introduce robustness against noisy labels. They introduce varying degree of noise in different datasets and benchmark their approach for robustness. In our experiments, we set the noise level to 0, making all the training labels accurate and true. This allows us to use their model as a standard classification problem, which in turn have been studied as a hyperparameter optimization problem in the scope of this thesis.
For the classifier, we use 2 layers of fully connected network with window size 3 and hidden size 256. We use Adam optimizer along with $\mathcal{L}2$ decay for learning the model parameters. Specific hyperparameters, their ranges and distributions used for this hyperparameter tuning problem has been listed in table 5.4.

Hyperparameter	Value Type	Range	Distribution
Learning Rate	Float	[1e-5, 1e-1]	Logarithmic
$\mathcal{L}2$ Decay Rate	Float	[1e-7, 1e-3]	Uniform
Batch Size	Categorical	{32, 64, 128, 256, 512}	Choice
Gold Fraction	Float	[0.1, 1]	Uniform

Table 5.4: Hyperparameters considered for experiments with Twitter part of speech dataset and Stanford sentiment treebank.

5.1.5. Stanford sentiment dataset

For experiments with the Stanford Sentiment Treebank, we predict the sentiment of each of the movie reviews present in the dataset. The dataset consists of 6,911 training samples, 872 samples for the validation set and finally 1,821 samples for the test set.
Similar to the twitter part of speech dataset, the model has been taken from the research article of Hendrycks et al. [8], and we set the noise level to 0, making all the training labels accurate and true. The model used has been directly taken from the experiment repository of Hendrycks et al. and redefined as a hyperparameter problem where the hyperparameters considered, their ranges and their ranges have been listed in table 5.4.

5.2. Experiment 1

In experiment 1, we try to answer the first research question from section 1.2, stated as follows.

Does varying budget constraints influence the hyperparameter tuning process in BOHB?

This question forms the backbone for the structure of our thesis, specifying the need for dynamic budget allocation. Defining budget constraints can be further broken down into two sub-problems: selecting an optimal budget type and setting the right budgetary values that produces reliable results on the selected budget type. Correspondingly, we break this research question into two sub-questions in an attempt to cumulatively look for underlying patterns in budget specification for hyperparameter tuning experiments.

5.2.1. Experiment 1 - a

In this sub-experiment, we answer the following research question.

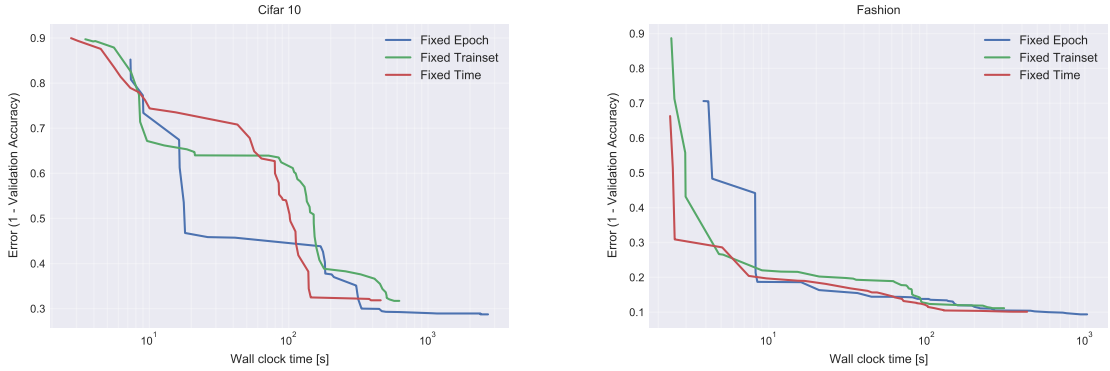
For a specified HPO task, is there a variation in performance of different budget types?

In this experiment, we select hyperparameter tuning tasks on Cifar 10 (section 5.1.1), Fashion MNIST (section 5.1.3) and MNIST (section 5.1.2) datasets. Here, we will execute three instances of vanilla BOHB for each of the HPO tasks. Each BOHB experiment simulates a unique budget type. The budgetary parameters for each of the three settings have been listed in table 5.5.

	Minimum Budget	Maximum Budget	Eta	Other parameters
Epoch based	1 epoch	16 epochs	2	
Trainset	6.25% training dataset	100% training dataset	2	Epochs per configuration (epc) = 4
Time	0.9375 seconds	15 seconds	2	

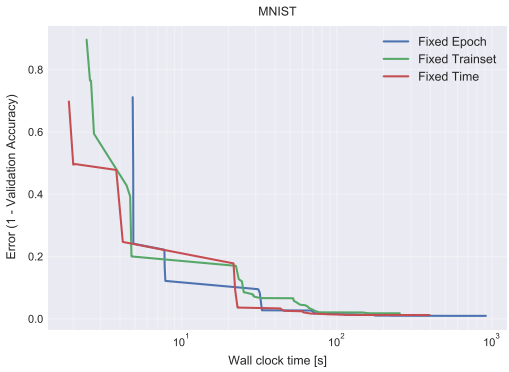
Table 5.5: Configurations considered for experiment 1-a on Cifar 10, Fashion MNIST and MNIST HPO tasks.

The underlying motivation for performing this experiment is to look for differences in efficacy of BOHB to optimize hyperparameters in different HPO tasks. The tuning problems selected offer diversity in the number of hyperparameters available to tune, the range and the variations for each of the hyperparameters. Such diversity should be able to simulate difference in the nature of optimization for each of the problems, possibly enabling difference in performance of each budget type. So, we hypothesize that while each of the budget types would converge to a good final performance, the rate of convergence might differ, owing to differences in the budget type.



