A Linear Programming Enhanced Genetic Algorithm for Hyperparameter Tuning in Machine Learning

Ankur Sinha, IEEE Senior Member
Centre for Data Science and Artificial Intelligence
Indian Institute of Management Ahmedabad
Ahmedabad, India 380015
asinha@iima.ac.in

Paritosh Pankaj
Department of Statistics and Data Science
Indian Institute of Technology Kanpur
Kanpur, India 208016
ppankaj21@iitk.ac.in

***Abstract*—In this paper, we formulate the hyperparameter tuning problem in machine learning as a bilevel program. The bilevel program is solved using a micro genetic algorithm that is enhanced with a linear program. While the genetic algorithm searches over discrete hyperparameters, the linear program enhancement allows hyper local search over continuous hyperparameters. The major contribution in this paper is formulation of the linear program that supports fast search over continuous hyperparameters, and can be integrated with any hyperparameter search technique. It can also be applied directly on any trained machine learning or deep learning model for the purpose of fine-tuning. We test the performance of the proposed approach on two datasets, MNIST and CIFAR, to show the efficacy of the proposed approach. Our results clearly demonstrate that using the linear program enhancement offers significant promise when incorporated with any population-based approach for hyperparameter tuning.**

***Index Terms*—Bilevel optimization, genetic algorithms, machine learning, hyperparameter tuning, linear program.**

I. Introduction

Hyperparameter optimization is an incredibly challenging task in machine learning, as hyperparameters are external to the model and can't be determined based on the training data alone. These common hyperparameters include, network architecture (for example, number of layers and number of neurons per layer), optimization parameters (for example, learning rate and momentum), and regularization parameter (for example, weight decay and dropout). The most common approach to identify the right set of hyperparameters involves training models with different hyperparameters on the training dataset and then evaluating the models on the validation dataset. The best performing model on the validation dataset is often the model that is chosen.

The hyperparameter optimization problem is intrinsically a bilevel optimization task where the upper level problem searches for the optimal hyperparameters and the lower level problem searches for the optimal model parameters for the corresponding hyperparameters. In the context of evolutionary algorithms as well a number of algorithm hyperparameters have to be tuned and their optimal choice can be made using a bilevel optimization approach. Formulating the hyperparameter optimization problem as a bilevel optimization task is a familiar approach in machine learning [3] and also in evolutionary computation [19].

A bilevel optimization problem involves two levels of optimization with each level having its own objective function, set of variables, and set of constraints. A large body of literature exists on bilevel optimization for which the readers may refer to [1], [6], [18]. A bilevel optimization problem is challenging because the upper level variables appear as parameters in the lower level optimization problem, while the lower level problem has to be optimized with respect to the lower level variables. Solving the lower level optimization problem for a given set of upper level variables and ensuring that the upper level constraints are satisfied lead to a feasible solution to the bilevel optimization problem. There are a number of approaches that are used to handle bilevel optimization problems efficiently; however, most of them often involve solving multiple optimization problems. In this paper, we will first formulate the hyperparameter optimization problem as a bilevel problem and will stick to the nomenclature commonly used by the machine learning community. Let the upper level variables (hyperparameters) be denoted by λ and the lower level variables (model parameters) be denoted by w. If the upper level objective (validation loss) is given as $F(\lambda, w)$ and the lower level objective (training loss) is given as $f(\lambda, w)$, then the hyperparameter optimization problem is defined as follows:

$$\begin{aligned} \min_{\lambda, w} \quad & F(\lambda, w; S^V) \\ \text{subject to} & \qquad\qquad\qquad\qquad (1) \\ w \in & \operatorname*{argmin}_{w} \{ f(\lambda, w; S^T) \} \end{aligned}$$

where S_T represents the set of training examples and S_V represents the set of validation examples. Under general cases, the upper level and the lower level problems may contain constraints as well.

In this paper, we will consider two types of hyperparameters to demonstrate our ideas. The first set of hyperparameters, denoted by λ_d, would include discrete hyperparameters for which we have chosen the network architecture parameters (number of layers and number of neurons). The second set of hyperparameters, denoted by λ_c, would include continuous hyperparameters for which we have chosen regularization hyperparameters (weight decay). In our proposed approach, we will handle the discrete and continuous parameters using a micro genetic algorithm (micro-GA), and perform a hyper local search for the continuous hyperparameters using a linear programming approach. The main contribution in this paper is the design of the linear program that would help us perform hyper local search within the micro-GA.

A wide range of techniques exists to address hyperparameter optimization problem defined in (1). The popular strategies include naive methods like grid search and random search, where a number of hyperparameter vectors are sampled from the hyperparameter space and models are optimized on the training dataset for each of the sample hyperparameter vector. The models are then evaluated on the validation dataset and the best performing model in terms of validation loss is chosen. Bergstra et al. [4] demonstrated that the random search surpasses grid search in terms of computational performance, thus making it preferable. Hyperband [11], which is an extension of random search, intelligently allocates computational resources to promising configurations via a multi-armed bandit technique while searching for the best hyperparameters. Bayesian optimization happens to be the gold standard for hyperparameter optimization [4], [8], [20], [21]. In these methods, a probabilistic model is created for the validation objective based on which an informed decision is made for sampling the next hyperparameter vector. A common requirement in most of these approaches is to strike the right balance between exploration and exploitation, as heavy exploration tends to be computationally very costly in hyperparameter optimization. Common methods used for probabilistic modeling of the objective function includes, tree Parzen estimator [4], Gaussian process estimation [20], and sequential model-based approach [8]. Most of these methods, model-free and model-based, suffer from the curse of dimensionality and perform poorly with increase in number of hyperparameters [14]. These approaches are also referred to as black-box approaches as they do not utilize the underlying structure of the hyperparameter optimization problem. Response surface estimation-based [7], [12]–[14], [17] and hypergradient-based [2], [15] optimization approaches have also been popular lately in the context of hyperparameter optimization. Some aspects of our study fall within the domain of gradient estimation for the bilevel hyperparameter optimization problem.

