Neurocomputing 409 (2020) 381–393

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Efficient hyperparameter optimization through model-based reinforcement learning

Jia Wu *, SenPeng Chen, XiYuan Liu

School of Information and Software Engineering, University of Electronic Science and Technology of China, China

ARTICLE INFO

Article history:
Received 9 January 2020
Revised 15 April 2020
Accepted 16 June 2020
Available online 23 June 2020
Communicated by Zhan Zhi-Hui

Keywords:
Hyperparameter optimization
Machine learning
Reinforcement learning

ABSTRACT

Hyperparameter tuning is critical for the performance of machine learning algorithms. However, a noticeable limitation is the high computational cost of algorithm evaluation for complex models or for large datasets, which makes the tuning process highly inefficient. In this paper, we propose a novel model-based method for efficient hyperparameter optimization. Firstly, we frame this optimization process as a reinforcement learning problem and then employ an agent to tune hyperparameters sequentially. In addition, a model that learns how to evaluate an algorithm is used to speed up the training. However, model inaccuracy is further exacerbated by long-term use, resulting in collapse performance. We propose a novel method for controlling the model use by measuring the impact of the model on the policy and limiting it to a proper range. Thus, the horizon of the model use can be dynamically adjusted. We apply the proposed method to tune the hyperparameters of the extreme gradient boosting and convolutional neural networks on 101 tasks. The experimental results verify that the proposed method achieves the highest accuracy on 86.1% of the tasks, compared with other state-of-the-art methods and the average ranking of runtime is significant lower than all methods by using the predictive model.

1. Introduction

Machine learning algorithms have been successfully applied in the fields of machine translation [1], speech recognition [2], and image recognition [3]. Despite their success, the application of machine learning algorithms still face several challenges, such as hyperparameter optimization (HPO) for complex models or on large datasets. Hyperparameters are different from the internal model parameters. Hyperparameters are set before model training and remain unchanged during training. However, internal model parameters are learned from the data during the model training phrase. HPO is critical for the performance of machine learning algorithms. However, manually setting the hyperparameters often requires users to have a deep professional background and practical experience, which is difficult and time-consuming for nonprofessional users.

To tackle this problem, automatic machine learning (AutoML) has emerged, which refers to automatically optimizing hyperparameters without human interference within a fixed computing budget (iteration or runtime). AutoML focused on, at first, the algorithm selection problem [4,5] and the hyperparameter optimization problem [6,7]. Recently, the combined algorithm selection and hyperparameter optimization (CASH) problem was raised [8].

For the HPO problem, we face the following three challenges. First, the optimization function is a black-box since the relationship between the hyperparameter configuration and its performance cannot be explicitly expressed and as a result, we cannot directly get an optimal solution based on traditional gradient-based methods. Second, since the given algorithm contains a variety of hyperparameters, and each hyperparameters has its own search space, the overall search space is very large. It is difficult for existing methods to efficiently explore the optimal solution. Finally, the evaluation of complex machine learning algorithms or large datasets can be extremely expensive since we need to train the target algorithm with the selected hyperparameter configuration to obtain the evaluation.

Thus far, many different kinds of methods have been used to solve the HPO problem. These methods can be mainly classified into three categories:

- Basic search methods: e.g., grid search method [9], random search method [6]. These methods sample without a guiding strategy, or according to a very simple rule.

* Corresponding author.
E-mail addresses: jiawu@uestc.edu.cn (J. Wu), chensp@std.uestc.edu.cn (S. Chen), liuxiyuan@std.uestc.edu.cn (X. Liu).

https://doi.org/10.1016/j.neucom.2020.06.064
0925-2312/

- Sample-based methods: e.g., Bayesian optimization-based methods [4,10,11], evolutionary algorithm-based method [12]. These methods use a policy to guide the sampling process and update the policy based on evaluating the new sample. A good policy helps to find a high quality configuration in a few iterations.
- Gradient-based methods, e.g., [13,14]. These methods perform hyperparameter optimization by directly calculating the partial derivative of loss function on the validation set.

All these methods have their own advantages and disadvantages, but none of them can handle all the challenges perfectly, especially the third one. Although there have been some excellent works for solving the third problem [15–17], these researches focus on improving the performance of the neural architecture search (NAS).

In this paper, we propose a new model-based method that applies reinforcement learning (RL) to solve the HPO problem. RL is a powerful framework for learning decision-making tasks. Concretely, we first treat the hyperparameter optimization as a sequential decision process and model it as a Markov decision process (MDP). Then, an RL agent constructed by the long short-term memory (LSTM) [18] is employed to sequentially choose the hyperparameters one by one. In this way, the search space can be greatly reduced. The agent aims to maximize the expected accuracy of the algorithm on a validation set. The measured accuracy of the optimized algorithm on the validation set is used as the reward signal to update agent's internal parameters.

It is note that the reward signal is obtained by training the optimized algorithm, which takes significant time and leads to low optimization efficiency. To accelerate training the RL agent, a model is used to predict the performance of the optimized algorithm on the validation set. However, the inaccuracy of the model is further exacerbated by long-term use, resulting in suboptimal solution or even collapse performance (as shown in Fig. 3). To solve this issue, we novelly control model use in the short-term by measuring the distance between policies before and after the model use in the action space and limiting it to a proper range. In this way, we can adaptively control the horizon (how many times) of the model use and improve the efficiency of optimization while ensuring accuracy.

In the experiment, we apply the proposed method to tune the hyperparameters of the extreme gradient boosting and convolutional neural networks on 101 real tasks. The experimental results verify that the proposed method achievers the highest accuracy on 86.1% of the tasks, compared with other state-of-the-art methods and runs much faster. The main contributions are as follows:

- We propose a method for extending the hyperparameter optimization to RL framework. In this way, we can employ an RL algorithm to optimize the black-box function based on sampling.
- We employ an agent which unrolls in multiple time-steps to select hyperparameters sequentially. The special process greatly reduces the search space at each step.
- We employ a model to evaluate the optimized algorithm with selected hyperparameter configuration on the validation set and propose a novel approach to dynamically control the model use. This approach offers new perspectives on how to deal with the third challenge of HPO as mentioned above.

The remaining five sections present the related work, the proposed method, including the framework based on RL and information on how to use a model to accelerate learning, experimental results and the conclusion.

2. Related work

2.1. HPO Problem

HPO is the process of choosing a set of hyperparameters that archive the best performance on the data in a reasonable budget. HPO can be formally defined as follows:

Let $\mathscr{A}$ denote a machine learning algorithm with a configuration space of the overall hyperparameters $\boldsymbol{\Lambda}$. We denote $\mathscr{A}_{\boldsymbol{\lambda}}$ as $\mathscr{A}$ with its hyperparameters $\boldsymbol{\lambda}$, where $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. The hyperparameter space $\boldsymbol{\Lambda}$ can include both discrete and continuous dimension spaces. Given a data set $\boldsymbol{D}$, the goal is to find the optimal hyperparameter configuration $\boldsymbol{\lambda}^*$ such that:

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \mathbb{E}_{(\boldsymbol{D}_{train},\boldsymbol{D}_{valid})\sim\boldsymbol{D}} L(\mathscr{A}_{\boldsymbol{\lambda}}, \boldsymbol{D}_{train}, \boldsymbol{D}_{valid}) \tag{1}$$

where $L(\mathscr{A}_{\boldsymbol{\lambda}}, \boldsymbol{D}_{train}, \boldsymbol{D}_{valid})$ denotes the loss of a model generated by algorithm $\mathscr{A}$ with hyperparameters $\boldsymbol{\lambda}$ on training dataset $\boldsymbol{D}_{train}$ and evaluated on validation dataset $\boldsymbol{D}_{valid}$.

Thus far, there have been various types of algorithms for solving HPO. For a small configuration space of hyperparameters, a grid search [9] or random search [6] are widely used. Some empirical evaluations have shown that the random search works more efficiently than the grid search [6]. However, as the search space increases, their performances become unstable and even degrade sharply. Moreover, the grid search suffers from the exponential combinations of the hyperparameters and the random search is inefficient since lacking of guidance.

Recently, Bayesian optimization (BO) methods have shown the success in hyperparameter optimization within limited resources budget [19,9,4]. Bayesian optimization (BO) is an iterative algorithm with two key components: a probabilistic surrogate model and an acquisition function to determine which point to evaluate next. The most popular acquisition function is the expected improvement (EI) [19]. Three most powerful variants based on BO are the sequential model-based algorithm configuration (SMAC) [4], the tree Parzen estimators (TPE) [9] and the Spearmint [19]. They have different surrogate models. SMAC and TPE are mainly tree-based models, while Spearmint integrates the Gaussian process (GP). SMAC employs random forests to model the surrogate model. Soon after these studies were conducted, authors in [20] improved the performance of the SMAC through using a warm-start technology. TPE is one of the best known BO variants. TPE uses a tree of Parzen estimators for conditional hyperparameters and demonstrates a good performance on the structured HPO tasks [9]. Spearmint is a Gaussian process-based Bayesian optimization method, which is a state-of-the-art approach for low-dimensional hyperparameter optimization. Those BO methods can find a good configuration by using only a few samples. However, as the number of iterations increases, each iteration consumes a large amount of resources.

