An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms

Amala Mary Vincent (amalamaryvincent@gmail.com)
National Institute of Technology Karnataka
Jidesh P.
National Institute of Technology Karnataka

Article

Keywords:

Posted Date: July 20th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1781731/v1

An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms

Amala Mary Vincent[1,*] **and P. Jidesh**[2]

[1]Department of Mathematical and Computational Sciences, National Institute of Technology Karnataka, Mangalore-575025, India
[*]amalamaryvincent@gmail.com

ABSTRACT

Machine learning strategy has changed the face of automated models by integrating themselves into many application domains. The spectrum of applications ranges over various domains, from atmospheric analysis to medical diagnosis. All these applications are design-sensitive, implying that the model's performance depends highly on the selected machine learning algorithm, training procedures, regularization methods, and most importantly, how the hyperparameters are tuned. With the advent of AutoML systems, all these processes including hyperparameter optimization can be automated, producing better performance and faster results. The ideal hyperparameter setting for machine learning models has a direct and significant impact on the model's performance. This paper studies different AutoML models and the hyperparameter techniques used by them for image classification problems. We also discuss some of the libraries for hyperparameter optimization and analyse how they work for different image classification problems. Moreover, the paper proposes a framework for hyperparameter optimization that combines Bayesian optimization and evolutionary algorithms. Experiments are carried out on image classification benchmark datasets to assess the performance of different optimization approaches including the proposed model.

Introduction

In the past decade, Machine Learning (ML) services have been integrated into a wide range of application domains including weather prediction, speech recognition, computer vision and medical diagnosis. Different ML models are appropriate for different applications. In order to find the most suitable algorithm, the simple method of applying and optimizing all known learning algorithms is not practical in most cases. This process of finding the best ML algorithm and creating the optimal architecture by setting the best hyperparameters is a complex and time-consuming process. Here comes the importance of an Automated Machine Learning (AutoML) system that determines the optimal configurations for a particular application with the best performance within the time constraints. For a deep learning network, AutoML not only performs Hyperparameter Optimization (HPO) to automatically set the optimal hyperparameters but also selects the right neural architecture for each layer. The following paragraphs briefly explain the structure of HPO and AutoML.

Every machine learning model has two types of parameters, a set that is trained by the model and another that controls the learning process. The former set of parameters are determined using the training dataset and are called the model parameters. The latter are values that can be tuned and adjusted by the user before running the model. They have a major role in determining the performance of the model and are called hyperparameters. The weights of a neural network are model parameters that are derived and fitted by training, whereas the learning rate of a neural network, the regularization parameter, and the kernel parameter are all examples of hyperparameters. Different machine learning algorithms require various sets of hyperparameters. Other than a few simple models like least square regression, most machine learning models have hyperparameters.

Different hyperparameter configurations are required for various datasets. So, we need to find a set that performs optimally for all kinds of datasets, for a given training algorithm. This is done by tuning the hyperparameters and the technique is called HPO. Once we have the hyperparameters, the algorithm learns the model parameters from the data.

The idea of AutoML is to automate the end-to-end process of machine learning involving automating 4 phases - data preparation, feature processing, model generation and estimation. The user only needs to submit data, and the AutoML system will automatically decide which strategy is optimal for a certain application. The primary goal of AutoML systems is to optimise the performance by automatically setting the best hyperparameters, i.e., automating the hyperparameter optimization part. In deep neural networks, the wide range of hyperparameters to choose from for the neural network's architecture and optimization is very much critical[1]. This is where the importance of automated HPO comes into play. It also has several other use cases: it decreases the amount of human labour required to apply machine learning, increases the effectiveness of machine learning algorithms, and makes scientific investigations more reproducible.

Every machine learning algorithm aims to identify a function that optimizes the loss. Let M be the given machine learning algorithm with $\{\lambda_1, \lambda_2, ...\lambda_n\}$ parameters to be tuned. Each hyperparameter λ_i can have a value within the interval $[a_i, b_i]$ in a hyperparameter configuration space $\Lambda = [a_1, b_1] \times ... \times [a_n, b_n]$. H is a function that determines the performance or loss value. H maps each possible configuration $\lambda \in \Lambda$ to a numerical value $H : \Lambda \rightarrow R^*$. The objective of hyperparameter optimization is to find the best configuration λ^* that minimizes $H(\lambda)$.

$$\lambda^* = \underset{\lambda \in \Lambda}{\operatorname{argmin}} H(\lambda) \tag{1}$$

The objective function H is a black-box function, meaning the actual function is unknown, but one can observe its output based on specific given inputs. Since the function cannot be accessed, we do not have any information about its derivative. The function evaluation is pretty time-consuming, implying that each iteration takes a considerable amount of time. This can be minutes for a small-scale dataset, while it can be hours and days for larger ones. As a result, solving the optimization problem is difficult, and obtaining the ideal configuration in a few trials necessitates a specific approach.

Some of the popular HPO methods are grid search and random search, Bayesian optimization, gradient descent, evolution-based techniques and multi-fidelity methods like hyperband and successive halving.

A comprehensive analysis of HPO and AutoML models

The first and basic approach proposed for performing HPO was grid search. Grid search performs an exhaustive search through the Cartesian product of manually specified, finite sets of hyperparameters. That is, the search space is a grid of hyperparameters and each point on the grid is assessed. Though grid search is a time-saving and resource-efficient tuning technique, it endures the problem of dimensionality. This is because the count of function evaluations increases exponentially with the hyperparameter space.

Similar to grid search, random search has the advantages of parallelism, ease of comprehension and implementation. Random search proves to be more efficient than grid search in high-dimensional spaces[2]. It selects the combination of hyperparameters from the search space randomly. Better results of random search in high-dimensional spaces are due to low intrinsic or effective dimensionality of the objective function. Low effective dimensionality suggests that the ultimate efficiency of the machine learning algorithm is influenced only by a limited number of hyperparameters. For instance, if a function of two variables, say $f(x,y)$ can be approximated by another function of one variable $g(z)$, then f has a low effective dimension. Though less efficient in low-dimensional spaces, they are highly cost-effective in high-dimensional search spaces.

