Automated ML and RL: Hyperparameter Optimization (HPO)

Pu Yang
School of Mathematical Sciences, Peking University

2022.3.16

Table of contents

A Demo of AutoGluon

demo

- pipeline of the demo
  - Task
  - Model
  - optimization
  - hyperparameters
- We use it for two purpose
  - quickly get a moderately good model
  - get a better performance

Markov Decision Processes (MDP) $\mathcal{M} =< \mathcal{S}, \mathcal{A}, P, R, \gamma >$[1]

- $\mathcal{S}$ - a set of states; $s \in \mathcal{S}$ - a state
- $\mathcal{A}$ - a set of actions; $a \in \mathcal{A}$ - an action
- P - transition probability function
- R - reward function
- γ - discounting factor for future rewards

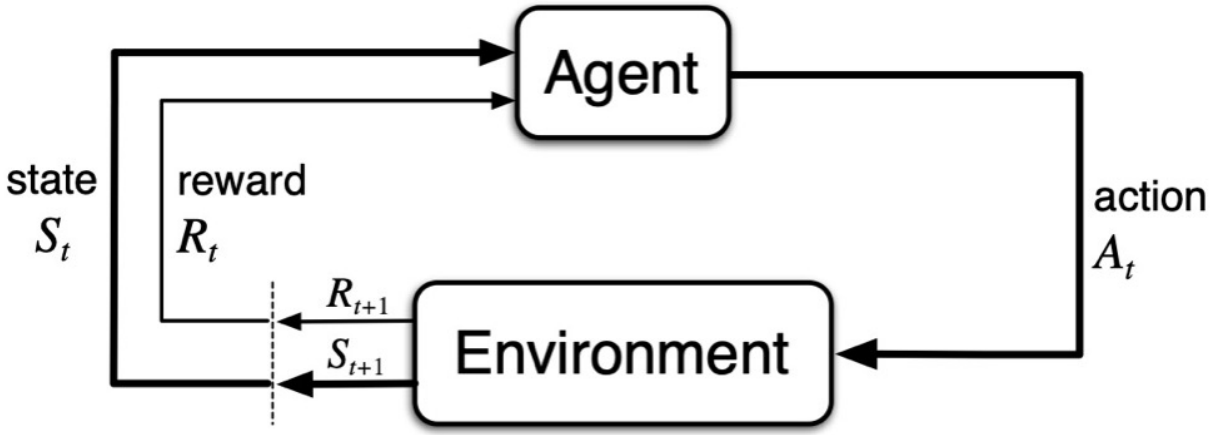


Figure 1: The agent–environment interaction in a Markov decision process.

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

What Needs to be Automated?[1]

	Deep Learning	**Reinforcement Learning**
Task	loss	MDP(reward, observation, environment)
Algorithm	optimizer	RL algorithm
Architecture	network	network
Hyperparameter	lr, decay, iteration	discount factor, lr, initialization scheme, code level implementation details, entropy and its coefficient, ...

[1] Jack Parker-Holder et al. "Automated Reinforcement Learning (AutoRL): A Survey and Open Problems". In: *arXiv preprint arXiv:2201.03916* (2022).

Hyperparameter Optimization (HPO)

HPO

Using an algorithm to learn good values for hyperparameters

Formulation

$$\min_{\zeta} f\left(\zeta, \theta^*\right) \text{ s.t. } \theta^* \in \arg\min_{\theta} J(\theta; \zeta)$$

where θ is a set of parameters; ζ is a set of hyperparameters; objective function J can be considered the inner loop, which may be equivalent to minimizing some loss; another objective function f can be considered the outer loop.
Examples: f can be defined as the validation loss/reward, and J can be considered the training loss/reward

What qualities make a good HPO algorithm?

- **Flexible**: It allow for somplicated hyperparameter configuration spaces.
- **Efficient**: It has a good runtime.
- **Effective**: It can find good hyperparameters.
- **Scalable**: Algorithm steps can be run in parallel.
- **Robust**: Stop considering ”hyper-hyper-· · ·-hyperparameters”.