Population-based methods are also widely used for HPO. One of the best known methods is the covariance matrix adaptation evolution strategy (CMA-ES) algorithm [21], which is an improved algorithm based on the evolutionary algorithm. Much more recently, CMA-ES has been demonstrated to be an excellent choice for parallel HPO, outperforming state-of-the-art BO tools when optimizing 19 hyperparameters of a deep neural network on 30 GPUs in parallel [22].

Another kind of method to solve the HPO problem is Hyperband [23]. It formulates HPO problem as a pure-exploration nonstochastic infinite-armed bandit problem. Hyperband has shown strong performance, especially for optimizing deep learning algorithms. Despite Hyperband has already achieved great success, it lacks guidance and can not quickly converge to the best configuration. To overcome this limitation, the recent method BOHB [24] combi-

nes the advantages of both Bayesian optimization and bandit-based methods, in order to achieve the best of both worlds: strong anytime performance and fast convergence to optimal configurations. Empirically, BOHB was shown to outperform many HPO methods in various of optimization tasks [24].

2.2. Reinforcement learning

The traditional RL problems are often modeled by an MDP, which defined by the tuple $\langle S, A, P, R, \gamma \rangle$ where S is the set of all valid states, A is the set of all valid actions, P defines transition probabilities for the environment, R is the reward function, $r \in \mathbb{R}$ is a scalar reward value, and γ is a discount factor balancing between immediate and future rewards. The agent interacts with the environment and maximizes the expected discount reward. The agent's action is governed by a policy, $\pi : S \rightarrow A$, which computes the state-action value, as:

$$Q_{\pi}(s,a) = E_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t | S_0 = s, A_0 = a] \tag{2}$$

To solve this maximization problem, the policy selects the action that maximizes the discounted cumulative reward, $\pi^*(s) \in argmax_a Q^*(s,a)$, where $Q^*(s,a)$ denotes the optimal state-action value. One of the best known value-based and off-policy methods for solving RL problems is Q-learning [25], which obtain the optimal policy by constantly estimating the optimal value function and obeys the fundamental identity, known as the Bellman equation:

$$Q^*(s,a) = E_{\pi}[r + \gamma max_{a'} Q^*(s',a') | S_0 = s, A_0 = a] \tag{3}$$

RL has achieved great success in many fields [26–28]. The RL algorithms can be roughly divided into two categories: model-free RL and model-based RL. The model-free RL has great potential for solving complex problems. However, this method often requires a large number of samples, which is inefficient. The model-based method allows the agent to learn a model of the environment. The agent directly interacts with the model, which can greatly improve training efficiency. Since the inaccuracy of the model (model bias [29]) is exacerbated and leads to catastrophic failures with long-term use, the horizon of model use should be effectively disciplined. Combining the advantages of both methods, Sutton [30] proposed the Dyna-Q method, which simultaneously trains the agent with samples from the model and the environment. Recently, Kurutach et al. [31] used an ensemble of models to maintain the model uncertainty and regularize the learning process to improve the sampling efficiency of TRPO.

Thus far, RL has achieved great success in AutoML. Many studies use RL to solve the NAS problem [32,33]. In this paper, we focus entirely on optimizing hyperparameters of traditional machine learning algorithms and neural networks, rather than neural architecture search, or automatic data acquisition and cleaning.

3. HPO Based on RL

We describe the details of our method, including why we treat it as a sequential decision problem, how to formulate the HPO problem in the framework of RL, the design of the agent, how to select the hyperparameters, and the training process.

3.1. Sequential decision problem

We consider HPO as a sequential decision process. Generally, a difficult problem can be solved effectively by breaking the problem into multiple subproblems that are easier to handle. Since the configuration space is very large, it is difficult for the agent to select all the hyperparameters in one step. However, if the agent selects one hyperparameter after another, the search space can be greatly reduced at each step, and the optimization efficiency can be improved. As authors in [34] claimed: any complex high-dimensional action can be selected incrementally, component by component, where each component's probability also depends on components already selected earlier. The decision process is naturally sequential [35].

To clearly show the advantages of sequential decision, we will analyze the search space for hyperparameter optimization. Assume that a machine learning algorithm has n hyperparameters, a simple solution is to directly select the whole hyperparameter configuration, where the overall search space is $\mathbf{\Lambda} = \mathbf{\Lambda}_1 \times \mathbf{\Lambda}_2 \times \ldots \mathbf{\Lambda}_n$ ($\times$ denotes the Cartesian product; $\mathbf{\Lambda}_i$ denote the search space of the i-th hyperparameter). The size of search space grows exponentially with the number of hyperparameters. On the other hand, if we treat the selection of hyperparameters as a sequential decision process where the agent selects hyperparameter one by one, and the agent select a new hyperparameter based on the result of the previous decisions, the search space is greatly reduced to $\mathbf{\Lambda}' = \mathbf{\Lambda}_1 \cup \mathbf{\Lambda}_2 \cup \ldots \mathbf{\Lambda}_n$. The size of the search space grows linearly. Obviously, the latter can greatly reduce the search space, thereby improve the optimization efficiency. Besides, the order in which the agent chooses the hyperparameters does not affect the final performance. We will study the influence of the order of selecting hyperparameters in the Section 6.7.

The RL-based method works as follows: for a given task, the agent first chooses n hyperparameters one by one. Then, the machine learning model with the selected hyperparameters is trained on the training set $\boldsymbol{D}_{train}$. The accuracy on the validation set $\boldsymbol{D}_{valid}$ is used as a reward signal to update the parameters of the agent by an RL algorithm. As a result, the agent learns how to tune hyperparameters over time.

3.2. MDP formulation

The sequential nature of HPO decisions allows us to formulate the problem as Markov decision processes (MDPs) and cast it in an RL framework. Since there are n hyperparameters to optimized, the horizon over which the agent will act is n. We formulate the HPO problem as an MDP with a 5-tuple $\langle S, A, P, R, \gamma \rangle$ with the following conditions:

- A is the set of all valid actions, a_t corresponds to a hyperparameter λ_t, which is sampled from the distribution $\mathscr{D}_t(\lambda_t)$ output by the agent.
- S is the set of all valid states. Since the agent sequentially selects hyperparameter and makes decision based on the previous decisions, i.e., $s_t = \mathscr{D}_{t-1}(\lambda_{t-1})$.
- R is the reward function. Since we use the accuracy of a validation set as a reward signal to update the agent, the undiscounted reward is obtained only after the final action. Thus, $r_t = 0$ for $t \in [1, n)$ and $r_n = accuracy$, where $accuracy$ denotes the validation performance of the algorithm with selected hyperparameters according to $a_{1:n}$.
- $P : S \times A \rightarrow \mathscr{P}(S)$ is a transition probability function, which is unknown for our problem.
- γ is a discount factor, and $\gamma = 1$.

3.3. Design of the agent

Fig. 1 presents the main structure and the workflow of the agent. The agent consists of an input embedding layer, an output embedding layer and a LSTM network, which is the core part of the agent for remembering previous decisions. The input embed-

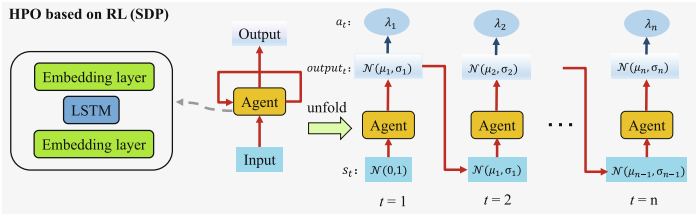


Fig. 1. Overview of the HPO based on RL. The agent selects hyperparameters sequentially.

ding and out embedding are composed of multilayer perceptron (MLP). The input is converted to a high-dimensional representation by an input embedding layer, which allows the agent to better observe the representation. The output of the LSTM is converted to a low-dimensional representation by an output embedding layer. The output of the output embedding layer is then fed to the input embedding layer in the next time-step. The core network of the agent consists of three LSTM layers. Although it is difficult to train the LSTM network, the LSTM cell has been indicated to be a powerful structure in solving the sequential problem. In addition, the LSTM network can discover conditionality in the configuration space.

At each episode k, the agent iterates n time-steps to sequentially select all the hyperparameters, where n is the number of hyperparameters. It is noted that the order of selecting hyperparameters is random and will not affect the final performance (We will prove that in Section 6.7). During the process of sequential selection of hyperparameters, the agent outputs a distribution $\mathscr{D}_t$ for hyperparameter λ_t at each time step t. Following [36,37], we use the normal distribution to represent the distribution of the hyperparameter, i.e., the output of the agent is $\mathscr{D}_t = \mathcal{N}(\mu_t, \sigma_t)$. The output $\mathcal{N}(\mu_t, \sigma_t)$ is fed to the agent at the next iteration $t+1$, i.e., $s_{t+1} = \mathcal{N}(\mu_t, \sigma_t)$, where $t \in [1, n-1]$. And the initial state $s_1 = \mathcal{N}(0, 1)$.

In this way, the selection process of the agent matches the sequence decision process very well. That is, the agent selects hyperparameter one by one while considering the conditionality among the configuration space by remembering previous decisions. Obviously, the search space of the HPO problem is greatly reduced at each time step.

