Experimental Report on Hyper-parameter Optimization

Xuedong Shang

No Institute Given

Abstract.

1 Introduction

The algorithms being considered here for the moment are Hyperband [1], a Bayesian-based approach TPE [2], two hierarchical approaches HOO [3], HCT [4], a baseline method random search, plus one new hyperband-liked heuristic on top of Top-Two Thompson Sampling (TTTS) [5]. Current datasets used are MNIST dataset and some datasets from UCI machine learning dataset archive.

2 Heuristics

2.1 TTTS coupled with Hyperband

We propose one new heuristic based on TTTS for the moment. The idea is similar to Hyperband. By running several brackets of TTTS with different number of configurations n and the same budget, we try to trade off between the number of configurations and the number of resources allocated to each configuration. Unlike Hyperband where the number of resources is fixed for each configuration in every bracket, here the number of resources is decided by the underlying Thompson Sampling.

This heuristic requires three inputs β, η and R, where η and R are the same inputs as in Hyperband: η is the discarding proportion and R is the maximum amount of resources that can be allocated to a single configuration. However, in this heuristic, they do not really make sense. The reason why they are required is just to keep the same budget as for Hyperband.

The subroutine used here is TTTS where we make use of a prior distribution Π_1 over a set of parameters Θ, where for each $\theta \in \Theta$, θ_i caraterizes configuration i for $i \in \{1 \dots n\}$. Based on a sequence of observations, we can update our beliefs to attain a posterior distribution Π_t. At each time step t, the subroutine samples θ from Π_t, then with probability β, the subroutine evaluates the configuration I with the best paramter $\theta_I = \max_{c \in C} \hat{\theta}_c$. Otherwise it samples a new θ from Π_t until we obtain a different configuration $J \neq I$ such that $\theta_J = \max_{c \in C} \hat{\theta}_c$.

Remark 1. Note that this algorithm does not require computing or approximating the optimal action probabilities, which could be computationally heavy. However in practice, sometimes it could be ridiculously long to sample a J that is different from I, especially when the best configuration is far better than the others. To avoid this issue, we can either explicitly compute the optimal action probabilities, or just play the second best one when this kind of situation occurs.

The pseudo-code is shown in Algorithm 1. Let us give some more details on how to update the posterior belief (Line. 17 in Algorithm 1). Currently, we assume a Beta prior which is usually associated with Bernoulli bandits. Here in our case, however, the reward (or more precisely, the loss) lies in $[0, 1]$, we thus need to adapt the Thompson Sampling process to the general stochastic bandits case. One way to tackle this is the binarization trick shown in Algorithm 2 inspired by [6].

3 MNIST

We consider here Logistic Regression, Multi-Layer Perceptron (MLP) and Convolutional Neural Networks (CNN) as classifiers to be hyper-optimized. This part of code (code for classifers with eventual usage of GPU) is based on code available at `http://deeplearning.net/`.

Algorithm 1: Heuristic (TTTS)

```
Input     : $\beta$; $\eta$; $R$
Initialize: $s_{\max} = \lfloor \log_\eta(R) \rfloor$; $B = (s_{\max} + 1)R$; $L = \emptyset$; $t = 0$
1  for $s \leftarrow s_{\max}$ to 0 do
2      $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$;
3      $C$ = get_hyperparameter_configurations(n);
4      for $c \in C$ do
5          $L = L \cup \{$run_then_return_val_loss($c$)$\}$;
6      end
7      $t = n$;
8      // Begin TTTS;
9      while $t < B$ do
10         Sample $\hat{\theta} \sim \mathit{\Pi}_t$; $I \leftarrow \arg\max_{c \in C} \hat{\theta}_c$;
11         Sample $b \sim$ Bernoulli($\beta$);
12         if $b = 1$ then
13             $L = L \cup \{$run_then_return_val_loss($I$)$\}$;
14         else
15             Repeat sample $\hat{\theta} \sim \mathit{\Pi}_t$; $J \leftarrow \arg\max_{c \in C} \hat{\theta}_c$ until $I \neq J$;
16             $L = L \cup \{$run_then_return_val_loss($J$)$\}$;
17         end
18         Update $\mathit{\Pi}_t$;
19         $t = t + 1$;
20     end
21 end
22 return Configuration with the smallest intermediate loss seen so far
```

Dataset The MNIST dataset is pre-split into three parts: training set D_{train}, validation set D_{valid} and test set D_{test}.

Hyper-parameters The hyper-parameters to be optimized are listed below in Table 1, 2 and 3. For logistic regression, the hyper-parameters to be considered are learning rate and mini-batch size (since we are doing mini-batch SGD). For MLP, we take into account an additional hyper-parameter which is the l_2 regularization factor. For CNN (or LeNet), we take into account the number of kernels used in the two convolutional-pooling layers.

Resource Allocation The type of resource considered here is the number of epochs, where one epoch means a pass of training through the whole training set using SGD. Note that this is similar to the original Hyperband paper where one unit of resources corresponds to 100 mini-batch iterations for example. One epoch may contain a various number of mini-batch iterations depending on the mini-batch size.

