Efficient Hyperparameter Optimization of Deep Learning Algorithms Using Deterministic RBF Surrogates

Ilija Ilievski
Graduate School for Integrative Sciences and Engineering
National University of Singapore
ilija.ilievski@u.nus.edu

Taimoor Akhtar
Industrial and Systems Engineering
National University of Singapore
erita@nus.edu.sg

Jiashi Feng
Electrical and Computer Engineering
National University of Singapore
elefjia@nus.edu.sg

Christine Annette Shoemaker
Industrial and Systems Engineering
National University of Singapore
isesca@nus.edu.sg

Abstract

Automatically searching for optimal hyperparameter configurations is of crucial importance for applying deep learning algorithms in practice. Recently, Bayesian optimization has been proposed for optimizing hyperparameters of various machine learning algorithms. Those methods adopt probabilistic surrogate models like Gaussian processes to approximate and minimize the validation error function of hyperparameter values. However, probabilistic surrogates require accurate estimates of sufficient statistics (*e.g.*, covariance) of the error distribution and thus need many function evaluations with a sizeable number of hyperparameters. This makes them inefficient for optimizing hyperparameters of deep learning algorithms, which are highly expensive to evaluate. In this work, we propose a new deterministic and efficient hyperparameter optimization method that employs radial basis functions as error surrogates. The proposed mixed integer algorithm, called HORD, searches the surrogate for the most promising hyperparameter values through dynamic coordinate search and requires many fewer function evaluations. HORD does well in low dimensions but it is exceptionally better in higher dimensions. Extensive evaluations on MNIST and CIFAR-10 for four deep neural networks demonstrate HORD significantly outperforms the well-established Bayesian optimization methods such as GP, SMAC and TPE. For instance, on average, HORD is more than 6 times faster than GP-EI in obtaining the best configuration of 19 hyperparameters.

Introduction

Deep learning algorithms have been extensively used for various artificial intelligence related problems in recent years. However, their adoption is severely hampered by the many hyperparameter choices one must make, *e.g.*, architectures of the deep neural networks, forms of the activation functions and learning rates. Determining appropriate values of these hyperparameters is of crucial importance which

however is a frustratingly difficult task. As a result, automatic optimization of hyperparameters for deep learning algorithms has been attracting much research effort (Bergstra et al. 2011; Snoek, Larochelle, and Adams 2012; Bergstra, Yamins, and Cox 2013; Swersky, Snoek, and Adams 2014; Snoek et al. 2015).

Hyperparameter optimization is a global optimization of a black-box error function f whose evaluation is expensive — the function f maps a hyperparameter choice $\mathbf{x}$ of D configurable hyperparameters to validation error of a deep learning algorithm with learned parameters θ. Optimizing f as follows gives a way to automatically search for optimal hyperparameters:

$$
\begin{aligned}
\min_{\mathbf{x}\in\mathbb{R}^D} \quad & f(\mathbf{x}, \theta; \mathcal{Z}_{\text{val}}), \\
\text{s.t.} \quad & \theta = \arg\min_{\theta} f(\mathbf{x}, \theta; \mathcal{Z}_{\text{train}})
\end{aligned}
\tag{1}
$$

Here $\mathcal{Z}_{\text{train}}$ and $\mathcal{Z}_{\text{val}}$ denote the training and validation datasets respectively, θ is learned through minimizing the training error and $\mathbf{x}$ is in a bounded set. In the following, we omit (in the notation only) the dependency of f on θ and $\mathcal{Z}$, and denote it as $f(\mathbf{x})$ for short.

Solving the problem in Eq. (1) is very challenging due to high complexity of the function f. A popular solution is to employ Bayesian optimization algorithms — which use a cheap *probabilistic* surrogate model to approximate the expensive error function (Mockus, Tiesis, and Zilinskas 1978), such as Gaussian process (GP) (Snoek, Larochelle, and Adams 2012) and Tree-structured Parzen Estimator (TPE) (Bergstra et al. 2011).

In this work, we propose a new *deterministic*-surrogate based hyperparameter optimization method that we show requires considerably fewer function evaluations for the optimization. The proposed algorithm uses radial basis function (RBF) as surrogate to approximate the error function of hyperparameters, and avoids the expensive computation of covariance statistics needed in GP methods. The proposed algorithm searches the surrogate via dynamic hyperparameter

coordinate search and is able to find a near optimal hyper-parameter configuration with few expensive function evaluations, for either integer or continuous valued hyperparameters.