(a) CIFAR 10

(b) Fashion MNIST



(c) MNIST

Figure 5.1: Each budget types yield different performance profiles on different HPO tasks.

Results and Inferences

Figure 5.1 showcases the performance of BOHB experiment for each budget type on the three HPO tasks considered. Contrary to our hypothesis, we observe that the difference is not only in the rate of convergence (any-time performance), but we also see difference in the final performance on Cifar 10 HPO task in figure 5.1a. Specifically, we see that the trainset based BOHB had an inferior performance compared to the the other two budget types. Even though time and epoch based experiments eventually converged to the same final performance, the epoch based BOHB converged to a better final performance faster than time based BOHB, implying better final performance. Additionally, we also observe that the epoch based BOHB was also lower than the remaining two trajectories in the Cifar 10 experiment, indicating better any-time performance.

Similarly, figure 5.1b showcases performance of epoch, time and trainset based BOHB for tuning hyperparameters on Fashion MNIST based HPO task. Contrary to Cifar 10 experiment, we see here that the difference in performance of each budget type is not significant. That said, we do observe marginally better performance in time based BOHB as compared to the rest of the budget types. The performance superiority of time based BOHB can be observed in its any-time performance (faster convergence as compared to the rest of the budget types) as well as its final performance (lower final converge as compared to the rest of the budget types).
Figure 5.1c showcases the same comparison on MNIST based HPO task. Similar to Fashion based HPO task, the differences are not as pronounced in this dataset as compared to Cifar 10. This could be attributed to the relative simplicity of the MNIST and Fashion MNIST datasets compared to Cifar 10. Smaller dimension of images along with use of monochromatic images causes significant reduction in the amount of computation required for convergence in Fashion MNIST and MNIST datasets. This property aside, we see that in MNIST, time based BOHB delivered better any-time performance than the rest of the budget types, establishing its superiority for MNIST based HPO task.

Conclusion
We see that the budget type does impart significant influence in hyperparameter tuning tasks. Complementing to this inference, we also see derive that no one budget type perform consistently better than any other budget type across HPO tasks.

5.2.2. Experiment 1 - b

In this sub-experiment, we answer the following research question.

> **Given an HPO task with specified budget type, does varying the constraint parameters influence the tuning process?**

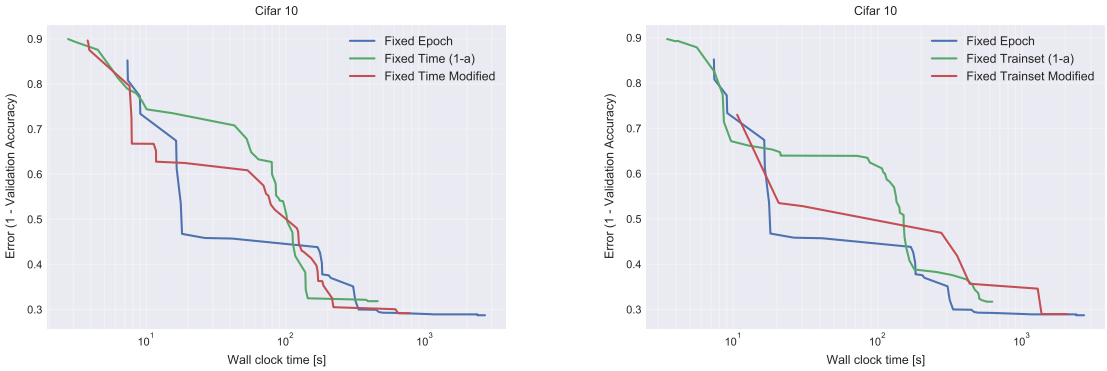
In experiment 1-a, we saw that different budget types performed differently in different HPO tasks. Let us now change our perspective to evaluate if for a specified budget type, does changing budget allocation parameters (minimum budget, maximum budget, eta and epochs-per-configuration for training set based budget) improve its performance to the level of an optimal budget type. We saw that for Cifar 10, epoch was the superior budget type. In this experiment, we will try iterations of the sub-optimal budget type and see if we can match the performance of the optimal budget type. We would be considering the final performance (converging to a better final validation accuracy) as well as the any-time performance (lower validation accuracy for any given time) in comparing the budget types. The underlying hypothesis here is that even by changing budget allocation parameters, a superior budget type will still out-perform an inferior budget type. Here, since epoch was the optimal budget type, we will benchmark epoch with varying time and training set based budgets. Table 5.6 lists the configurations of budget considered for this experiment.

	Minimum Budget	Maximum Budget	Eta	Other parameters
Fixed Epoch based	1 epoch	16 epochs	2	
Fixed Trainset (1-a)	6.25% training dataset	100% training dataset	2	Epochs per configuration (epc) = 4
Fixed Trainset (Modified)	6.25% training dataset	100% training dataset	2	Epochs per configuration (epc) = 16
Fixed Time (1-a)	0.9375 seconds	15 seconds	2	
Fixed Time (Modified)	1.875 seconds	30 seconds	2	

Table 5.6: Configurations considered for experiment 1-b on Cifar 10 for gauging performance with varying budget constraints.