Experimental Design In this section, we focus on neural network-typed classifiers, we will thus be maximizing the likelihood of the training set D_{train} under the model parameterized by θ (in this section, θ corresponds to (W, b) where W is the weight matrix and b is the bias vector):

$$\mathcal{L}(\theta, D_{\text{train}}) = \frac{1}{|D_{\text{train}}|} \sum_{i=1}^{|D_{\text{train}}|} \log(\mathbb{P}(Y = y^{(i)} | x^{(i)}, \theta),$$

which is equivalent to minimize the loss function:

$$\ell(\theta, D_{\text{train}}) = -\mathcal{L}(\theta, D_{\text{train}}).$$

At each time step t, we give one unit of resources (one epoch of training here) to the current algorithm, who will run one epoch of training on the training set D_{train}. The trained model is then used to predict output values $\hat{y}_{\text{pred},t}$ and $\tilde{y}_{\text{pred},t}$ respectively over validation set D_{valid} and test set D_{test}. We then compute

Algorithm 2: Detailed TTTS with Beta Prior for General Stochastic Bandits

```
   Input      : $n \leftarrow |C|$(number of arms), $\alpha_0$, $\beta_0$, $\beta$
   Initialize: $\forall c \in C, S_c = 0$(number of successes), $F_c = 0$(number of failures)
 1 for $t \leftarrow 1$ to $B$ do
 2     $\forall c \in C$, sample $\hat{\theta}_c \sim \text{Beta}(S_c + \alpha_0, F_c + \beta_0)$; $I \leftarrow \arg\max_{c \in C} \hat{\theta}_c$;
 3     Sample $b \sim \text{Bernoulli}(\beta)$;
 4     if $b = 1$ then
 5         Evaluate configuration $I$, and observe loss $\tilde{l}_t$;
 6     else
 7         Repeat $\forall c \in C$, sample $\hat{\theta}_c \sim \text{Beta}(S_c + \alpha_0, F_c + \beta_0)$; $J \leftarrow \arg\max_{c \in C} \hat{\theta}_c$ until $I \neq J$;
 8         Set $I \leftarrow J$;
 9         Evaluate configuration $I$, and observe loss $\tilde{l}_t$;
10     end
11     Sample $r_t \sim \text{Bernoulli}(1 - \tilde{l}_t)$;
12     if $r_t = 1$ then
13         $S_I \leftarrow S_I + 1$;
14     else
15         $F_I \leftarrow F_I + 1$;
16     end
17     $t = t + 1$;
18 end
```

Hyper-parameter	Type	Bounds
learning_rate	$\mathbb{R}^+$	$[10^{-3}, 10^{-1}]$ (log-scaled)
batch_size	$\mathbb{N}^+$	$[1, 1000]$

Table 1. Hyper-parameters to be optimized for logistic regression with SGD.

the number that were misclassified by the model, a.k.a. the zero-one loss on the validation and test set:

$$\ell_t(\hat{\theta}_t, D_{\text{valid}}) = \frac{1}{|D_{\text{valid}}|} \sum_{i=1}^{|D_{\text{valid}}|} \mathbb{1}_{\{\hat{y}^{(i)}_{\text{pred},t} \neq y^{(i)}\}},$$

$$\ell_t(\hat{\theta}_t, D_{\text{test}}) = \frac{1}{|D_{\text{valid}}|} \sum_{i=1}^{|D_{\text{valid}}|} \mathbb{1}_{\{\tilde{y}^{(i)}_{\text{pred},t} \neq y^{(i)}\}}.$$

During the experiment, we keep track of the best validation error and its associated test error. At each time step t, if the new validation error is smaller than the current best validation error, then we update the best validation error, and report the new test error. Otherwise we just report the test error associated with the previous best validation error.

Note that there is a very important 'keep training' notion here, which means if we are running the classifier with a same hyper-parameter configuration, then we do not restart the training from scratch. In contrary, we always keep track of previously trained weight matrix and bias vector with respect to the current hyper-parameter configuration, and train the model from these pre-trained parameters.

The total budget for Hyperband and the heuristic would be $B = R \times s_{\max}$, and each configuration can be evaluted for $R_{\max}$ times (this $R_{\max}$ depends only on R and $s_{\max}$). For HCT, we just need to feed the algorithm the total budget B, and the number of times that each configuration is evaluated will be decided by the algorithm itself. While for TPE, HOO and Random Search, we evaluate each configuration for $R_{\max}$ times in order to make a fair comparison, which means $B/R_{\max}$ configurations will be evaluted.

Comparison All plots here (from Fig. 1 to Fig. 2) are averaged on 10 trials of experiments.

Discussion In the current setting, Hyperband seems to be a plausible choice since it may explore more configurations compared to other algorithms.

Hyper-parameter	Type	Bounds
learning_rate	$\mathbb{R}^+$	$\left[10^{-3}, 10^{-1}\right]$ (log-scaled)
batch_size	$\mathbb{N}^+$	$[1, 1000]$
l_2_reg	$\mathbb{R}^+$	$\left[10^{-4}, 10^{-2}\right]$ (log-scaled)

Table 2. Hyper-parameters to be optimized for MLP with SGD.

