Learning to Mutate with Hypergradient Guided Population

Zhiqiang Tao[1,4][*], **Yaliang Li**[2], **Bolin Ding**[2], **Ce Zhang**[3], **Jingren Zhou**[2], **Yun Fu**[4]
[1]Department of Computer Science and Engineering, Santa Clara University
[2]Alibaba Group
[3]Department of Computer Science, ETH Zürich
[4]Department of Electrical & Computer Engineering, Northeastern University
`ztao@scu.edu, {yaliang.li, bolin.ding, jingren.zhou}@alibaba-inc.com`
`ce.zhang@inf.ethz.ch, yunfu@ece.neu.edu`

Abstract

Computing the gradient of model hyperparameters, *i.e.*, hypergradient, enables a promising and natural way to solve the hyperparameter optimization task. However, gradient-based methods could lead to suboptimal solutions due to the non-convex nature of optimization in a complex hyperparameter space. In this study, we propose a hyperparameter mutation (HPM) algorithm to explicitly consider a learnable trade-off between using global and local search, where we adopt a population of *student* models to simultaneously explore the hyperparameter space guided by hypergradient and leverage a *teacher* model to mutate the underperforming students by exploiting the top ones. The teacher model is implemented with an attention mechanism and is used to learn a mutation schedule for different hyperparameters on the fly. Empirical evidence on synthetic functions is provided to show that HPM outperforms hypergradient significantly. Experiments on two benchmark datasets are also conducted to validate the effectiveness of the proposed HPM algorithm for training deep neural networks compared with several strong baselines.

1 Introduction

Hyperparameter optimization (HPO) [4, 10] is one of the fundamental research problems in the field of automated machine learning. It aims to maximize the model performance by tuning model hyperparameters automatically, which could be achieved either by searching a fixed hyperparameter configuration setting [3, 20, 30, 8] from the predefined hyperparameter space or by learning a hyperparameter schedule along with the training process [15, 23]. Among existing methods, hypergradient [2, 24] forms a promising direction, as it naturally enables gradient descent on hyperparameters.

Hypergradient is usually defined as the gradient of a validation loss function w.r.t hyperparameters. Previous methods mainly focus on computing hypergradients by using reverse-mode differentiation [2, 5, 24], or designing a differentiable response function [11, 23] for hyperparameters, yet without explicitly considering the non-convex optimization nature in a complex hyperparameter space. Thus, while hypergradient methods could deliver highly-efficient local search solutions, they may easily get stuck in local minima and achieve suboptimal performance. This can be clearly observed on some synthetic functions which share a similar shape of parameter space to the HPO problem (see Sec. 4.1). It also leads to the question: *can we find a way to help hypergradient with global information?*

The population based hyperparameter search methods work as a good complementary to the hypergradient, such as evolutionary search [25], particle swarm optimization [7], and the population based

[*]Work was done when Zhiqiang Tao interned at Alibaba Group and worked at Northeastern University.

training [15, 19], which generally employ a population of agent models to search different hyperparameter configurations and update hyperparameters with a mutation operation. The population could provide sufficient diversity to globally explore hypergradients throughout the hyperparameter space. However, it is non-trivial to incorporate hypergradients in the population based methods due to a possible conflict between the hand-crafted mutation operation (*e.g.*, random perturbation) and the direction of hypergradient descent.

To address the above challenges, we propose a novel hyperparameter mutation (HPM) scheduling algorithm in this study, which adopts a population based training framework to explicitly learn a *trade-off* (*i.e.*, a mutation schedule) between using the hypergradient-guided local search and the mutation-driven global search. We develop the proposed framework by alternatively proceeding model training and hyperparameter mutation, where the former jointly optimizes model parameters and hyperparameters upon gradients, while the latter leverages a student-teaching schema for the exploration. Particularly, HPM treats the population as a group of student models and employs a teacher model to mutate the hyperparameters of underperforming students. We instantiate our teacher model as a neural network with attention mechanism and learn the mutation direction towards minimizing the validation loss. Benefiting from learning-to-mutate, the mutation is adaptively scheduled for the population based training with hypergradient.

In the experiments, we extensively discuss the properties of the proposed HPM algorithm and show that HPM significantly outperforms hypergradient and global search methods on synthetic functions. We also employ the HPM scheduler in training deep neural networks on two benchmark datasets, where experimental results validate the effectiveness of HPM compared with several strong baselines.

2 Related Work

Roughly we divide the existing HPO methods into two categories, namely, hyperparameter *configuration* search and hyperparameter *schedule* search. Hyperparameter configuration search methods assume that the optimal hyperparameter is a set of fixed values, while hyperparameter schedule search methods relax this assumption and allow hyperparameters to change in a single trail.

Hyperparameter configuration. For hyperparameter configuration search methods, we may divide existing methods into three subcategories: model-free, Bayesian optimization, and the gradient-based methods. The first subcategory includes grid search [29], random search [3], successive halving [16], Hyperband [20], etc. Grid search adopts an exhausting strategy to select hyperparameter configurations in pre-defined grids, while the random search method randomly selects a hyperparameter configuration from the hyperparameter space with a given budget. Inspired by the amazing success of random search, successive halving [16] and Hyperband [20] are further designed with multi-arm bandit strategies to adjust the computation resource of each hyperparameter configuration upon their performance.

All the above HPO methods are model-free as they do not have any distribution assumption about the hyperparameters. Differently, Bayesian optimization methods [30, 14, 6]) assume the existence of a distribution about the model performance over the hyperparameter search space. This category of methods estimates the model performance distribution based on the tested hyperparameter configurations, and predicts the next hyperparameter configuration by maximizing an acquisition function. However, due to the distribution estimation, the computation cost of Bayesian optimization methods can be high and thus the hyperparameter searching is time-consuming. Recently, BOHB [30, 8] utilizes model-free methods such as Hyperband to improve the efficiency of Bayesian optimization.

The gradient-based HPO method is closely related to this work. Pioneering works [2, 5] propose to employ the reverse-mode differentiation (RMD) to calculate hypergradients on the validation loss based on the minimizer given by a number of model training iterations. Following this line, research efforts [24, 28] have been made to reduce the memory complexity of RMD to handle the large-scale HPO problem. A forward-mode differentiation algorithm is proposed in [11] to further improve the efficiency of computing hypergradients based on the chain rule and a dynamic system formulation.