Consider figure 5.2a, where we tried two variations of time based budget to compare with the optimal budget type (epoch based BOHB). Between the two time based trajectories, we do see improvement in the final performance as well as the any-time performance in tuning hyperparameters. Specifically, we see that the 'Fixed Time Modified' trajectory is generally lower than the 'Fixed Time (1-a)' trajectory. In figure 5.2b, where we look at two variations of training dataset based budget, we observe improvements similar improvement to that in time based budgets. We observe that the modified training dataset

parameters render better final performance as well as better any-time performance. The final validation error was minimized in both the cases. However, it should be noted even though the performance of each of the sub-optimal budget types improved with modifications to the constraint parameters, they were still not as good as the epoch based budget, which consistently rendered better performance as compared to all other budget types, in both final as well as any-time performance.



(a) CIFAR 10 with varying time budget

(b) Cifar 10

Figure 5.2: Modifying constraints for the sub-optimal budget types improved their performance relative to the same budget type, but were still not on par with the optimal budget type.

In fact, even if we increased the epc for training dataset based budget, the any-time performance would only get worse, owing to more time and resource allocated per configuration evaluation. On the other hand, while proportionately doubling the minimum and maximum budgets rendered better performance than the old time-based budget, further increase would also increase the minimum-budget spent per configuration, causing the any-time performance to deteriorate. Furthermore, it is impossible to reliably predict the optimal configuration for using BOHB even if we know the convergence points for a machine learning task.

Conclusion
In this experiment, we see that while we were able to improve the performance of sub-optimal budget types from experiment 1-a, these modified budget constraints still did not render performance on par with the optimal budget type. Additionally, we also discuss why sub-optimal budget types are unlikely to yield better results than an optimal budget type in vanilla BOHB setup.

5.3. Experiment 2

In this experiment, we benchmark the 2D infinite horizon budget allocation schemes, and try to answer the following research question:

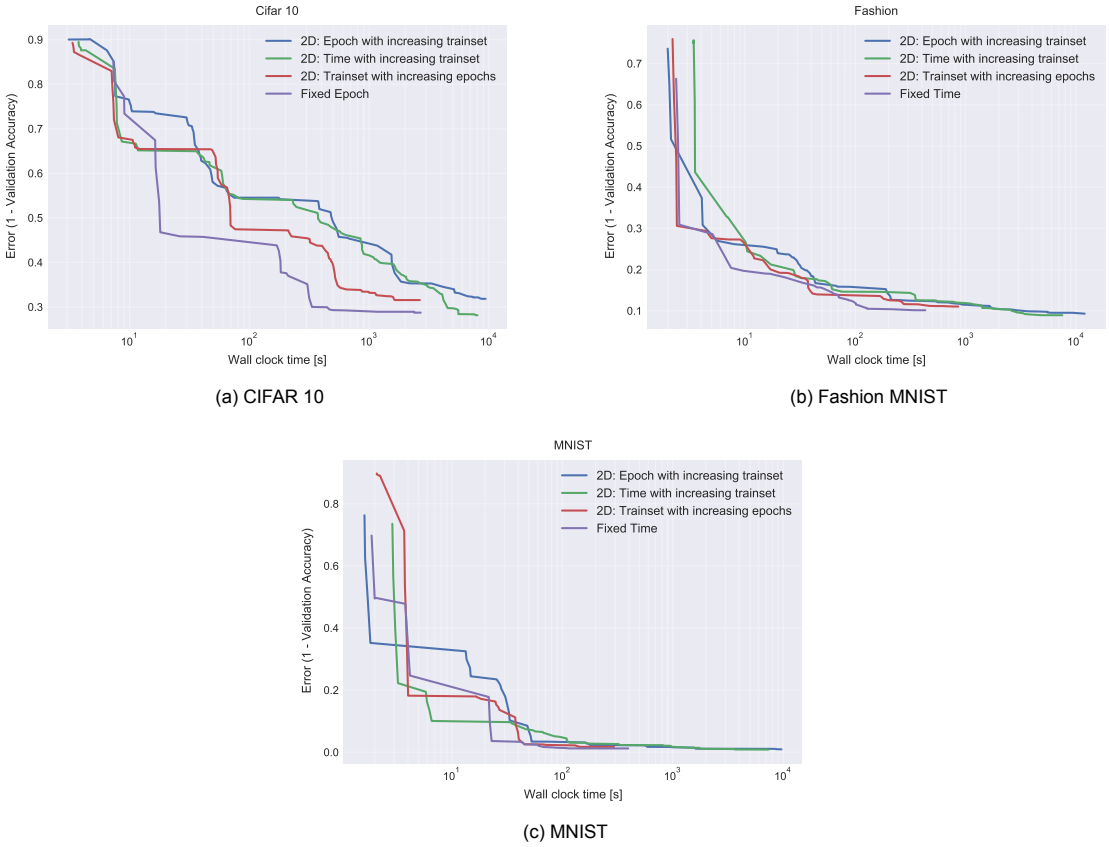
Having specified a budget type, can we alleviate the need for setting budget constraints?

Experiment 1-b presented a glaring challenge of vanilla BOHB, of defining budget constraints to converge to the best performing configuration as early as possible. This experiment formed the motivation to build dynamic budget allocation schemes that dynamically defined granular yet increasingly accurate fidelities of the objective function. With this, we defined 3 variants of MultiTune 2D in section 4.5, enabling dynamic allocation of budgets for hyperparameter tuning problems.
In this experiment, we wish to evaluate each of the MultiTune 2D algorithms discussed in section 4.4 on CIFAR 10(5.1.1), Fashion MNIST(5.1.3) and MNIST(5.1.2) HPO problems. We will monitor their performance over the fixed horizon BOHB, which have been configured for optimal performance based on experiment 1. Table 5.7 specifies the fixed horizon BOHB configurations selected for each of the HPO tasks.
The underlying hypothesis here is that at least one of the MultiTune 2D algorithm would be able to

Dataset	Budget Type	Minimum Budget	Maximum Budget	Eta
Cifar 10	Epoch based	1 epoch	16 epochs	2
MNIST	Time	0.9375 seconds	15 seconds	2
Fashion MNIST	Time	0.9375 seconds	15 seconds	2

Table 5.7: Vanilla BOHB configurations considered for experiment 2 on different HPO tasks to benchmark the performance of the three MultiTune 2D algorithms/values.

converge to a good final performance. We pessimistically look for only one superior algorithm per HPO task, since we have established that different HPO tasks have different preferred budget type for hyperparameter tuning.