The paper is organized as follows. To begin with, we propose the model fine-tuning approach with respect to continuous hyperparameters in Section II, followed by a detailed discussion on its integration in micro-GA in the form of hyper local search in Section III. In Section IV we provide extensive results on two datasets to demonstrate the effectiveness of our approach. Finally, we conclude the paper in Section V. The central notations used in our study are summarized in Table I.

II. Fine-tuning Machine Learning Models

In this section, we discuss a model fine-tuning approach that can be applied on any model that has been learned on a given training dataset. The approach works by refining the chosen continuous hyperparameters and the optimized model parameters in its vicinity using a linear programming approach. Let us focus only on continuous hyperparameters that can be varied and let us assume the discrete hyperparameters to be fixed. In this case, the bilevel optimization problem for optimizing the continuous hyperparameters can be written as follows:

$$\begin{aligned} \min_{\lambda_c, w} \quad & F(\lambda_c, w; S^V) \\ \text{subject to} \quad & \\ & w \in \operatorname*{argmin}_{w} \{ f(\lambda_c, w; S^T) \} \end{aligned} \tag{2}$$

Let us say that for a given value of the continuous hyperparameter, λ_c°, we optimize the model parameters, i.e. solve the lower level problem, and obtain w°. We wish to fine-tuning the model $M(\lambda_c^\circ, w^\circ)$, by moving in a direction, (d_{λ_c}, d_w), such that we obtain a new model $M(\lambda_c^\circ + t d_{\lambda_c}, w^\circ + t d_w)$ (for some non-negative value of t) that provides a better upper level function value and $w^\circ + t d_w$ remains optimal for $\lambda_c^\circ + t d_{\lambda_c}$. It would be ideal to choose the direction in such a way that it leads to the steepest descent for the upper level objective function while satisfying the lower level optimality conditions. Such a direction is nothing but the negative of the gradient of the bilevel optimization problem (2). We will next attempt to derive this direction of steepest descent.

The assumptions for the derivation are that the upper level function $F(\lambda_c, w; S^V)$ is at least once differentiable and the lower level function $f(\lambda_c, w; S^T)$ is at least twice differentiable. We also assume that for any given value of λ_c, there always exists a solution $w \in \operatorname*{argmin}_{w} \{ f(\lambda_c, w; S^T) \}$.

Theorem 1. *At a given point $(\lambda_c^\circ, w^\circ)$, such that, $w^\circ \in \operatorname*{argmin}_{w} \{ f(\lambda_c^\circ, w; S^T)$ the steepest descent direction for (2) can be obtained by solving the following problem:*

$$\begin{aligned} \min_{d_{\lambda_c}, d_w} \quad & \nabla_{\lambda_c} F(\lambda_c^\circ, w^\circ; S^V)^T d_{\lambda_c} + \nabla_w F(\lambda_c^\circ, w^\circ; S^V)^T d_w \\ \text{subject to} \quad & \\ & d_w \in \operatorname*{argmin}_{d_w} \left\{ \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}^T \nabla^2_{(\lambda_c, w)} f(\lambda_c^\circ, w^\circ; S^T) \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix} \right\} \\ & -1 \le d_{\lambda_c} \le 1 \end{aligned} \tag{3}$$

Proof. For a given direction vector (d_{λ_c}, d_w) and gradient of the upper level function $(\nabla_{\lambda_c} F(\lambda_c^\circ, w^\circ; S^V), \nabla_w F(\lambda_c^\circ, w^\circ; S^V))$ at the point $(\lambda_c^\circ, w^\circ)$, clearly (d_{λ_c}, d_w) represents the upper level descent direction if the dot product $\nabla_{\lambda_c} F(\lambda_c^\circ, w^\circ; S^V)^T d_{\lambda_c} + \nabla_w F(\lambda_c^\circ, w^\circ; S^V)^T d_w < 0$. Also if D° is the acceptable set of direction vectors at $(\lambda_c^\circ, w^\circ)$, then the following would lead to the steepest descent

TABLE I: Central Notation

Category	Notation	Description
Dataset	S_T	$S_T = \{(x_i, y_i)\}_{i=1}^{N^T}$; training set, where x and y are combination of input features and output classes, and N^T is the number of training examples
	S_V	$S_V = \{(x_i, y_i)\}_{i=1}^{N^V}$; validation set, where x and y are combination of input features and output classes, and N^V is the number of validation examples
Bilevel variables	$\lambda = (\lambda_d, \lambda_c)$	discrete and continuous hyperparameters (upper level variables)
	w	model parameters (lower level variables)
Objectives	$F(\lambda, w; S^V)$	Upper level objective function
	$f(\lambda, w; S^T)$	Lower level objective function
Loss function	l	Training $l(w; S^T)$ and validation loss $l(w; S^V)$
Regularization	Θ	regularization function (L_2 regularization is used in this paper)
Direction vectors	d_{λ_c}	the descent (with respect to F) direction vector for continuous hyperparameters
	d_w	the descent (with respect to F) direction vector for model parameters

direction for (2) at the point $(\lambda_c^\circ, w^\circ)$.