Random/Grid Search

- **Random Search**[1]: randomly samples hyperparameter configurations from the search space
- **Grid Search**: divides the search space into fixed grid points which are evaluated

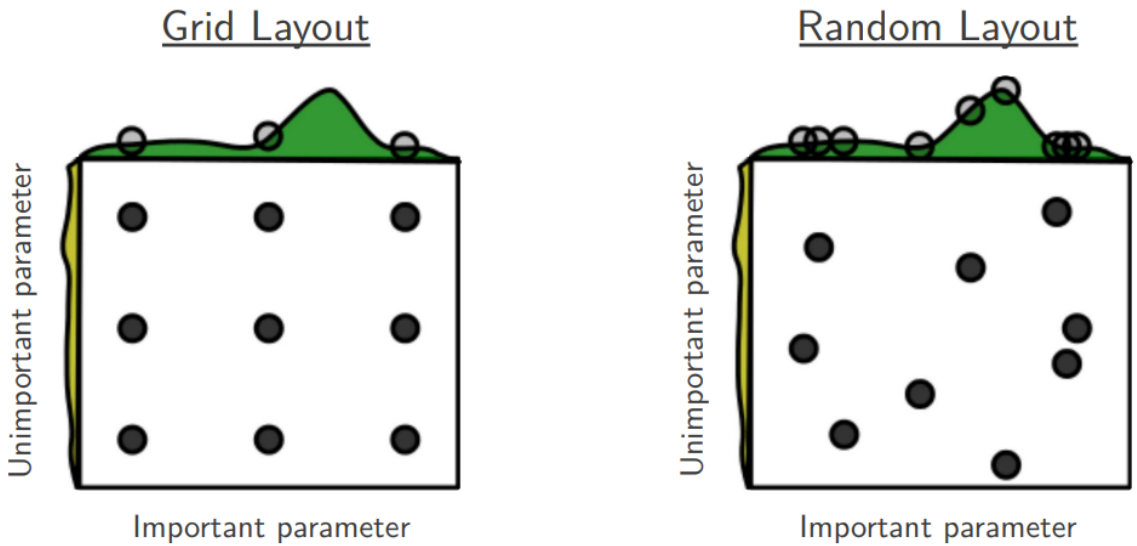


Figure 2: Grid and random search of nine trials for optimizing a function $f(x, y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow.

[1] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization.". In: *Journal of machine learning research* 13.2 (2012).

Where to Improve?

- Evenly distribute computing resources $\rightarrow$ a bandit-based configuration
- not use prior knowledge to select the next set of hyperparameters $\rightarrow$ Bayesian optimization (evolutionary method, meta-gradient method)

Multi-armed Bandit (MAB)

Multi-armed Bandit

The multi-armed bandit model

The multi-armed bandit can be seen as a set of real distributions $B = \{R_1, \ldots, R_K\}$, each distribution being associated with the rewards delivered by one of the levers.

Let $\mu_1, \ldots, \mu_K$ be the mean values associated with these reward distributions.

The gambler iteratively plays one lever per round and observes the associated reward. The objective is to maximize the sum of the collected rewards.

The **regret** ρ after T rounds is defined as the expected difference between the reward sum associated with an optimal strategy and the sum of the collected rewards:

$$\rho = T\mu^* - \sum_{t=1}^{T} \hat{r}_t$$

where $\mu^* = \max \mu_k$ is the maximal reward mean, and $\hat{r}_t$ is the reward in round t.

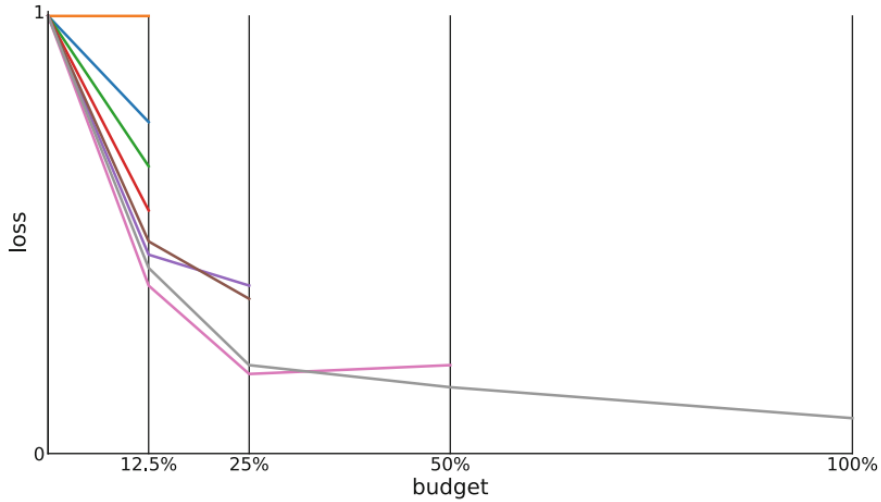
HyperBand[1]

- Notations
  - B: total computational resource, called budget
  - n: number of sets of hyperparameter configurations
- Trade-off:
  - Random/Grid Search: trade-off between n and B/n
  - HyperBand: trade-off of the resource allocation between the n sets of hyperparameter configurations
- A pure-exploration (non-)stochastic n-armed bandit problem

[1] Lisha Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816.