(a) CIFAR 10

(b) Fashion MNIST

(c) MNIST

Figure 5.3: The figure demonstrates performance of the three MultiTune 2D algorithms on Cifar 10, Fashion MNIST and MNIST HPO tasks. At least one variant of the algorithm always converges to a good final performance. While this solved the need for specifying budgetary constraints, we still need to choose the best budget type.

Let us begin in figure 5.3a, where we study the performance of different MultiTune 2D algorithms over an epoch based budget for Cifar 10. We selected epoch based budget, since it showed superior performance in experiment 1. In line to our hypothesis, we observe differences in performance of different budget type. One of the unexpected patterns that emerged here is that MultiTune 2D algorithms did not follow the preferred budget type of fixed horizon setting. In experiment 1 we observed that epoch based BOHB rendered superior performance. But in this experiment, we see that the infinite algorithm with time as the primary budget type yielded superior performance. Additionally, even though the any-time performance of each of the MultiTune 2D experiments were worse than epoch based BOHB, time with increasing trainset was able to converge to the same final performance, without needing the users to specify any budgetary requirement. The loss in any-time performance could be attributed to defining more granular fidelities in the beginning stages.

Continuing to figure 5.3b,where we study performance of MultiTune 2D algorithms on Fashion MNIST HPO task, we observe similar patterns. While the difference is not as pronounced as Cifar 10, we do see variations in the performance of different budget types' final as well as any-time performance. Furthermore, we see that each of the MultiTune 2D algorithms were able to converge to very similar final performance. In experiment 1, we observed that Fashion MNIST performed superior in a time based budget. Similar to the case in Cifar 10, the preferred budget type did not persist in infinite horizon algorithms. Furthermore, similar to the case in Cifar 10, we observed consistent loss in the any-time performance in each of the MultiTune 2D algorithms.

Finally in figure 5.3c, we study performance of MultiTune 2D algorithms on MNIST HPO task. The patterns observed above continue, even though it is less pronounced than Fashion MNIST as well as Cifar 10 based HPO tasks. Owing to the simplicity of MNIST dataset, the differences in the performance becomes less significant. We can see that each of the MultiTune 2D algorithms could converge to the final performance similar to fixed horizon BOHB in experiment 1, in the duration it took for fixed-horizon BOHB to converge to the final accuracy.

Conclusion
So, to summarize the key observations,

- The difference in performance of different budget type still persists. This difference is observed in the rate of convergence to an optimal configuration, as well as the final convergence value.
- At least one of the three MultiTune 2D algorithm consistently converged to a good final performance, satisfying the desiderata of convergence without specifying budget constraints.

5.4. Experiment 3

In this experiment, we validate the miniature evaluation algorithm proposed in section 4.5.1, on its efficacy to derive an optimal budget type.Specifically, we wish to answer the following research question.

> **Without exhaustively evaluating each budget type, can we identify a preferred type for an HPO task?**

From experiment 2, we saw that each of the 3 MultiTune 2D algorithms performed differently in three different HPO tasks. The fundamental difference in the 3 MultiTune 2D algorithms is the difference in the primary budget type. In this experiment, we wish to establish a heuristic to predict the better performing budget type, without having to exhaustively evaluate each budget types. Knowing the preferred budget type a priori could induce significant performance improvement, especially in practical machine learning problems that require extensive resources for training the model.

The hypothesis for this experiment is that early performance of budget types would be indicative of the overall performance of the budget type. More specifically, we try to validate if the miniature algorithm discussed in section 4.5.1 can reliably derive the better performing budget type. To briefly reiterate this algorithm, we focus on the first two terminations of the cheapest successive halving iteration for each of the budget types. Specific evaluations for each of the MultiTune 2D algorithms can be found in table 4.6. We track the lowest validation error over these evaluations, along with the time taken for the algorithm to converge to that validation error. We perform this analysis for each of the HPO tasks considered in experiment 2 and see if a pattern emerges.

Consider an instance of each of the HPO tasks evaluated in experiment 2. As discussed above, we focus our observation only on the first two termination cycles of the cheapest successive halving iteration for each algorithm. The readings for this experiment have been directly sourced from experiment 2, so as to enable direct comparison between predictions made by the miniature algorithm and the actual performance observed by each of the MultiTune 2D algorithms on each HPO task. We will look at results obtained during experiment 2 to make predictions about the preferred budget type based on the miniature algorithm and evaluate if the predicted algorithm type indeed perform better than the other algorithms.

Budget Type	Best Accuracy Received	Duration to converge to the best accuracy
Epoch with increasing trainset	32.08%	39 seconds
Training dataset with increasing epochs	33.84%	47 seconds
Training Duration (Time) with increasing trainset	37.99%	43 seconds

(a) CIFAR 10

Budget Type	Best Accuracy Received	Duration to converge to the best accuracy
Epoch with increasing trainset	76.94%	26 seconds
Training dataset with increasing epochs	80.93%	26 seconds
Training Duration (Time) with increasing trainset	83.712%	46 seconds

(b) Fashion MNIST

Budget Type	Best Accuracy Received	Duration to converge to the best accuracy
Epoch with increasing trainset	75.55%	14 seconds
Training dataset with increasing epochs	87.79%	17 seconds
Training Duration (Time) with increasing trainset	89.36%	34 seconds

(c) MNIST

Table 5.8: Miniature evaluations for CIFAR 10, Fashion MNIST and MNIST datasets.

