Efficient hyperparameter optimization through model-based reinforcement learning

최원우

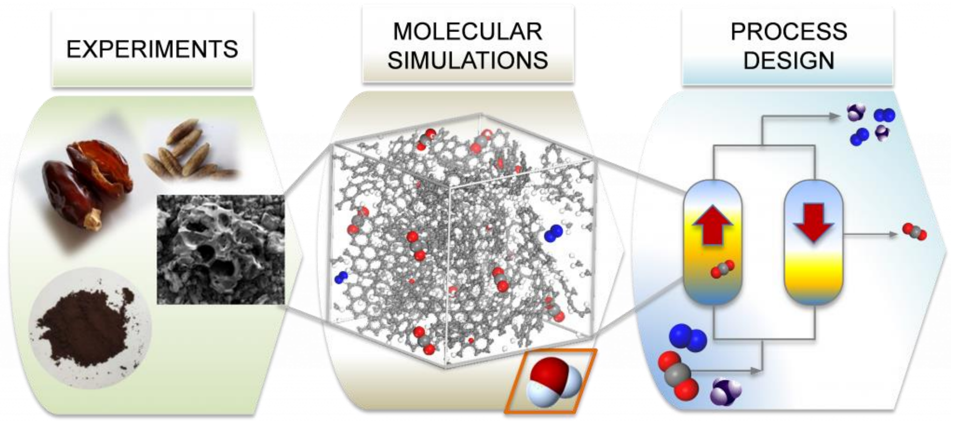
2021. 05. 17

1. Motivation

■ 대부분의 R&D에서 하는일…

시뮬레이션을 통한 최적화

- 회로 : SPICE
- 소자 : TCAD
- 광학 : Setfos
- 유체 : Abaqus
- 플라즈마 : Pegasus
- 재료 : Schrödinger

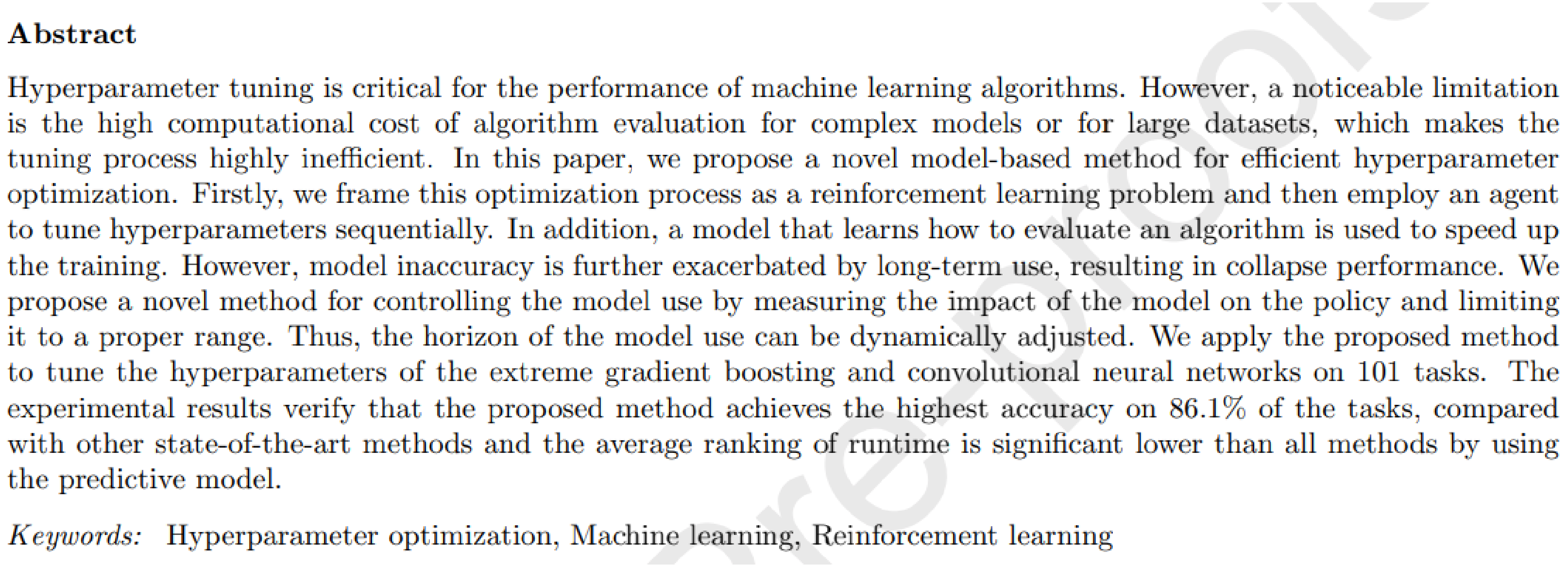


→ 다 파라미터 최적화

→ 딥러닝 (지도학습)을 쓰기엔 데이터 확보에 한계가 있다.