Hyperparameter Schedule. Two representative ways of changing hyperparameters are gradient-based methods such as self-tuning networks (STN) [23] and mutation-based methods such as population based training (PBT) [15, 19]. STN employs hypernetworks [22] as a response function to map hyperparameters to model parameters so that it could obtain hypergradient by backpropagating the validation error through the hypernetworks. PBT performs an evolutionary search over the hyperparameter space with a population of agent models. It provides a discrete mutation schedule via random perturbation. The other two interesting works related to this regime include hypergradient descent [1] and online meta-optimization [33]. However, these two works both focus more on online learning rate adaptation rather than a generic HPO problem. The proposed HPM algorithm also belongs to the category of hyperparameter schedule. Different from existing methods, the proposed HPM explicitly learns suitable mutations when optimizing hypergradient in a complex hyperparameter space.

3 Hyperparameter Mutation (HPM)

3.1 Preliminary

Given input space $\mathcal{X}$ and output space $\mathcal{Y}$, we define $f(\cdot;\theta,h):\mathcal{X}\to\mathcal{Y}$ as a model parameterized by θ and h, where $\theta\in\mathbb{R}^D$ represents model parameters and $h\in\mathbb{R}^N$ vectorizes N hyperparameters sampled from the hyperparameter configuration space $\mathcal{H}=\mathcal{H}_1\times\cdots\times\mathcal{H}_N$. $\mathcal{H}_i$ is a set of configuration values for the i-th hyperparameter. Let $\mathcal{D}_{trn},\mathcal{D}_{val}:\{(x,y)\}$ be the training and validation set. We define $\mathcal{L}(\theta,h):\mathbb{R}^D\times\mathbb{R}^N\to\mathbb{R}$ as a function of parameter and hyperparameter by

$$\mathcal{L}(\theta,h)=\sum_{(x,y)\in\mathcal{D}}\ell(f(x;\theta,h),y),\tag{1}$$

where $\ell(\cdot,\cdot)$ denotes a loss function and $\mathcal{D}$ refers to $\mathcal{D}_{trn}$ or $\mathcal{D}_{val}$. Upon Eq. (1), we further define $\mathcal{L}_{trn}$ and $\mathcal{L}_{val}$ as the training and validation loss functions by computing $\mathcal{L}(\theta,h)$ on $\mathcal{D}_{trn}$ and $\mathcal{D}_{val}$, respectively. Generally, we train the model f on $\mathcal{D}_{trn}$ with the fixed hyperparameter h or a human-crafted schedule strategy, and peek at the model performance by $\mathcal{L}_{val}$ with the learned parameter θ. Thus, the validation loss is usually bounded to the hyperparameter selection.

Hyperparameter optimization (HPO) solves the above issue, and it could be formulated as

$$\min_{h\in\mathcal{H}}\mathcal{L}_{val}(\theta^*,h)\text{ s.t. }\theta^*=\operatorname*{argmin}_{\theta}\mathcal{L}_{trn}(\theta,h),\tag{2}$$

which seeks for an optimal hyperparameter configuration h^* or an optimal hyperparameter schedule. Hypergradient [2, 24, 11] provides a natural way to solve Eq. (2) by performing gradient descent. However, due to the non-convex nature of a hyperparameter space, this kind of method may get stuck in local minima and thus lead to suboptimal performance. In contrast, the population based methods utilize a mutation-driven strategy to search the hyperparameter space thoroughly, which provides the potential to help hypergradient escape from local valleys. In this study, we focus on developing a *trade-off* solution between using hypergradient and the mutation-driven search.

3.2 Population Based Hyperparameter Search

We adopt a similar population based training framework as proposed in [15]. Let $\mathcal{S}_t=\{S_t^k\}_{k=1}^K$ be a population of agent models w.r.t $f(\cdot;\theta,h)$ at the t-th training step, where S_t^k refers to the k-th agent, T represents the total training steps, and K denotes the population size. Generally, the iterative optimization method (*e.g.,* stochastic gradient decent) is used to optimize model weights for each agent. Hence, for $\forall k$, one training step could be described as

$$\theta_{t+1}^k\leftarrow S_t^k(\theta_t^k,h_t^k),\tag{3}$$

where the agent model S_t^k updates model parameters from θ_t^k to θ_{t+1}^k with a fixed hyperparameter h_t^k during the training step. The population based hyperparameter search is given by

$$k^*=\operatorname*{argmin}_{k}\{\mathcal{L}_{val}(\theta_T^k,h_T^k)\}_{k=1}^K.\tag{4}$$

In Eq. (4), $\theta_T^k=S_{T-1}^k(S_{T-2}^k(\dots S_0^k(\theta_0^k,h_0^k)\dots,h_{T-2}^k),h_{T-1}^k)$ is obtained by chaining a sequence of update steps with Eq. (3) and the hyperparameters are updated through some pre-defined or rule-based mutation operations (*e.g.*, random perturbation). More specifically, we summarize the searching process with population based training [15] as follows.

- ***Train step*** updates θ_{t-1}^k to θ_t^k and evaluates the validation loss $\mathcal{L}_{val}(\theta_t^k,h_t^k)$ for each k. One training step could be one epoch or a fixed number of iterations. An agent model is *ready* to be exploited and explored after one step.

- ***Exploit*** $\mathcal{S}_t$ by selection methods, *e.g.*, the truncation selection, which divides $\mathcal{S}_t$ into three sets of *top*, *middle*, and *bottom* agents in terms of validation performance. The agent models in *bottom* exploit the *top* ones by cloning their model parameters and hyperparameters, *i.e.*, $(\theta_t^k, h_t^k) \leftarrow (\theta_t^*, h_t^*)$, where $k \in$ *bottom* and $*$ represents the index of a top performer.
- ***Explore*** the hyperparameters with a mutation operation, denoted as Φ. As in [15], Φ keeps non-bottom agents unchanged, and randomly perturbs a bottom agent's hyperparameter.