Table 5.8 summarises the readings required to simulate the miniature algorithm. Let us begin in table 5.8a, where we perform the analysis on Cifar 10 HPO task. We can see here that the best overall validation accuracy at the end of miniature evaluation was received in time based MultiTune 2D algorithm. Looking back at figure 5.3a, we can see that indeed time based MultiTune 2D had superior performance as compared to the rest of the infinite algorithms in terms of better final performance. Based on similar analysis on the remaining two HPO tasks considered, we can see that for Fashion MNIST, the preferred budget type based on the miniature algorithm is time based MultiTune 2D. In case of MNIST based HPO task, we see that while the highest validation accuracy of 89.36% was received in time based budget, notice that training dataset based MultiTune 2D converged to close to 87% validation accuracy in almost half the time. The miniature algorithm proposed in this case would choose training dataset as the preferred budget type, since the delta in accuracy gain is minimal as compared to the additional compute investment required for time based MultiTune 2D. Looking back in figure 5.3b and 5.3c, we can see that the algorithms predicted by the miniature algorithm indeed had superior performance compared to the rest of the algorithms.

Conclusion
Based on this experiment, we can conclude that the miniature algorithm can in fact identify better performing budget types without having to exhaustively evaluate each budget type. This enables us to solve the second bottle-neck of fidelity specification in BOHB: choosing the budget type.

5.5. Experiment 4

In this experiment, we answer the following research question.

> **How can we integrate the budget type selection and dynamic fidelity definitions in a unified algorithm?**

Having established the efficacy of miniature algorithm in experiment 3, we can identify the better budget type for any given HPO experiment without having to fully evaluate each budget type. While this allowed us to continue only the best performing algorithm or the budget type, terminating the sub-optimal

budget types. However, it is not desirable to simply discard results that we have already invested compute resources for. So we proposed a method to reuse results from miniature evaluations of all of the budget type to build a Bayesian model of the objective function in section 4.5.2. This reuse in Bayesian model could introduce speed-ups in the HPO experiment in terms of finding better performing configurations early on, lowering the need to spend more resources op potentially sub-par hyperparameter configurations. Building this Bayesian model requires specifying the model performance (validation error) along with the resource spent in the evaluation, in terms of the primary budget type (the optimal budget type). This entails that, to utilize each of the miniature evaluations for a Bayesian model built on the optimal budget type, we would need to translate the readings from each sub-optimal budget type into an equivalent reading from the optimal budget type. Based on the underlying heuristic of the amount of compute resources spent for each miniature evaluation, we proposed in section 4.5.2 an algorithm that can translate results from any budget type to any other budget type.

In this experiment, we wish to evaluate the efficacy of this translation algorithm and validate the speed-up. We do this by evaluating MultiTune 3D algorithm proposed in section 4.5 with and without utilization of the readings from sub-optimal budget types. To briefly reiterate MultiTune-3d algorithm, we evaluate miniatures of each budget type and continue the tuning experiment with the optimal budget type. The underlying hypothesis in this experiment is that utilizing readings from the sub-optimal budget types would actually improve the anytime performance, lowering the validation error received for at given timestamp.

Consider the Cifar 10 based HPO task, described in section 5.1.1. Figure 5.4 indicates the performance of MultiTune 3D algorithm for Cifar 10 based HPO tasks, with and without utilization of results from the miniature evaluation.

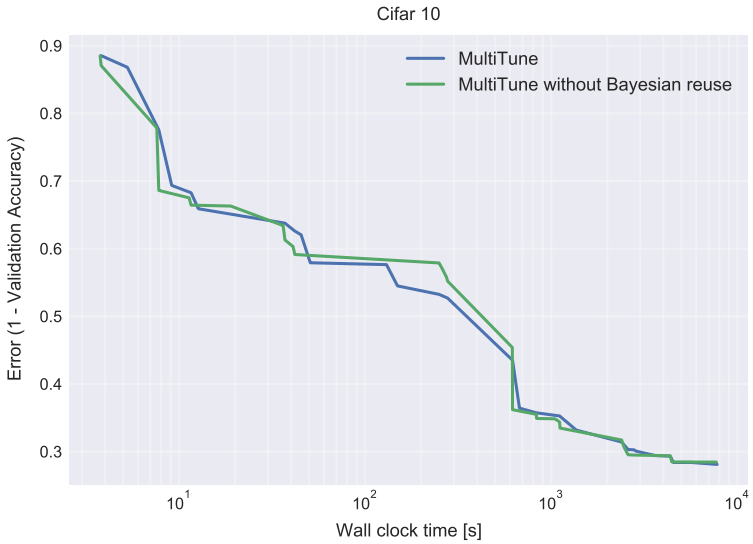


Figure 5.4: Benchmarking MultiTune 3D with and without utilization of miniature results in its Bayesian model over Cifar 10.

Consider the figure 5.4, which showcases trajectory of the validation error received with respect to wall clock time for the two algorithms under consideration: MultiTune 3D where we simply discarded readings from of sub-optimal budget type from the miniature evaluation and MultiTune 3D where we utilised readings from sub-optimal budget types to the build the Bayesian model for the unified HPO task. We see that MultiTune 3D which utilized readings form the miniature evaluations generally had lower trajectory than that without utilization of miniature results. This indicates that better anytime performance of the algorithm with reuse, resulting in lower validation for any given time as compared to the algorithm without reuse. However, the difference in the two trajectories is not substantial, indicating possibility for more research.