Hyper-parameter	Type	Bounds
learning_rate	$\mathbb{R}^+$	$\left[10^{-3}, 10^{-1}\right]$ (log-scaled)
batch_size	$\mathbb{N}^+$	$[1, 1000]$
k_2	$\mathbb{N}^+$	$[10, 60]$
k_1	$\mathbb{N}^+$	$[5, k_2]$

Table 3. Hyper-parameters to be optimized for CNN with SGD.

4 UCI Datasets

We consider here Adaptive Boosting (AdaBoost), Gradient Boosting Machine (GBM), k-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), Support Vector Machine (SVM), Decision Tree and Random Forest from Scikit-learn.

Dataset Several datasets on UCI dataset archive (e.g. Wine, Breast Cancer, etc) are being used. They are all pre-split into a training set D_{train} and a test set D_{test}.

Hyper-parameters The hyper-parameters to be optimized are listed below in Table 4, 5, 6, 7, 8, 9 and 10.

Parameter	Type	Bounds
`learning_rate`	$\mathbb{R}^+$	$\left[10^{-5}, 10^{-1}\right]$
`n_estimators`	Integer	$\{5, \ldots, 200\}$

Table 4. Hyper-parameters to be optimized for AdaBoost models.

Resource Allocation One unit of resources in this setting is one iteration of training, which means one complete training of each classifier/regressor over the whole training set.

Experimental Design In this section we use the logarithmic loss, also known as cross-entropy for classification problem, defined by:

$$\ell(\theta, D) = -\frac{1}{|D|} \sum_{i=1}^{|D|} \sum_{j=1}^{m} y_j^{(i)} \log(\hat{p}_j^{(i)}),$$

where $\hat{p}_{ij}$ is the predicted probability of a sample i belonging to class j, and m is the number of classes considered. And for regression problems, the loss that we use is the typical mean squared error, defined by:

$$\ell(\theta, D) = -\frac{1}{|D|} \sum_{i=1}^{|D|} \left(y^{(i)} - \hat{y}_{\text{pred}}^{(i)}\right)^2.$$

And for this part of experiments, we choose to perform a shuffled $k = 5$ cross-validation scheme on D_{train} at each time step t. In practice, this means that we fit 5 models with the same architecture to different train/validation splits and average the loss results in each. More precisely, for every cross-validation split $\texttt{cv}_j, j = 1 \ldots 5$, we get a loss $\ell_{j,t}(\hat{\theta}_{j,t}, D_{\text{valid},j,t}) = \frac{1}{n} \sum_{i=1}^{n} \left(y_j^{(i)} - \hat{y}_{\text{pred},j,t}^{(i)}\right)^2$, where $n =$

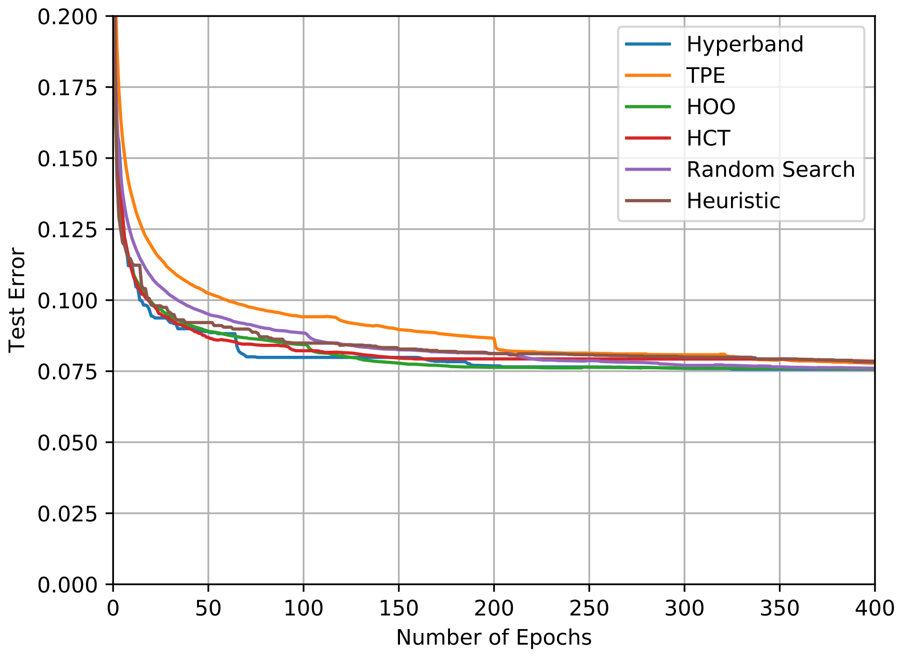

Fig. 1. Performance (log-scale) comparison of different hyper-parameter optimization algorithms on Logistic Regression, trained on MNIST Dataset.

Parameter	**Type**	**Bounds**
`learning_rate`	$\mathbb{R}^+$	$[10^{-5}, 10^{-2}]$
`n_estimators`	Integer	$\{10, \ldots, 100\}$
`max_depth`	Integer	$\{2, \ldots, 100\}$
`min_samples_split`	Integer	$\{2, \ldots, 100\}$

Table 5. Hyper-parameters to be optimized for GBM models.