Sequential Model-Based Optimization (SMBO) is a formalization of Bayesian Optimization (BO)[3–7]. The BO approach treats the black-box objective function as a random function and assumes a prior distribution over the loss function which is updated from new observations to a posterior distribution. In other words, it constructs a probabilistic model that maps the hyperparameters to a probability score, denoted as $p(x|y)$. This model is a surrogate for the expensive-to-evaluate objective function. SMBO is a sequential model that runs multiple trials one after another, finding a more promising set of hyperparameters each time. The criterion used for selecting the next best hyperparameter values is called an acquisition function, which uses the surrogate function information to obtain the next point to be evaluated. These hyperparameter values are then used to evaluate the objective function and the (probability score, hyperparameter) pairs are finally used to update the surrogate model.

Mostly used surrogate models are Gaussian Process (GP), Random Forest Regressions, and Tree Parzen Estimator (TPE) while the most preferred choice for acquisition function is Expected Improvement. Unlike grid search and random search, SMBO keeps track of past evaluation results. Even though the method is inherently serial and difficult to parallelize, it runs faster than random search[8].

One of the major limitations of SMBO is the uncertainty regarding the choice of acquisition function. Since it is a sequential model, achieving parallelization is almost futile. Another issue is that the expense for different data varies significantly as the function evaluation step is a laborious procedure.

Another popular optimization technique is gradient-based[9]. It is an iterative algorithm that identifies the local optimum of a differentiable function. In each step of the iteration, the optimizer either modifies the gradient or the learning rate or both. There are three variants of gradient descent depending on the quantity of data used for computing the objective function's gradient. They are batch gradient descent, stochastic gradient descent and mini-batch gradient descent. The mini-batch gradient descent incorporates the advantages of both batch gradient and stochastic gradient descent variants and is the most popular among the three. Various challenging factors like the choice of proper learning rate and learning rate schedules led to the discovery of new optimizer algorithms as an extension to the vanilla mini-batch gradient descent. The ones that are most commonly used are Momentum, RMSprop, Adam, Nadam, AMSGrad etc.[10,11].

Further, there is another class of algorithms that are population-based, nature-inspired metaheuristic approaches. The term metaheuristics come from two words, meta meaning "beyond" and heuristics meaning "to find". Metaheuristics are an

algorithmic framework that efficiently modifies domain-specific knowledge into heuristics to produce better solutions. This modification generally involves two procedures: exploration (diversification) and exploitation (intensification) and is done with the help of heuristic operators. These heuristic operators are usually inspired by nature and are different in different evolutionary algorithms. Diversification aims to cover as much of the search area as feasible, whereas intensification aims to fully exploit the search space in order to arrive at better solutions as rapidly as possible.

Like all other optimization algorithms, this family of algorithms also try to optimize the objective function in an iterative fashion, by updating the parameter (solution) values. This can be represented by the equation, $y_i = y_{i-1} + \Delta y_{i-1}$ where i indicates the number of iterations, then y_{i-1} is the solution vector in the previous iteration, y_i is the new vector of solutions, and Δy_{i-1} signifies the change in the solution vector after one iteration. Based on how Δy_{i-1} is determined, there are two general categories of algorithms: "Evolutionary Algorithms (EA)" and "Swarm Intelligence (SI)"[12,13]. The former includes algorithms like Genetic Algorithms (GA), Genetic Programming, Evolutionary Strategies (ES), Evolutionary Programming etc. that are based on biological evolution, with selection, crossover (recombination), and mutation phases, while the latter includes Particle Swarm Optimization and Ant Colony Optimization, and are influenced by the social conduct of natural organisms like birds, fish and ants[14,15].

GA are a class of evolutionary algorithms, hence both iterative and population-based[16]. On each iteration, they work with several solutions collectively called a population rather than a single solution. A GA initialises a random population as the solution and updates this solution with the help of three operators, namely, selection, crossover and mutation (variation operators). This is continued until the stopping criteria is met. A single iteration is designated as one generation of GA. The term genetic algorithm comes from the similarity of the representation of solutions to chromosomes and that of GA operators to genetic operators.

Particle Swarm Optimization (PSO) is an algorithm inspired by the behaviour of fish and birds moving in a group. They are closely similar to GA as they initialize a random population as a solution and iteratively update generations to find the optimal solution. But the difference is regarding how the update is made. They do not use the evolution operators; instead, they use information like the position and velocity of the candidate solutions (here particles). Over the iterations, each particle moves faster towards its best-known position as an impact of exploitation and towards the best-known solution of the population as an effect of exploration. Hence results in the swarm moving toward the best optimal solution[17,18].

Ant Colony Optimization (ACO) is another category of swarm intelligence algorithms which is influenced by the nature of ant colonies searching for food. Ants communicate with each other with the help of an organic compound called pheromone. They lay pheromone trails in different proportions while travelling. An ant isolated from others tends to move in any random direction. When it comes in contact with a pheromone trail, it chooses that path with high probability and follows it, strengthening the trail with its pheromone. Thus making that path more prominent for other ants to follow. This way, when an ant finds the shortest path to food, more ants tend to follow this path and find the food destination due to the presence of a larger concentration of pheromone along that path[19,20].

Grid search, random search and population-based methods like the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) are the common model-free paradigms used for hyperparameter tuning in AutoML systems. Apart from these model-free black-box optimization techniques, Bayesian optimization, HPO in AutoML focuses on multi-fidelity methods that are cheaper compared to the pure black-box methods. This includes an early stopping algorithm, a simple selection algorithm and other adaptive choices of fidelity.

A major hurdle in the optimization procedure is the exhaustive runtime of the algorithms which increases with the amount of dataset and number of hyperparameters. Multi-fidelity methods focus on finding a solution to the time and resource constraints in HPO. Reducing the data set for training and bringing down the number of features, iterations, and cross-validation folds are a few manual approaches used to accelerate the tuning process. These ideas are used by low fidelity methods to find an approximation of the actual objective function to minimise[21]. The approximation often results in a tradeoff between the performance and execution of the algorithm. But in most cases, the reduced time complexity compensates for the approximation error. Also, these methods get rid of the poorly performing configurations after each iteration and only the well-performing parameter settings are assessed over the entire training set.