$$\operatorname*{argmin}_{d_{\lambda_c}, d_w}\{\nabla_{\lambda_c} F(\lambda_c^\circ, w^\circ; S^V)^T d_{\lambda_c} + \nabla_w F(\lambda_c^\circ, w^\circ; S^V)^T d_w \\ : (d_{\lambda_c}, d_w) \in D^\circ\}$$

We know that $w^\circ \in \operatorname*{argmin}_{w}\{f(\lambda_c^\circ, w; S^T)\}$, therefore, $\nabla_w f(\lambda_c^\circ, w; S^T) = 0$. If λ_c changes infinitesimally, as $\lim_{t \to 0} \lambda_c^\circ + t d_{\lambda_c}$, we would like to know $\lim_{t \to 0} w^\circ + t d_w$, such that,

$$d_w \in \operatorname*{argmin}_{d_w}\{f(\lambda_c^\circ + t d_{\lambda_c}, w^\circ + t d_w; S^T)\} \tag{4}$$

We made the assumption that the lower level problem always has an optimal solution for any given upper level vector, so an optimal d_w exists for a given value of t and d_{λ_c}. Essentially, at $(\lambda_c^\circ, w^\circ)$ when the upper level vector changes along the direction d_{λ_c}, we want to know the direction d_w along which the lower level vector should change so that it remains optimal for the lower level problem.

We have assumed $f(\lambda_c^\circ, w^\circ)$ to be twice differentiable, therefore, we can write its Taylor's expansion around $(\lambda_c^\circ, w^\circ)$ with second-order approximation as follows (dropping S^T for brevity):

$$f(\lambda_c^\circ + t d_{\lambda_c}, w^\circ + t d_w) = f(\lambda_c^\circ, w^\circ) + \\ \nabla_{\lambda_c} f(\lambda_c^\circ, w^\circ)^T t d_{\lambda_c} + \nabla_w f(\lambda_c^\circ, w^\circ)^T t d_w + \\ \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}^T \nabla^2_{(\lambda_c, w)} f(\lambda_c^\circ, w^\circ) \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}$$

The above expansion has been written as 4 terms, where the second and third terms are first-order terms written in two parts. Since $\nabla_w f(\lambda_c^\circ, w) = 0$, the third term can be ignored. Therefore, we get the following:

$$\min_{d_w} f(\lambda_c^\circ + t d_{\lambda_c}, w^\circ + t d_w; S^T) = \\ f(\lambda_c^\circ, w^\circ) + \nabla_{\lambda_c} f(\lambda_c^\circ, w^\circ)^T t d_{\lambda_c} + \\ \min_{d_w} \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}^T \nabla^2_{(\lambda_c, w)} f(\lambda_c^\circ, w^\circ) \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}$$

which implies,

$$\operatorname*{argmin}_{d_w}\left\{f(\lambda_c^\circ + t d_{\lambda_c}, w^\circ + t d_w; S^T)\right\} = \\ \operatorname*{argmin}_{d_w}\left\{\begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}^T \nabla^2_{(\lambda_c, w)} f(\lambda_c^\circ, w^\circ) \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}\right\}$$

Therefore, (4) can be written as follows:

$$d_w \in \operatorname*{argmin}_{d_w}\left\{\begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}^T \nabla^2_{(\lambda_c, w)} f(\lambda_c^\circ, w^\circ) \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix}\right\} \tag{5}$$

solving which gives us an optimal d_w for a given d_{λ_c}. We want that (d_{λ_c}, d_w) pair that leads to the steepest descent direction while ensuring d_w optimality for any d_{λ_c}, which we get by solving (3). Note that we additionally have $-1 \leq d_{\lambda_c} \leq 1$ as a constraint at the upper level which restricts the magnitude of the vector otherwise (3) will be unbounded. This completes the proof of the theorem. □

Interestingly, the same results can also be arrived at as a special case of the results discussed in [16]. Next, we attempt to simplify the results further. Given that the lower level problem is an unconstrained optimization problem, we can write the first-order optimality conditions for the lower level problem in (3).