→ 강화학습으로 파라미터 최적화?

Abstract



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.

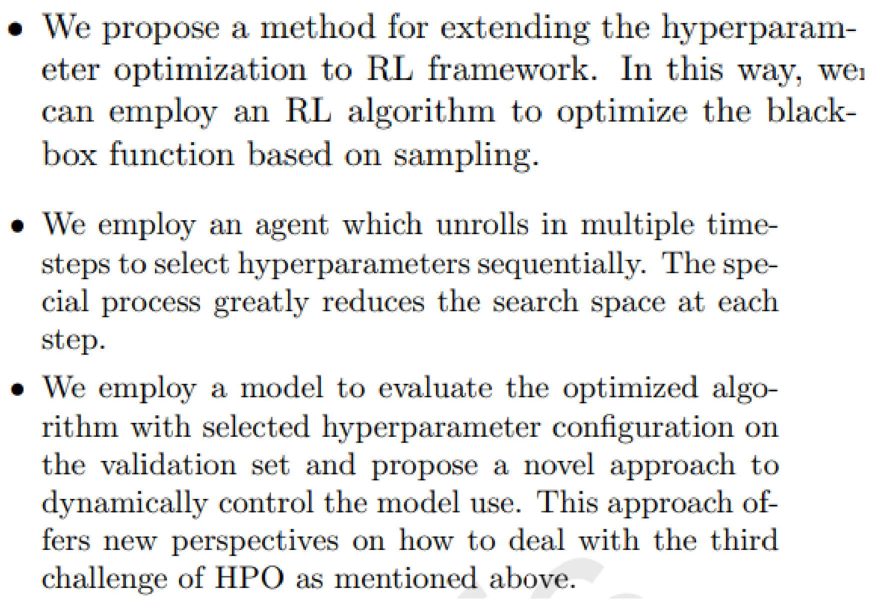
Keywords: Hyperparameter optimization, Machine learning, Reinforcement learning

• Hyper-Parameter Optimization(HPO)는 기계학습 성능향상을 위해 매우 중요하지만, 굉장히 비효율적인 과정이다.

•본 논문은 강화학습 Agent가 순차적으로 파라미터를 조정하도록 하고, 평가 모델을 이용하여 학습 속도를 향상시켰다. 이 평가모델을 오랫동안 의지하여 사용하면 부정확성이 높아지기에 모델의 정책에 대한 영향도를 조절하며 적절히 사용하는 방법론을 도입하였다.

•기존의 CNN과 Gradient Boosting을 101 task에 대해 실험하였더니, 86.1%의 task들에 대해 기존 SOTA보다 더 높은 성능을 더 빠르게 도출하였다.

Introduction



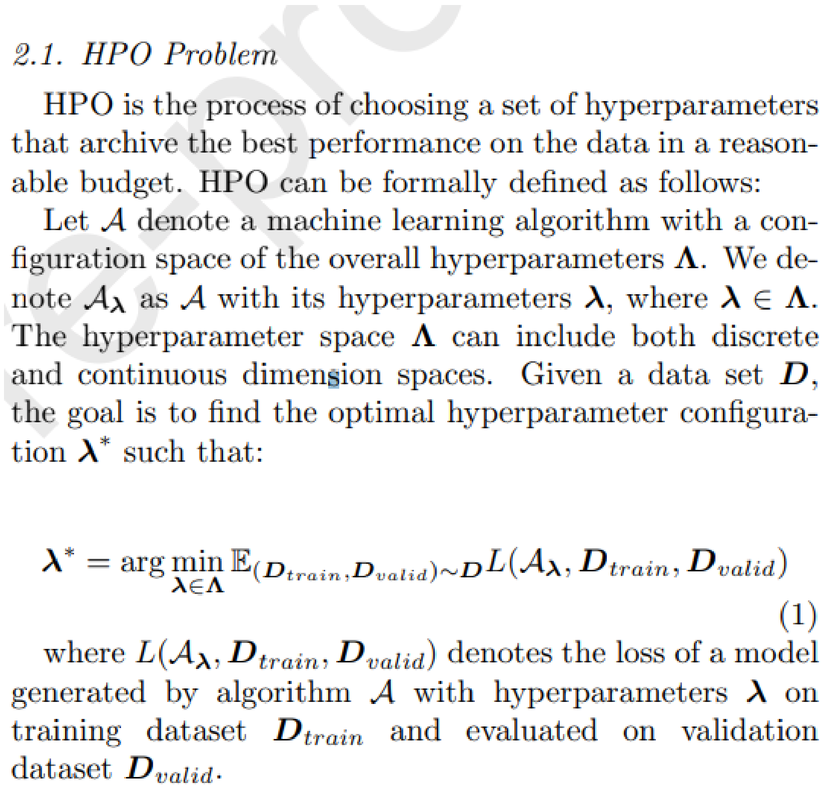
- We propose a method for extending the hyperparameter optimization to RL framework. In this way, we can employ an RL algorithm to optimize the black-box function based on sampling.
- We employ an agent which unrolls in multiple time-steps to select hyperparameters sequentially. The special process greatly 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.