Learning curve-based prediction is one such method that gets rid of configurations that are anticipated to perform badly[22]. It's an early stopping algorithm that examines the learning curve during HPO and stops the training operation for a certain hyperparameter setting if the curve isn't expected to meet the performance level of the best model produced up until that point in the optimization process. Implementation of these models combined with the Bayesian optimization technique called the Freeze-Thaw Bayesian optimization is mentioned in Swersky et. al.[23]. The algorithm maintains a set of frozen configurations and uses an information-theoretic decision framework to either thaw (defreeze) a setting chosen by the Bayesian optimization and continue training or train a new configuration to find the best hyperparameter settings.

Selection algorithms like Successive Halving (SH) and Hyperbands are bandit-based strategies that are pure-exploration focused resource allocation problems. Pure exploration problems are also called best-arm identification problems. The goal is

to choose the best arm (here, the best settings with the maximum expected performance) with maximum confidence. Successive halving is one such non-stochastic best-arm identification problem proposed by Jamieson et. al.[24]. The algorithm can be summarized in 3 steps. 1) At first, it uniformly allocates resources to each set of hyperparameter configurations. 2) And then assess how they perform. 3) Lastly, the algorithm removes half of the worst-performing group. The process is iterated till a single configuration remains. Through the sequential elimination step, the algorithm guarantees more resources for more reliable configurations[25].

Hyperband is an extension of the successive halving algorithm put forward by Li et. al.[26]. In successive halving, the fixed budget of resources is uniformly distributed to all configurations initially. If there are n configurations and a total budget B of resources, then SH allocates B/n resources for each setting. Whereas hyperband takes into consideration the fact that a large number of the configurations (large n) will require only a small amount of resource (small B/n) or vice versa. This is called the n versus B/n issue and hyperband addresses this by letting different possible values of n for a fixed B and allocating a minimum resource r for all the configurations before the elimination step. And calls successive halving techniques on random samples of configurations as a subroutine[27].

The multi-task Bayesian optimization technique is an adaptive fidelity technique that learns from previously trained models or from a trained subset of a large dataset[28]. They use multi-task Gaussian process models for fastening the Bayesian optimization by studying the correlation among tasks. Other adaptive choices of fidelity include algorithms like Bayesian Optimization with Hyperband (BOHB)[29], Multi-Fidelity Gaussian Process Upper Confidence Bound (MFGP-UCB)[30] and Bayesian optimization with Continuous Approximations (BOCA)[31] that employs an Upper Confidence Bound (UCB) acquisition function to aid the optimization process.

In recent years, ML has had a great deal of success, and it's now being employed in an increasing number of sectors. However, this success is reliant on human machine learning experts. Cleaning and pre-processing the data, selecting and building relevant attributes, selecting a suitable model family, optimising model hyperparameters, designing neural network topologies (if deep learning is used), and postprocessing machine learning models, as well as skeptically examining the results, all of this requires human expertise. Because the intricacy of these activities is mostly beyond the capabilities of non-ML professionals, the growing prevalence of machine learning applications has necessitated the need for off-the-shelf machine learning technologies that can be deployed rapidly and without specialised expertise. AutoML provides tools and approaches for enabling ML to be accessible to non-experts, increasing performance, and speeding up ML research. Some of the popular AutoML frameworks and the optimizers they use are listed below:

1. ML models-based frameworks

   (a) Auto-WEKA
   Auto-WEKA is a fully automated Automl system that includes feature selection. It uses a Bayesian optimization framework for HPO[32].

   (b) Auto-sklearn
   Auto-sklearn has feature processing and pre-processing units. It considers 15 classifiers and 110 hyperparameters and considers previous performances on similar datasets. Auto-sklearn can also construct ensembles models using a TPE-based BO for HPO[33].

   (c) Tree-based Pipeline Optimization Tool (TPOT)
   It is a genetic programming-based AutoML. TPOT optimizes a sequence of feature preprocessors and machine learning models to enhance the classification accuracy by making use of GA for hyperparameter tuning[34].

2. DL models-based frameworks

   (a) AutoKeras
   AutoKeras focuses on the deep learning tasks and uses BO to guide neural architecture search. It also employs a neural network kernel and a tree-structured acquisition function optimization technique to effectively evaluate the search space[35].

   (b) Auto-PyTorch
   Auto-PyTorch is a NAS library based on the same principles as AutoKeras, but with a PyTorch backend. It employs BO combined with hyperband for HPO[36].

   (c) AutoGluon
   AutoGluon uses multi-layer ensemble models and performs complex data processing and deep learning[37].

(d) H2O AutoML
Make use of one layer of ensemble stacking combined with bagging and a strong base model - XGBoost tree ensemble[38]. H2O uses random search for hyperparameter tuning.

3. Hyperparameter tuners
Packages that use Bayesian optimization include SMAC[39], Spearmint[40], Hyperopt[41], Scikit-optimize, BoTorch etc. Packages like DEHB[42], DEAP[43] and Nevergrad make use of evolutionary algorithms, whereas Optuna, Orion and RayTune implement both Bayesian optimization and evolutionary algorithms.

Motivation for a new model

Hyperparameter optimization is an important and ubiquitous problem in machine learning that can drastically affect the performance of a model. Despite multi-fidelity optimization's popularity and success, there are machine learning challenges that, due to their scale, have not been directly addressed by existing HPOs and may require unique methodologies. Because it is so expensive to train even a small neural network on massive datasets, no work on HPO for deep neural networks on such datasets has been done yet. Therefore, it is beneficial to study and analyse prevailing techniques to determine effective ways to improve them and to discover novel approaches that outperform existing ones. This work on HPO techniques can improve the efficiency of the existing machine learning and deep-learning frameworks.

Proposed method

The proposed model combines evolutionary algorithms with Bayesian optimization for HPO. The working of Bayesian Optimization and details of each evolutionary algorithm used in the proposed study are described briefly in the following paragraphs.

Bayesian Optimization is an optimization technique that uses a probabilistic method based on the Bayes theorem for identifying the global optimum of a black-box function. It has two major parts. The first component is a surrogate model that is probabilistic and comprises a prior distribution which represents the unknown objective function. The acquisition function is the second component, and it is optimised for selecting the next point to sample.