Let the symmetric matrix $\nabla^2_{(\lambda_c,w)} f(\lambda_c^\circ, w^\circ; S^T)$ be denoted as:

$$\Big[h_{ij}\Big]_{i=1,j=1}^{p+q,p+q} = \nabla^2_{(\lambda_c,w)} f(\lambda_c^\circ, w^\circ; S^T),$$

where p and q denote the dimensions of λ_c (or d_{λ_c}) and w (or d_w), respectively. Then, the first order conditions for the lower level problem in (3) can be written as follows:

$$\Big[h_{ij}\Big]_{i=p+1,j=1}^{p+q,p+q} \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix} = 0$$

This reduces formulation (3) into a linear program solving which provides us the steepest descent direction for (2):

$$\begin{aligned} \min_{d_{\lambda_c}, d_w} \quad & \nabla_{\lambda_c} F(\lambda_c^\circ, w^\circ; S^V)^T d_{\lambda_c} + \nabla_w F(\lambda_c^\circ, w^\circ; S^V)^T d_w \\ \text{subject to} \quad & \\ & \Big[h_{ij}\Big]_{i=p+1,j=1}^{p+q,p+q} \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix} = 0 \\ & -1 \le d_{\lambda_c} \le 1 \end{aligned} \tag{6}$$

We relax the equality constraints in the linear program into inequalities by choosing a small value δ, which leads to the following program:

$$\begin{aligned} \min_{d_{\lambda_c}, d_w} \quad & \nabla_{\lambda_c} F(\lambda_c^\circ, w^\circ; S^V)^T d_{\lambda_c} + \nabla_w F(\lambda_c^\circ, w^\circ; S^V)^T d_w \\ \text{subject to} \quad & \\ & -\delta \le \Big[h_{ij}\Big]_{i=p+1,j=1}^{p+q,p+q} \begin{bmatrix} d_{\lambda_c} \\ d_w \end{bmatrix} \le \delta \\ & -1 \le d_{\lambda_c} \le 1 \end{aligned} \tag{7}$$

Let the optimal solution to the above problem be denoted as $d^*_{\lambda_c}, d^*_w$, then new models along the descent direction can be generated as follows:

$$M(\lambda_c^\circ + t d^*_{\lambda_c}, w^\circ + t d^*_w) : t > 0 \tag{8}$$

One may choose a model in the vicinity of $M(\lambda_c^\circ, w^\circ)$ for a particular value of t (say t^*), such that the validation loss for $M(\lambda_c^\circ + t^* d^*_{\lambda_c}, w^\circ + t^* d^*_w)$ is smaller than the validation loss for $M(\lambda_c, w^\circ)$.

For the experimentation in this section, the upper and the lower level objective functions in (3) have been chosen as follows:

$$F(w) = l(w; S^V) \tag{9}$$

$$f(\lambda_c, w) = l(w; S^T) + \Theta(w, \lambda_c) \tag{10}$$

where l is the average cross-entropy loss function, and Θ is the L$_2$-regularization function. Note that with these choice of functions, the upper level objective is defined only with respect to w and does not directly involve λ_c. The optimal model parameter vector is a function of the hyperparameter vector, therefore, one often denotes the optimal model parameters as $w(\lambda_c)$, due to which F has an indirect dependency on λ_c. With a single regularization hyperparameter we have, $\Theta(w, \lambda_c) = \lambda_c \sum_{i=1}^q w_i^2$. Such a regularization approach is also known as weight decay as it promotes the model parameters to be smaller in magnitude, thus preventing overfitting on the training examples. L$_2$-regularization can be extended with additional terms when λ_c is a vector, with each term in λ_c penalizing a different set of sum of squared weights.

The model $M(\lambda_c^\circ, w^\circ)$ represents a feasible solution to the bilevel optimization problem in (3) for which we identify the steepest descent direction by solving the linear program (7). Let the solutions of the linear program be denoted as $(d^*_{\lambda_c}, d^*_w)$ and the new models can be generated along this direction as follows:

$$\begin{bmatrix} \lambda_c^n \\ w^n \end{bmatrix} = \begin{bmatrix} \lambda_c^\circ \\ w^\circ \end{bmatrix} + t \begin{bmatrix} d^*_{\lambda_c} \\ d^*_w \end{bmatrix}$$

If $M(\lambda_c^\circ + t d^*_{\lambda_c}, w^\circ + t d^*_w)$ denotes various models along the steepest descent direction, then let $M(\lambda_c^\circ + t^* d^*_{\lambda_c}, w^\circ + t^* d^*_w)$ be the model that minimizes the validation loss $l(w; S^V)$. We refer to this exercise as a fine-tuning exercise that will be later incorporated into a genetic algorithm for the purpose of hyper local search. Figure 1 shows the fine-tuning exercise graphically and provides a visual representation on how validation and training loss may change along this direction. While validation loss is expected to improve along the descent direction, not much can be inferred about the training loss. Depending on the starting point $(\lambda_c^\circ, w^\circ)$ the training loss may improve or get worse.

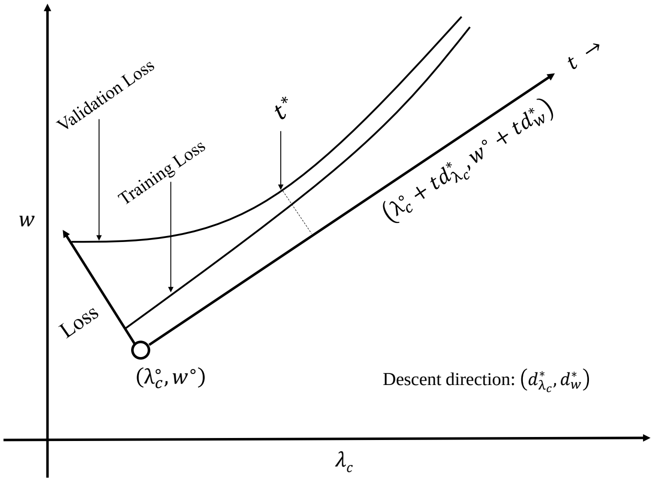

Validation Loss
Training Loss
t^*
$t \nearrow$
$(\lambda_c^\circ + t d^*_{\lambda_c}, w^\circ + t d^*_w)$
w
Loss
$(\lambda_c^\circ, w^\circ)$
Descent direction: $(d^*_{\lambda_c}, d^*_w)$
λ_c

Fig. 1: (λ_c, w) space with the descent direction $(d^*_{\lambda_c}, d^*_w)$ from $(\lambda_c^\circ, w^\circ)$. The training and validation loss are also shown along the descent direction.