$|D_{\text{valid}}|$ (here we take MSE as an example, it's the same for log-loss). Thus the validation loss at time t is

$$\frac{1}{5}\sum_{j=1}^{5}\ell_{j,t}(\hat{\theta}_{j,t}, D_{\text{valid},j,t}) = \frac{1}{5n}\sum_{j=1}^{5}\sum_{i=1}^{n}\left(y_j^{(i)} - \hat{y}_{\text{pred},j,t}^{(i)}\right)^2 .$$

Just like in the previous section, we can then compute and report the test error on the holdout test set D_{test}:

$$\ell_t(\hat{\theta}_t, D_{\text{test}}) = \frac{1}{|D_{\text{test}}|}\sum_{i=1}^{|D_{\text{test}}|}\left(y^{(i)} - \tilde{y}_{\text{pred},t}^{(i)}\right)^2 .$$

Note that under this experimental environment, 'keep training' does not make sense anymore. Thus for HOO, TPE and Random Search, we only need to evaluate each configuration once, contrarily to what we did in the previous setting. While for Hyperband, we still need to evaluate each configuration for a certain times based on R and $s_{\max}$.

Comparison Each plot here is averaged on 20 runs of experiments. Fig. 3 to 7 display the results for Wine Dataset.

Fig. 8 to 12 display the results for Breast Cancer Dataset.

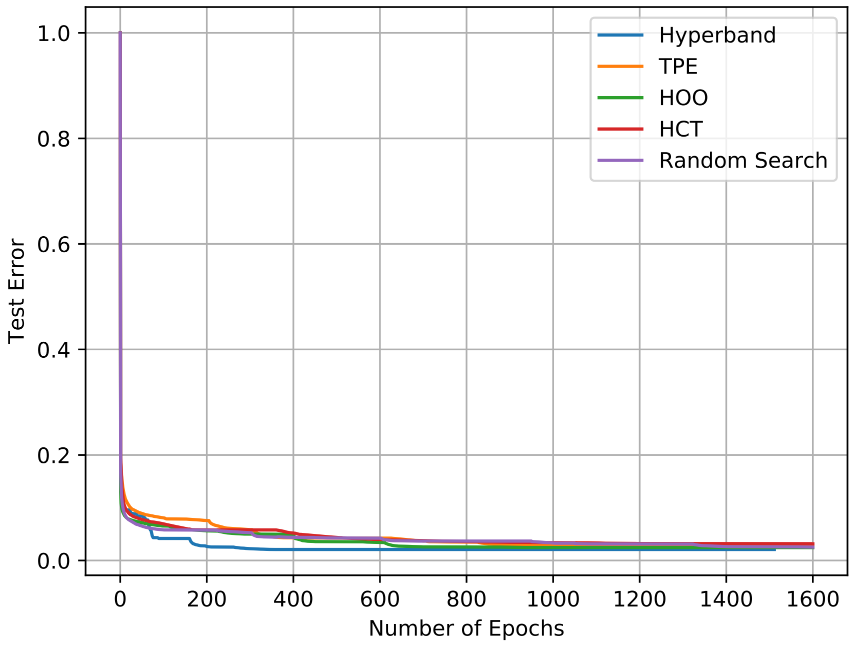

Fig. 2. Performance comparison of different hyper-parameter optimization algorithms on MLP, trained on MNIST Dataset.

Parameter	**Type**	**Bounds**
k	Integer	$\{10, \ldots, 50\}$

Table 6. Hyper-parameters to be optimized for KNN models.

Discussion In this setting, Hyperband seems to loose its advantage of exploring more configurations. It appears to me that there is no reason for HOO, TPE and Random Search to evaluate one point several times as does Hyperband.

References

1. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). Hyperband: A novel bandit-based approach to hyperparameter optimization. arXiv preprint arXiv:1603.06560.
2. Bergstra, J. S., Bardenet, R., Bengio, Y., & Kgl, B. (2011). Algorithms for hyper-parameter optimization. In Advances in neural information processing systems (pp. 2546-2554).
3. Bubeck, S., Munos, R., Stoltz, G., & Szepesvri, C. (2011). X-armed bandits. Journal of Machine Learning Research, 12(May), 1655-1695.
4. Azar, M. G., Lazaric, A., & Brunskill, E. (2014). Online stochastic optimization under correlated bandit feedback. In Proceedings of the 31st International Conference on Machine Learning (ICML-14) (pp. 1557-1565).
5. Russo, D. (2016, June). Simple bayesian algorithms for best arm identification. In Conference on Learning Theory (pp. 1417-1418).
6. Agrawal, S., & Goyal, N. (2012, June). Analysis of thompson sampling for the multi-armed bandit problem. In Conference on Learning Theory (pp. 39-1).

Parameter	Type	Bounds
`hidden_layer_size`	Integer	$[5, 50]$
`alpha`	$\mathbb{R}^+$	$[0, 0.9]$

Table 7. Hyper-parameters to be optimized for MLP models.

Parameter	Type	Bounds
C	$\mathbb{R}^+$	$[10^{-5}, 10^5]$ (log-scaled)
γ	$\mathbb{R}^+$	$[10^{-5}, 10^5]$ (log-scaled)

Table 8. Hyper-parameters to be optimized for SVM models.