A surrogate model is the probability representation of the objective function. It generates a posterior probability distribution using the Bayes rule, which represents possible $H(\lambda)$ values in a candidate configuration λ. Whenever H is observed at a new λ, this posterior distribution is updated. The Gaussian Process (GP), which gives a mean function $\mu : \Lambda \to R$ and a definite positive covariance function $k(\lambda, \lambda') : \Lambda^2 \to R$, also known as the kernel, is the most commonly employed surrogate model. The equation for GP is given below:

$$GP(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

The covariance function c or the Gaussian RBF kernel (also known as the squared exponential or exponentiated quadratic kernel) is defined as:

$$k(x, x_i) = \sigma^2 e^{-\frac{1}{2l^2}||x-x_i||^2} \tag{3}$$

where l is the length scale. x and x_i are l distance apart. The bigger the kernel function's value, the nearer two input space points are.

The BO algorithm begins with a collection of k configurations$\{\lambda_i\}_{i=1}^k$ and their corresponding function values $\{y_i\}_{i=1}^k$ where $y_i = H(\lambda_i)$. We use the Bayes rule to update the GP model at each iteration, resulting in a posterior distribution that is dependent on the present training set. This set $S_t = \{\lambda, y_i\}_{i=1}^t$ contains the previously evaluated configurations and observations. The posterior mean $\mu_t(\lambda)$ and posterior variance $\sigma_t^2(\lambda)$ of the GP, dependent on S_k, are given by:

$$\mu_t(\lambda) = c(\lambda)^T [C + \tau^2 I]^{-1} y \tag{4}$$

$$\sigma_t^2(\lambda) = k(\lambda, \lambda) - c(\lambda)^T [C + \tau^2 I]^{-1} c(\lambda) \tag{5}$$

where C is the $t \times t$ matrix whose entries are $C_{i,j} = k(\lambda_i, \lambda_j)$, $c(\lambda)$ is a $t \times 1$ vector of covariance terms between λ and $\{\lambda_i\}_{i=1}^k$, y is the $t \times 1$ vector whose ith entry is y_i and τ^2 is the noise variance.

At each iteration, we choose and evaluate a new point λ_{t+1} that gives $y_{t+1} = H(\lambda_{t+1})$. And we add this point to the existing training set S_t to obtain a new training set $S_{t+1} = S_t \cup \{\lambda_{t+1}, y_{t+1}\}$.

We use the following equation to determine the next query point λ_{t+1}:

$$\lambda_{t+1} = \underset{\lambda \in \Lambda}{\operatorname{argmax}}\ U_t(\lambda; S_t) \tag{6}$$

where U_t is the acquisition function to maximize. The generally employed acquisition function is called expected improvement and is defined as:

$$U_t(\lambda; S_t) = (f^* - \mu_t(\lambda)) \cdot \Phi(f^*; \mu_t(\lambda); \sigma_t(\lambda)) + \sigma_t(\lambda) \cdot N(f^*; \mu_t(\lambda); \sigma_t(\lambda)) \tag{7}$$

where N and Φ are the normal distribution function and the normal cumulative distribution function, respectively. Expected improvement comprises two parts: the first part of the sum can be improved by lowering the mean, while the other part can be improved by raising the variance. These phrases strike a compromise between exploitation and exploration. The illustration of BO is shown in figure 1.

In this paper, we use evolutionary algorithms like Differential Evolution, Genetic Algorithm and Evolutionary Strategy for maximizing U_t. The foundation of every evolutionary algorithm is biological evolution, which is natural selection and continual blending of variation via recombination and mutation. New individuals (candidate solutions) are produced in each iteration (generation) by variation, typically in a stochastic way, from the existing parental individuals. Then, based on their fitness score, some individuals are chosen to become the parents of the following generation. After each generation, individuals with greater and better fitness values are generated in this manner.

A GA is a search strategy that is based on Charles Darwin's idea of natural selection. Initialising the population, Calculating the fitness function, Selection, Crossover and Mutation are the five phases in GA. The process begins with a group of people known as the population. Each individual is denoted by a finite-length vector of components, similar to a chromosome and represents a solution in search space for a given problem. Genes are equivalent to these variable components. Thus, each chromosome (individual) is made up of multiple genes (variable components). The fitness function determines an individual's ability to compete against others. It specifies each individual a score. This fitness score determines the likelihood of an individual being selected for the next generation.

The goal of the selection phase is to find the fittest individuals and enable them to carry on their traits to the coming generation. This is done by calculating the fitness score for each individual and selecting the highest scored pair. Physically fit individuals have a higher possibility of being selected for generating offspring. The most crucial part of the genetic algorithm is the crossover phase. A crossover site within the DNA is picked at random for each pair of parents to be mated. The genes at this point are then swapped, resulting in the creation of a totally new individual called the offspring. These new children are then included in the population. Some genes in new offspring are vulnerable to mutation, meaning that some of the bits in the DNA can be swapped. The benefits of mutation include maintaining population diversity and avoiding premature population convergence. If the population has converged, it will not generate children who are distinct from the previous generation, causing the algorithm to cease.

DE is an evolutionary algorithm proposed by Storn et. al.[44]. DE makes minimal, if any, assumptions about the underlying optimization issue and can rapidly explore enormous design spaces. Unlike standard evolutionary algorithms, it can deal with multi-dimensional real-parameter optimization problems.

DE is also a population-based stochastic approach. Like other evolutionary algorithms, genome/chromosome is the term given to each solution. Each chromosome goes through mutation followed by recombination. Unlike the genetic algorithm that represents candidate solutions using sequences of bits, DE keeps a population of candidate solutions in the form of real-valued vectors, also called target vectors. The target vector is made of a certain number of decision variables. In each loop of the algorithm, a new donor vector is generated after mutation and a trial vector formed out of recombination. Once the trial vectors have been generated, the best solution is selected by a greedy approach conducted among all target and trial vectors.

During mutation, DE creates new vectors called donor vectors by multiplying a third vector to the weighted difference between two population vectors. Donor vector (V) of a chromosome X_i is created as:

$$V = X_{\lambda_1} + F(X_{\lambda_2} - X_{\lambda_3}) \tag{8}$$

where F is the scaling factor (0, 1 or 2), $\lambda_1, \lambda_2, \lambda_3$ are random solutions $\in \{1, 2, 3, ..., N_p\}$ and $\lambda_1 \neq \lambda_2 \neq \lambda_3 \neq i$. Four vectors are involved in the generation of the target vector via mutation. Therefore the size of the population N_p should be greater than or equal to 4. The target vector is not involved in mutation.