The population based training (PBT) methods [15, 19] simultaneously explore the hyperparameter space with a group of agent models. PBT inherits the merits of random search and leverages exploit & explore strategy to alternatively optimize the model parameter θ (by training step) and hyperparameter h (by mutation). This leads to a joint optimization over θ and h, and eventually provides an optimal hyperparameter schedule, *i.e.*, $h_0^{k^*}, \dots, h_{T-1}^{k^*}$ given by Eq. (4), among the population of agents. However, PBT has two limitations. 1) For each training step, $S_t(\theta_t, h_t)$ updates θ_t by fixing h_t, hence the joint optimization stays at a coarse level. 2) The hyperparameters are mainly updated by the mutation operation, yet a learnable mutation is under-explored.

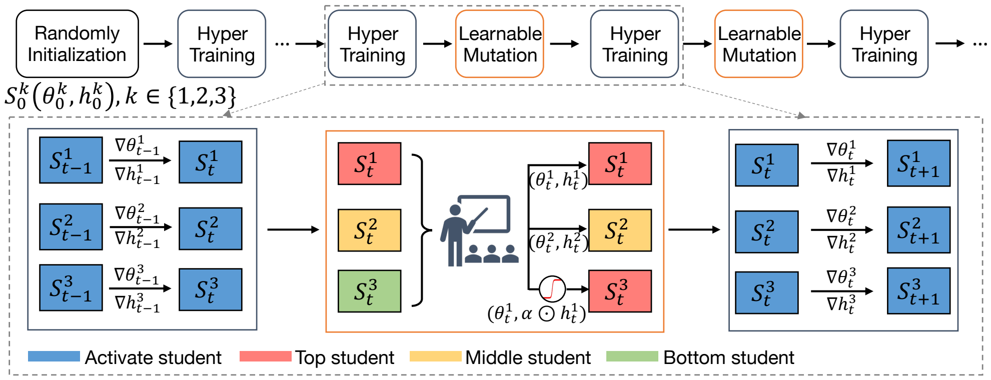


Figure 1: Illustration of the proposed HPM algorithm with an example of three student models. After performing one hypertraining step for all the students, HPM exploits the population by using the truncation selection and explores the cloned hyperparameter for the bottom student models with learnable mutations given by a teacher model.

3.3 Hypergradient Guided Population

We propose to use hypergradient to guide the population based hyperparameter search. To obtain hypergradient, we define $\theta(h) : \mathbb{R}^N \to \mathbb{R}^D$ as a response function of hyperparameter h to approximate the model parameter θ. By using $\theta(h)$, we could extend the agent model to $S_t(\theta_t(h_t), h_t)$, and formulate our hyperparameter mutation (HPM) scheduling algorithm as

$$\min_{h_T} \{\mathcal{L}_{val}(\theta_T^k(h_T^k), h_T^k)\}_{k=1}^K. \tag{5}$$

where $(\theta_T^k(h_T^k), h_T^k)$ is obtained by alternatively proceeding one *hypertraining* step and one *learnable mutation* step as shown in Fig. 1.

Hypertraining jointly optimizes θ and h with hypergradients. Specifically, (θ, h) is updated by

$$\begin{aligned} \theta_t &= \theta_{t-1}(h_{t-1}) - \eta_\theta \nabla\theta, \\ h_t &= h_{t-1} - \eta_h \nabla h, \end{aligned} \tag{6}$$

where $\nabla\theta = \partial\mathcal{L}_{trn}/\partial\theta$ is the gradient of model parameter and ∇h is the hypergradient computed by

$$\nabla h = \frac{\partial \mathcal{L}_{val}(\theta(h), h)}{\partial\theta}\frac{\partial\theta}{\partial h} + \frac{\partial \mathcal{L}_{val}(\theta(h), h)}{\partial h}. \tag{7}$$

The computation of hypergradient in Eq. (7) is mainly depended on the response function $\theta(h)$. In this work, $\theta(h)$ is implemented by hypernetworks [22, 23], which provide a flexible and efficient way to compute hypergradients.

Algorithm 1 Hyperparameter Optimization via HPM

```
Let $\mathcal{S}$ be a set of student models, and $T$ be the given budget
for $t = 1$ to $T$ do
    for $S^k_{t-1} \in \mathcal{S}_{t-1}$ (could be parallelized) do
        Update $S^k_{t-1}(\theta^k_{t-1}, h^k_{t-1})$ to $S^k_t(\theta^k_t, h^k_t)$ by one hypertraining step with Eq. (6) and Eq. (7)
    Divide $\mathcal{S}_t$ into $top$, $middle$, $bottom$ students by the truncation section method
    for $S^k_t \in bottom$ do
        Clone model parameters as $\theta^k_t \leftarrow \theta^*_t$ where $(\theta^*_t, h^*_t) \in top$
        Train the teacher network $g_\phi(h^k_t)$ with Eq. (10) conditioning on $(\theta^*_t, h^*_t)$
        Mutate the hyperparameter with Eq. (8) as $h^k_t \leftarrow g_\phi(h^k_t) \odot h^*_t$
return $\{h^*_0, \dots, h^*_{T-1}\}$, $\theta^*_T$
```

Learnable mutation employs a similar exploit strategy as in Section 3.2 (without $h^k_t \leftarrow h^*_t$) and develops a student-teaching schema [9, 32] for exploration. Particularly, after updating $\mathcal{S}_{t-1}$ to $\mathcal{S}_t$ via one hypertraining step, we treat each agent $S^k_t \in \mathcal{S}_t$ as a student model and learn a teacher model to mutate the underperforming student's hyperparameters. The mutation module Φ is developed as

$$h^k_t = \Phi(h^k_t, h^*_t) = \alpha \odot h^*_t, \tag{8}$$

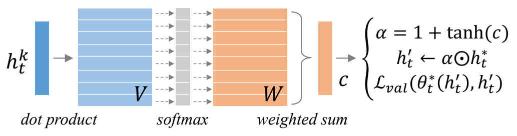
where $h^k_t \in bottom$, $h^*_t \in top$, $\odot$ is the hadamard product, and $\alpha \in \mathbb{R}^N$ denotes the mutation weights. In the following, we will show how to learn α with the teacher network.