Next, we present some results for demonstrating the effectiveness of the approach on MNIST dataset [10] in the context of multi-layer perceptron (MLP) architecture. In MNIST dataset the objective is to solve a multi-classification problem, for which we create an MLP model with 5 hidden layers and 50 neurons in each layer. We randomly sample 5,000 data points for training, 2,500 data points for validation and 10,000 data points for testing. The reason for choosing fewer samples for training is to allow overfitting to happen when we create our first model $M(\lambda_c^\circ, w^\circ)$ without any regularization, i.e. $\lambda_c^\circ = 0$, using stochastic gradient descent. Thereafter, we consider three cases:

1) MNIST (1HP): Solve the linear program in (7) with 1 regularization hyperparameter, i.e. a single regularization hyperparameter for the hidden layers and the output layer
2) MNIST (2HP): Solve the linear program in (7) with 2 regularization hyperparameters, i.e. 1 regularization hyperparameter for the hidden layers and 1 regularization hyperparameter for the output layer
3) MNIST (6HP): Solve the linear program in (7) with 6 regularization hyperparameters, i.e. 5 regularization hyperparameters for the hidden layers and 1 regularization hyperparameter for the output layer

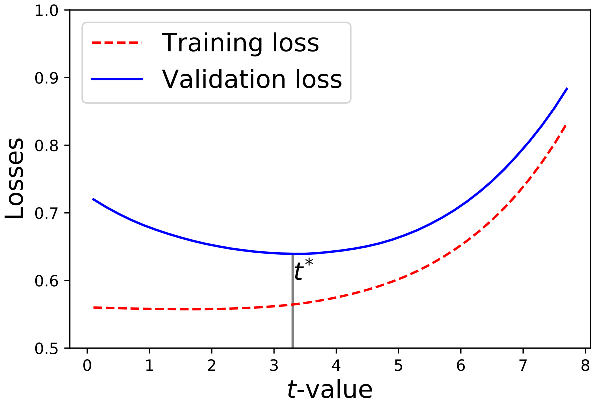


Fig. 2: Training and validation losses while moving along the steepest descent direction for MNIST (1HP).

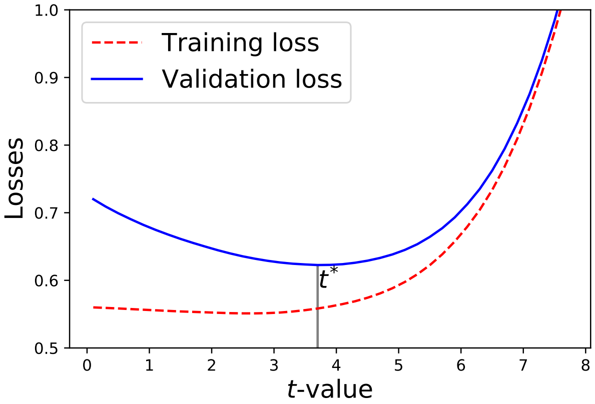


Fig. 3: Training and validation losses while moving along the steepest descent direction for MNIST (2HP).

We present the results of fine-tuning for MNIST (1HP), MNIST (2HP) and MNIST (6HP) through Figures 2, 3 and 4, respectively. For different values of t, we get models with (approximately) locally optimal weights on the training data. The figures show how the training and the validation accuracy for these models change as we increase t. Moving along the steepest descent direction leads to a better validation loss with the best being at t^*. The models corresponding to t^* are considered to be the fine-tuned model for each of the cases. Figure 5 shows the validation and test accuracy of the models corresponding to t^* for all the three cases. All the models have a better test performance than the base model with no regularization hyperparameters for which the testing accuracy was 0.6776. Interestingly, the models improve with increase in number of regularization hyperparameters; however, this is expected to continue only until overfitting does not happen on the validation dataset. With too many regularization hyperparameters there can be overfitting on the validation dataset leading to poor performance on the test dataset.

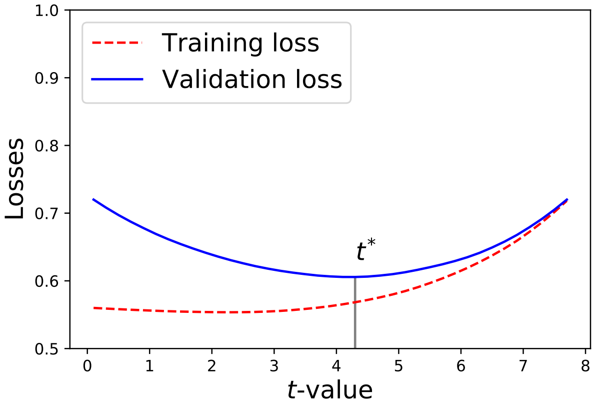


Fig. 4: Training and validation losses while moving along the steepest descent direction for MNIST (6HP).