3.4. Sampling from $\mathcal{N}(\mu, \sigma)$

The value of a hyperparameter λ is determined by sampling from the distribution $\mathcal{N}(\mu, \sigma)$. Random sampling causes large variations during training since the ranges of the possible values of hyperparameters are significantly different, e.g., for the extreme gradient boosting [38], the range of *n_estimators* is from 50 to 1200 and *learning_rate* varies from 0.001 to 0.1 (as shown in Table 1). As a result, it is difficult for the agent to efficiently explore the search space. To solve the problem, the following customized normalization procedure is taken:

- Scale the means of the distributions μ to μ' by the *tanh* function in the range $(-1, 1)$;
- Sample values h from the new distributions $\mathcal{N}(\mu', \sigma)$;
- Scale h into the range of value $[h_L, h_U]$ by the following method:

$$h' = h_L + (h_U - h_L) \times (1 + h)/2 \tag{4}$$

$$\lambda = clip\_and\_convert(h', h_L, h_U) \tag{5}$$

where h_U and h_L represent the upper and lower bounds of a hyperparameter Table 1, respectively. The *clip_and_convert* function removes the value h' outside of the interval $[h_L, h_U]$ and performs a type of conversion. Thus, we can handle continuous or discrete hyperparameters. Through the above operations, the training process becomes more efficient.

3.5. Training the agent

A policy π is a rule used by an agent to decide what actions to take. Suppose that θ represents the parameter of a policy π_θ. A trajectory τ is a sequence of states and actions: $\tau = (s_1, a_1, \ldots, s_n, a_n)$, which corresponds to the process of sequentially selecting hyperparameters. The return is the cumulative reward over a trajectory $R(\tau) = \sum_{t=1}^{n} r_t$. The goal of the agent is to find the θ that can maximize the expected return:

$$\max_\theta J(\pi_\theta) = arg\max_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \tag{6}$$

We choose a policy optimization method to train the agent since it works in a more stable and fast manner. The algorithm optimizes θ directly by applying a gradient ascent algorithm to $J(\pi_\theta)$:

$$\theta_{k+1} = \theta_k + \alpha \nabla_{\theta_k} J(\pi_{\theta_k}) \tag{7}$$

$\nabla_\theta J(\pi_\theta)$ is given by:

Table 1
Search space of hyperparameters for XGBoost and a CNN net, *Lower* and *Upper* denote the upper and lower bounds of a hyperparameter, respectively.

Alg.	Hyperparameter	*Lower*	*Upper*	Alg.	Hyperparameter	*Lower*	*Upper*
XGBoost Algorithm	max_depth	1	25	CNN	batch size	24	128
	learning_rate	0.001	0.1		convolution stride(2)	1	5
	n_estimators	50	1200		convolution kernel(2)	2	5
	gamma	0.05	0.9		convolution channel(2)	24	128
	min_child_weight	1	9		pooling kernel(2)	2	5
	subsample	0.5	1.0		pooling stride(2)	1	5
	colsample_bytree	0.5	1.0		pooling type(2)	0	1
	colsample_bylevel	0.5	1.0		fc layer nodes(2)	128	1100
	reg_alpha	0.1	0.9		learning rate	0.001	0.05
	reg_lambda	0.01	0.1		–	–	–

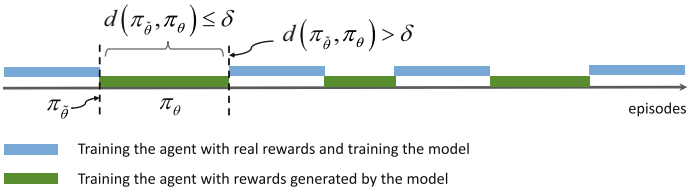


Fig. 2. The process of the model use.

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{n} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)] \tag{8}$$

Eq. (8) can be estimated by a sample mean:

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{n} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau_i) \tag{9}$$

where $\tau_i, i = 1, \ldots, m$ is a set of trajectories.

However, the small differences of updates in θ can have very large differences in the performance. It is very dangerous to use a large step size that can collapse the whole system. To solve this problem, a newly developed policy optimization algorithm, proximal policy optimization (PPO), constrains the updates of θ so that the new policies are close to the old ones [39]. In this paper, we use the PPO-clip method to update θ. The objective function of the PPO-clip method is defined as:

$$\max_\theta(J(\pi_\theta)) = arg\max_\theta \mathbb{E}_{s,a\sim\pi_{\theta_k}}[L(s, a, \theta_k, \theta)] \tag{10}$$

where L is given by:

$$\begin{aligned} L(s, a, \theta_k, \theta) = \; & min(\tfrac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \\ & clip(\tfrac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_k}}(s, a)) \end{aligned} \tag{11}$$

where ϵ is a hyperparameter that controls the change to the new policy from the old policy, $\epsilon = 0.2$. A is the advantage function, which is defined as $A^{\pi_{\theta_k}} = R(\tau_k) - b$, where $R(\tau_k)$ denotes the accuracy of the k^{th} sample (configuration) and b is an exponential moving average of the accuracy of the previous samples. b is also called the baseline function, which is used to reduce the training variance.

4. Efficiency improving based on a model

According to the workflow of our method, the given algorithm with selected hyperparameters is trained on $\boldsymbol{D}_{train}$, and the cross-validation result is used as a reward to update the agent. It is noted that the evaluation process is very time consuming, which makes the learning process highly inefficient. To accelerate the learning process, a model is used to predict the performance of the selected configuration rather than training the algorithm. In this section, we describe the techniques in detail, including how to design and train the model and how to control the number of times that the model use.

4.0.1. Structure and training of the model

The input to the model is the selected configuration and the output is the predictive performance on the validation set. An MLP is employed to construct the model, and we train it by supervised learning methods. We choose a simple structure for the model since training a complex model requires more samples and time. Theoretically, more samples are provided, and the model is more accurate. However, we cannot afford an adequate number of samples to train the model because of the cost. Considering several factors, the following principle is followed: the number of training samples is generally 5–10 times the number of parameters of the model.

4.0.2. Controlling the model use

A model that predicts the performance of the selected configuration can accelerate the learning process. However, the model inaccuracy is further exacerbated by the long-term use, resulting in considerable noise in the parameter space and collapse performance (as shown in Fig. 3). Although the noise may have risk, it was shown by [40] that the addition of parameter noise in a reasonable range leads to better exploration and obtains a policy that exhibits a larger variety of behaviors. Hence, we limit the influence of the model on the policy by dynamically controlling the horizon (the number of times) of the model use.

A straightforward method is to discipline the model use in the short-term horizon. After the short-term model use, the model is trained again with new samples. The model training and the model use are alternately repeated. However, the horizon of the model use cannot be fixed since the inaccuracy of the model varies over time. Additionally, it is far harder to estimate the horizon of the model use. We novelly propose a simple solution that resolves all of the aforementioned difficulties in an easy and straightforward way (as shown in Fig. 2). This is achieved by relating the horizon of the model use to the degree that the policy is perturbed. More specifically, we can define a distance measure between policies before and after the model use in the action space:

$$d(\pi_{\tilde{\theta}}, \pi_\theta) = D_{KL}(\pi_{\tilde{\theta}}||\pi_\theta) = \sum_{t=1}^{n} D_{KL}(\pi_{\tilde{\theta}}(a_t|s_t)||\pi_\theta(a_t|s_t)) \tag{12}$$

where $\pi_{\tilde{\theta}}$ and π_θ denote policies before and after the model use, respectively (as shown in Fig. 2). $\pi_\theta(a_t|s_t) = \mathcal{N}(\mu_t, \sigma_t)$ and $\pi_{\tilde{\theta}} = \mathcal{N}(\tilde{\mu}_t, \tilde{\sigma}_t)$ represent the conditional probability distribution of the action of policies π_θ and $\pi_{\tilde{\theta}}$, respectively.

We use the KL divergence between π_θ and $\pi_{\tilde{\theta}}$ to measure the impact of the model use on the policy. To limit it in a proper range, set:

$$d(\pi_{\tilde{\theta}}, \pi_\theta) \leqslant \delta \tag{13}$$

where $\delta > 0$ is a threshold value and we will analyze the effect of δ on the final performance and recommend a value in the experiments. In this way, the model use can adapt to changes in policy and is dynamically controlled.

If $d(\pi_{\tilde{\theta}}, \pi_\theta) > \delta$, the change in policy is too large, i.e., the inaccuracy of the model increases and the model needs to be retrained. The model training and model use are repeated alternately (see Fig. 2). Therefore, we can take full advantage of the predictive model to highly improve the learning efficiency.

5. Overall framework

To clarify the proposed method, we present the entire optimization process. The RL-based method works as follows: for a given task, the agent chooses n hyperparameters one by one based on

the previous decisions. Then, the machine learning model with the selected hyperparameters is trained on a training set $\boldsymbol{D}_{train}$. The accuracy of a validation set $\boldsymbol{D}_{valid}$ is used as a reward signal to update the parameters of the agent by an RL algorithm. In the overall framework, the following two processes are performed alternately (see Algorithm 1): train the agent with the real rewards and train the model (lines 2 to 8); train the agent with the reward generated by the model to accelerate learning (lines 9 to 12). We dynamically control the horizon of the model use by limiting the distance of policies before and after the model use in parameter space, i.e., $d(\pi_{\tilde{\theta}}, \pi_\theta) \leqslant \delta$.