3.4 Learning to Mutate

We formulate our teacher model g_ϕ as a neural network with attention mechanism parameterized by $\phi = \{W, V\}$, where $W \in \mathbb{R}^{N\times M}$, $V \in \mathbb{R}^{N\times M}$ are two learnable parameters and M represents the number of attention units, as shown in Fig. 2. It takes input as a bottom student's hyperparameter h^k_t and computes the mutation weights by

$$\alpha = g_\phi(h^k_t) = 1 + \tanh(c), c = W\text{softmax}(V^{\mathrm{T}} h^k_t), \tag{9}$$

where $\alpha \in [0, 2]^N$ and $c \in \mathbb{R}^N$ is a mass vector that tries to characterize the mutation degree for each dimension of h. The benefits of using attention mechanism lie in two folds. 1) It provides sufficient model capability with a key-value architecture, which uses the key slots stored in V to address different underperforming hyperparameters and assign the mutations with the corresponding memory slots in W. 2) g_ϕ enables a learnable way to adaptively mutate hyperparameters along with the training process, where $\alpha \to 1$ gives a mild mutation for a small exploration (update) step, and $\alpha \to 0$ or $\alpha \to 2$ encourages an aggressive exploration to the hyperparameter space.



h^k_t
V
W
c
$\alpha = 1 + \tanh(c)$
$h'_t \leftarrow \alpha \odot h^*_t$
$\mathcal{L}_{val}(\theta^*_t(h'_t), h'_t)$
dot product
softmax
weighted sum

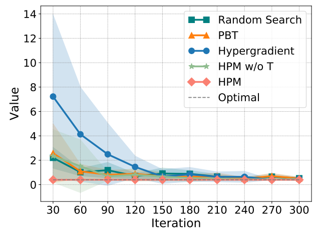
Figure 2: Illustration of our teacher model implemented by an attention network.

We aim to learn the mutation direction towards minimizing $\mathcal{L}_{val}$. To this end, we train our teacher model g_ϕ conditioning on (θ^*_t, h^*_t) by

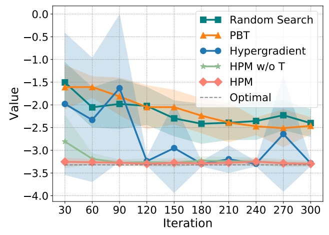
$$\min_{\phi=\{W,V\}} \mathcal{L}_{val}(\theta^*_t(h'_t), h'_t), \tag{10}$$

where $h'_t = \alpha \odot h^*_t = g_\phi(h^k_t) \odot h^*_t$. The parameters of g_ϕ are updated by backpropagating the hypergradients given in Eq. (7) through the chain rule. By freezing the cloned model parameters and hyperparameters (θ^*_t, h^*_t), g_ϕ could be focused on learning the mutations to minimize $\mathcal{L}_{val}$. Please refer to the supplementary material for more details about training the teacher model.

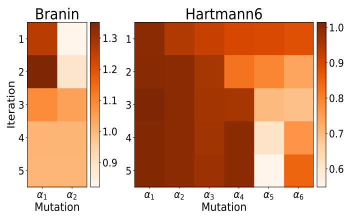
Algorithm 1 summarizes the entire HPM scheduling algorithm. Particularly, HPM computes hypergradients with hypernetworks [22, 23], which add a linear transformation between hyperparameters and model parameters layer-wisely. The hypernetwork can be efficiently computed via feed-forward and backpropagation operations. Moreover, since the teacher network is trained with the frozen student model, the additional computing cost it brings in is much less than training a student model. Thus, the time complexity of HPM is mainly subject to the population size K. While the hypertraining step could be parallelized, the whole population cannot be asynchronously updated due to the centralized teaching process. This can be effectively addressed by introducing an asynchronous HPM, similar to [15]. We leave it as future work and focus on learning to mutate in this study.



(a) Branin



(b) Hartmann6D



(c) Learned Mutations

Figure 3: Experiments on synthetic functions. (a)-(b) The mean performance computed by different methods along with the standard deviation over 10 trials, in terms of different given budget of iterations. (c) The average mutation values learned by HPM over 10 trials. In each trial, HPM runs 30 iterations in total with a population size of 5, resulting in 6 training steps and 5 mutations.

4 Experiments

4.1 Synthetic Functions

One common strategy for exploring the properties of hyperparameters is to perform hyperparameter optimization on synthetic loss functions [34]. These loss functions usually have many local minima and different shapes, and thus could well simulate the optimizing behavior of the real hyperparameters, yet work as much computationally cheaper testbeds than real-world datasets.

Experimental Settings. We employ the Branin and Hartmann6D function provided by the HPOlib[2] library, where Branin is defined in a two-dimensional space with three global minima ($f(h^*) = 0.39787$) and Hartmann6D is defined over a hypercube of $[0, 1]^6$ with one global minima ($f(h^*) = -3.32237$). We compare the proposed HPM with three baseline methods, including 1) random search [3], 2) population based training (PBT) [15], and 3) Hypergradient. We also compare HPM with HPM w/o T, which is the ablated HPM model without using a teacher network. It uses a random perturbation (α is randomly chosen from $[0.8, 1.2]$) for mutation instead. We ran the random search algorithm in HPOlib library and implement the PBT scheduler according to [15]. Note that, as we use the synthetic function f to mimic the loss function of hyperparameters h, the hypergradient is directly given by $\partial f/\partial h$ and is optimized with Stochastic Gradient Descent (SGD) algorithm.

Hyperparameter Optimization Performance. Fig. 3a and Fig. 3b compare the performance of different HPO methods on the Branin and Hartmann6D functions, respectively, where we have several interesting observations. 1) The hypergradient method generally performs better than the global search methods (*e.g.*, random search and PBT) on Hartmann6D rather than Branin, which is consistent with the fact that Hartmann6D has a less number of global minima than Branin. 2) There should be a trade-off between using hypergradient and global search methods (*e.g.*, PBT) according to their opposite performance on these two test functions. 3) The proposed teacher network leads to a more stable and faster convergence performance for HPM compared with HPM w/o T.

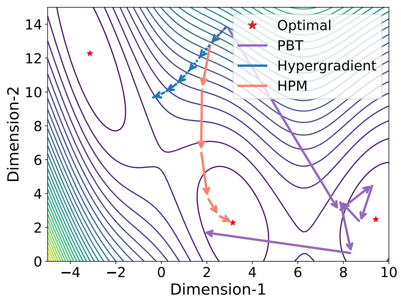


Figure 4: Illustration of the optimization behavior of PBT, Hypergradient, and HPM on the landscape of the Branin function. We run these three methods from the same initialization point with a budget of 30 iterations. HPM and PBT perform 6 updates as they both adopt a population size of 5.