• Manual Hyperparameter Tuning은 전문가의 경험과 노하우에 의존하며 많은 시간을 소요한다. Hyperparameters는 성능과 비선형 관계를 가지고 있으며, search space가 굉장히 크고 평가하는데 매우 오래 걸린다. 그래서 AutoML 연구가 필요하다.

• 방법론은 크게 1. Basic search method (grid/random saerch), 2. Sample-based methods(Bayesian Optimization, Evoluatoinary algorithm), 3. Gradient-based methods(직접적으로 partial derivative을 계산하여 다음 조합 수행)가 있다.

• 이 논문에서는 HPO를 sequential decision process문제로 여기고 MDP로 문제를 모델링하여, LSTM기반의 RL로 문제를 해결, 그리고 학습 효율화를 위해 (오래걸리는 성능평가 부분을) model based로 대체하여 reward를 계산하였다.

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 $\mathcal{A}$ denote a machine learning algorithm with a configuration space of the overall hyperparameters $\boldsymbol{\Lambda}$. We denote $\mathcal{A}_{\boldsymbol{\lambda}}$ as $\mathcal{A}$ 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:

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

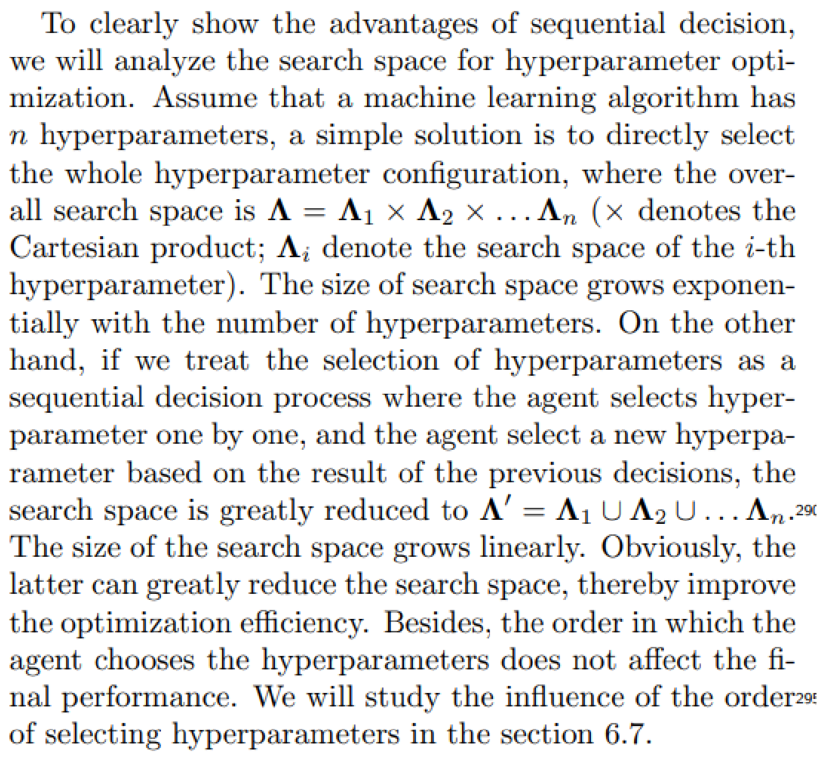
where $L(\mathcal{A}_{\boldsymbol{\lambda}}, \boldsymbol{D}_{train}, \boldsymbol{D}_{valid})$ denotes the loss of a model generated by algorithm $\mathcal{A}$ with hyperparameters $\boldsymbol{\lambda}$ on training dataset $\boldsymbol{D}_{train}$ and evaluated on validation dataset $\boldsymbol{D}_{valid}$.

- 기존의 많이 쓰였던 방법론들로는 Bayesian Optimization, Genetic Algorithm, Population-based method, Hyperband 등이 있다.