Since RL method uses an iterative training process, the time complexity is approximated as $O(n \times t)$, where n is the number of iterations, and t is the time for each iteration. To precisely analyse the time complexity, we define the following symbols: t_{real} denotes the time taken at each iteration without the model use; t_{use} is the time taken at each iteration with model use, obviously, $t_{use} < t_{real}$; t_{train} is the time for training the predictive model. Without the model use, the algorithm roughly takes $n \times t_{real}$. When using the predictive model, the time decreases sharply, which is approximated as: $t_{real} \times n_{real} + t_{use} \times n_{use} + t_{train} \times n_{train}$, where n_{real} and n_{use} refer to the number of times without model use and with model use, respectively, $n = n_{real} + n_{use}$; n_{train} denotes the number of times the model to be trained, which is a small value. It can be clearly seen that for complex model or large task, $t_{use} \ll t_{real}$, the efficiency of the whole process is improved. We will conduct further analysis through experiments in Section 6.8 and demonstrate the effectiveness of the model.

Algorithm 1: Overall Framework

Input: Initialize policy $\pi_\theta, \pi_{\tilde{\theta}}$.
Initialize the predictive model $\mathscr{F}$.
Initialize data set $\mathscr{D}_{\mathscr{F}} = \varnothing$.
Procedure:
1: **while** not done bf do
2: **while** $\mathscr{D}_{\mathscr{F}}$ not full **do**
3: Agent selects a configuration $\boldsymbol{\lambda}$ and obtains the accuracy r on $\boldsymbol{D}_{valid}$
4: Use $(\boldsymbol{\lambda}, r)$ to update π_θ by PPO
5: Add $(\boldsymbol{\lambda}, r)$ to $\mathscr{D}_{\mathscr{F}}$
6: **end while**
7: Save the current policy, $\pi_{\tilde{\theta}} = \pi_\theta$
8: Fit the model $\mathscr{F}$ with $\mathscr{D}_{\mathscr{F}}$
9: **while** $d(\pi_{\tilde{\theta}}, \pi_\theta) \leqslant \delta$ **do**
10: Agent selects a configuration $\boldsymbol{\lambda}$ and the model predicts the accuracy $\hat{r}$ on $\boldsymbol{D}_{valid}$
11: Use $(\boldsymbol{\lambda}, \hat{r})$ to update policy π_θ by PPO
12: **end while**
13: **end while**

We believe that the reasons that a model can effectively accelerate the training while ensuring the improvement of performance are as follows:

- During the process of training the agent, we employ a predictive model to directly evaluate the performance of hyperparameter configuration rather than obtaining the reward by training the algorithm, which speeds up the training process;
- We use the KL divergence between π_θ and $\pi_{\tilde{\theta}}$ to dynamically control the model use, which avoids the model bias problem [29] and improves the performance of the proposed method.

6. Experiments

In this section, we will verify the performance of the proposed method by applying it to tune the hyperparameters of a well-known model - the extreme gradient boosting (XGBoost) and a convolutional neural network on 101 tasks (datasets). Five main questions about the proposed method that we will investigate are as follows:

- Firstly, can the proposed method achieve better optimization performance than other state-of-the-art methods and the baseline?
- Is the sequential selection of hyperparameters feasible and effective? In other words, is the sequential selection of hyperparameters more advantageous than directly output configuration in one step?
- How would the performance of the proposed method be affected if the order in which the hyperparameters are selected is random on each trial?
- Can the use of the model improve the time efficiency while ensuring the optimization results?
- How does δ affect the final performance? and what are the advantages of dynamically controlling the model use by adjusting δ?

6.1. Datasets

To evaluate the robustness and generalization of the proposed method on a broad range of datasets, we gather 101 binary and multiclass classification datasets with different scales[1] from the OpenML [41] and UCI repositories as the target tasks. The size of datasets ranges from thousands to hundreds of thousands. Importantly, these datasets cover a diverse range of applications, such as digit and letter recognition, and other classification tasks for specific scenarios. The 5-fold cross-validation method is used for training, and the test set results are used to verify the performance of the hyperparameter configuration. Specifically, for small datasets (the number of instances is less than 10,000), the partition ratio is 8 (training set)/2 (test set), and the partition ratio is 9 (training set)/1 (test set) for big datasets (the number of instances is larger than 10,000).

6.2. Comparison methods

In our experiments, the proposed method is called MBRL-SDP (MBRL: model-based RL; SDP: sequential decision process), which uses an RL agent to tune hyperparameters sequentially and employs a model to accelerate training. We also propose other two variants of our method to validate the various components of our method. First, RL-SDP employs an RL agent to select hyperparameters sequentially but without using a model. By comparing MBRL-SDP and RL-SDP, we can demonstrate the effectiveness of the model. Second, RL-DOC (DOC: directly output configuration) method directly outputs the whole hyperparameters in one step, and the process of the agent selecting hyperparameters is shown in Fig. 4. The other settings are the same as RL-SDP method. We will study whether sequential decisions can improve optimization performance by comparing RL-SDP and RL-DOC. In addition, we compare our proposed method with other optimization methods: random search method[2], two Bayesian optimization-based methods TPE[2] and Spearmint[3], a well-known evolutionary method CMA-ES[4], a

[1] https://github.com/uestc-chensp/datasets
[2] Code from https://github.com/hyperopt/hyperopt/
[3] Code from https://github.com/fmfn/BayesianOptimization/
[4] Code from https://github.com/AIworx-Labs/chocolate

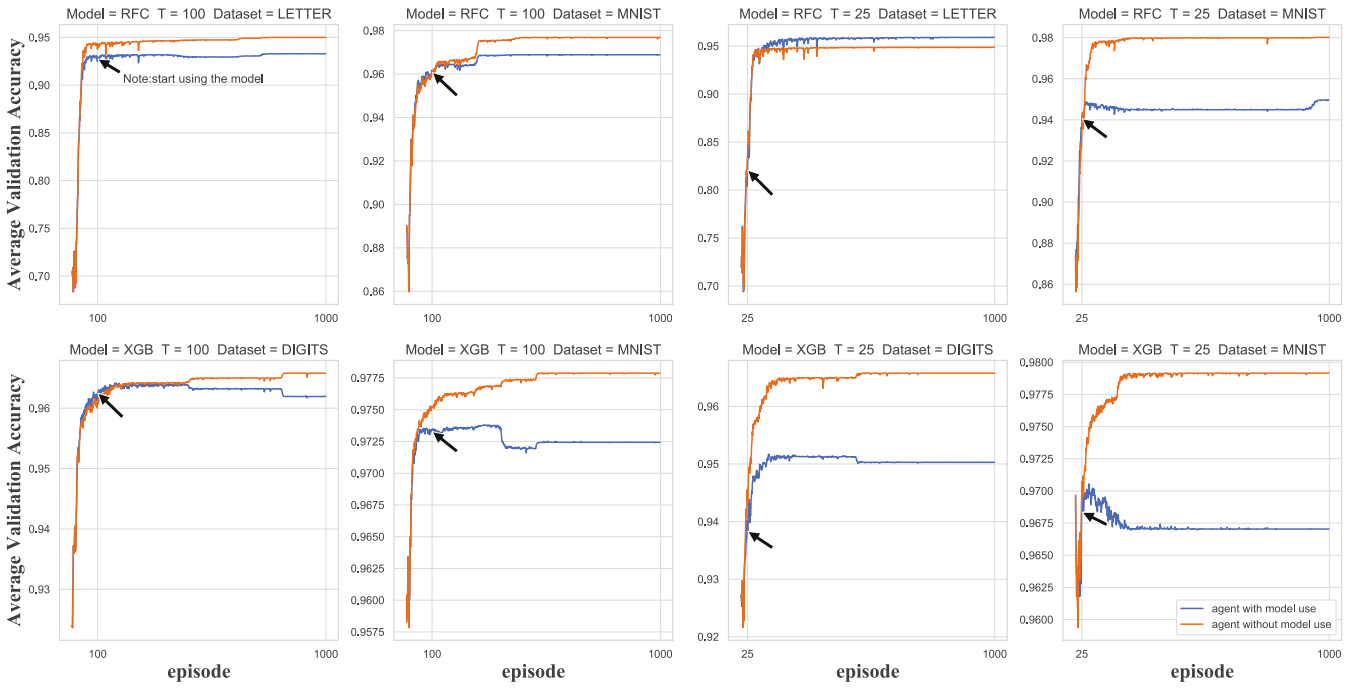


Fig. 3. Comparison between RL agents with and without long-term model use. "Agent without model use" denotes the learning curve of RL agent without model use during 1000 episodes training. "Agent with model use" presents the learning curve of RL agent with long-term model use from the 25^{th} or 100^{th} episode (indicated by a black arrow) to the 1000^{th} episode. "T" is the number of samples (episodes) taken to train the predictive model.

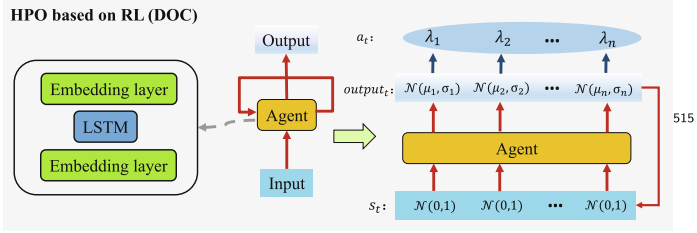


Fig. 4. This figure shows the hyperparameter selection process of RL-DOC method. The agent of RL-DOC directly outputs the whole configuration at one step. Suppose there are n hyperparamters to optimize. The inputs to the agent are n standard normal distributions $\mathcal{N}(0,1)$; the outputs of the agent are n normal distributions corresponding to the distribution of n hyperparmeters.

bandit-based method Hyperband[5], a recent method BOHB [5] combining bandit-based and Bayesian optimization, and the default configuration of XGBoost (Baseline).