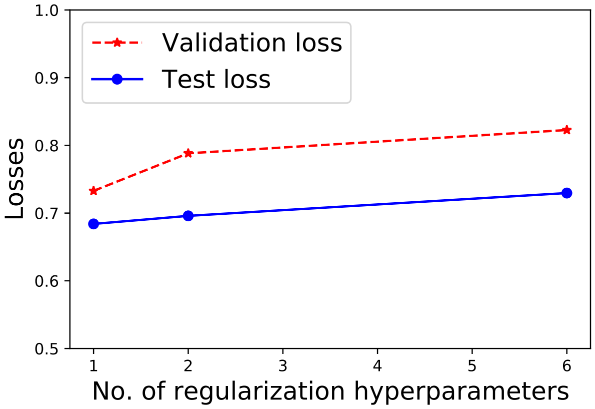


Fig. 5: Validation and test accuracy with increase in number of regularization hyperparameters. The base model with no regularization hyperparameters had the lowest test accuracy of 0.6776.

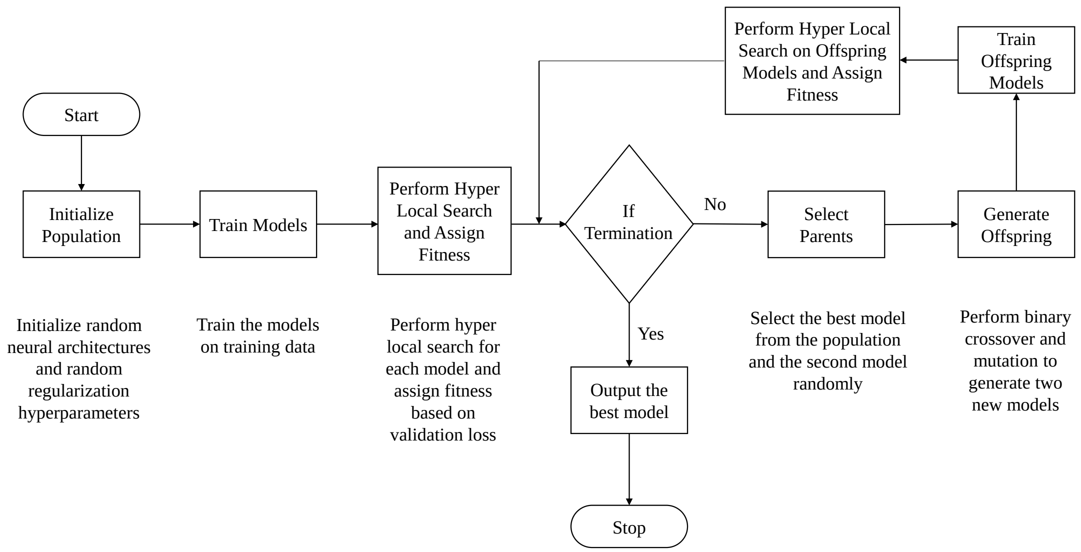
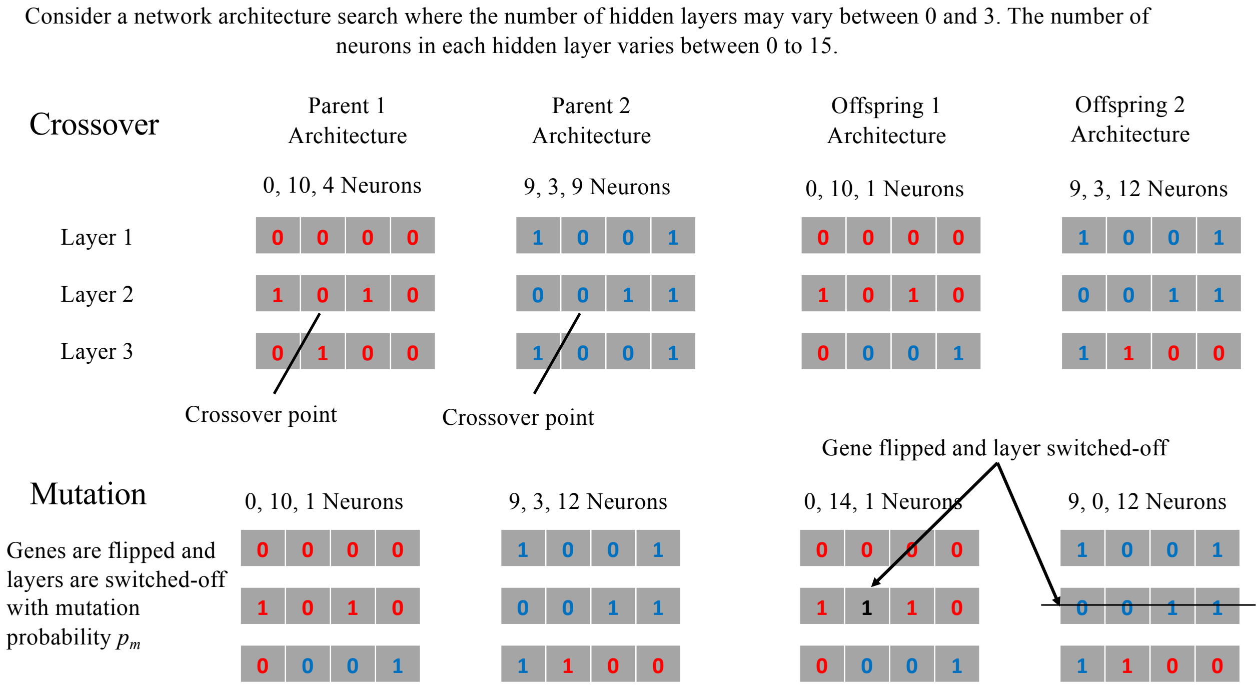