Parameter	Type	Bounds
`max_features`	$\mathbb{R}^+$	$[0.01, 0.99]$
`max_depth`	Integer	$\{4, \ldots, 30\}$
`min_samples_split`	$\mathbb{R}^+$	$[0.01, 0.99]$

Table 9. Hyper-parameters to be optimized for Decision Tree models.

Parameter	Type	Bounds
`max_features`	$\mathbb{R}^+$	$[0.1, 0.5]$
`n_estimators`	Integer	$\{10, \ldots, 50\}$
`min_samples_split`	$\mathbb{R}^+$	$[0.1, 0.5]$

Table 10. Hyper-parameters to be optimized for Random Forest models.

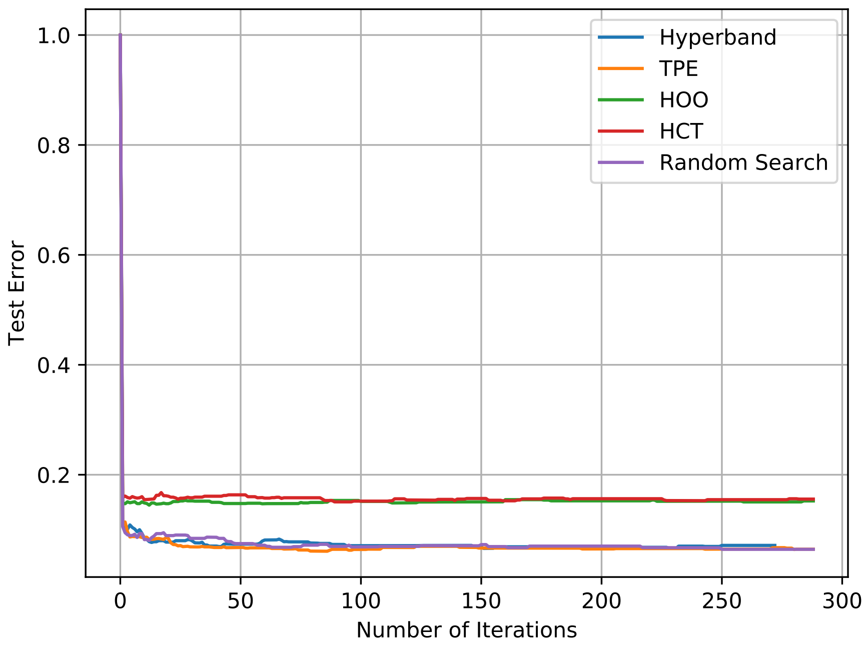

Fig. 3. Performance comparison of different hyper-parameter optimization algorithms on AdaBoost, trained on Wine Dataset.

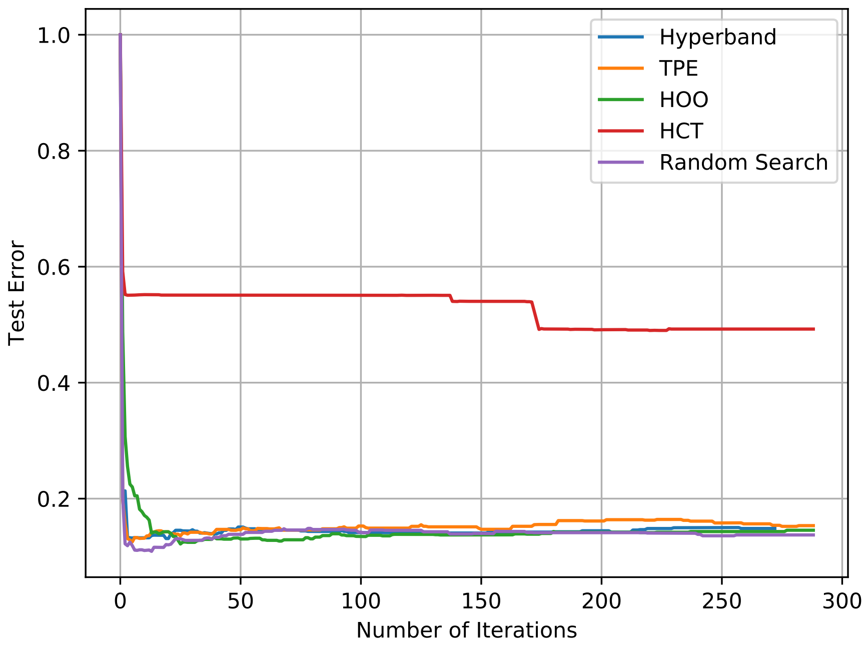

Fig. 4. Performance comparison of different hyper-parameter optimization algorithms on GBM, trained on Wine Dataset.