6.3. Evaluation criterion

We follow the evaluation criterion of the state-of-the-art works [4,42], and use the test performances as the basic evaluation criterion of the optimization methods. In addition, we also evaluate the time to achieve the best optimization result in a fixed number of samples, which denotes the time efficiency of the optimization methods. To show the performance difference more clearly, a statistic measure, *Average Rank*, is used for evaluating the rank of the test performance:

$$Average\ Rank = \frac{1}{D}\sum_{i\in D} rank_i \tag{14}$$

where D represents the number of datasets evaluated and $rank_i$ is the ranking of the algorithm with the test performance achieved for a particular dataset i.

6.4. Experimental details

The hidden layers of the predictive model consists of 3 dense layers, and each layer contains 16, 16 and 8 nodes, respectively. The number of nodes in the input layer is the number of hyperparameters of the model to be optimized. The output layer has only one node to output a predictive performance. The samples provided for training the predictive model were 80 and 240, for the XGBoost and the convolutional neural network optimization tasks, respectively. Since the convolutional neural network has more hyperparameters to be optimized, we need more samples to prevent the predictive model from underfitting. For the agent, the input embedding layer has 35 nodes, the output embedding layer has 2 nodes, the LSTM network consists of 3 layers, and each layer contains 35 nodes. The settings of δ and the learning rate for each optimization task are 0.4 and 0.007, respectively. We employ the

[5] Code from https://github.com/automl/HpBandSter

Table 2
The average rank of test set results over 101 datasets (300 samples). "*" and "+" denotes that the differences between ours and other methods are statistically significant with $p < 0.01$ and $p < 0.05$, respectively.

Measure	MBRL-SDP(ours)	BOHB	Hyperband	Spearmint	CMA-ES	TPE	Random	Baseline
Average rank	1.119(+)(*)(*) (*)(*)(*)(*)	2.495	5.277	3.782	3.168	5.713	6.891	7.554
Std	0.3799	1.4325	0.9023	1.1990	0.5982	1.1110	0.9217	0.4970

PPO algorithm to train the RL agent and use its default parameter settings.

6.5. Compare with other methods

6.5.1. HPO for XGBoost classifier

6.5.1.1. Search space. We chose to optimize the hyperparameters of an advanced classification algorithms, XGBoost, based on the following reasons: the XGBoost algorithm has recently been dominating the Kaggle competition; the performance of the algorithm is sensitive to the hyperparameter configuration. The code of XGBoost is based on scikit-learn [43]. Ten hyperparameters of XGBoost are chosen to be optimized (see Table 1).

6.5.1.2. Results. We first measured the *Average Rank* of each method on 101 optimization tasks (as shown in Table 2). In addition, we used the Friedman test to validate the statistical significance of differences between the evaluated methods [44]. The null hypotheses that the 8 methods perform the same and the observed differences are merely random was rejected with $p < 0.05$. Meanwhile, the Wilcoxon post hoc tests is applied and the results suggest that the differences between MBRL-SDP(ours) and other methods were found to be statistically significant with $p < 0.01$ or $p < 0.05$ (Table 2). In Table 2, we found that the average ranking of all the optimization methods are lower to the *Baseline*, which indicates that tuning hyperparameters indeed improves the performance. It is clear that MBRL-SDP outperforms the other methods both in performance and consistency (i.e., low average rank and small standard deviation). Importantly, our method is significantly better than random search method, which suggests that the RL agent learns how to turn hyperparameter. In addition to obtaining the top average ranking (1.119), our approach has achieved a perfect optimization results on 87 of the 101 tasks (see Table 3). As shown in Table 3, the number of tasks with top performance achieved by our method is significantly higher than any of the other methods and baseline. To further investigate the effectiveness of each method, Fig. 5 presents the number of tasks that each optimization method outperforms the *Baseline* on 101 tasks. It is clear to see that MBRL-SDP is better than *Baseline* on more tasks, which demonstrates that our method is much more effective in hyperparameter tuning processes.

Secondly, we measured the average ranking of runtime of each method on 101 optimization tasks (as shown in Table 4). The "runtime" denotes the time to achieve the best hyperparameter configuration in 300 samples. Meanwhile, we performed the same statistical test as before. We can see that MBRL-SDP also achieves the best time performance in a fixed number of samples, compared with other methods (i.e., low average ranking and small standard deviation).

To conclude, all these results show that the proposed method MBRL-SDP significantly improves the optimization performance for XGBoost in term of accuracy and runtime.

6.5.2. HPO for convolutional neural network

6.5.2.1. Search space. We study further to optimize a convolutional neural network with the main structure shown in Figure 6, which is similar to the one proposed by [45]. It includes two convolution layers (Conv1, Conv2), two pooling layers (Pool1, Pool2), and two fully connected layers (FC1, FC2). 16 hyperparameters are chosen to be optimized, including the stride size, kernel size, and channel size in each convolutional layer; the pooling type, kernel size, and stride size in each pooling layer; the number of hidden nodes in each fully connected layer; the batch size and the learning rate. The range of values of hyperparameters is shown in Table 1.

6.5.2.2. Results. We conducted experiments on *MNIST* [46] and *Fashion MNIST* [47] datasets, both of which are commonly used to evaluate the performance of convolutional neural networks. The results are presented in Table 5. The error rate of the one [45] is considered as the *Baseline*. We can see that the error rate of the *Baseline* is lower than some optimization methods, which indicates that *Baseline* is very competitive and some optimization methods can not work well in large hyperparameter space. MBRL-SDP (ours) outperforms other methods in terms of optimiza-

Table 3
Number of tasks with the top performance of each method over 101 tasks.

Method Name	Number of Datasets with Top Performance
BOHB	11 (10.9%)
Hyperband	2 (2.0%)
Spearmint	14 (13.9%)
CMA-ES	7 (6.9%)
TPE	1 (1.0%)
Random	0 (0.0%)
Baseline	0 (0.0%)
MBRL-SDP	**87 (86.1%)**

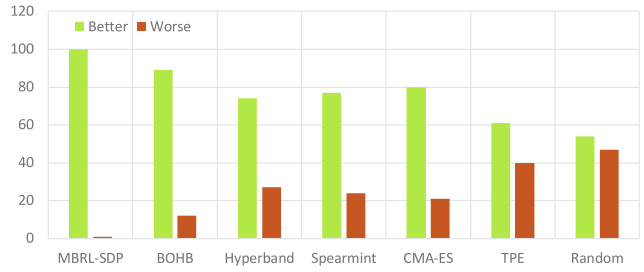


Fig. 5. The number of datasets that the evaluated method performs better or worse than *Baseline* (accuracy on the test set), across 101 datasets. "Better" denotes the evaluated method performs better than *Baseline* and "worse" means the evaluated method performs worse than *Baseline*.

Table 4
The average rank of runtime to achieve the best performance in 300 samples over 101 datasets. "*" and "+" denotes that the statistically significant difference from other values in the same line is $p < 0.01$ and $p < 0.05$, respectively.

Measure	MBRL-SDP	BOHB	Hyperband	Spearmint	CMA-ES	TPE	Random
Average rank	1.832 (*)(+) (*)(*) (*)(*)	3.198	2.574	6.257	6.178	4.683	3.277
Stdev	0.8568	1.5671	1.0749	0.8402	0.8489	1.0141	1.7299

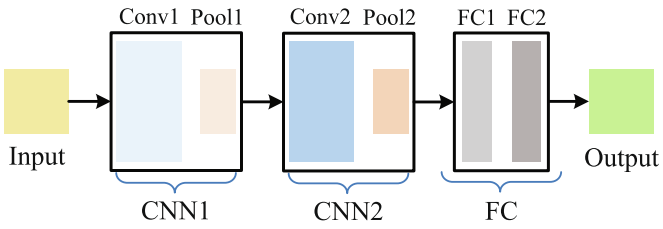


Fig. 6. Convolutional neural network structure.

tion results and standard deviation on both datasets. It is noted that MBRL-SDP is very competitive in time performance. The reason is that for most of the time, MBRL-SDP uses a predictive model to infer the accuracy of the CNN on the validation set without directly training it.

We used the Friedman test to verify the statistical significance among all methods. The null hypothesis that all the methods perform the same and the differences are merely random was rejected by $p < 0.05$.

6.6. Study of agent's structure

In this part, we would like to investigate whether the sequential selection of hyperparameters feasible and effective. We compared two RL-based methods, RL-SDP (output hyperparameters sequentially) and RL-DOC (directly output all hyperparameters in one step).

6.6.1. Experiments

The experiment is implemented on 12 tasks to optimize hyperparameters of XGBoost. The way of partitioning the dataset and training the algorithms are the same as before. Each method samples 300 times.

6.6.2. Results and analysis

The experimental results are presented in Fig. 7. Although RL-DOC performs well in some tasks, the performance fluctuates sharply and is quite unstable, even fails to work in some tasks (see Fig. 7-(d), (g), (j)). The experiment result indicates that RL-DOC may easily fall into suboptimal solution.