Start
Initialize Population
Train Models
Perform Hyper Local Search and Assign Fitness
If Termination
No
Yes
Select Parents
Generate Offspring
Train Offspring Models
Perform Hyper Local Search on Offspring Models and Assign Fitness
Output the best model
Stop
Initialize random neural architectures and random regularization hyperparameters
Train the models on training data
Perform hyper local search for each model and assign fitness based on validation loss
Select the best model from the population and the second model randomly
Perform binary crossover and mutation to generate two new models

Fig. 6: Flowchart for the steady state micro-GA enhanced with linear program-based hyper local search for hyperparameter optimization.


Consider a network architecture search where the number of hidden layers may vary between 0 and 3. The number of neurons in each hidden layer varies between 0 to 15.

Genes are flipped and layers are switched-off with mutation probability p_m

Fig. 7: Implementation of crossover and mutation on neural architectures.

III. Micro Genetic Algorithm with Fine-tuning

In this section we propose a steady-state micro-GA that utilizes the linear program-based hyper local search (fine-tuning) for the purpose of hyperparameter optimization. We will consider both discrete and continuous hyperparameters in our micro-GA. A steady state micro-GA starts with a small population and updates only a few solutions in each generation. Given the computational cost involved in hyperparameter optimization a steady state micro-GA is a viable option for hyperparameter search, which further gets enhanced with a linear program-based hyper local search on continuous hyperparameters. The flowchart for the micro-GA is provided in Figure 6. In this paper, we consider three hyperparameters for our experiments, namely, number of hidden layers, number of neurons in each hidden layer, and regularization hyperparameters. On neural architecture hyperparameters (discrete) we use a binary crossover and mutation operator as shown in Figure 7, while for regularization hyperparameters (continuous) we use the simulated binary crossover (SBX) and polynomial mutation operators [5]. The algorithm terminates based on the maximum number of generations. The parameter settings for the micro-GA are as follows:

1) Crossover probability: $p_c = 0.9$
2) Mutation probability: $p_m = 0.1$
3) Population size: 10
4) Maximum generations: 15
5) Offspring produced in each generation: 2

The micro-GA can be run with or without hyper local search that we will explicitly specify while presenting the results.

IV. Results

In this section, we provide the results of micro-GA (with and without hyper local search) on two datasets. We also provide results for grid search and random search, with and without hyper local search. The objective is not to compare the performance of the GA against naive techniques like grid search and random search, but to demonstrate that the linear program based-hyper local search proposed in the paper provides benefits in all the approaches where it is incorporated. The two datasets considered in the paper are MNIST [10] and CIFAR [9] with both involving a multi-class classification problem to be solved. We work with the multi-layer perceptron architecture with the following settings throughout the paper.

1) Hidden layers: 0-3 (hyperparameter)
2) Number of neurons in each layer: 0-15 (hyperparameter)
3) L_2 Regularization: 1-dimensional where all weights are penalized in a single term
4) Activation functions: ReLU in hidden layers and Softmax in output layer
5) Optimizer: Adam

A. MNIST Dataset

TABLE II: Validation and test accuracy from 40 samples of grid search, random search and micro-GA for MNIST dataset with 1 hyperparameter.

MNIST	Without hyper local search		With hyper local search	
	Va. Acc.	Te. Acc	Va. Acc.	Te. Acc
Grid search	0.7288	0.7128	0.7356	0.725
Random search	0.7212	0.7142	0.7652	0.7626
micro-GA	0.7368	0.7361	0.7941	0.7788

TABLE III: Validation and test accuracy from 40 samples of grid search, random search and micro-GA for CIFAR-10 dataset with 1 hyperparameter.

CIFAR-10	Without hyper local search		With hyper local search	
	Va. Acc.	Te. Acc	Va. Acc.	Te. Acc
Grid search	0.1768	0.1722	0.2272	0.2204
Random search	0.1872	0.1819	0.2432	0.2415
micro-GA	0.2496	0.2511	0.2781	0.2701

In the original MNIST dataset, there are 60,000 training data points and 10,000 testing data points with 10 classes consisting of handwritten digits. The digits are 28×28 pixel gray-scale images. From the original training dataset we randomly sample 5000 data points that we use for training, and 2500 data points that we use for validation. The entire 10000 data points from the original testing dataset are used as testing data points. We report results for three approaches in this section, i.e., grid search, random search and genetic algorithm. For all the three approaches we report the results with and without linear program-based hyper local search. Number of models searched using grid search, random search and genetic algorithm are restricted to 40 in number. Table II provides detailed results in terms of losses for validation and testing for various models. It is quite clear that the results with hyper local search are better in all cases. Figure 9 shows the convergence of the micro-GA over 15 generations for a run with and without hyper local search. The losses in the case of hyper local search are much lower than the losses in the case of no hyper local search right from the start of the algorithm.