HPO Based on RL

3.1. Sequential Decision Problem

- 어려운 문제는 보통 여러 쉬운문제로 쪼게서 품.
- configuration space가 엄청 크기 때문에 agent가 한번에(one step)에 parameter를 선택하는게 힘들다.
- 반면에 하나씩 parameter를 선택해 나가면 search space가 획기적으로 줄어들어 최적화가 효과적이다. 이러면 선택한 parameter에 따라 다음에 선택한 parameter에 영향을 주어 자연스럽게 sequential decision process가 된다.



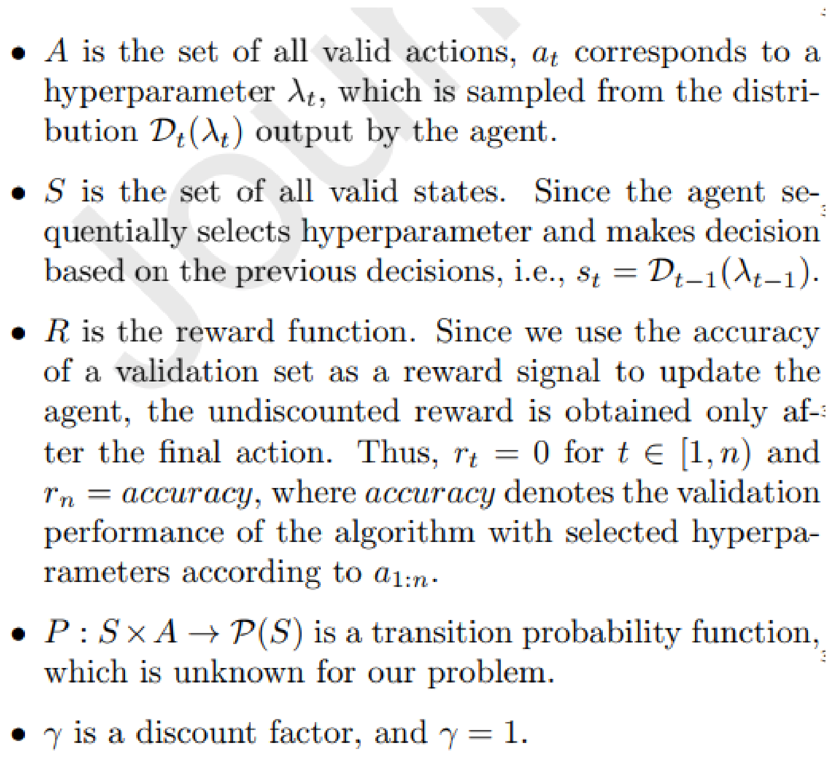
To clearly show the advantages of sequential decision, we will analyze the search space for hyperparameter optimization. Assume that a machine learning algorithm has n hyperparameters, a simple solution is to directly select the whole hyperparameter configuration, where the overall search space is $\mathbf{\Lambda} = \mathbf{\Lambda}_1 \times \mathbf{\Lambda}_2 \times \ldots \mathbf{\Lambda}_n$ ($\times$ 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.

HPO Based on RL

3.2. MDP Formulation

•steps = hyperparameter 수

A: t번째의 parameter값
S: set of all valid states
R: validation accuracy(t-1까지는 0, t에서는 accuracy)
P: 우리문제에서는 모른다.
gamma: 1



- A is the set of all valid actions, a_t corresponds to a hyperparameter λ_t, which is sampled from the distribution $\mathcal{D}_t(\lambda_t)$ 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., $s_t = \mathcal{D}_{t-1}(\lambda_{t-1})$.
- 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 $a_{1:n}$.
- $P : S \times A \to \mathcal{P}(S)$ is a transition probability function, which is unknown for our problem.
- γ is a discount factor, and $\gamma = 1$.