The above ablation experiments suggest that it is more reasonable to output hyperparameters sequentially. We believe that the reasons is as follows: if the search space is very large, it is difficult for RL-DOC to explore a good policy in such a large space. However, RL-SDP method outputs hyperparameters one by one and makes a new decision based on the previous ones, the search space reduces at each time-step. Therefore, it is much easier for the agent to handle the problem.

6.7. Effect of optimization order

6.7.1. Experiments

We aim to study whether the performances of the proposed method is affected by the order in which the hyperparameters are selected. In this experiment, we use RL-SDP method to optimize hyperparameters of XGBoost over 12 datasets of different scales. Three different orders for selecting hyperparameters are randomly set (as shown in Table 6). The performances of RL-SDP method with different orders are compared in Table 7.

6.7.2. Results and analysis

We can see from Table 7 that the performances of RL-SDP with three different orders are similar. The results show that the order of selecting hyperparameters will not affect the final optimization performance. The reason is that:

Let $P(X_1X_2 \ldots X_n)$ denote the probability of selection a hyperparameter configuration, where X_i denote the random variable of selection a hyperparameter λ_i. Since the agent selects hyperparameters one by one, where the probability of selection one hyperparameter depends on hyperparameters already selected earlier, $P(X_1X_2 \ldots X_n)$ equals:

$$\begin{aligned} P(X_1X_2 \ldots X_n) &= P(X_1)P(X_2|X_1) \ldots P(X_n|X_1 \ldots X_n) \\ &= P(X_2)P(X_3|X_2) \ldots P(X_1|X_2 \ldots X_n) = \ldots \end{aligned} \tag{15}$$

The second line in Eq. (13) indicates that the order of selecting hyperparamters is $X_1 \rightarrow X_2 \rightarrow \ldots X_n$ and the third line denotes the order is $X_2 \rightarrow X_3 \rightarrow \ldots X_1$. We can easily see that the order of selecting hyperparameters will not affect the final result.

6.8. Effectiveness of the model

6.8.1. Experiments

In this section, we compared the performance of MBRL-SDP (with model use) with RL-SDP (without model use) on 20 optimization tasks to verify the effectiveness of the model. The performance results are presented in Table 8. Each value reports the average performance of 5 independent experiments. Each method samples 300 times in each experiment.

6.8.2. Results and analysis

In Table 8, it is clear that the time to achieve the best performance of MBRL-SDP is lower than RL-SDP on all tasks, which suggests that directly evaluating the reward using the model can speed up the tuning process. It is noted that MBRL-SDP can reduce runtime by more than half on big scale dataset, such as MNIST and Fashion MNIST. However, the advantage of time efficiency is not obvious on small scale task, like *Anuran C* and *Breast C*. To further analyze the reason, we define the following symbols: t_{real} refers to

Table 5
Optimizing hyperparameters of a convolutional neural network on the MNIST and Fashion MNIST datasets. "err" means the test set error. "std" means the standard deviation of the test set error over multiple experiments. "time" denotes the time to achieve the best optimization results. Each value reports the average performance of 5 independent experiments. Each method samples 300 times in each experiment. Values in bold denote the best values. Friedman test shows that the differences between our method and all other methods were found to be statistically significant with $p < 0.05$.

Methods	MNIST			Fashion MNIST		
	err	std(%)	time(h)	err	std(%)	time(h)
MBRL-SDP	**0.0051**	**0.64**	**6.04**	**0.1065**	**1.27**	**5.35**
BOHB	0.0057	1.99	19.37	0.1080	2.67	6.44
Hyperband	0.0101	3.41	20.54	0.1516	3.80	7.38
Spearmint	0.0068	0.98	44.72	0.1354	1.73	15.18
CMA-ES	0.0099	7.07	25.49	0.1097	1.79	6.71
TPE	0.0101	2.70	41.28	0.1418	4.64	13.31
Random	0.0104	7.74	27.91	0.1903	5.93	13.95
Baseline	0.0073	–	–	0.1074	–	–

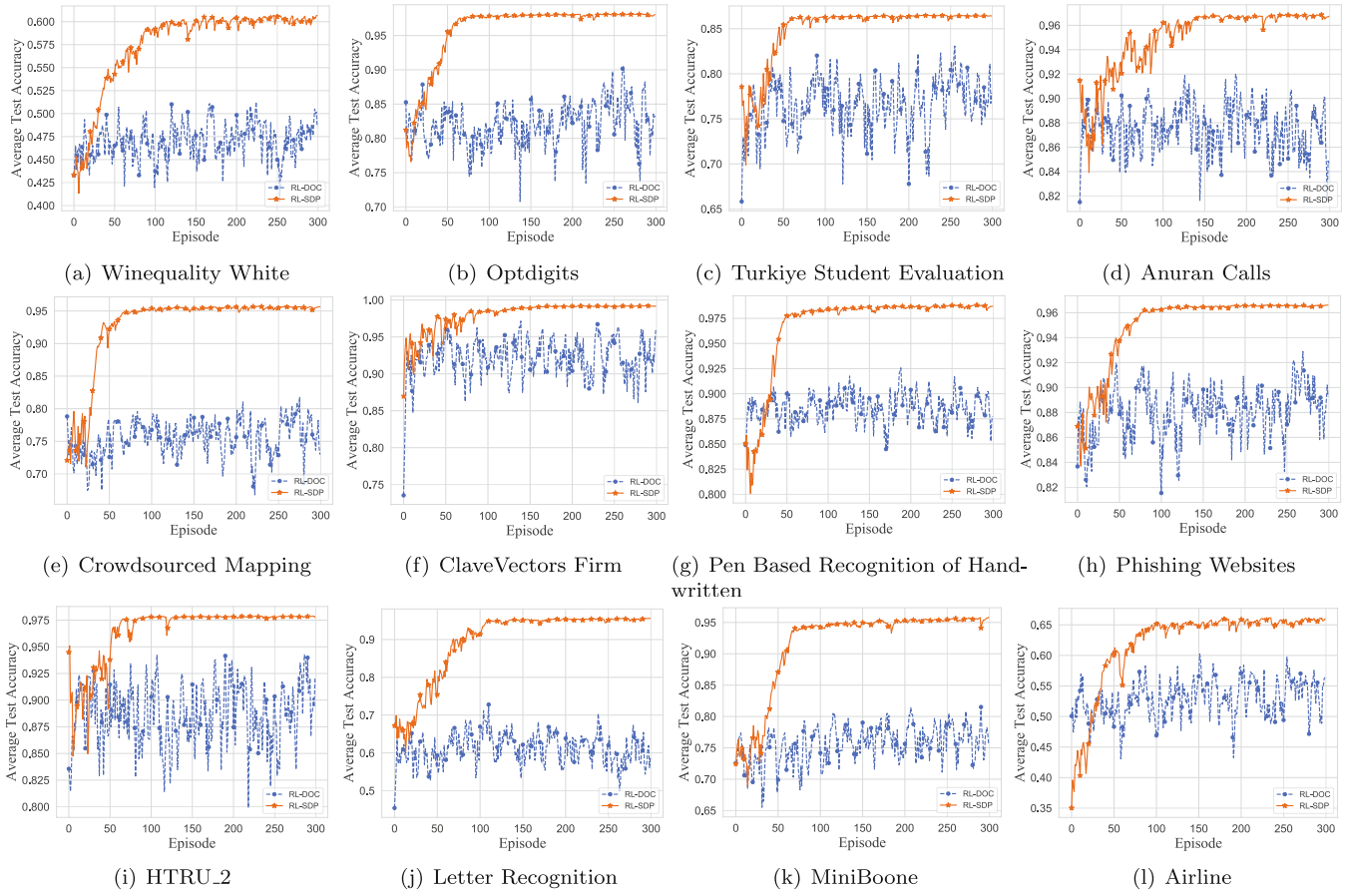


Fig. 7. Comparison between RL-SDP and RL-DOC on the 12 target tasks. Each figure shows the test performance in 300 episodes.

Table 6
Random order of selecting hyperparameters of XGBoost algorithm

No.	Hyperparameter Optimization Order
Order 1	⇒max_depth ⇒n_estimators ⇒min_child_weight ⇒colsample_bytree ⇒reg_alpha ⇒learning_rate ⇒gamma ⇒subsample ⇒colsample_bylevel ⇒reg_lambda
Order 2	⇒learning_rate ⇒gamma ⇒subsample ⇒colsample_bylevel ⇒reg_lambda ⇒max_depth ⇒n_estimators ⇒min_child_weight ⇒colsample_bytree ⇒reg_alpha
Order 3	⇒colsample_bytree ⇒reg_alpha ⇒learning_rate ⇒max_depth ⇒n_estimators ⇒min_child_weight ⇒gamma ⇒subsample ⇒colsample_bylevel ⇒reg_lambda

the time taken for optimizing hyperparameters without model use; t_{train} refers to the time taken by training the prediction model, and t_{use} refers to the time taken by using the prediction model. Therefore, the time saved is $t_{save} = t_{real} - t_{train} - t_{use}$, where t_{real} is determined by the size of the optimization task (dataset size and number of hyperparameters). Obviously, this value is big on large scale task. t_{train} and t_{use} are determined by the size of the predictive model and are almost the same for different tasks. Therefore, for small scale tasks, the sum of t_{train} and t_{use} is comparable to t_{real}, and $t_{save} \approx 0$ or even $t_{save} < 0$; while for large scale tasks, $t_{train} + t_{use}$ is much smaller than t_{real}, and the model use saves a lot of time.