B. CIFAR-10 Dataset

CIFAR-10 dataset has been used in our study that consists of 50,000 data points in training dataset and 10,000 data points in testing dataset with 10 classes. Each data point is a 32×32 pixel coloured image of an object. For CIFAR-10 we randomly sample 5000 data points for training, 2500 data points for validation and 10000 data points for testing from the original datasets. The results are presented in a similar manner as before for grid search, random search and genetic algorithm. Table III clearly demonstrates the benefit of hyper local search for all the cases once again. Figure 9 shows the convergence of the micro-GA over generations for a run with and without hyper local search. Clearly, right from the start of the algorithm, the losses in the case of hyper local search are significantly lower than the losses in the case of no hyper local search.

V. Conclusions

In our work, we have proposed a linear program-based approach that can be used to fine-tune any machine learning model by searching for better continuous hyperparameters

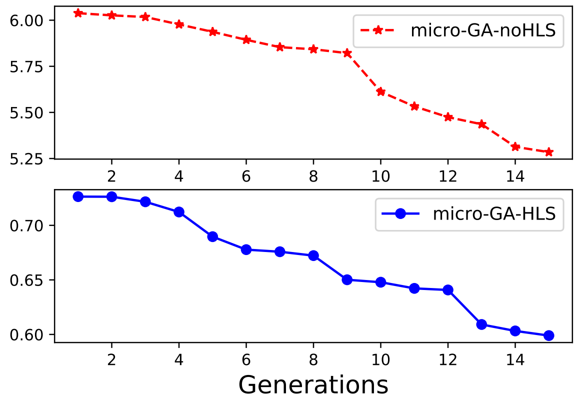


Fig. 8: Validation loss performance of micro-GA with hyper local search (micro-GA-HLS) and without hyper local search (micro-GA-noHLS) over 15 generations on MNIST dataset.

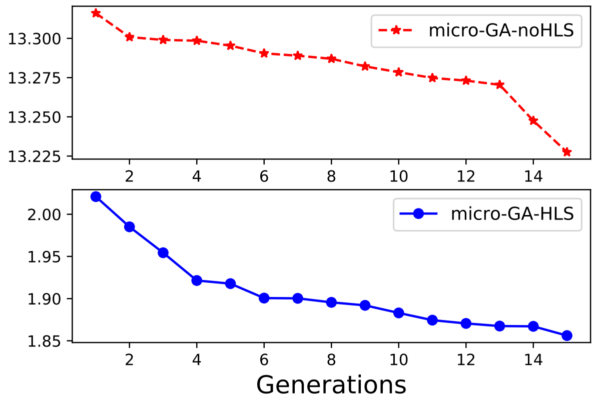


Fig. 9: Validation loss performance of micro-GA with hyper local search (micro-GA-HLS) and without hyper local search (micro-GA-noHLS) over 15 generations on CIFAR-10 dataset.

in the vicinity of the hyperparameters chosen by the user. We formulated the hyperparameter optimization problem as a bilevel program and then showed how the gradient of the bilevel program can be used for fine-tuning the continuous hyperparameters and the model parameters. We first demonstrated the working of this principle on individual models and then incorporated this idea as hyper local search in a steadstate micro-GA. Our results show that when the proposed idea is incorporated in naive techniques like grid search and random search, or in a genetic algorithm, it benefits by producing models that perform better on validation and test data. We evaluated the idea on two datasets, MNIST and CIFAR-10, and the results obtained from all the runs are very promising. We believe that this is a fundamental contribution as the approach can be incorporated in any hyperparameter optimization algorithm.

VI. Acknowledgements

The authors would like to acknowledge the support provided by AK Vishwanathan in performing the initial test study of the ideas discussed in the paper.

References

[1] J. F. Bard. *Practical bilevel optimization: algorithms and applications*, volume 30. Springer Science & Business Media, 2013.

[2] Y. Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

[3] K. P. Bennett, G. Kunapuli, J. Hu, and J. Pang. Bilevel optimization and machine learning. In *Proceedings of the 2008 World Congress on Computational Intelligence*, pages 25–47, 2008.

[4] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[5] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.

[6] S. Dempe. *Foundations of bilevel programming*. Springer Science & Business Media, 2002.

[7] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. *arXiv preprint arXiv:1806.04910*, 2018.

[8] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[9] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[10] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. 2010.

[11] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[12] J. Lorraine and D. Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

[13] M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *arXiv preprint arXiv:1903.03088*, 2019.

[14] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[15] F. Pedregosa. Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355*, 2016.

[16] G. Savard and J. Gauvin. The steepest descent direction for the nonlinear bilevel programming problem. *Operations Research Letters*, 15(5):265–272, 1994.

[17] A. Sinha, T. Khandait, and R. Mohanty. A gradient-based bilevel optimization approach for tuning hyperparameters in machine learning. *arXiv preprint arXiv:2007.11022*, 2020.

[18] A. Sinha, P. Malo, and K. Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2017.

[19] A. Sinha, P. Malo, P. Xu, and K. Deb. A bilevel optimization approach to automated parameter tuning. In *Proceedings of the 16th Annual Genetic and Evolutionary Computation Conference (GECCO 2014)*. New York: ACM Press, 2014.

[20] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

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