HPO Based on RL

3.3. Design of the Agent

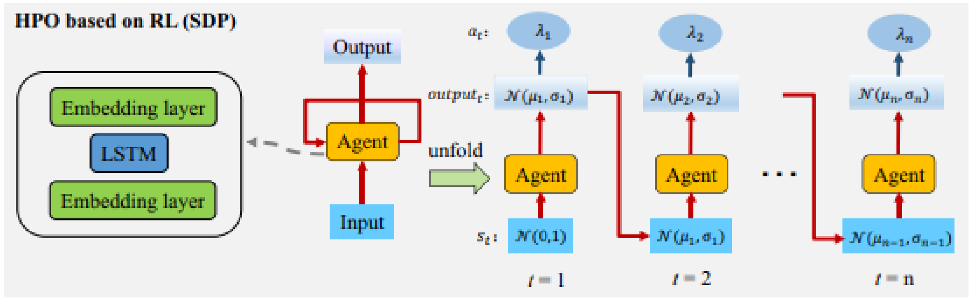
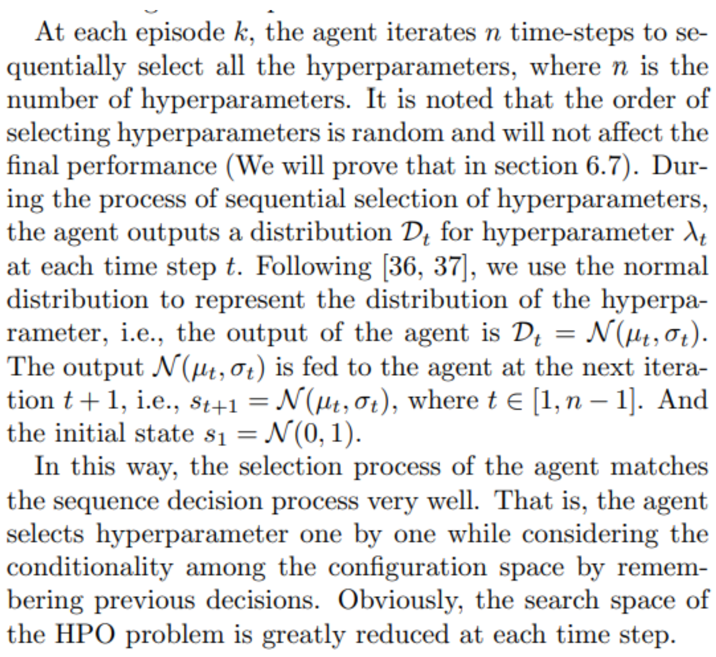


Figure 1: Overview of the HPO based on RL. The agent selects hyperparameters sequentially.


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 $\mathcal{D}_t$ 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 $\mathcal{D}_t = \mathcal{N}(\mu_t, \sigma_t)$. 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.

Embeding layer: MLP로 구성, input을 high-dimensional representation, 또는 lstm의 output을 low-dimentinal representation.

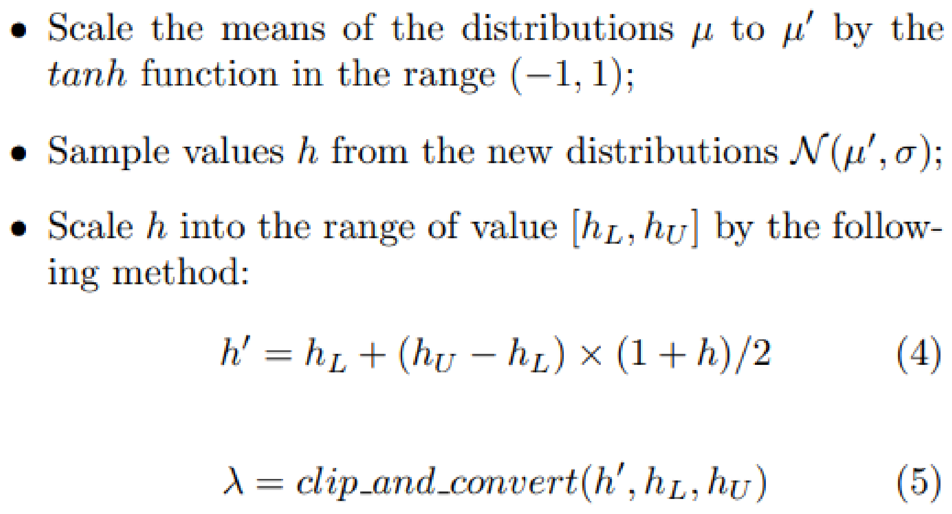
LSTM layer: 3층을 사용, 학습시키기 어렵지만, sequential problem에 적합. LSTM은 configuration space에서의 conditionality를 찾을 수 있음

각 episode 마다, n(number of parameters)번 반복하며 순차적으로 parameter를 선택.

파라미터 선택 순서는 중요하지 않음.