Successive-Halving Algorithm[1]

1. Uniformaly allocate a budget to a set of hyperparameter configurations
2. keep the top 50% and train them further with twice the budget
3. Iteratively do 1 and 2, until only one set of hyperparameter configurations remains



[1] Kevin Jamieson and Ameet Talwalkar. "Non-stochastic best arm identification and hyperparameter optimization". In: *Artificial intelligence and statistics*. PMLR. 2016, pp. 240–248.

HyperBand

- The performance of **Successive-Halving** can be sensitive to the number of culling steps performed before reaching the maximum budget per run.
- **HyperBand** solves this by evenly splitting resources between running. **Successive-Halving** with multiple values of this sensitive hyper-hyperparameter.
- All values between the extremes of no culling and aggressive culling are tried.

Pseudocode

Algorithm 1: Pseudocode for SuccessiveHalving used by Hyperband as a subroutine.

input : initial budget b_0, maximum budget b_{max}, set of n configurations $C = \{c_1, c_2, \dots c_n\}$

$b = b_0$
while $b \leq b_{max}$ **do**
  $L = \{\tilde{f}(c, b) : c \in C\}$
  $C = \text{top}_k(C, L, \lfloor |C|/\eta \rfloor)$
  $b = \eta \cdot b$

Algorithm 1: Pseudocode for Hyperband using SuccessiveHalving (SH) as a subroutine.

input : budgets b_{min} and b_{max}, η

$s_{max} = \lfloor \log_\eta \frac{b_{max}}{b_{min}} \rfloor$
for $s \in \{s_{max}, s_{max} - 1, \dots, 0\}$ **do**
  sample $n = \lceil \frac{s_{max}+1}{s+1} \cdot \eta^s \rceil$ configurations
  run SH on them with $\eta^s \cdot b_{max}$ as initial budget

- $b_{\max}$: the maximum amount of resource that can be allocated to a single configuration
- η: an input that controls the proportion of configurations discarded in each round of **Successive-Halving**

Pseudocode

Algorithm 1: HYPERBAND algorithm for hyperparameter optimization.

input : R, η (default $\eta = 3$)
initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$

for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  // begin SUCCESSIVEHALVING with (n, r) inner loop
  $T =$ `get_hyperparameter_configuration`(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{$`run_then_return_val_loss`$(t, r_i) : t \in T\}$
    $T =$ `top_k`$(T, L, \lfloor n_i/\eta \rfloor)$
  end
end
return *Configuration with the smallest intermediate loss seen so far.*

Example Application

We work with the MNIST dataset and optimize hyperparameters for the LeNet trained using SGD. The following hyperparameters are considered.

Hyperparameter	Scale	Min	Max
Learning Rate	log	1e-3	1e-1
Batch size	log	1e1	1e3
Layer-2 Num Kernels (k2)	linear	10	60
Layer-1 Num Kernels (k1)	linear	5	k2

We set the input of **HyperBand** as: $R = 81, \eta = 3$, then $s_{\max} = 4, B = 405$. Indeed, we consider 4 **Successive-Halving**, 1 **Random Search** and 1 **HyperBand**.