Mutation Schedule. Fig. 4 shows the optimization steps of three methods on the Branin function, where we run PBT, hypergradient, and HPM from the same random initialization point with a budget of 30 iterations. As can be seen, the hypergradient decreases well along with the direction of gradient yet may get stuck in local minima. In contrast, while the PBT method could fully explore the hyperparameter space, it cannot achieve the global minimum without using the gradient guidance. Guided by the teacher network and hypergradient information, the proposed HPM moves towards the

[2]`https://github.com/automl/HPOlib`

Table 1: Performance comparison for the image classification task on the CIFAR10 dataset by validation/test loss and the language modeling task on the PTB corpus dataset by perplexity (PPL).

	Method	CIFAR10		PTB	
		Val Loss	Test Loss	Val PPL	Test PPL
Fixed	Grid Search	0.7940	0.8090	97.32	94.58
	Random Search	0.9210	0.7520	84.81	81.86
	Bayesian Optimization	0.6360	0.6510	72.13	69.29
	Hyperband [20]	0.7156	0.7491	71.25	68.39
Schedule	PBT [15]	0.6253	0.6437	72.07	69.33
	STN [23]	0.5892	0.5878	71.49	68.29
	HPM w/o T	0.5724	0.5802	73.18	70.48
	HPM	**0.5636**	**0.5649**	**70.49**	**67.88**

global optimum adaptively, where HPM skips over several areas quickly and half steps to the end. Interestingly, this is consistent with the mutation schedule as shown in Fig. 3c on Branin, where HPM employs a larger mutation in the first three steps ($\alpha \to 0$ or $\alpha \to 2$) and mild mutations ($\alpha \to 1$) in the last two. Hence, benefiting from the learned mutation schedule, the proposed HPM is a good trade-off between using the hypergradient and mutation-driven update.

4.2 Benchmark Datasets

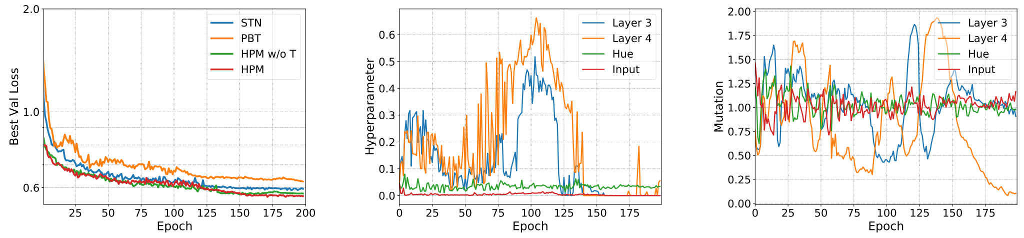
We validate the effectiveness of HPM for tuning hyperparameters of deep neural networks on two representative tasks, including image classification with CNN and language modeling with LSTM.

Experimental Settings. For a fair comparison to hypergradient, all the experiments in this section follow the same setting as in self-tuning networks [23], which is specifically designed for optimizing hyperparameters of deep neural networks with hypergradients. Particularly, we tune 15 hyperparameters, including 8 dropout rates and 7 data augmentation hyperparameters for AlexNet [18] in the CIFAR10 image dataset [17], and 7 RNN regularization hyperparameters [12, 31, 27] for LSTM [13] model in the Penn Treebank (PTB) [26] corpus dataset. We compare our approach with two groups of HPO methods as 1) *fixed hyperparameter* and 2) *hyperparameter schedule* methods. The first group tries to find a fixed hyperparameter configuration over the hyperparameter space, including grid search, random search, Bayesian Optimization[3] and Hyperband [20]. The second group learns a dynamical hyperparameter schedule along with the training process, such as population based training (PBT) [15] and self-tuning network (STN) [23]. Our HPM belongs to the second category.

Implementation Details. We implement PBT with different baseline networks (*e.g.*, AlexNet and LSTM) and use the truncation selection with random perturbation for exploitation and exploration according to [15]. For STN, we directly run the authors' code. We implement our HPM algorithm by using STN as a student model to proceed the hypertraining. HPM employs the same exploit strategy as in PBT and performs learnable mutation with a teacher model (*e.g.*, an attention neural network) for exploration. For both PBT and HPM, we take one training epoch as one *training step*, and do *exploit & explore* operation after each step. The teacher model in HPM is trained by one epoch on the validation set each time called by an underperforming student model. We also implement a strong baseline model as HPM w/o T, which incorporates hypergradient in the population based training without using a teacher network.

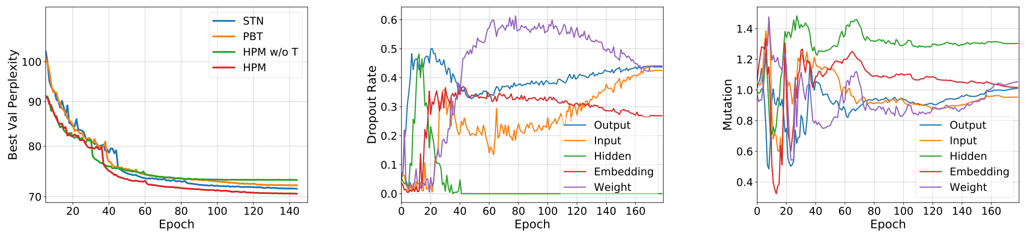
All the codes on benchmark datasets were implemented with Pytorch library. We set the population size as 20 and the truncation selection ratio as 20% for PBT, HPM w/o T, and HPM. We employed the recommended optimizers and learning rates for all the baseline networks and STN models following [23]. Our teacher network was implemented with 64 key slots and was trained with Adam optimizer with a learning rate of 0.001. For the fixed hyperparameter methods, we used the Hyperband [20] implementation provided in [21] and posted the results of the others reported in [23]. For all the hyperparameter schedule methods, we ran the experiments in the same computing environment. STN usually converges within 250 (150) epochs on the CIFAR-10 (PTB) dataset. Thus, we set T as 250 and 150 for all the population based methods on CIFAR-10 and PTB, respectively.

[3]`https://github.com/HIPS/Spearmint`



(a) Loss (b) Hyperparameter schedule (c) Mutation α

Figure 5: Experiments on the CIFAR-10 image dataset. (a) The best validation loss of different methods over training epochs. (b) The learned schedule for 4 hyperparameters by HPM. (c) The mutation schedule given by HPM.