HPO Based on RL

3.4. Sampling from N (μ, σ)



- 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:

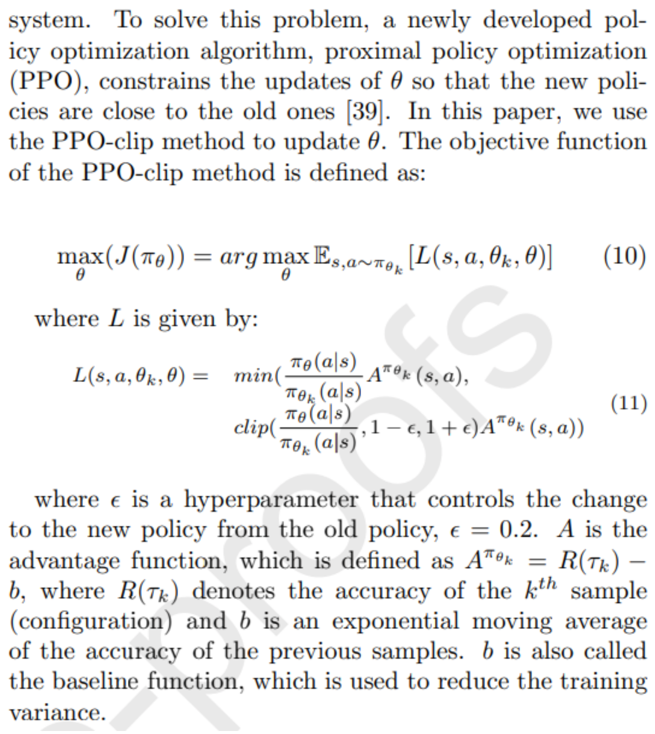
$$h' = h_L + (h_U - h_L) \times (1 + h)/2 \tag{4}$$

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

각각의 parameter scale이 달라 sampling할때 search 하기 어렵다.
Customized Scaling 적용

HPO Based on RL

3.5. Training the agent



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:

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

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.

policy pi, expected reward를 maximize하는 pi의 parameter theta를 찾는게 모적임.
Proximal Policy Optimization (clip)사용

Efficiency Improving Based on a Model

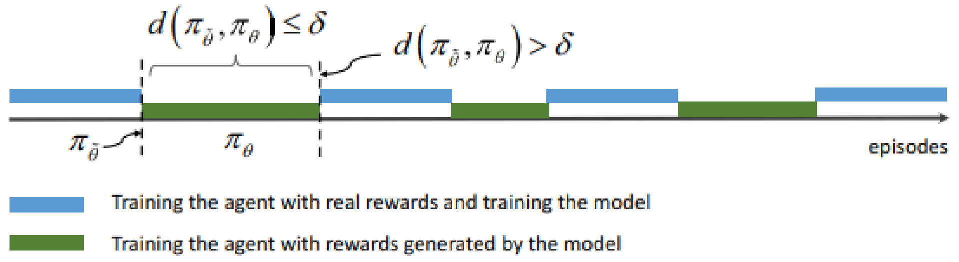
선택된 hyperparameter들로 학습하고, validation으로 평가
평가하는데 엄청 오래 걸림.
전체적인 강화학습을 비효율적이게 함.
선택된 hyperparameter들로 학습하는 대신에, 평가모델을 사용

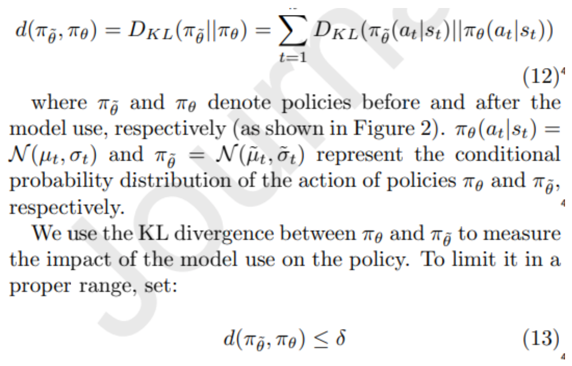
4.0.1. Structure and Training of the Model

model의 입력은 선택된 configuration, 출력은 validation 평가값이다.
지도학습으로 학습한 MLP구조의 model
충분히 많은 학습데이터셋을 확보하기가 힘들다.
가정 : number of traning sample = 보통 number of parameters의 5~10배

Efficiency Improving Based on a Model

4.0.2. Controlling the Model Use


$d(\pi_{\tilde\theta},\pi_\theta)\le\delta$
$d(\pi_{\tilde\theta},\pi_\theta)>\delta$
$\pi_{\tilde\theta}$
π_θ
episodes
Training the agent with real rewards and training the model
Training the agent with rewards generated by the model



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

where $\pi_{\tilde\theta}$ and π_θ denote policies before and after the model use, respectively (as shown in Figure 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:

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

평가델의 사용은 학습을 속도에 견인하지만 장기적으로 계속 사용하게되면 잘못된 성능평가로 모델의 성능을 떨어트린다.
horizon of the model : 언제까지 모델을 써야할지 말아야할지를 결정해야 되는데,
모델을 썼을때와 안썼을때의 policy들의 거리를 KL을 써서 계산하고 이게 threshhold보다 작으면 inaccuracy가 높을 확율이 크기 때문에 다시 학습

5. Overall Framework

- 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.