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

Example Application

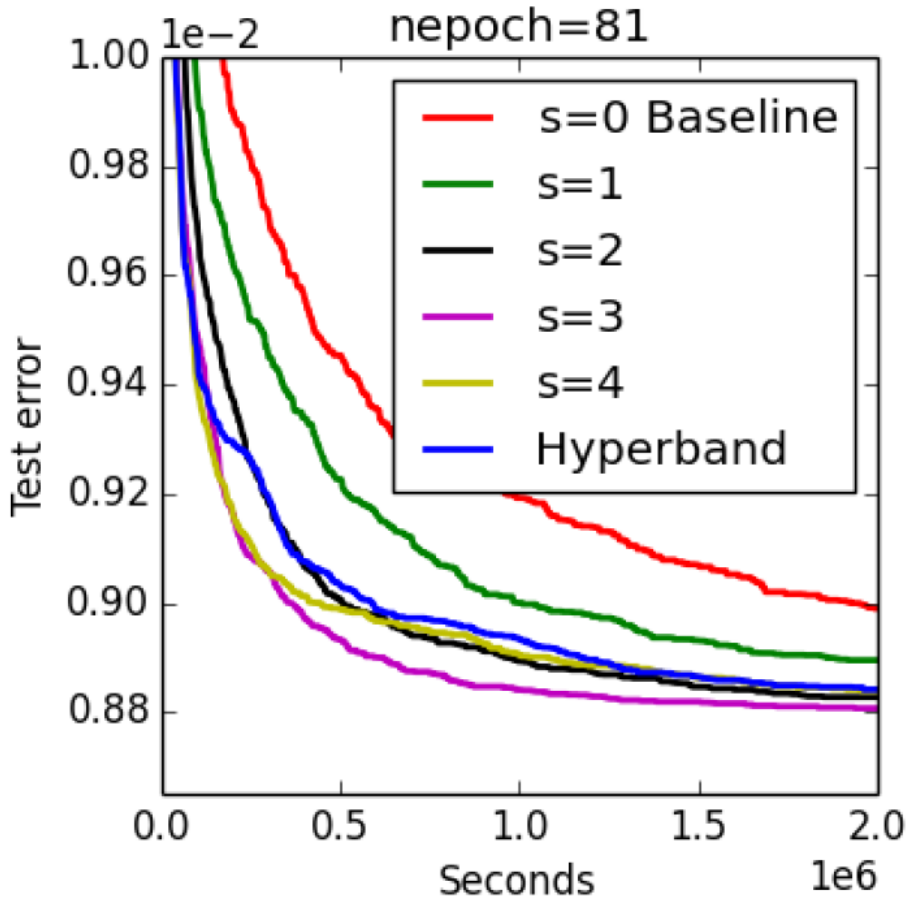


Figure 3: Performance of individual **Successive-Halving** s and **HyperBand**

Bayesian Optimization Pipeline

1. A prior function (commonly a Gaussian Process) is used to construct uncertainty estimates on the blackbox function $f(\cdot)$ given previous evaluation data.
2. An acquisition function is constructed from the regressor, which represents the explore-exploit trade-off on f.
3. The argmax of the acquisition function is used for the next trial.

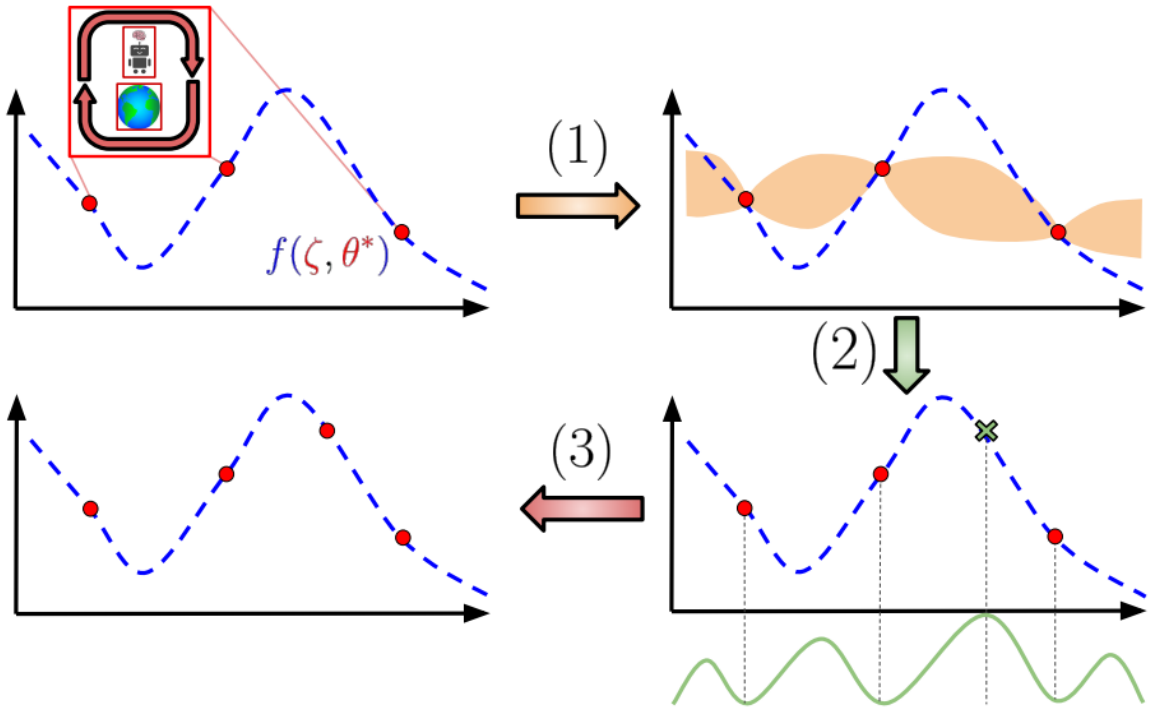


Figure 4: Bayesian Optimization, which consists of 3 main steps.