Table 7
Performance of RL-SDP with three random orders of selecting hyperparameters. We performed 5 independent runs of RL-SDP with each order. "err" and "time" are similar to those of Table 5.

No.	Order1		Order2		Order3	
	err	time(h)	err	time(h)	err	time(h)
Winequality W.	0.3918	2.90	0.3932	2.96	0.3901	2.84
Optdigits	0.0285	2.80	0.0277	2.86	0.0281	2.64
Turkiye S.E.	0.1330	2.82	0.1327	2.90	0.1337	2.74
Anuran C.	0.0283	0.82	0.0280	0.85	0.0283	0.83
Crowd S.M.	0.0430	5.85	0.0451	5.62	0.0446	5.74
Clave V.F.T.	0.0009	0.29	0.0007	0.28	0.0009	0.29
Pen B.R.H.	0.0129	3.62	0.0131	3.79	0.0133	4.21
Phishing W.	0.0390	1.91	0.0340	2.33	0.0357	1.85
HTRU_2	0.0205	1.30	0.0221	1.26	0.0208	2.87
Letter.R	0.0653	7.17	0.0668	7.23	0.0652	6.90
Miniboone	0.0329	13.36	0.0323	10.52	0.0339	12.90
Airline	0.3295	15.18	0.3279	13.55	0.3303	15.73

Table 8
Comparison between MBRL-SDP (with model use) and RL-SDP (without model use) on 20 optimization tasks. The definitions of "err", "std" and "time" are the same as those of Table 5. The value in bold indicates the best value between the two.

Alg.	Dataset	MBRL-SDP			RL-SDP		
		err	std	time	err	std	time
XGBoost	Anuran C.	**0.0234**	**0.11%**	**0.72 h**	0.0283	0.14%	0.82 h
	Breast C.	**0.0294**	**0.11%**	**0.26 h**	0.0297	0.26%	0.29 h
	Car E.	**0.0327**	**0.23%**	**3.00 h**	0.0379	0.49%	3.92 h
	CTG	**0.1364**	**0.25%**	**1.75 h**	0.143	0.29%	2.75 h
	Clave V.F.T.	**0.0003**	**0.01%**	**0.15 h**	0.0009	0.02%	0.29 h
	Crowd S.M.	**0.0423**	**0.01%**	**2.92 h**	0.0430	0.10%	5.85 h
	Diabetic R.D.	**0.3227**	**0.06%**	**0.23 h**	0.3321	0.56%	0.54 h
	Digits	0.0412	**0.35%**	**0.23 h**	**0.0326**	0.40%	0.78 h
	HTRU2	0.0207	**0.02%**	**0.06 h**	**0.0205**	0.03%	1.30 h
	Image S.	0.0067	**0.03%**	**0.96 h**	**0.0066**	0.05%	1.82 h
	Iris	**0.0417**	**0.04%**	**0.04 h**	0.0424	0.48%	0.08 h
	Letter R.	**0.0618**	**0.21%**	**3.49 h**	0.0653	0.25%	7.17 h
	Optdigits	**0.0248**	**0.16%**	**2.17 h**	0.0285	0.19%	2.80 h
	Pen B.R.H.	**0.0096**	0.10%	**1.48 h**	0.0129	**0.06%**	3.62 h
	Phishing W.	**0.0379**	**0.04%**	**1.68 h**	0.0390	0.22%	1.91 h
	Turkiye S.E.	**0.1322**	0.16%	**0.82 h**	0.1330	**0.12%**	2.82 h
	Wilt	**0.0248**	**0.11%**	**0.21 h**	0.0287	0.20%	0.46 h
	Winequality W.	**0.3836**	**0.17%**	**2.01 h**	0.3918	0.32%	2.90 h
CNN	MNIST	**0.0051**	**0.64%**	**6.04 h**	0.0085	1.39%	16.35 h
	Fashion MNIST	**0.1065**	**1.27%**	**5.35 h**	0.1075	2.05%	12.53 h

Table 9
Study the influence of δ on the accuracy and the runtime. "err" and "time" are the same as those of Table 5.

Algorithm-Dataset	δ=0.1		δ=0.2		δ=0.3		δ=0.4		δ=0.5		δ=0.6		δ=0.7		δ=0.8		δ=0.9	
	err	time (h)	err	time (h)	err	time (h)	err	time (h)	err	time (h)	err	time (h)	err	time (h)	err	time (h)	err	time (h)
XGB-Letter	0.0621	10.2	0.0618	9.4	0.0624	6.7	0.0618	3.5	0.0741	3.1	0.0886	2.9	0.0891	2.4	0.0910	2.1	0.0974	2.0
CNN-Fashion MNIST	0.1070	10.6	0.1065	8.4	0.1067	6.8	0.1065	5.4	0.1174	4.7	0.1209	4.1	0.1281	3.3	0.1309	3.1	0.1343	2.7

In addition, we can see that the test error of MBRL-SDP is lower than RL-SDP on most tasks, which indicates that the inaccuracy of the model does not degrade performance; in contrast, the added noise to the parameter space contributes to explore better hyperparameter configurations. Moreover, MBRL-SDP also shows the lowest standard deviation value on the most tasks, which suggests that MBRL-SDP runs more stable.

6.9. Advantages of dynamical control of model use

6.9.1. Experiments

In this part, we conducted two experiments to demonstrate the advantage of dynamical control of model use:

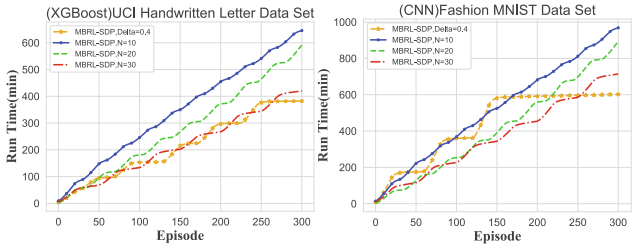
- According to Eq. (13), δ is a quite important hyperparameter to control the horizon of model use. A series of experiments with different values of δ are implemented to study how δ affects the final performance.
- We compare our method with the one that fixes the horizon of model use to demonstrate that the dynamical control of model use has more advantages. Here, we set three different values for the horizon of model use, $N = \{10, 20, 30\}$.

6.9.2. Results and analysis

Table 9 shows the experiment results of different values of δ. We can see that δ does affect the final performance and the runtime. Firstly, it is clear that δ does not greatly affect the optimization results when $\delta \in$ [0.1, 0.2, 0.3, 0.4]. When $\delta > 0.4$, the optimization results become worse. Secondly, the runtime gradually decreases as δ increases.

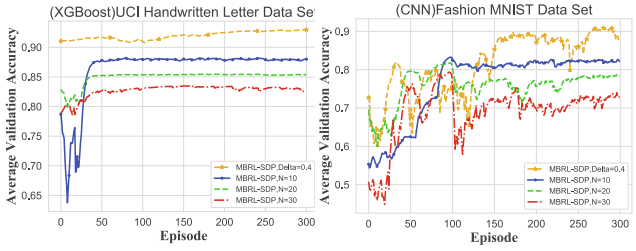
We empirically analyze the above experimental results. δ is a threshold that determines how long or how many times the model can be used. If δ is too small, the change of policy is limited to a small range. That means the horizon of the model use is short, and the model is frequently updated, resulting in time-consuming but the final accuracy can be guaranteed. Conversely, large δ value allows a big change of policy. That means the horizon of the model use is long, and the inaccuracy of the model may be exacerbated, resulting in poor performance of the optimization. However, the training time can be greatly saved. In fact, δ balances the runtime and optimization results. According to our experiments, the optimal balance of runtime and performance can be achieved near 0.4.

Figs. 8 and 9 present the comparison results between the dynamic horizon of model use and the fixed one. Compared with the fixed-horizon of model use, the dynamic control method has more advantages in terms of runtime and optimization performance. Firstly, it is noted in Fig. 8 that the runtime curve of the method with fix horizon rises regularly, i.e., the training (the rising part of the curve) and use (the flat part of the curve) of the model alternate strictly. In contrast, the horizon of dynamic model use varies in length. This is because our method dynamically adjusts the horizon of model use according the change on the policy. If the change is less than δ, the agent continues to use the model to obtain samples. Therefore, the dynamic adjustment method reduces runtime significantly. Secondly, we found in Fig. 9 that the method with dynamic controlling of horizon achieves a higher accuracy than the method with fixed-horizon. Besides, for the method with fixed-horizon, the final accuracy decreases as N increases. The reason is that as the inaccuracy of the model



(a) XGBoost (b) CNN

Fig. 8. Runtime of optimizing hyperparameters of XGBoost and CNN over UCI Handwritten letter dataset and Fashion MNIST dataset, respectively. Two methods of controlling horizon of model use is compared: δ values is 0.4 for the dynamic one. The fixed one sets the horizon of model use N, where $N = \{10, 20, 30\}$.