Conclusion
In line with our hypothesis for this experiment, we saw that the Bayesian reuse model proposed in section 4.5.2 did improve the anytime performance hyperparameter tuning tasks. However, we saw that the improvement was in fact marginal, calling for the need of enhanced Bayesian model integration.

5.6. Experiment 5

From experiment 1 through 3, we have established

- the need for dynamic budget allocation schemes which were realised in the form of MultiTune 2D
- a heuristic that allows us to determine a good budget type for a given machine learning task
- a way to avoid wastage of resources in the process of evaluating budget type

Combining these inferences, we proposed the final MultiTune algorithm in section 4.5, which explores the 3 budget types before choosing an optimal budget type and executes hyperparameter tuning. This alleviates the users from the burden to specify budget types and constraints while performing hyperparameter tuning tasks using the BOHB. Given that BOHB is the state of the art HPO technique and budget specification is one of the key challenges of BOHB, MultiTune could be instrumental in increasing adaptation of BOHB for a wide range of machine learning tasks.

	Minimum Budget	Maximum Budget	Eta	Other parameters
Fixed Epoch	1 epoch	16 epochs	2	
Fixed Trainset	0.41% training dataset	100% training dataset	3	Epochs per configuration (epc) = 2
Fixed Time	0.185 seconds	15 seconds	3	

Table 5.9: Configurations considered for experiment 1-a on Cifar 10, Fashion MNIST and MNIST HPO tasks.

So, in this experiment, we wish to gauge the performance of MultiTune and study it on each of HPO tasks discussed in section 5.1. In context of MultiTune, we hypothesize that while there could be a delay in getting the first readings for hyperparameter tuning owing to the miniature evaluations, the algorithm should consistently converge to a good final performance. In the experiment, we will benchmark MultiTune with three different fixed horizon BOHB configurations of three different budget types. The configurations have been summarised in table 5.9. We break down this experiment into two parts, each handling a unique genre of machine learning tasks.

5.6.1. Experiment 5 - a

This sub-experiment answers the following question.

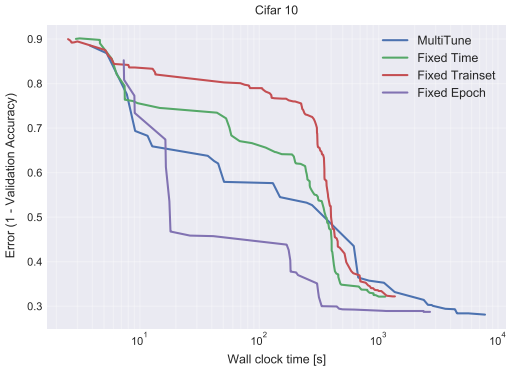
> **Does MultiTune consistently perform better than mis-specified vanilla BOHB for hyperparameter tuning tasks on image based datasets?**

Let us begin this experiment by considering Cifar 10 based HPO task. Figure 5.5 showcases the hyperparameter optimization performance of the MultiTune algorithm with the fixed horizon BOHB optimizers described in table 5.9.

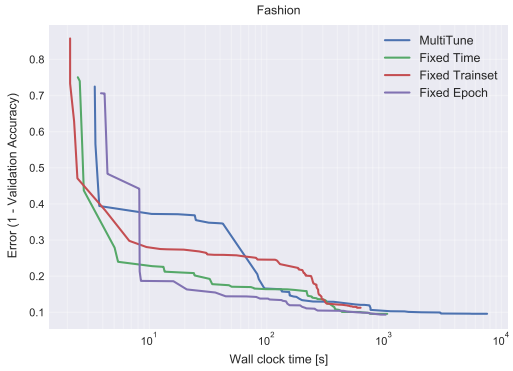
We see that the fixed epoch out-performed every other optimizer. However, that could be attributed to the fact that epoch is the preferred budget type for Cifar 10 HPO tasks as inferred from prior experiments. On the other hand, MultiTune outperformed fixed time as well as fixed trainset based optimizers. MultiTune yielded better any time performance, while being able to converge to slightly better final performance than every other optimizer considered. Even though we observe close to 2000 seconds difference in convergence time of Fixed Epoch and MultiTune, we argue that the additional compute is an investment for not having to specify budget types or budget constraints, and still reaching to a good final performance.

Similarly, figure 5.5b represents the optimization performance of MultiTune compared with the three fixed horizon configurations on Fashion MNIST based HPO task. Unlike Cifar 10, the differences here

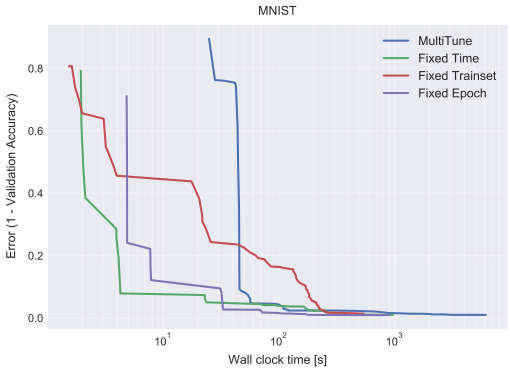
are much less pronounced. The properties observed on Cifar 10 persists in Fashion dataset as well, in form of MultiTune obtaining better any-time as compared to trainset based fixed horizon optimizer as well as MultiTune eventually converging to the best final performance.