Bayesian Optimization Algorithm

Algorithm 1 Sequential Model-Based Optimization

Input: $f, \mathcal{X}, S, \mathcal{M}$
$\mathcal{D} \leftarrow \text{INITSAMPLES}(f, \mathcal{X})$
for $i \leftarrow |\mathcal{D}|$ **to** T **do**
  $p(y \mid \mathbf{x}, \mathcal{D}) \leftarrow \text{FITMODEL}(\mathcal{M}, \mathcal{D})$
  $\mathbf{x}_i \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}, p(y \mid \mathbf{x}, \mathcal{D}))$
  $y_i \leftarrow f(\mathbf{x}_i)$ ▷ Expensive step
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{x}_i, y_i)$
end for

Bayesian optimization algorithm

- f: objective function
- $\mathcal{X}$: hyperparameter space
- $\mathcal{M}$: prior probabilistic model set
- S: acquisition function

Prior Probabilistic Model

- Suppose the objective function has a prior of Gaussian Process (GP)

$$f(x) = \mathcal{GP}(m(x), k(x, x'))$$

- A common prior is to set

$$m(x) = 0, k(x_i, x_j) = \exp\left(-\frac{1}{2}||x_i - x_j||^2\right)$$

- $\mathcal{D} = (x_i, y_i)_{i=1}^t$ has t observed point, then we get the covariance matrix

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_t, \mathbf{x}_1) & \dots & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix}$$

and

$$\mathbf{f}_{1:t} = f(\mathbf{x}_{1:t}) \sim \mathcal{N}(0, \mathbf{K})$$

Prior Probabilistic Model

- With $f_{t+1} = f(x_{t+1})$, we have

$$\begin{bmatrix} \mathbf{f}_{1:t} \\ f_{t+1} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k\left(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}\right) \end{bmatrix}\right)$$

$$\mathbf{k} = \begin{bmatrix} k\left(\mathbf{x}_{t+1}, \mathbf{x}_1\right) & k\left(\mathbf{x}_{t+1}, \mathbf{x}_2\right) & \cdots & k\left(\mathbf{x}_{t+1}, \mathbf{x}_t\right) \end{bmatrix}$$

- Finally, based on the observed t points, we can get the posterior distribution

$$P(f_{t+1} \mid \mathcal{D}, x_{t+1}) = \mathcal{N}\left(\mu_t(x_{t+1}), \sigma_t(x_{t+1})\right)$$

where

$$\mu_t(x_{t+1}) = \mathbf{k}^T \mathbf{K}^{-1} \mathbf{f}_{1:t}$$

$$\sigma_t(x_{t+1}) = k(x_{t+1}, x_{t+1}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k}$$

Acquisition Function

- Intuition:
  If x_{t+1} has large mean, sample it will give us more information of f.
  If x_{t+1} has large var, it means that we know very little about this point.
- Probability of improvement (PI)

$$\text{PI}(x) = P(f(x) \leq f(x^*) - \xi) = \Phi\left(\frac{f^* - \xi - \mu(x)}{\sigma(x)}\right)$$

  where $f^* = f(x^*) = \min_{x_i \in \mathbf{x}_{1:t}} f(x_i)$

- Expected improvement (EI)
  the improvement at x can be expressed as

$$I(x) = \max(f^* - f(x), 0)$$

  then calculate the expectation of it

$$\text{EI}(x) = \int \max(f^* - y, 0) P(y \mid x) dy$$

Tree Parzen Estimator (TPE)

- Instead of modeling $P(y \mid x, \mathcal{D})$, TPE models density function

$$l(x) = P(x \mid y < \alpha, \mathcal{D})$$
$$g(x) = P(x \mid y > \alpha, \mathcal{D})$$

- Then, with Bayesian formula, calculate EI as follow

$$\begin{aligned}
\text{EI}(x) &= \int_{-\infty}^{\alpha} (\alpha - y) p(y \mid x) dy \\
&= \int_{-\infty}^{\alpha} (\alpha - y) \frac{p(x \mid y) p(y)}{p(x)} dy \\
&= \frac{\gamma \alpha l(x) - l(x) \int_{-\infty}^{\alpha} P(y) dy}{\gamma l(x) + (1 - \gamma) g(x)} \qquad (\text{let } \gamma = P(y < \alpha)) \\
&\propto (\gamma + \frac{g(x)}{l(x)}(1 - \gamma))^{-1}
\end{aligned}$$

- $\max \text{EI}(x) \iff \max l(x)/g(x)$

Bayesian Optimization + HyperBand (BOHB)

- Start with vanilla **HyperBand** but store all validation scores for each (config, budget) pair run.
- When a sufficient amount of data has been collected for a budget, fit a TPE surrogate model and start using this for future configuration selection.
- Use surrogate model for largest budget with enough available data.
- Continue to sample random configurations with some small frequency for robustness guarantees.