Recombination is performed by binomial crossover. This step increases the diversity in the population. The trial vector is created out of recombination as follows:

$$u^j = \begin{cases} v^j & \text{if } r \leq p_c \text{ OR } j = \delta \\ x^j & \text{if } r > p_c \text{ AND } j \neq \delta \end{cases} \tag{9}$$

where p_c is the crossover probability, r is a random number (0 or 1), δ is a randomly selected variable location $\{1,2,...D\}$ (D is the number of decision variables in target vector), u^j is the jth variable of trial vector, v^j is the jth variable of donor vector and x^j is the j th variable of target vector. δ assures that at least one variable from the donor vector is selected. The value of p_c is generally set high, indicating more crossover, i.e., more variables from the donor.

During selection, a fitness function f_U for each offspring is evaluated. Population is updated using greedy selection:

$$\left.\begin{array}{l} X_i = U_i \\ f_i = f_{U_i} \end{array}\right\} \quad \text{if } f_{U_i} < f_i \tag{10}$$

X and f remain the same if $f_{U_i} > f_i$. The selection procedure is carried out only after all solutions generate offspring.

Another algorithm we use is a variant of ES. ES is a kind of evolutionary algorithm that is based on the scientific idea of natural selection in evolution. It does not use crossover like other evolutionary algorithms; instead, candidate solution modification is limited to mutation operators. The algorithm uses a population of potential solutions that are created at random. Each iteration of the method begins with an evaluation of the population of solutions, followed by truncation selection, which entails removing all but a subset of the best solutions. The remaining solutions (parents) are each utilised to generate a set of new candidate solutions (mutation) that replace or compete with the parents for a place

This approach has several variations. The population size is measured in λ, while the number of parents chosen per iteration is measured in μ. One of the variations is represented as (μ, λ)-ES, a version where children take the place of parents. Another variation is represented as $(\mu + \lambda)$-ES where children and parents are added to the population.

CMA-ES is the most powerful variation of ES[45]. For an initial population $P = \{x_1, x_2, ...x_\mu\}$ with μ parents and λ offsprings, the major differences of CMA-ES and classic ES are:

1. Offsprings are not generated by mutation of a single individual but by the weighted mean of the current population:

   $$\hat{x_i} = \bar{x} + \delta_i z \tag{11}$$

   where $\delta_i > 0$ for $i = 1,2,...,\mu$, z is a random vector and

   $$\bar{x} = \frac{1}{\sum_{j=1}^{\mu} w_j} \sum_{j=1}^{\mu} w_j x_j, \quad w_j > 0 \tag{12}$$

   w_j is the set of positive weight coefficients for recombination and x_j is the set of best individuals out of the current population.

2. Unlike standard ES, z is not chosen to be independent of i, but chosen such that:

   $$z \sim N(0, C) \tag{13}$$

   where C is the covariance matrix. x_i is computed by choosing another random vector $\bar{z} \sim N(0,1) = N(0,1)^n$ and then computing the singular value decomposition of C:

   $$C = U \Sigma V^T \tag{14}$$

   and setting

   $$\hat{x_i} = \bar{x} + \delta_i U \Sigma^{\frac{1}{2}} \bar{z}, \tag{15}$$

   $$\Sigma^{\frac{1}{2}} = diag\left(\left(\sigma_i^{\frac{1}{2}}\right)_{i=1}^{n}\right) \tag{16}$$

3. C is updated in each iteration using a rank-1 update:

   $$\hat{C} = (1-\alpha)C + \alpha \hat{p} \hat{p}^T, \quad \alpha \in (0,1] \tag{17}$$

   using

   $$\hat{p} = (1-\beta)p + \gamma U \Sigma^{\frac{1}{2}} \left(\bar{\hat{x}} - \bar{x}\right) \tag{18}$$

   The parameters α, β, γ are selected approximately.

4. Step size control is integrated into CMA-ES. That is, δ in equation 11 is chosen in a way that prevents the population from converging prematurely.

The major advantage of CMA-ES over other evolutionary algorithms is that the population size can be freely chosen. There is no inherent need to use large population sizes.

Results and discussion

As part of the study, we have conducted three sets of experiments, all of which uses the same sets of data and are run on the same configuration. We use an Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz with 1 NVIDIA Tesla V100 GPU. The datasets used include CIFAR-10, MNIST, SVHN and CALTECH-101.

CIFAR-10 comprises 6000 images belonging to 10 different classes. It includes images of aeroplanes, birds, frogs, cats, deer, horses, cars, ships, dogs and trucks. MNIST consists of 60,000 grayscale square images (28×28 pixels) of handwritten single digits from 0 to 9. Street View House Number dataset contains 6,00,000 RGB images (32×32 pixels) of printed digits from 0 to 9 clipped from photographs of house number plates. CALTECH dataset has 101 object categories, with about 40 to 800 images belonging to each category (300 x 200 pixels).

Setup 1:

AutoML systems with different optimizers are evaluated based on their performance in image classification problems. AutoKeras and TPOT are used for this experiment as AutoKeras provides three different choices of hyperparameter optimization. It allows the user to select between BO, random search and hyperband for optimization. In comparison, TPOT uses a genetic algorithm for the same. The comparison of test error rates and wallclock time for experiment setup 1 are shown in figures 2 and 3.

Figure 2 represents a line graph showing the error rates obtained for the classification of CIFAR, MNIST, SVHN and CALTECH data using BO, random, hyperband and genetic algorithm for hyperparameter optimization. Overall, random search gives better results than the other three HPO techniques in terms of test error rate for all four datasets.

The green line in the graph represents genetic algorithm. It is observed that genetic algorithm has the highest error rates implying low classification accuracy for three of the datasets. The blue line depicting the performance of bayesian optimization shows that it produces a better outcome compared to GA in all four cases. Hyperband performs better than GA in most cases, but for CALTECH data, it gives an error rate of 0.7, indicating a very low accuracy. Random search, presented by the red line, has the lowest error rates for all classification problems.