(a) CIFAR 10



(b) Fashion MNIST



(c) MNIST

Figure 5.5: MultiTune benchmarks with different fixed-budget optimizers for hyperparameter tuning on CIFAR 10, Fashion MNIST and MNIST datasets.

Finally, figure 5.5c benchmarks the optimization performance of MultiTune on MNIST based HPO task. Here, after a delay of about 40 seconds, we see that MultiTune drops almost with a perpendicular slope to a good final performance. With this, even though MultiTune was slower than time based and epoch based optimizers, the delay could be attributed to evaluation period for the miniature algorithm. Persistent with the other two image based datasets, we do observe that MultiTune eventually converged to the best final performance. Owing to simplicity of the underlying dataset, the rest of the optimizers were able to converge to similar final performance, albeit minor differences in the time taken for convergence.

Conclusion

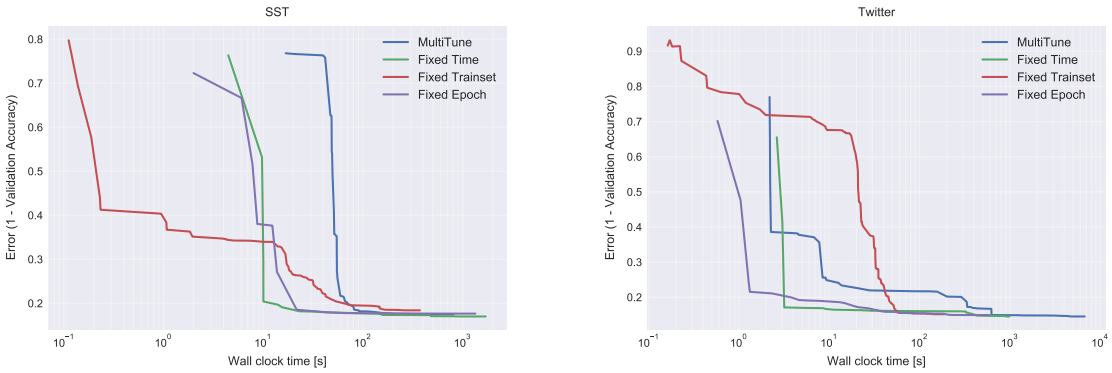
Overall, our hypothesis with regard to MultiTune persisted, in that BOHB enabled with MultiTune algorithm consistently converged to the best final performance on all image based datasets. On datasets such as MNIST and Fashion MNIST, which are relatively cheap to evaluate given the small image size and a single channel image, we observed extremely steep drop in the convergence of the MultiTune trajectory, albeit after a slight delay. In the duration that it took MultiTune to evaluate each of the budget types, the optimizers of the pre-set budget type already performed numerous calculations, making it seem like better any-time performance. However, the delay is minimal, and in the scheme of machine learning applications where training duration extends up to weeks, this delay is negligible. In machine learning applications with even slightly more complex dataset such as Cifar 10, we saw that MultiTune not only delivered the best final performance, it also delivered superior any-time performance.

5.6.2. Experiment 5 - b

This sub-experiment answers the following question.

> **Does MultiTune consistently perform better than mis-specified vanilla BOHB for hyperparameter tuning tasks on text based datasets?**

In this experiment, we continue to evaluate MultiTune on Twitter part of speech dataset and the Stanford sentiment treebank. The motivation behind this experiment is that while Falkner et al. did incorporate an exhaustive test suite for BOHB [6], the report did not show evaluation on any text based dataset. Similar to experiment 5 - a, the underlying hypothesis here is that while MultiTune might have delay in initial any-time performance, we expect MultiTune to converge the best final performance as compared to the rest of the fixed-horizon optimizers.



(a) Stanford sentiment treebank

(b) Twitter

Figure 5.6: MultiTune benchmarks with different fixed-budget optimizers for hyperparameter tuning on text based datasets.

Figure 5.6 confirms our hypothesis of the relative performance of MultiTune compared to the rest of the fixed horizon optimizers from table 5.9. Specifically, in figures 5.6a and 5.6b, we can observe that MultiTune converged to the optimal final performance, and rendered superior performance as compared to mis-specified budget parameters in different budget types. In line with our conclusion from experiment 5 - a, we also see initial delay in the performance observed for MultiTune, owing to miniature algorithm evaluations. We also observe near vertical drop in the validation error after the initial delay, establishing the superior performance of MultiTune based budget allocation scheme.

Conclusion

Similar to image based datasets, we observed that MultiTune consistently converged to a good final performance. Similarly, the delay observed in the initial part can be attributed to evaluation of the miniature algorithm, in the course of which other preset optimizers progressed through computation.

6

Conclusion

We begin this work by empirically evaluating the state of the art hyperparameter tuning technique, BOHB (Bayesian Optimization and HyperBand). We found that mis-specification of budgets led to sub-optimal performance. For example, setting a high minimum-budget worsened the anytime performance of BOHB, owing to the large wastage of computation that is to be spent on every sub-optimal hyperparameter evaluated. Further more, high minimum-budget also required setting a high maximum-budget to maintain a diversity in fidelities. In machine learning models which had low requirement for convergence, this entailed a significant wastage of resources. A more critical trend was observed on the other end of the spectrum, wherein if the maximum-budget was not sufficiently large, we saw that the algorithm could not converge to a good final performance.

We also explored three different means to create fidelities in BOHB: using epochs or training iterations across the model, using subsets of training data for tuning the model, and using time duration as a budget training the model. We found that different machine learning tasks exhibited unique trajectory trend for each budget type, and no one budget type was globally optimum.