Sampling details

Replace the random sampling in **HyperBand** into the following strategy.

Algorithm 2: Pseudocode for sampling in BOHB

input : observations D, fraction of random runs ρ, percentile q, number of samples N_s, minimum number of points N_{min} to build a model, and bandwidth factor b_w

output : next configuration to evaluate

1 **if** *rand()* $< \rho$ **then return** random configuration
2 $b = \arg\max \{D_b : |D_b| \geq N_{min} + 2\}$
3 **if** $b = \emptyset$ **then return** random configuration
4 fit KDEs according to Eqs. (2) and (3)
5 draw N_s samples according to $l'(\boldsymbol{x})$ (see text)
6 **return** sample with highest ratio $l(\boldsymbol{x})/g(\boldsymbol{x})$

- line 2: Always uses the model for the largest budget for which enough observations are available.
- line 4: Kernel Density Estimation (KDE) is a non-parametric way to estimate the PDF of a RV. Split D_b into two subset to model density function in TPE.
- line 5: To optimize EI, sample N_s points from $l'(x)$, which is the same KDE as $l(x)$ but with all bandwidths multiplied by a factor b_w to encourage more exploration.

Experiments

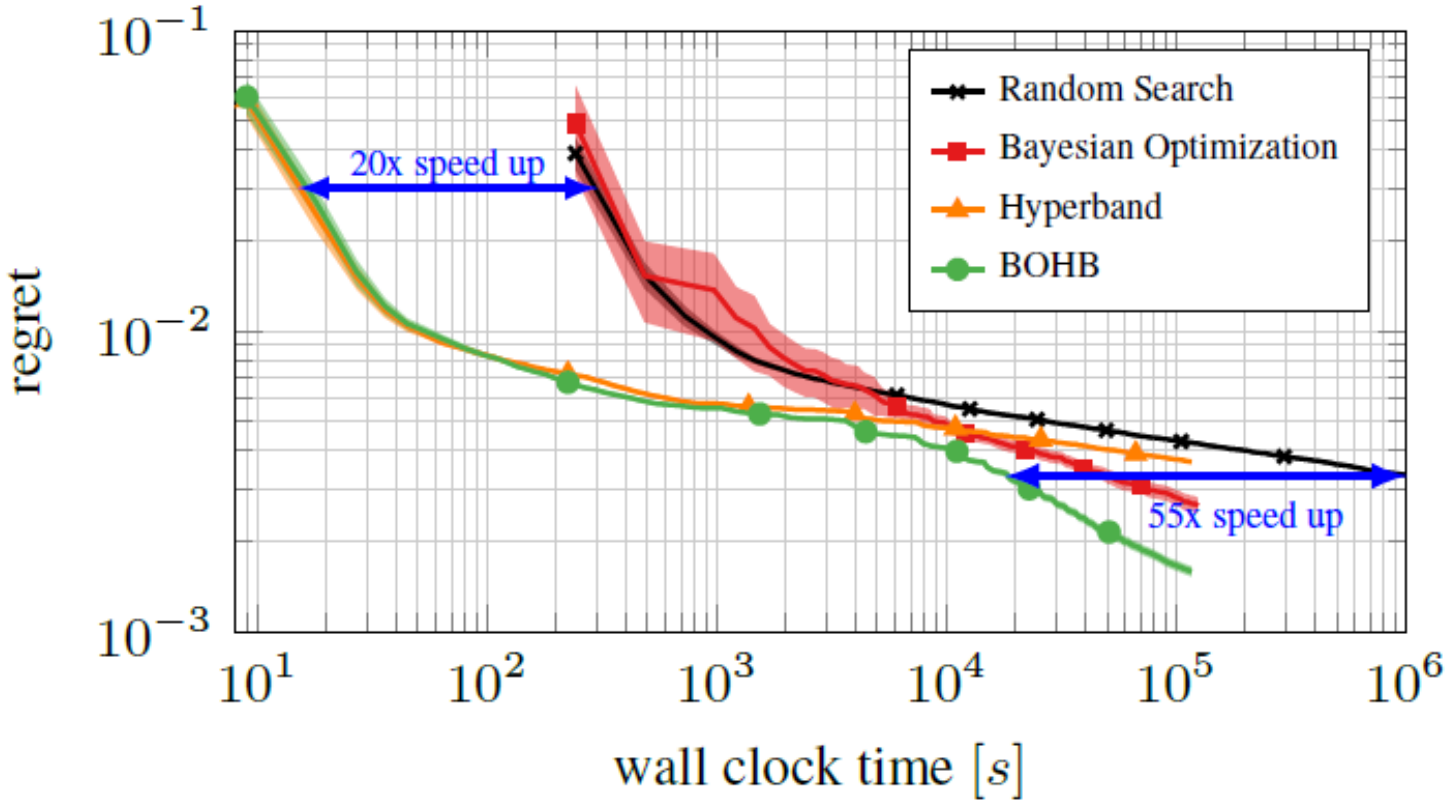


Figure 5: Illustration of typical results obtained, here for optimizing six hyperparameters of a neural network. We show the immediate regret of the best configuration found by 4 methods as a function of time.

Experiments

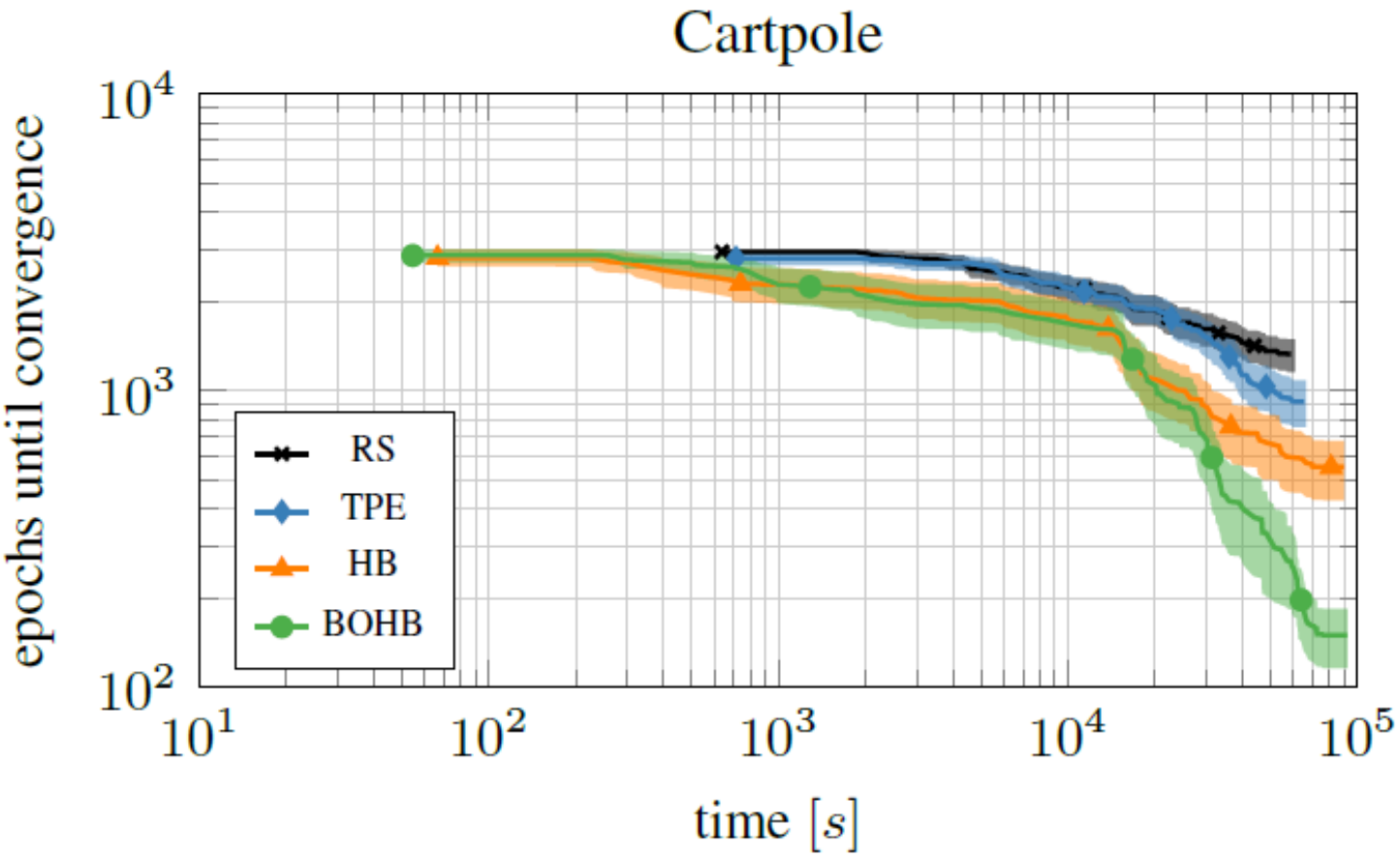
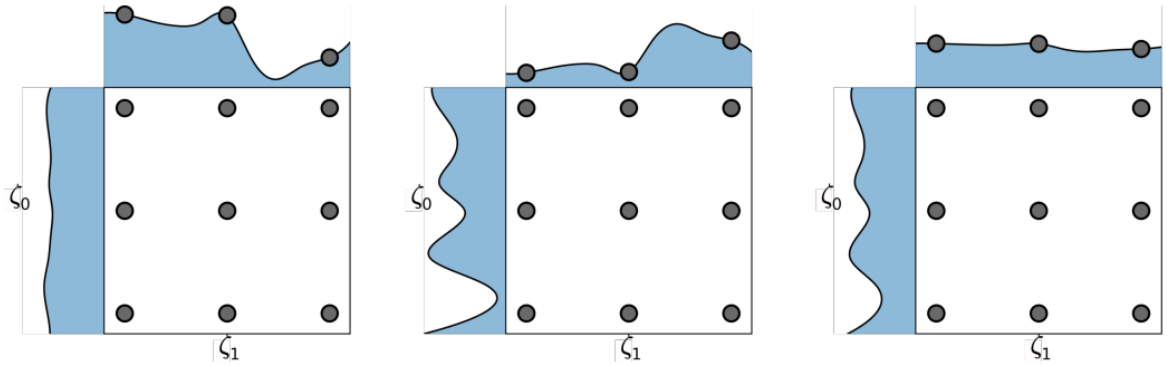