Figure 3 shows datasets used on the x-axis and the time taken by AutoKeras (with BO, random search and hyperband used for HPO) and TPOT (uses GA for HPO) on the y-axis. The graph is a semi-logarithmic plot that uses a logarithmic scale on the y axis to indicate the variation in wallclock time from seconds to hours.

It is seen from the graph that hyperband takes only minutes to perform classification for all four datasets, whereas GA requires hours to days to complete the same task. Random search takes the longest time (around 1 and half hours) for classification on MNIST, while BO took more than twice the time (3 hours 18 minutes) on the same dataset. Overall, hyperband proves to be time-efficient compared to other HPO methods. TPOT is recommended only in the case where time is not considered a constraint.

Setup 2:

The second experiment carried out compared optimizer packages that use BO and EA. Same datasets are used for this analysis. The packages are used to optimize the hyperparameters of an AlexNet, a deep learning architecture with 8 layers. It consists of 5 convolution layers and a mix of max-pooling layers. There are three layers that are fully connected. Alex introduced the use of non-saturating ReLU function as activation function, which resulted in improved training accuracy over tanh function and sigmoid. Two dropout layers are used. The output layer uses a Softmax activation function.

AlexNet is a fast GPU execution of a convolutional neural network. AlexNet won the ImageNet Large Scale Visual Recognition Challenge in 2012. The main conclusion of the original research on AlexNet was that the model's depth was critical for its high performance, which was computationally costly but made possible by the use of GPUs during training[46].

The experiment considers an AlexNet model and 4 of its hyperparameters. Table 1 gives the list of hyperparameters and the range of values used for tuning. The optimizers used in the experiment are Sequential Model-based Algorithm Configuration (SMAC) and Differential Evolution Hyperband (DEHB). The main core of SMAC combines Bayesian Optimization with a mechanism to quickly determine which of the two configurations is the better performer. On the other hand, DEHB combines the advantages of the popular bandit-based HPO method hyperband and the evolutionary search approach of Differential Evolution. Figure 4 illustrates the comparison of the two HPO packages based on error rates obtained using a t-test on samples from 10 iterations with a statistical significance of p-value < 0.01. From figure 4, it is clearly noticeable that the green line representing DEHB has lower error rates for all four datasets compared to the pink depicting SMAC. DEHB with the advantages of both hyperband and differential evolution proves to be more promising than SMAC.

Setup 3:

The third analysis is executed by implementing the proposed models on the same set of data. We propose three different HPO models with a Bayesian Optimization framework that employs differential optimization, genetic algorithm and evolutionary

strategy for acquisition function maximization. The results are compared in terms of the error rate obtained using a t-test on samples from 10 iterations with a statistical significance of p-value < 0.01 and is given in figure 5.

All variants of BO proposed by the study are compared to the standard Bayesian optimization. Figure 5 shows that the overall efficiency of BO combined with CMA-ES has better classification accuracy compared to combinations of BO with other evolutionary algorithms in most cases. BO combined with DE produces slightly better results than GA. The graph also proves that the performance of standard BO can be improved by its combination with DE and CMA-ES.

Conclusion

Hyperparameters of ML models must be tuned to fit particular datasets before being deployed to practical problems. However, because the volume of created data has substantially grown in practice, and manually setting hyperparameters is tremendously resource-intensive, it has become critical to optimise hyperparameters automatically. We have seen the efficacy of AutoML systems in handling large-scale image classification datasets. We have also observed the performance of SMAC and DEHB packages for hyperparameter optimization. Out of the proposed HPO frameworks, it is examined that the efficiency of BO-CMAES proves it to be worth adopting in AutoML systems. The results from the paper can be extended to other models and datasets. The number of hyperparameters can also be scaled along with the range of values considered for tuning. The above experiments are carried out for 10 iterations. For better results, the number of iterations can be increased. As future work, we plan to conduct the experiments for real-time image classification problems.

Data availability

Data underlying the results presented in this paper are publicly available and can be downloaded from Kaggle.

References

1. Feurer, M. & Hutter, F. *Automated Machine Learning* (Springer, 2018).
2. Bergstra, J. & Bengio, Y. Random search for hyperparameter optimization. *J. Mach. Learn. Res.* **13**, 281–305, DOI: 10.5555/2188385.2188395 (2012).
3. Hutter, F., Lucke, J. & Schmidt-Thieme, L. Beyond manual tuning of hyperparameters. *KI - Kunstliche Intelligenz* **29**, 329–337, DOI: 10.1007/s13218-015-0381-0 (2015).
4. Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, DOI: 10.5555/2986459.2986743 (ACM, 2011).
5. Victoria, A. H. & Maragatham, G. Automatic tuning of hyperparameters using bayesian optimization. *Evol. Syst.* **12**, 217–223, DOI: 10.1007/s12530-020-09345-2 (2021).
6. Dewancker *et al.* Bayesian optimization primer (2015).
7. Hazan, E., Klivans, A. & Yuan, Y. Hyperparameter optimization: A spectral approach (2018).
8. Snoek, J., Larochelle, H. & Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances on Neural Information Processing Systems*, DOI: 10.5555/2999325.2999464 (2012).
9. Lemarechal, C. Cauchy and the gradient method. *Documenta Math.* 251–254 (2012).
10. Pedregosa, F. Hyperparameter optimization with approximate gradient (2016).
11. Ruder, S. An overview of gradient descent optimization algorithms (2017).
12. Blum, C. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.* **35**, 268–308, DOI: 10.1145/937503.937505 (2003).
13. Maier, H. R. *et al.* Introductory overview: Optimization using evolutionary algorithms and other metaheuristics. *Environ. Model. Softw.* **114**, 195–213, DOI: 10.1016/j.envsoft.2018.11.018 (2019).
14. Deb, K. Practical optimization using evolutionary methods.
15. Orive, D., Sorrosal, G., Borges, C. E., Martin, C. & Alonso-Vicario, A. Evolutionary algorithms for hyperparameter tuning on neural network models. In *Environmental Modelling and Software* (2014).
16. Goldberg, D. E. Genetic algorithms in search, optimization and machine learning, DOI: 10.5555/534133 (1989).
17. Kennedy, J. & Eberhart, R. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, DOI: 10.1109/ICNN.1995.488968 (IEEE, 1995).

18. Lorenzo, P. R., Nalepa, J., Kawulok, M., Ramos, L. S. & Pastor, J. R. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference*, 481–488, DOI: 10.1145/3071178.3071208 (2017).