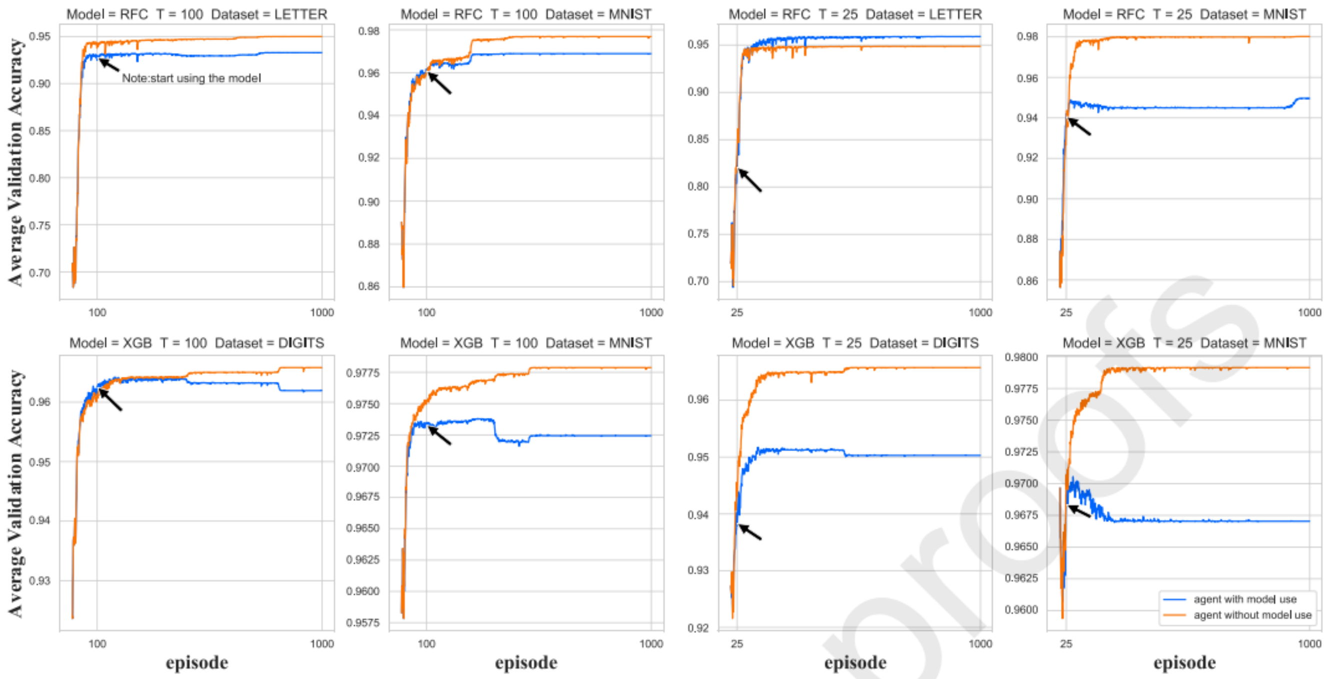
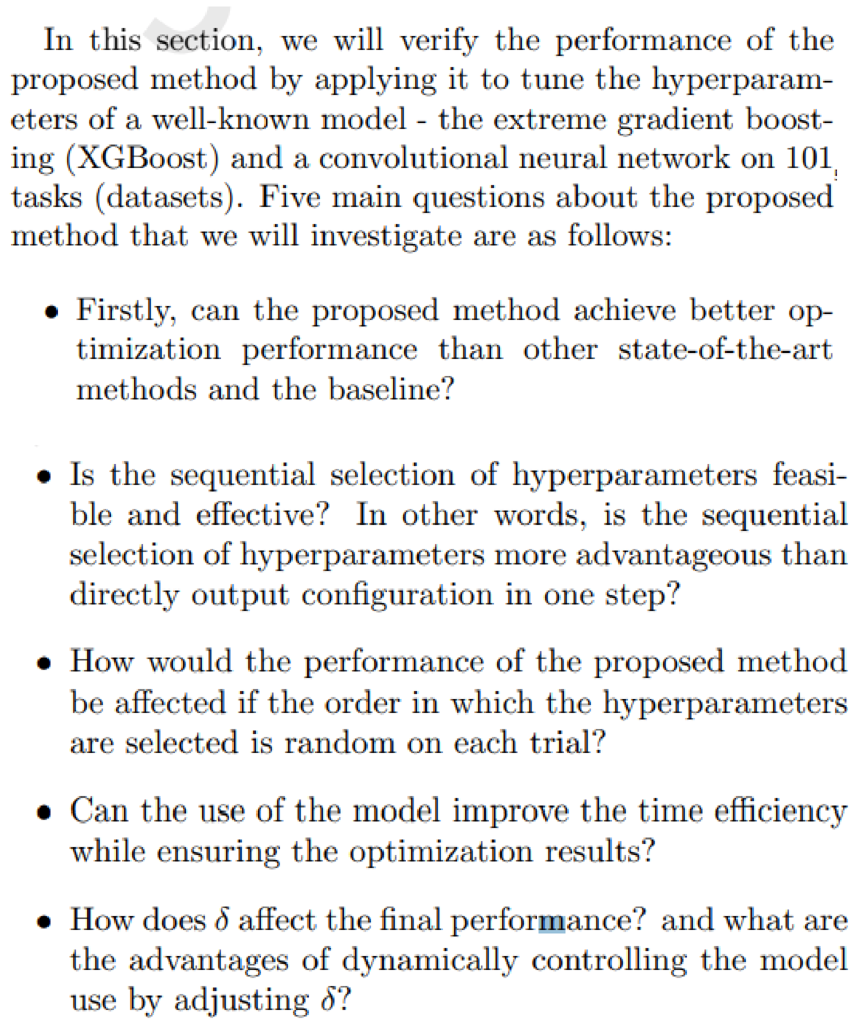