Figure 6: Hyperparameter optimization of 8 hyperparameters of PPO on the cartpole task.

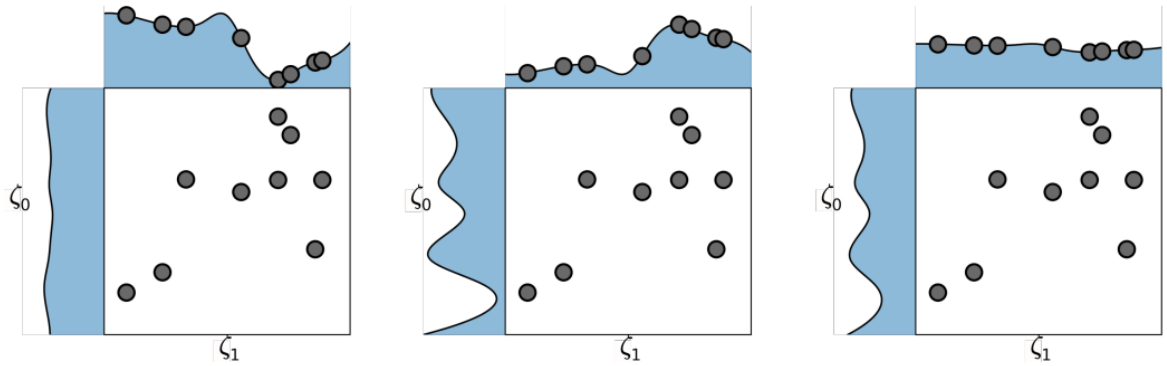
Non-Stationarity of the Optimization Problem

- Some hyperparameters need to be adapted over time in deep learning
- An example: learning rates
  - a large learning rate will results in poor convergence
  - a small learning rate will lead to slow convergence or convergence to a local optimum
  - solution: begin with a large learning rate and decay during training
- The nature of RL further amplifies the potential non-stationarity of hyperparameters, examples:
  - for multi-agent task, other agents change all the time during training
  - the environment changes during training
  - an RL agent generates its own training data, which is highly dependent on the current instantiation of the agent

An Illustration of Non-Stationarity of Auto-RL Problem



(a) Grid Search at $t = 0$, $t = T$ and averaged over t.



(b) Random Search at $t = 0$, $t = T$ and averaged over t.

Figure 7: The non-stationarity of the AutoRL problem. Generally, grid and random search keep the hyperparameters ζ_0 and ζ_1 fixed throughout a run. If the **loss landscape** changes during the run, classical hyperparameter optimization methods optimize the average performance over time.

Thanks