19. Dorigo, M., Maniezzo, V. & Colorni, A. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Syst. Man, Cybern. Part B (Cybernetics)* 29–41, DOI: 10.1109/3477.484436 (1996).

20. Costa, V. O. & Rodrigues, C. R. Hierarchical ant colony for simultaneous classifier selection and hyperparameter optimization. In *IEEE Congress on Evolutionary Computation (CEC)*, DOI: 10.1109/CEC.2018.8477834 (IEEE, 2018).

21. Yang, L. & Shami, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* **415**, 295–316, DOI: 10.1016/j.neucom.2020.07.061 (2020).

22. Domhan, T., Springenberg, J. T. & Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI'15: Proceedings of the 24th International Conference on Artificial Intelligence*, 3460–3468, DOI: 10.5555/2832581.2832731 (2015).

23. Swersky, K., Snoek, J. & Adams, R. P. Freeze-thaw bayesian optimization (2014).

24. Jamieson, K. & Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2015).

25. Karnin, Z., Koren, T. & Somekh, O. Almost optimal exploration in multi-armed bandits. *Dasgupta McAllester* 1238–1246 (2013).

26. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. Hyperband a novel bandit based approach to hyperparameter optimization. *J. Mach. Learn. Res.* **18**, 1–52 (2018).

27. Dores, S. C. N. d., Soares, C. & Ruiz, D. Bandit-based automated machine learning. In *7th Brazilian Conference on Intelligent Systems (BRACIS)*, DOI: 10.1109/BRACIS.2018.00029 (2018).

28. Swersky, K., Snoek, J. & Adams, R. P. Multi-task bayesian optimization. *Adv. Neural Inf. Process. Syst.* 121–126 (2013).

29. Falkner, S., Klein, A. & Hutter, F. Bohb: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35 th International Conference on Machine Learning* (Stockholm, Sweden, PMLR 80, 2018).

30. Kandasamy, K., Dasarathy, G., Oliva, J., Schneider, J. & Póczos, B. Gaussian process bandit optimisation with multi-fidelity evaluations. In *30th Conference on Neural Information Processing Systems*, 992–1000 (2016).

31. Kandasamy, K., Dasarathy, G., Oliva, J., Schneider, J. & Póczos, B. Multi-fidelity bayesian optimisation with continuous approximations. In *Precup and Teh*, 1799–1808 (2015).

32. Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F. & Leyton-Brown, K. Auto-weka: Automatic model selection and hyperparameter optimization in weka. *Autom. Mach. Learn.* 81–95, DOI: 10.1007/978-3-030-05318-5_4 (2018).

33. Feurer, M. *et al.* Auto-sklearn: Efficient and robust automated machine learning. *Autom. Mach. Learn.* DOI: 10.1007/978-3-030-05318-5_6 (2018).

34. Olson, R. S. & Moore, J. H. Tpot: A tree-based pipeline optimization tool for automating machine learning. *Autom. Mach. Learn.* DOI: 10.1007/978-3-030-05318-5_8 (2018).

35. Jin, H., Song, Q. & Hu, X. Auto-keras: An efficient neural architecture search system. In *KDD '19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1946–1956, DOI: 10.1145/3292500.3330648 (2019).

36. Zimmer, L., Lindauer, M. & Hutter, F. Autopytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis Mach. Intell.* (2020).

37. Erickson, N. *et al.* Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505* (2020).

38. LeDell, E. & Poirier, S. H2o automl: Scalable automatic machine learning. In *7th ICML Workshop on Automated Machine Learning* (2020).

39. Hutter, F., Hoos, H. H. & Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, 507–523, DOI: 10.1007/978-3-642-25566-3_40 (Springer, 2011).

40. Snoek, J., Larochelle, H. & Adams, R. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems* (2011).

41. Komer, B., Bergstra, J. & Eliasmith, C. Hyperopt-sklearn. *Autom. Mach. Learn.* DOI: 10.1007/978-3-030-05318-5_5 (2018).

42. Awad, N., Mallik, N. & Hutter, F. Dehb: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization. In *Proceedings of IJCIA* (2021).

43. Fortin, F., De Rainville, F., Gardner, M., Parizeau, M. & Gagné, C. Deap: Evolutionary algorithms made easy. *J. Mach. Learn. Res.* (2012).

44. Storn, R. & Price, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* (1997).

45. Hansen, N. & Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.* (2001).

46. Krizhevsky, A., Sutskever, I. & E. Hinton, G. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012).

Author contributions statement

A.M.V. carried out experiments and wrote the manuscript. P.J. reviewed the manuscript and supervised the research.

Competing interests

The authors declare no competing interests.