(a) Perplexity (b) Hyperparameter schedule (c) Mutation α

Figure 6: Experiments on the PTB corpus dataset. (a) The best validation perplexity of different methods over training epochs. (b) The learned schedule for 5 dropout rates by HPM. (c) The mutation schedule given by HPM.

Image Classification. Table 1 reports the performance of the fixed hyperparameter and hyperparameter schedule methods on the CIFAR-10 dataset in terms of validation and test loss, respectively. As can be seen, the hyperparameter schedule methods generally perform better than the fixed ones and the proposed HPM scheduler achieves the best performance, which demonstrates the effectiveness of using HPM in tuning deep neural networks. Fig. 5a shows the best validation loss of different methods over training epochs, where the loss of HPM is consistently lower than PBT and STN. We also show the hyperparameter and mutation schedule learned by HPM in Fig. 5b and Fig. 5c. Specifically, we select four hyperparameters including the dropout rates of the input, the third and fourth layer activation, and the rate of adding noise on the hue of an image. We observe that the mutation has a consistent behavior with the hyperparameter. For example, HPM schedules the dropout rate of Layer 3 with a high variance at the early training stage and assigns it a stable small value after the 150-th epoch. Accordingly, the mutation α of Layer 3 oscillates between $[0.5, 1.75]$ before 150 epochs and then tends to be 1. For another example, as Hue and Input have a relatively stable schedule, their mutation weights spread around 1 with a small variance. These observations indicate that HPM can learn a meaningful mutation schedule during the training process.

Language Modeling. We summarize the validation and test perplexity of all the methods on the PTB corpus dataset in Table 1, where HPM also outperforms all the compared methods. One may note that HPM w/o T performs much worse than PBT and STN. This might be due to the conflict between hypergradient and the exploration of random perturbation, which justifies that HPM is not a trivial combination of PBT and STN, and supports that the proposed teacher network plays a key role in finding the mutation schedule. Fig. 6 shows the best validation perplexity of different methods over training epochs on the PTB dataset, as well as the hyperparameter and mutation schedules given by HPM, where a similar observation to the image classification experiment could be obtained.

4.3 Ablation Study

The proposed HPM method adopts a population-based training framework and learns the hyperparameter schedule by alternatively proceeding with the hypertraining and learnable mutation steps. To investigate the impact of different components in HPM, we provide more ablated models other than HPM w/o T as follows: 1) `RS+STN` combines STN [23] and random search (RS). We ran RS with the same given budget as the population size in HPM, *i.e.*, $K = 20$. 2) `HPM w/o H` freezes hyperparame-

Table 2: Ablation study on the CIFAR10 dataset by validation/test loss and the PTB corpus dataset by perplexity (PPL). We investigate the ablated models from four different aspects, including the hypertraining step, population-based training (PBT), learnable mutation, and teacher network.

Methods	Model Components				CIFAR10		PTB	
	Hypertraining	PBT	Mutation	Teacher	Val Loss	Test Loss	Val PPL	Test PPL
RS + STN	✓				0.5817	0.5832	72.03	69.88
HPM w/o H		✓	✓	✓	0.5944	0.6031	73.76	70.73
HPM w/o M	✓	✓			0.6139	0.6267	75.24	72.85
HPM w/o T	✓	✓	✓		0.5724	0.5802	73.18	70.48
HPM (T-MLP)	✓	✓	✓	✓	0.5696	0.5745	71.12	68.57
HPM †	✓	✓	✓	✓	**0.5636**	**0.5649**	**70.49**	**67.88**

† indicates the full proposed model.

ters in the hypertraining step and only updates hyperparameters with learnable mutations. Thus, it could be treated as a PBT model with hypergradient-guided mutations. 3) `HPM w/o M` disables the mutation operation in HPM and, instead, performs one more hypergradient descent step on the cloned hyperparameters for the exploration purpose. 4) In HPM, the mutation is learned by a teacher model implemented with attention networks. Here `HPM (T-MLP)` employs a different implementation for the teacher model. Specifically, it implements the teacher model $g_\phi(h) = 1 + \tanh(W\sigma(V^{\mathrm{T}}h))$ by setting σ as LeakyRelu rather than the softmax function in Eq. (9), in which case, it turns the attention networks as multilayer perceptron (MLP) networks.

Table 2 shows the ablation study results on two benchmark datasets, where our full model HPM consistently outperforms all the ablated models. On the one hand, RS+STN achieves a similar performance compared to STN [23], indicating that, without leveraging an effective *exploit & explore* strategy, a simple combination between local gradient and global search may not boost the performance significantly. On the other hand, while HPM w/o H adopts a learnable mutation, it only performs hypergradient descent with the teacher model, leading to hyperparameters will be updated slowly and cannot be seamlessly tuned along with model parameters. Hence, both hypertraining and learnable mutations are useful for optimizing hyperparameters.

We further compare HPM with two ablated models without using mutations (HPM w/o M) and the teacher network (HPM w/o T). Particularly, HPM w/o M degrades the performance due to over-optimizing hyperparameters and the lack of mutation-driven search; HPM w/o T underperforms since the potential conflict between hypergradient descent and the random-perturbation based mutation. Hence, the ablation studies in Table 2 demonstrate the effectiveness of learning mutations with a teacher model. Moreover, we also provide an alternative implementation of the teacher model with MLP networks, *i.e.*, HPM (T-MLP), which delivers comparative performance to the proposed HPM.

5 Conclusions

We proposed a novel hyperparameter mutation (HPM) algorithm for solving the hyperparameter optimization task, where we developed a hypergradient-guided population based training framework and designed a student-teaching schema to deliver adaptive mutations for the underperforming student models. We implemented a learning-to-mutate algorithm with the attention mechanism to learn a mutation schedule towards minimizing the validation loss, which provides a trade-off solution between using the hypergradient-guided local search and the mutation-driven global search. Experimental results on both synthetic and benchmark datasets clearly demonstrated the benefit of using the proposed HPM over hypergradient and the population based methods.