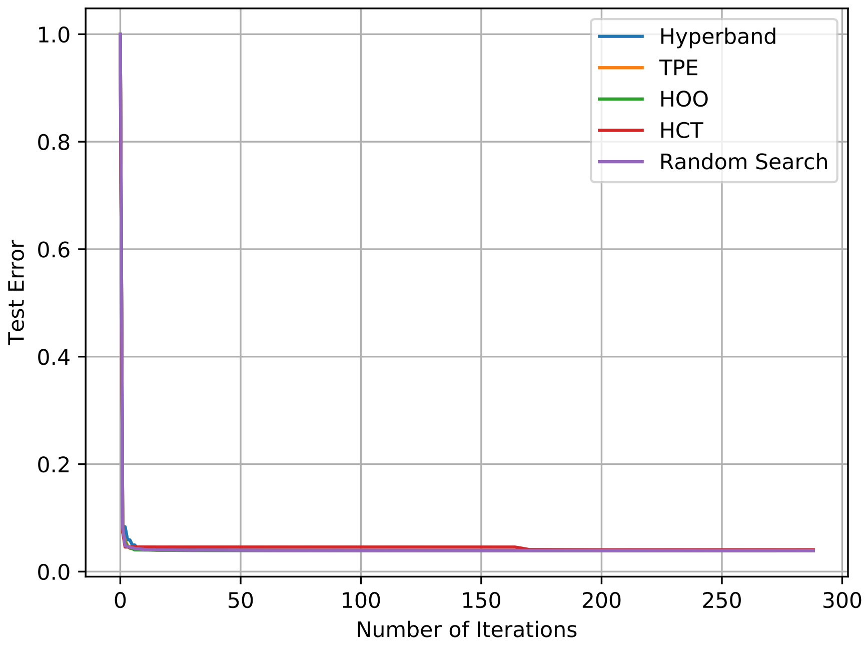

Fig. 5. Performance comparison of different hyper-parameter optimization algorithms on KNN, trained on Wine Dataset.

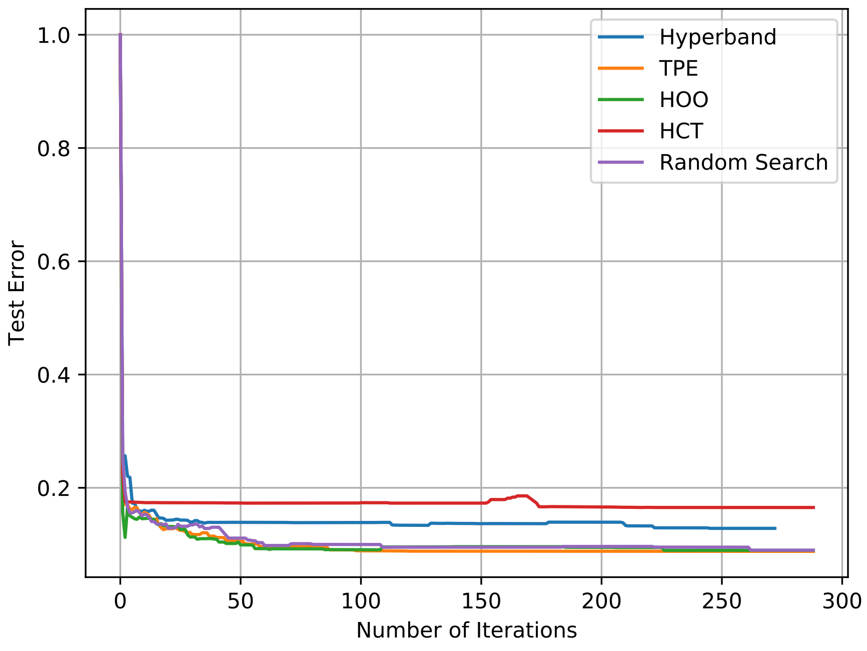

Fig. 6. Performance comparison of different hyper-parameter optimization algorithms on MLP, trained on Wine Dataset.

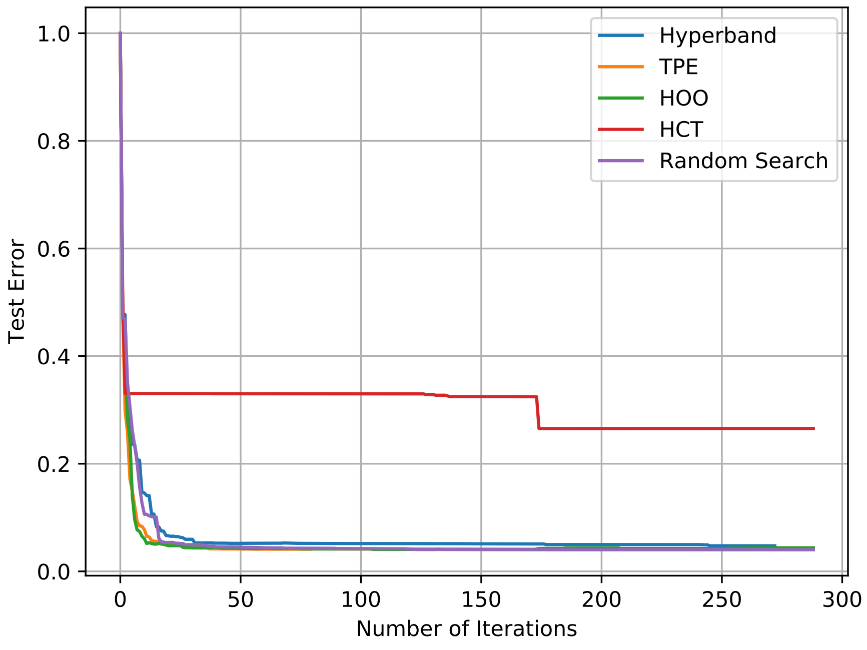

Fig. 7. Performance comparison of different hyper-parameter optimization algorithms on SVM, trained on Wine Dataset.