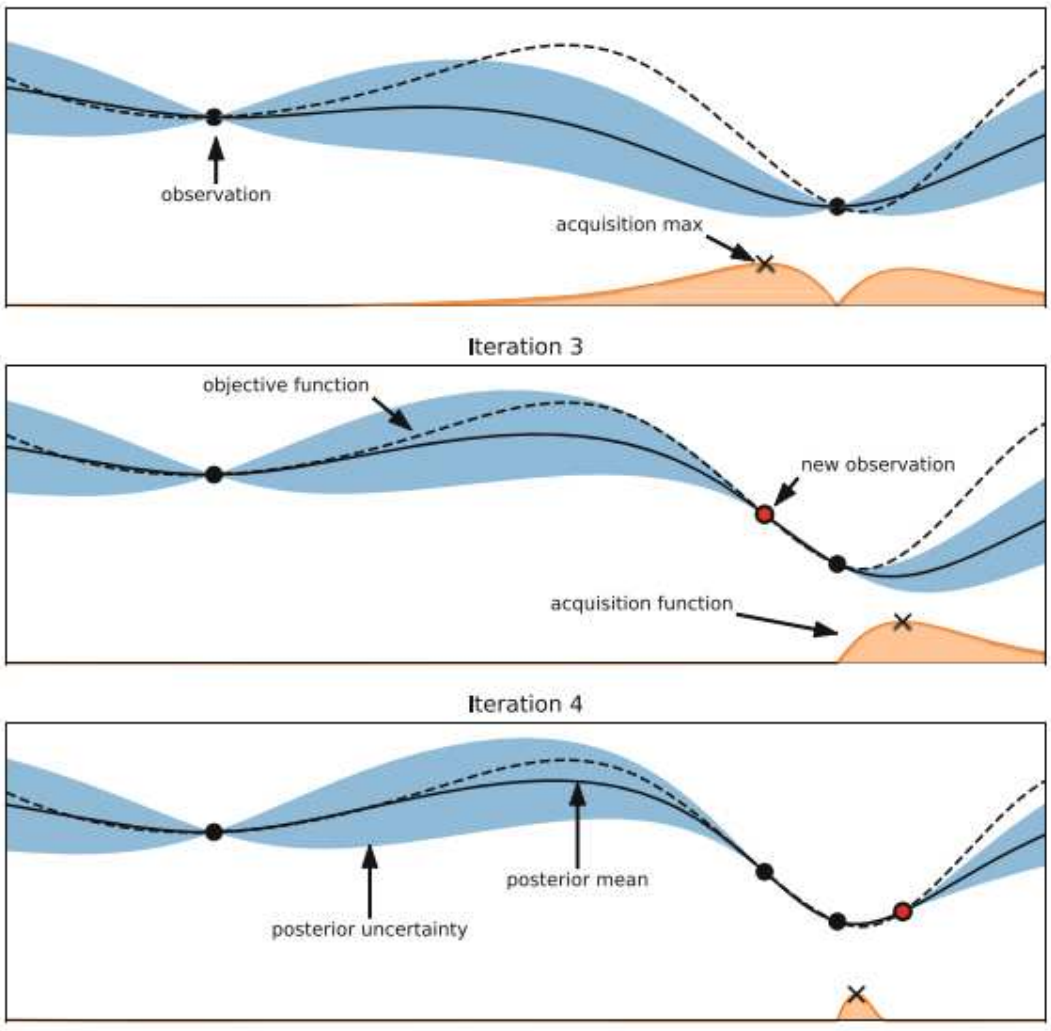

Figure 1. Illustration of Bayesian Optimization on a 1*D* function. The graph depicts a Gaussian process approximation of the objective function and an acquisition function in the lower portion. The mean of the objective function is represented by the dashed line and the blue region shows the predictive uncertainty. Acquisition function is represented by the orange curve. The acquisition function is maximum where the Gaussian process gives a low objective value with high uncertainty. *Note.* Adapted from *Automated Machine Learning*, by Feurer, M. and Hutter, F., 2018, Springer, p. 10.

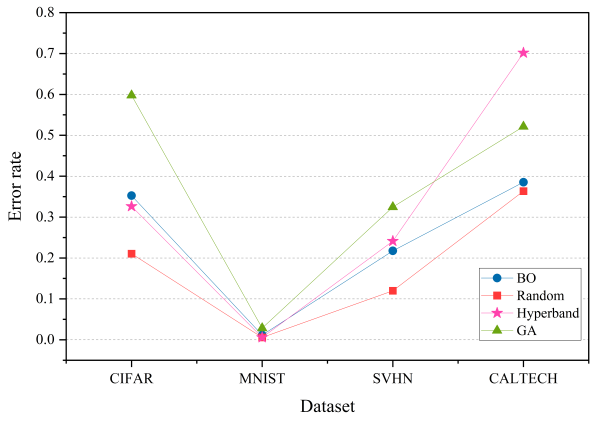

Figure 2. Test error rate plot for experiment setup 1

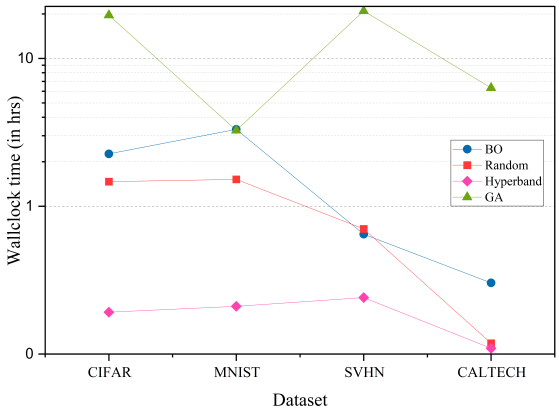

Figure 3. Wallclock time comparison plot

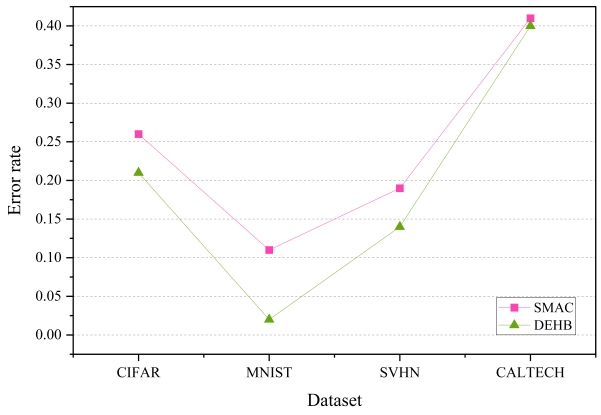

Figure 4. Test error rate plot for experiment setup 2

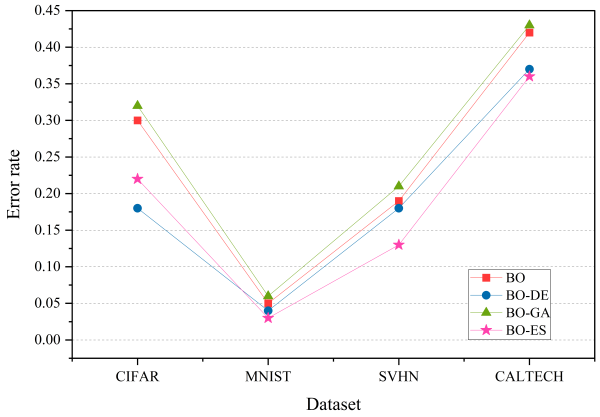

Figure 5. Test error rate plot for experiment setup 3

Sl. No	Hyperparameter	Range
1	Learning rate	[0.0001 - 1.0]
2	Batch size	[32 - 1024]
3	Epochs	[10 - 100]
4	Momentum	[0.9 - 0.999]

Table 1. Hyperparameter values for tuning