Broader Impact

The proposed HPM algorithm addresses the challenge of combining local gradient and global search for solving the hyperparameter optimization problem. The proposed framework could be incorporated in many automated machine learning systems to provide an effective hyperparameter schedule solution. The outcome of this work will benefit both the academic and industry communities by liberating researchers from the tedious hyperparameter tuning work.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments and valuable suggestions. This work was supported by Alibaba DAMO Academy and the SMILE Lab (https://web.northeastern.edu/smilelab/) at Northeastern University.

References

[1] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*, 2018.

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

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

[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] J. Domke. Generic methods for optimization-based modeling. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, pages 318–326, 2012.

[6] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, page 3, 2013.

[7] H. J. Escalante, M. Montes, and E. Sucar. Particle swarm model selection. *Journal of Machine Learning Research*, 10:405–440, 2009.

[8] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1437–1446, 2018.

[9] Y. Fan, F. Tian, T. Qin, X.-Y. Li, and T.-Y. Liu. Learning to teach. In *International Conference on Learning Representations*, 2018.

[10] M. Feurer and F. Hutter. *Hyperparameter Optimization*, pages 3–33. Springer International Publishing, 2019.

[11] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1165–1173, 2017.

[12] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29*, pages 1019–1027. 2016.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] 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.

[15] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

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

[17] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 05 2012.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[19] A. Li, O. Spyra, S. Perel, V. Dalibard, M. Jaderberg, C. Gu, D. Budden, T. Harley, and P. Gupta. A generalized framework for population based training. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1791–1799, 2019.

[20] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18-185:1–52, 2018.

[21] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[22] J. Lorraine and D. Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *CoRR*, abs/1802.09419, 2018.

[23] M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations (ICLR)*, 2019.

[24] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2113–2122, 2015.

[25] N. Maheswaranathan, L. Metz, G. Tucker, D. Choi, and J. Sohl-Dickstein. Guided evolutionary strategies: augmenting random search with surrogate gradients. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4264–4273, 2019.

[26] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[27] S. Merity, B. McCann, and R. Socher. Revisiting activation regularization for language rnns. *CoRR*, abs/1708.01009, 2017.

[28] F. Pedregosa. Hyperparameter optimization with approximate gradient. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 737–746, 2016.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12:2825–2830, 2011.

[30] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2951–2959, 2012.

[31] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1058–1066, 2013.

[32] L. Wu, F. Tian, Y. Xia, Y. Fan, T. Qin, L. Jian-Huang, and T.-Y. Liu. Learning to teach with dynamic loss functions. In *Advances in Neural Information Processing Systems 31*, pages 6466–6477. 2018.

[33] Y. Wu, M. Ren, R. Liao, and R. Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018.

[34] M. Zöller and M. F. Huber. Survey on automated machine learning. *CoRR*, abs/1904.12054, 2019.

Supplementary Material

We provide more details of training the teacher network in Section A, more experimental results on synthetic functions in Section B, and the hyperparameter settings for benchmark datasets in Section C.

A Details of Training Teacher Network

Different from the *exploit & explore* strategy in PBT [15], which directly clones $(\theta^*, h^*) \in top$ to an underperforming agent model $(\theta^k, h^k) \in bottom$ and mutates the cloned hyperparameter by random perturbation, we employ a teacher model g_ϕ to adaptively learn the mutation weights α for the exploration purpose, *i.e.*, $(\theta^k, h^k) \leftarrow (\theta^*, \alpha \odot h^*)$ where $\alpha = g_\phi(h^k)$. Here, we omit the iteration subscript t for simplicity. The teacher network g_ϕ takes the *bottom* hyperparameter h^k as input and outputs mutation weights α for exploring the local hyperparameter space centering on h^*. We formulate g_ϕ with Eq. (9) as $\alpha = g_\phi(h^k) = 1 + \tanh(W\text{softmax}(V^{\mathrm{T}} h^k))$, and train $g_{\phi=\{W,V\}}$ by minimizing the validation loss given in Eq. (10).

Let $h' = \alpha \odot h^*$ denote the mutated hyperparameter during the training of g_ϕ. To solve Eq. (10), we obtain the hypergradient regarding to α and backpropagate it to $\phi = \{W \in \mathbb{R}^{N\times M}, V \in \mathbb{R}^{N\times M}\}$. By substituting $h' = \alpha \odot h^*$ in Eq. (7), we obtain the hypergradient of α by

$$\nabla\alpha = \frac{\partial\mathcal{L}_{val}(\theta(h'), h')}{\partial\theta}\frac{\partial\theta}{\partial h'}\frac{\partial h'}{\partial\alpha} + \frac{\partial\mathcal{L}_{val}(\theta(h'), h')}{\partial h'}\frac{\partial h'}{\partial\alpha} = \nabla h|_{(\theta,h)=(\theta^*,h')} \odot h^*. \tag{11}$$

It is worth noting that, Eq. (11) is computed by fixing the model parameters and hyperparameters, (θ^*, h^*), of the top student model, so that the proposed HPM focuses on training the teacher network g_ϕ by learning the mutation weights to minimize the validation loss. To update the parameters of g_ϕ, we compute the gradient of ϕ by employing the backpropagation algorithm with $\nabla\alpha$ as follows:

$$\begin{aligned} \frac{\partial\mathcal{L}_{val}}{\partial W} &= \nabla\alpha(\sigma(V^{\mathrm{T}}h^k))^{\mathrm{T}}, \\ \frac{\partial\mathcal{L}_{val}}{\partial V} &= h^k\delta^{\mathrm{T}}, \delta = (W^{\mathrm{T}}\nabla\alpha) \odot \sigma'(V^{\mathrm{T}}h^k), \end{aligned} \tag{12}$$

where $\sigma(\cdot)$ represents the softmax function and $\sigma'(\cdot)$ denotes its derivative. Thus, by using Eq. (11) and Eq. (12), we could train the teacher network by solving Eq. (10) with the SGD algorithm.

As shown in Algorithm 1, we train the teacher network one step when each time it is called by an underperforming student model, where the step refers to one iteration on synthetic functions and one epoch of the validation set on benchmark datasets in the experiment.

B Experiments on Synthetic Functions

In Section 4.1, we have shown the experimental results of HPM on two population synthetic functions, *i.e.*, the Branin and Hartmann6D functions. In the following, we will provide more details about synthetic functions and the implementation, as well as more results on the other two functions.

Experimental Settings. We used the Branin and Hartmann6D functions in Section 4.1. We show the details of these two functions as follows.