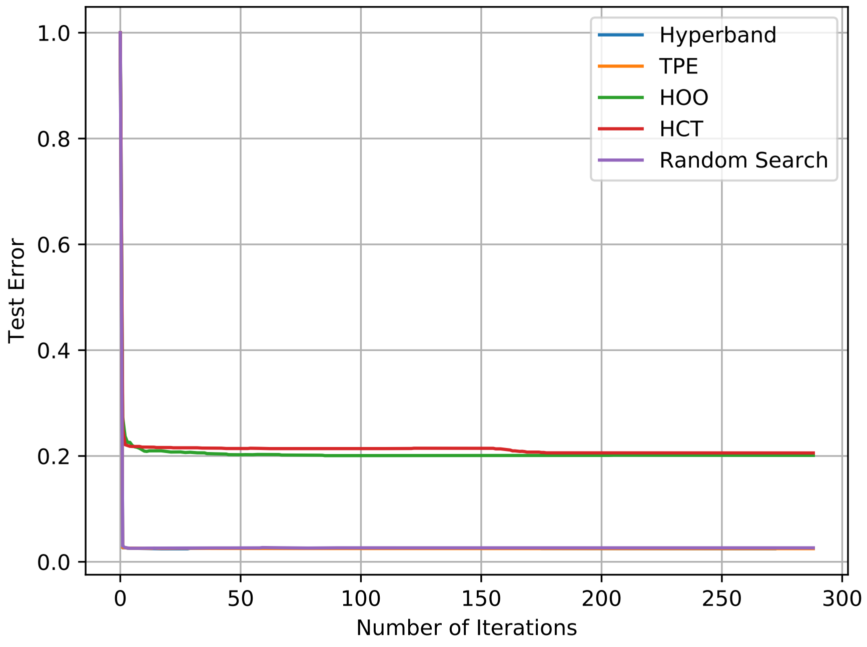

Fig. 8. Performance comparison of different hyper-parameter optimization algorithms on AdaBoost, trained on Breast Cancer Dataset.

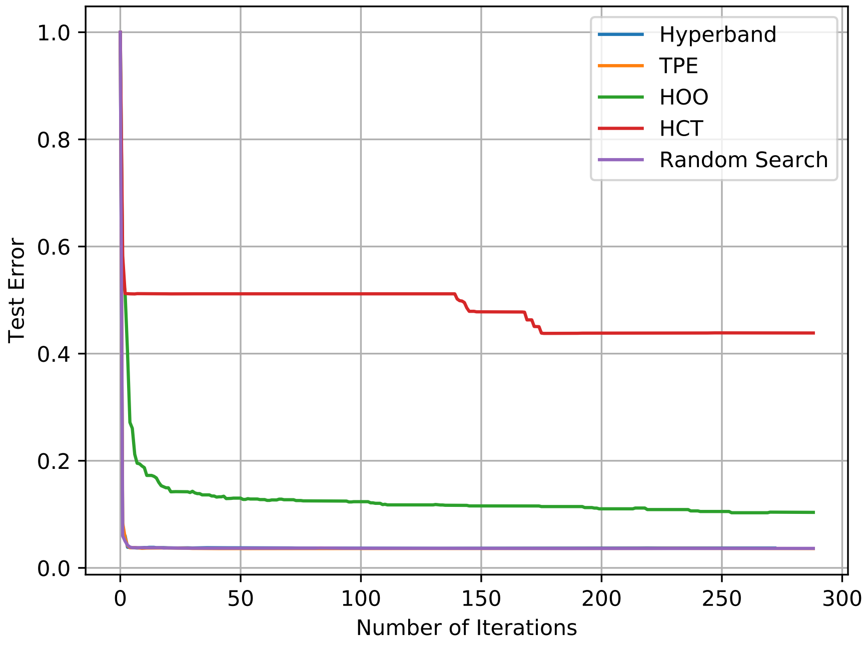

Fig. 9. Performance comparison of different hyper-parameter optimization algorithms on GBM, trained on Breast Cancer Dataset.