We then approached both of these problems in a modular fashion: we began by defining infinite horizon budget schemes for each budget type, followed by defining an algorithm to dynamically choose an optimum budget type. Efficacy of both of these modules were validated in 3 different machine learning tasks. The infinite horizon budget schemes were benchmarked with an optimally configured vanilla BOHB for each budget type. We found that the algorithms consistently converged to a good final performance, at the cost of a lowered anytime performance. But we justified this hindrance, by ensuring lower and faster convergence than a sub-optimally configured BOHB. Furthermore, we found that initial performance of a budget type for infinite horizon BOHB yielded a sufficiently unique heuristic allowing us choose the optimal budget type without exhaustive evaluation.

While integrating both these components into MultiTune, we also introduced means to reuse readings made during exploration of different budget types into the Bayesian model of the optimal budget type (derived post evaluation of each budget types). While we did observe marginal improvement in any-time performance of the optimization trajectories, the performance gain was minimal. This could be attributed to the fact that these evaluations were from extremely cheap fidelities, and did not carry significant information of the underlying objective function.

Finally, we benchmarked MultiTune with optimally and sub-optimally configured vanilla BOHB optimizers. We found that MultiTune consistently converged to the final performance similar to or better than BOHB with the optimal configuration, albeit with not-as-superior anytime performance. On the other hand, we observed that MultiTune had comparable or better any-time performance than sub-optimal budget types. In line with the no free lunch theorem, which states that no optimization technique is globally optimum, we find that MultiTune performs sub-optimally in cases wherein the inherent amount of computation required for training a model was less. In these cases, even a poorly specified BOHB was able to converge to a good performance without significant overhead, and MultiTune

seemed like an overkill.

So to conclude, in this thesis, we attempted to alleviate the challenge of budget specification for multi-fidelity hyperparameter optimization optimizers like BOHB. Our empirical analysis shows promising results, enabling BOHB to consistently converge to a good final performance.

6.1. Limitations and Future work

Some of the most critical avenues of research for extending this work have been listed below.

Better sampling algorithms for drawing configurations
The number of configurations to be drawn at different phases of optimization in BOHB is driven by Hyperband. In a draw, configuration for each hyperparameter is either chosen at random or from a Bayesian model, based on the BOHB algorithm. However, we observed that same configurations were being sampled multiple times during a single successive halving Bracket. While this might be a Bayesian model driven decision, we argue that it is sub-optimal. It would be more advantageous to have a wider range of configurations to evaluate rather than having multiple instances of the same configuration, since apart from inherent randomness, we would expect consistent performance for a given configuration. Evaluating different configurations would be more valuable, since it would entail evaluating a wider search space, rather than evaluating one configuration multiple times. We propose an add-on to the sampler algorithm, to keep the configuration pool for a successive halving bracket diverse.

Evaluation of parallel scalability
Parallel scalability is extremely important for tuning hyperparameters on machine learning problems with high computational requirements, since it can significantly reduce the time required for computation. BOHB scales almost linearly with increasing parallel resources [6]. We hypothesize that MultiTune should also be able to attain similar scalability out of the box, since the underlying heuristics that enable scalability in BOHB persists in MultiTune as well. However, we could not invest time during this thesis to gauge the performance of MultiTune on parallel scalability.

Exploration of more methods to define fidelities
Approximation of the machine learning model into multiple fidelities form a critical component of speeding up the hyperparameter tuning process. In our research, we were able to define fidelities in granular fashion with 2 dimension exploration for each of the primary budget types with their respective auxiliary budget type. For example, epoch as a primary budget was enabled granular control by varying the amount of training dataset available. However, we only performed explorations in 2 dimensions, with variations in the primary budget type based on Hyperband, and variations on auxiliary budget based on gradient of gain. Through our empirical analysis, we hypothesize that improving the definition of fidelities could in fact enhance the anytime performance of MultiTune, resulting in better validation accuracy at any given time. The enhancement could be in the form of exploration of more dimensions for definition of fidelities, exploring more budget types, transitioning from gradient based search in the auxiliary budget to a better search algorithm for the budget itself.

Bibliography

[1] Loss Landscape Visualizer. URL http://www.telesens.co/loss-landscape-viz/viewer.html.

[2] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. 2012.

[4] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. 2011.

[5] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010.

[6] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. 2018.

[7] Matthias Feurer and Frank Hutter. Hyperparameter optimization. 2018.

[8] Dan Hendrycks, Mantas Mazeika, Duncan Wilson, and Kevin Gimpel. Using trusted data to train deep networks on labels corrupted by severe noise. *Advances in Neural Information Processing Systems*, 2018.

[9] Frank Hutter and Joaquin Vanschoren. Frank hutter and joaquin vanschoren: Automatic machine learning (neurips 2018 tutorial), 2018.

[10] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. Beyond manual tuning of hyperparameters. 07 2015.

[11] Kevin Jamieson and Ameet Talwalkar. Non-stochastic Best Arm Identification and Hyperparameter Optimization. 2015.

[12] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. Multi-fidelity Bayesian optimisation with continuous approximations. 2017.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2015.

[14] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. 2016.

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[16] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. 1998.

[17] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. 2017.

[18] Matthias Poloczek, Jialei Wang, and Peter Frazier. Multi-information source optimization. 2017.

[19] Herbert Robbins and Sutton Monro. A stochastic approximation method. 1951.

[20] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. 2013.

[21] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization, 2014.

[22] Jiazhuo Wang, Jason Xu, and Xuejun Wang. Combination of hyperband and bayesian optimization for hyperparameter optimization in deep learning. 2018.

[23] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. 1997.

[24] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.