The Branin function is defined in a two-dimensional space of $x_1 \in [-5, 10]$ and $x_2 \in [0, 15]$. It is computed by

$$f(x) = (x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(x_1) + 10.$$

The Branin function has three global minima located at $[-\pi, 12.275]$, $[\pi, 2.275]$, and $[9.42478, 2.475]$, with a global optimal value of $f(x^*) = 0.397887$.

The Hartmann6D function is defined over a hypercube as

$$f(x) = -\sum_{i=1}^{4}\alpha_i \exp(-\sum_{j=1}^{6}A_{ij}(x_j - P_{ij})^2),$$

where $x_j \in (0, 1)$, $\alpha = [1.0, 1.2, 3.0, 3.2]^{\mathrm{T}}$, A and P are two constant parameter matrices. The global minimum of the Hartmann6D function is $f(x^*) = -3.32237$ at $x^* = [0.20169, 0.150011, 0.476874, 0.275332, 0.311652, 0.6573]$.

We also conduct experiments on the other two population functions, including the Rosenbrock and the Bohachevsky functions. The Rosenbrock function is a valley-shaped function defined on the hypercube $x_i \in [-5, 10], i = 1, \dots, d$. The Rosenbrock function is given by

$$f(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2],$$

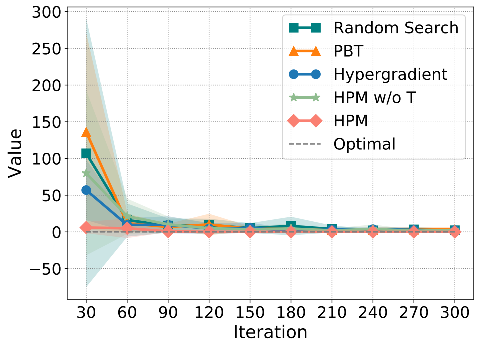
where we set $d = 2$ in our experiments. It has a global minimum value $f(x^*) = 0$ at $x^* = (1, \dots, 1)$.

The Bohachevsky function is a bowl-shape function defined on the square $x_1, x_2 \in [-100, 100]$, which has a global minimum value $f(x^*) = 0$ at $x^* = (0, 0)$. In our experiment, we use the definition of the Bohachevsky function given by

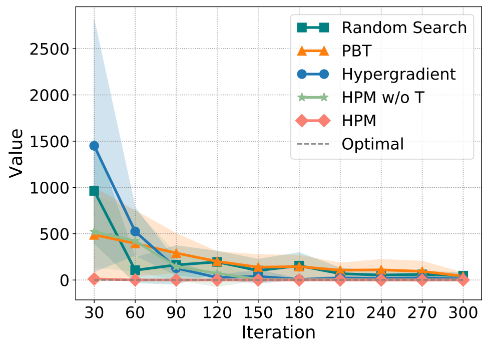
$$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7.$$

Implementation Details. In the experiment, we compare with random search, population based training (PBT), and hypergradient on synthetic functions. We adopt the random search algorithm in HPOlib library, and implement the PBT scheduler according to [15]. It is worth noting that, since the synthetic function only simulates the validation loss function (*i.e.*, $\mathcal{L}_{val}$), it is in essence not suitable to the HPO methods that consider the model selection, like PBT. Hence, we use a simplified implementation here, where PBT performs one *training step* using random search and follow the same *exploit & explore* strategy in Sec. 3.2. The hypergradient is obtained by using the Stochastic Gradient Descent (SGD) algorithm with the synthetic function. We implement our HPM algorithm by treating a set of SGD optimizers as a population of student models, each of which proceeds the hypertraining as one gradient descent step. The same exploit strategy in PBT, *i.e.*, truncation selection [15], is used in our model. HPM w/o T is the ablated model of HPM without using the teacher network, which uses the random perturb (*i.e.*, α is randomly chosen from $[0.8, 1.2]$) instead and could be regarded as a hypergradient guided PBT scheduler.

All the codes on the synthetic functions were implemented with Autograd. The population size of PBT, HPM w/o T and HPM were set to be 5 and the ratio of truncation selection was set as 20%. The learning rate for all the SGD optimizers was 0.01, and we employed $M = 64$ keys for the attention mechanism in our teacher network.



(a) Rosenbrock



(b) Bohachevsky

Figure 7: Experimental results on the Rosenbrock (a) and Bohachevsky (b) functions.

Experimental Results. Fig. 7 compares the performance of different tuning methods on the Rosenbrock and Bohachevsky functions. Same to the Fig. 3 in Section 4.1, we show the mean performance of different methods and the standard deviation (std) over 10 trials. For a fair comparison, each method is given a certain budget of iterations, varied from 30 to 300 with a fixed increasing step of 30, and each iteration only evaluates the synthetic loss function once. As can be seen, the proposed HPM algorithm converges faster than the global search and gradient-based methods, which exhibits a similar observation result as in Fig. 3.

C Hyperparameters on Benchmark Datasets

We show the details of hyperparameters we tuned on the benchmark datasets as follows.

For image classification on the CIFAR-10 dataset, we study AlexNet [18] as the baseline model and employ a configuration space of 15 hyperparameters, including 8 dropout rates assigned on per-layer activations and input, and 7 data augmentation hyperparameters on controlling the noise added to hue, saturation, brightness, and contrast of an image, as well as the length and the number of cut-out holes. Following [23], all the dropout rates are ranged in $[0, 0.75]$; the noise ratios of saturation, brightness, and contrast are ranged in $[0, 1]$; the noise ratio of hue is ranged in $[0, 0.5]$; the number and the length of cut-out holes are in the ranges of $[0, 4]$ and $[0, 24]$, respectively.

For language modeling on the PTB corpus, we tune 7 RNN regularization hyperparameters for the LSTM [13] model, where we use a two-layer LSTM with 650 hidden units per layer and also set the word embedding size as 650. The tied weight is used for the embedding and softmax layer. Following [23], we adopt the hyperparameters as follows: 3 variational dropout [12] rates on the input, hidden states and output, 1 embedding dropout [12] rate, 1 DropConnect [31] rate on the recurrent hidden-to-hidden matrices, and 2 scaling coefficients for activation regularization and temporal activation regularization [27], respectively. All the dropout rates are ranged in $[0, 0.95]$, and the scaling coefficients of regularization are ranged in $[0, 4]$.