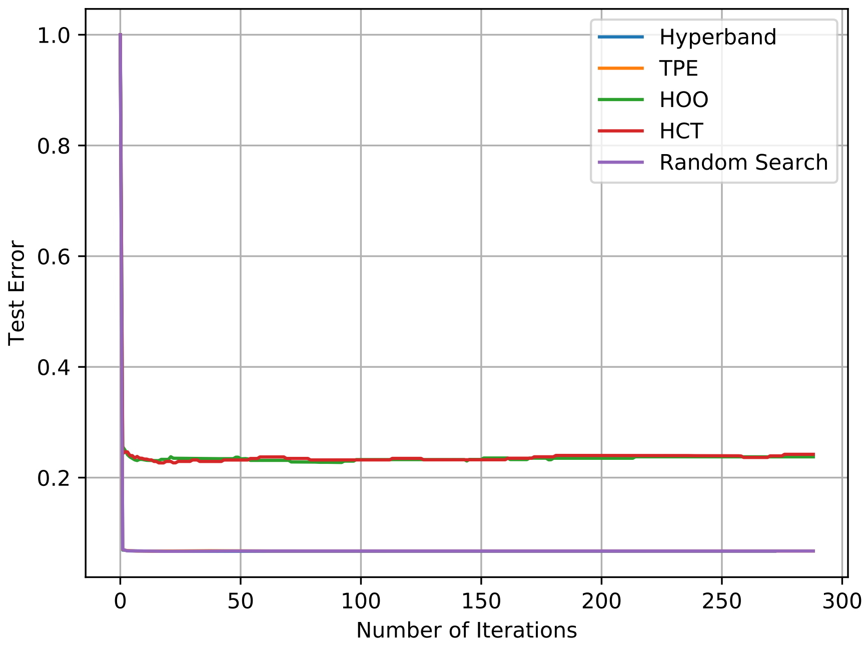

Fig. 10. Performance comparison of different hyper-parameter optimization algorithms on KNN, trained on Breast Cancer Dataset.

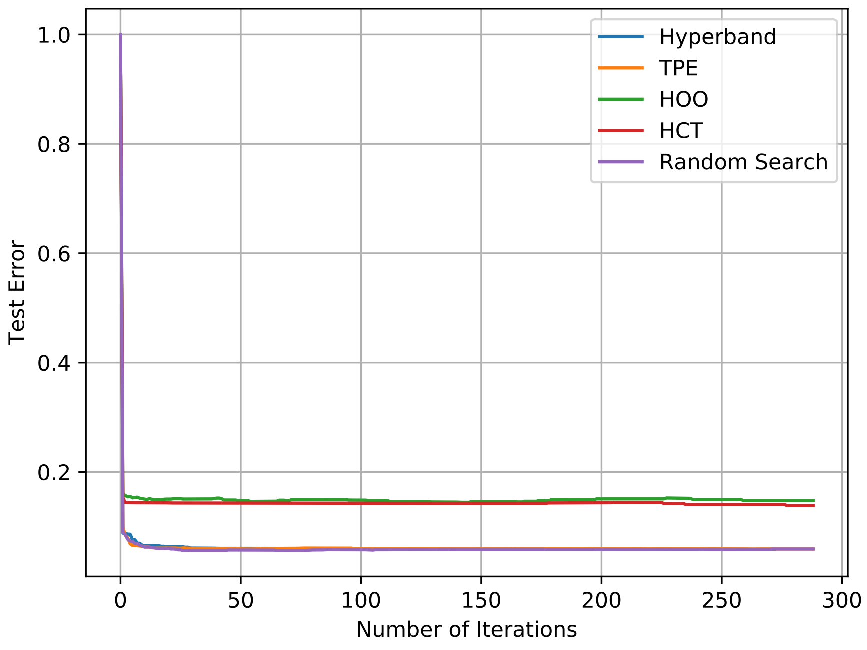

Fig. 11. Performance comparison of different hyper-parameter optimization algorithms on MLP, trained on Breast Cancer Dataset.

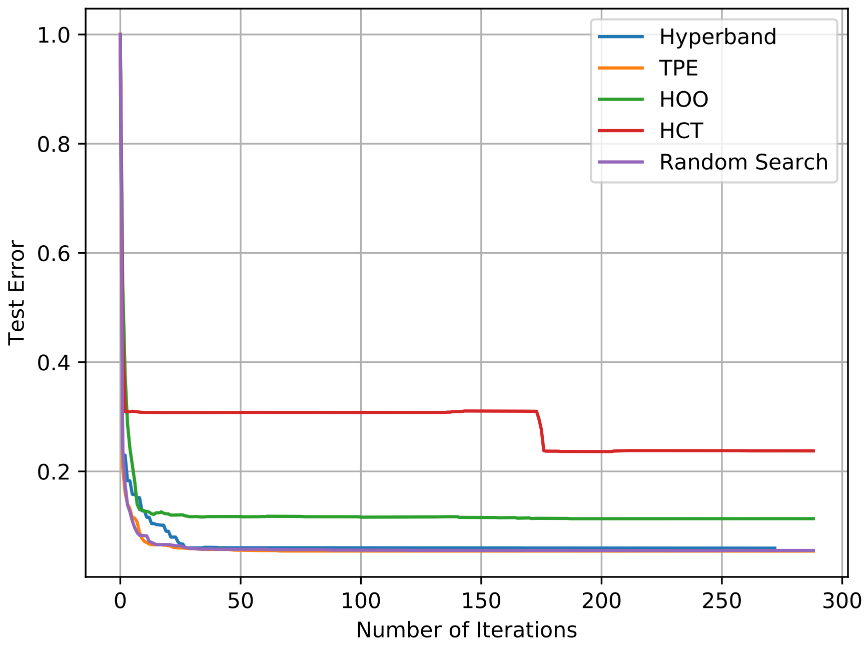

Fig. 12. Performance comparison of different hyper-parameter optimization algorithms on SVM, trained on Breast Cancer Dataset.