Figure 3: Comparison between RL agents with and without long-term model use. "Agent without model use" denotes the learning curve of RL agent without model use during 1000 episodes training. "Agent with model use" presents the learning curve of RL agent with long-term model use from the 25^{th} or 100^{th}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

5. Overall Framework

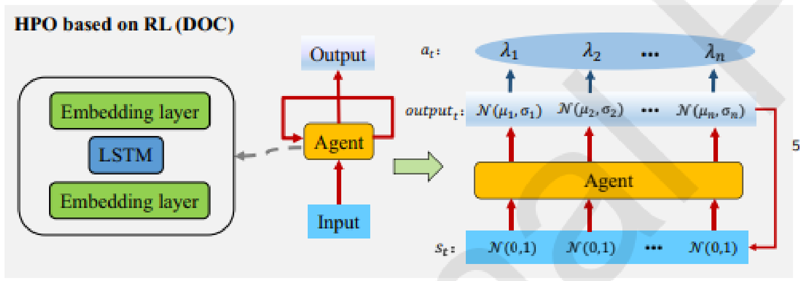


In this section, we will verify the performance of the proposed method by applying it to tune the hyperparameters of a well-known model - the extreme gradient boosting (XGBoost) and a convolutional neural network on 101 tasks (datasets). Five main questions about the proposed method that we will investigate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

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		-	-	-



HPO based on RL (DOC)
Embedding layer
LSTM
Embedding layer
Output
Agent
Input
a_t:
λ_1 λ_2 ⋯ λ_n
$output_t$: $\mathcal{N}(\mu_1,\sigma_1)$ $\mathcal{N}(\mu_2,\sigma_2)$ ⋯ $\mathcal{N}(\mu_n,\sigma_n)$
Agent
s_t: $\mathcal{N}(0,1)$ $\mathcal{N}(0,1)$ ⋯ $\mathcal{N}(0,1)$

Figure 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.

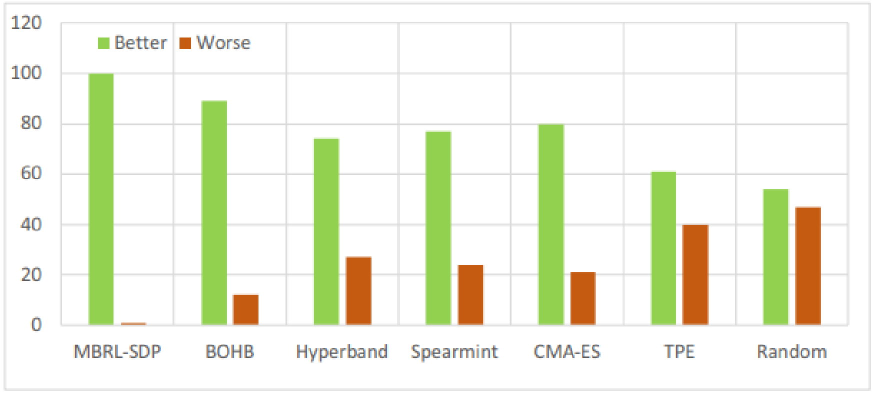


Figure 5: The number of datasets that the evaluated method performs better or worse than *Baseline* (accuracy on the test set), across 101 datasets. "Better" denotes the evaluated method performs better than *Baseline* and "worse" means the evaluated method performs worse than *Baseline*.