(a) XGBoost (b) CNN

Fig. 9. Average validation accuracy of optimizing hyperparameters of XGBoost and CNN over UCI Handwritten letter dataset and Fashion MNIST dataset, respectively. Two methods of controlling horizon of model use is compared: δ values is 0.4 for the dynamic one. The fixed one sets the horizon of model use N, where $N = \{10, 20, 30\}$.

increases, the policy of the agent is greatly perturbed, and the differences in parameter space can have very large influence in performance, so a single bad step can collapse the performance. The method with dynamic–horizon controls the degree of perturbation on policy by satisfying a special constraint on how close the policies before and after model-use are allowed to be. Therefore, the dynamic one achieves a higher accuracy compared to the fixed one.

7. Conclusions

In this paper, we propose an efficient model-based method to solve the HPO problem. First, we formulate the HPO problem as a sequential decision problem and model it as a Markov decision process. Then, an RL agent is used to tune the hyperparameters of a given algorithm. To speed up the training process of the agent, we employ a model to evaluate the performance of the selected configuration. Since the inaccuracy of the model is further exacerbated by long-term use, we dynamically control the horizon of model use by measuring the distance between policies before and after the model use in parameter space. In the experiment, we apply the proposed method to tune XGBoost on 101 tasks and a CNN on 2 tasks. The experimental results indicate that our method outperforms other methods in terms of accuracy and runtime. In the future, we will extend our work to solve the combined algorithm selection and hyperparameters tuning problem.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Jia Wu: Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **SenPeng Chen:** Software, Visualization, Writing - original draft, Writing - review & editing. **XiYuan Liu:** Investigation, Validation.

References

[1] M. Fandong, Z. Jinchao, Dtmt: A novel deep transition architecture for neural machine translation, arXiv preprint arXiv:1812.07807.
[2] M. Brian, R. Delip, Listening to the world improves speech command recognition, in: Proceedings of the 32st AAAI Conference on Artificial Intelligence, 2017, pp. 378–385.
[3] X. Jie, L. Lei, D. Cheng, H. Heng, Bilevel distance metric learning for robust image recognition, Adv. Neural Inform. Process. Syst. 31 (2017) 4198–4207.
[4] F. Hutter, H.H. Hoos, K. Leytonbrown, Sequential model-based optimization for general algorithm configuration, in: Learning & Intelligent Optimization-international Conference, 2012.
[5] C. Thornton, F. Hutter, H.H. Hoos, K. Leyton-Brown, Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, 2013, pp. 847–855.
[6] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (1) (2012) 281–305.
[7] S. Samantha, C. Christophe, Giraud, Informing the use of hyperparameter optimization through metalearning, IEEE Int. Conf. Data Mining (2017) 1051–1056.
[8] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: Advances in Neural Information Processing Systems 28, 2015, pp. 2962–2970.
[9] B. James, B. Remi, B. Yoshua, K. Balazs, Algorithms for hyper-parameter optimization, in: International Conference on Neural Information Processing Systems, 2011.
[10] H. Yi-Qi, Q. Hong, Y. Yang, Sequential classification-based optimization for direct policy searc, in: The 31st AAAI Conference on Artificial Intelligence, 2017, pp. 2029–2035.
[11] L. Marius, H. Frank, Warmstarting of model-based algorithm configuration, in: The 32nd AAAI Conference on Artificial Intelligence, 2018, pp. 1355–1362.
[12] Y. Sun, B. Xue, M. Zhang, G.G. Yen, Completely automated cnn architecture design based on blocks, IEEE Trans. Neural Networks Learn. Syst. (2019) 1–13.
[13] M. Dougal, D. David, P.A. Ryan, Gradient-based hyperparameter optimization through reversible learning, International Conference on Machine Learning (15'ICML) (2015) 2113–2122.
[14] P. Fabian, Hyperparameter optimization with approximate gradient, in: International Conference on Machine Learning(17'ICML), 2017, pp. 737–746.
[15] Y. Sun, H. Wang, B. Xue, Y. Jin, G.G. Yen, M. Zhang, Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor, IEEE Trans. Evol. Comput. 24 (2) (2020) 350–364.
[16] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, A.C.I. Malossi, Tapas: Train-less accuracy predictor for architecture search, arXiv:1806.00250..
[17] B. Baker, O. Gupta, R. Raskar, N. Naik, Accelerating neural architecture search using performance prediction, in, in: Proceedings of the 6th International Conference on Learning Representations, Workshop Track Proceedings, 2018.
[18] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation.
[19] S. Jasper, L. Hugo, A. Ryan P., Practical bayesian optimization of machine learning algorithms, Neural Information Processing Systems(NIPS'12).
[20] M. Lindauer, F. Hutter, Warmstarting of model-based algorithm configuration, in: Association for the Advancement of Artificial Intelligence(18'AAAI), 2018, pp. 1355–1362.
[21] N. Hansen, The CMA Evolution Strategy: A Comparing Review, Springer, Berlin, 2006.
[22] L. Ilya, H. Frank, Cma-es for hyperparameter optimization ofdeep neural networks, International Conference on Learning Representations workshop.
[23] L. Li, K.G. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization, J. Mach. Learn. Res. 18 (2017), pp. 185:1–185:52.
[24] S. Falkner, A. Klein, F. Hutter, BOHB: robust and efficient hyperparameter optimization at scale, in: Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 1436–1445.
[25] C. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (1992) 279–292.
[26] C. Tessler, Y. Efroni, S. Mannor, Action robust reinforcement learning and applications in continuous control, in: Proceedings of the 36th International Conference on Machine Learning, 2019, pp. 6215–6224.
[27] H. Cuayáhuitl, D. Lee, S. Ryu, Y. Cho, S. Choi, S.R. Indurthi, S. Yu, H. Choi, I. Hwang, J. Kim, Ensemble-based deep reinforcement learning for chatbots, Neurocomputing 366 (2019) 118–130, https://doi.org/10.1016/j.neucom.2019.08.007.
[28] F. Li, Q. Jiang, S. Zhang, M. Wei, R. Song, Robot skill acquisition in assembly process using deep reinforcement learning, Neurocomputing 345 (2019) 92–102, https://doi.org/10.1016/j.neucom.2019.01.087.
[29] M.P. Deisenroth, C.E. Rasmussen, Pilco: A model-based and data-efficient approach to policy search., in: International Conference on International Conference on Machine Learning, 2011.

[30] R.S. Sutton, Integrated architectures for learning, planning, and reacting based on approximating dynamic programming - machine learning proceedings 1990, in: Proc of International Conference on Machine Learning, 1990.
[31] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, P. Abbeel, Model-ensemble trust-region policy optimization, International Conference on Learning Representations(18'ICLR).
[32] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, 2017.
[33] H. Cai, T. Chen, W. Zhang, Y. Yu, J. Wang, Efficient architecture search by network transformation, in, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, 2018, pp. 2787–2794.
[34] J. Schmidhuber, Reinforcement learning upside down: Don't predict rewards – just map them to actions, arXiv:1912.02875.
[35] X. Chang, Q. Tao, W. Gang, L. Tie-Yan, Reinforcement learning for learning rate control, arXiv preprint arXiv:1705.11159.
[36] S. John, L. Sergey, A. Pieter, J. Michael I., M. Philipp, Trust region policy optimization, International Conference on Machine Learning(15'ICML).
[37] H. Tuomas, Z. Aurick, H. Kristian, T. George, H. Sehoon, T. Jie, K. Vikash, Z. Henry, G. Abhishek, A. Pieter, L. Sergey, Soft actor-critic algorithms and applications, arXiv preprint arXiv:1812.05905.
[38] C. Tianqi, H. Tong, B. Michael, xgboost: Extreme gradient boosting, Available from: https://github.com/dmlc/xgboost.
[39] S. John, W. Filip, D. Prafulla, R. Alec, K. Oleg, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.0634.
[40] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R.Y. Chen, C. Xi, T. Asfour, P. Abbeel, M. Andrychowicz, Parameter space noise for exploration, International Conference on Learning Representations(18'ICLR).
[41] J. Vanschoren, J.N. van Rijn, B. Bischl, L. Torgo, Openml: networked science in machine learning, CoRR abs/1407.7722. arXiv:1407.7722.
[42] C. Yao, D. Cai, J. Bu, G. Chen, Pre-training the deep generative models with adaptive hyperparameter optimization, Neurocomputing 247 (2017) 144–155, https://doi.org/10.1016/j.neucom.2017.03.058.
[43] P. Fabian, V. Gael, G. Alexandre, M. Vincent, T. Bertrand, G. Olivier, B. Mathieu, P. Peter, V.D. RonWeiss, V. Jake, P. Alexandre, C. David, B. Matthieu, P. Matthieu, D. Edouard, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. (2011) 2825–2830.
[44] J. Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[45] Tensorflow, https://github.com/tensorflow/models/.
[46] L. Yann, B. Leon, B. Yoshua, H. Patrick, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[47] H. Xiao, R. Kashif, V. Roland, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747.

Jia Wu received the M.S. degree in Computer Science from University of Electronic Science and Technology of China in 2006 and the Ph.D. degree in Automation from Université de Technologies Belfort-Montbéliard (UTBM), France in 2011. She is currently an associate professor with University of Electronic Science and Technology of China. Her main research interests are deep reinforcement learning, meta-learning and intelligent transportation systems.

SenPeng Chen is currently a graduate student from University of Electronic Science and Technology of China. His main research are deep reinforcement learning, meta-learning and data analysis.

XiYuan Liu is currently a graduate student from University of Electronic Science and Technology of China. His main research are deep reinforcement learning, meta-learning and data analysis.