We name our algorithm as Hyperparameter Optimization via RBF and Dynamic coordinate search, or HORD for short. We compare our method against the well-established GP-based algorithms (*i.e.*, GP-EI and GP-PES) and against the tree-based algorithms (*i.e.*, TPE and SMAC) for optimizing the hyperparameters of two types of commonly used neural networks applied on the MNIST and CIFAR-10 benchmark datasets.

Our contributions can be summarized as follows:

- We provide a mixed integer deterministic-surrogate optimization algorithm for optimizing hyperparameters. The algorithm is capable of optimizing both continuous and integer hyperparameters of deep neural networks and performs equally well in low-dimensional hyperparameter spaces and exceptionally well in higher dimensions.

- Extensive evaluations demonstrate the superiority of the proposed algorithm over the state-of-the-art, both in finding a near-optimal configuration with fewer function evaluations and in achieving a lower final validation error. As discussed later, HORD obtains speedups of 3.7 to 6 fold over average computation time of other algorithms with 19-dimensional problem and is faster on average for all dimensions and all problems tested.

Related Work

Surrogate-based optimization (Mockus, Tiesis, and Zilinskas 1978) is a strategy for the global optimization of expensive black-box functions over a constrained domain. The goal is to obtain a near optimal solution with as few as possible function evaluations. The main idea is to utilize a surrogate model of the expensive function that can be inexpensively evaluated to determine the next most promising point for evaluation. Surrogate-based optimization methods differ in two aspects: the type of model used as a surrogate and the way the model is used to determine the next most promising point for expensive evaluation.

Bayesian optimization is a type of surrogate-based optimization, where the surrogate is a probabilistic model to compute the posterior estimate of distribution of the expensive error function values. The next most promising point is determined by optimizing an acquisition function of choice given the distribution estimation. Various Bayesian optimizations methods mainly differ in how to estimate the error distribution and definition on the acquisition function.

GP-EI (Snoek, Larochelle, and Adams 2012) and GP-PES (Hernández-Lobato, Hoffman, and Ghahramani 2014) methods use Gaussian processes to estimate distribution of the validation error given a hyperparameter configuration using the history of observations. The methods use the expected improvement (EI) and the predictive entropy search (PES) respectively as an acquisition function. Although GP is simple and flexible, GP-based optimization involves inverting an expensive covariance matrix and thus

scales cubically with the number of observations. To address the scalability issue of GP based methods, TPE is proposed by (Bergstra et al. 2011). TPE is a non-standard Bayesian-based optimization algorithm, which uses tree-structured Parzen estimators to model the error distribution in a non-parametric way. SMAC (Hutter, Hoos, and Leyton-Brown 2011) is another tree-based algorithm that uses random forests to estimate the error density.

In (2013) Eggensperger et al. empirically showed that Spearmints GP-based approaches reach state-of-the-art performance in optimizing few hyperparameters, while TPE's and SMAC's tree-structured approaches achieve best results in high-dimensional spaces. Thus, we compare our algorithm against these four methods.

The authors of (Eggensperger et al. 2015) propose an interesting way of using surrogate based models for hyperparameter optimization. They propose to build a surrogate of the validation error given a hyperparameter configuration to *evaluate and compare* different hyperparameter optimization algorithms. However, they do not unify the surrogate models with an optimization algorithm.

In (Snoek et al. 2015) the authors address the scalability issue of GB-based models and explore the use of neural networks as an alternative surrogate model. Their neural network based hyperparameter optimization algorithm uses only a fraction of the computational resources of GP-based algorithms, especially when the number of function evaluation grows to 2,000. However, as opposed to our proposed method, their approach does not outperform the GP-based algorithms nor the tree-based algorithms such as TPE and SMAC.

Description of The Proposed Method HORD

Here we introduce a novel hyperparameter optimization called Hyperparameter Optimization using RBF and Dynamic coordinate search, or HORD for short.

The surrogate model

We use the radial basis function (RBF) interpolation model as the surrogate model (Powell 1990) in searching for optimal hyperparameters. Given D number of hyperparameters, and n hyperparameter configurations, $\mathbf{x}_{1:n}$, with $\mathbf{x}_i \in \mathbb{R}^D$, and their corresponding validation errors $f_{i:n}$, where $f_i = f(\mathbf{x}_i)$, we define the RBF interpolation model as:

$$S_n(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i \phi(\|\mathbf{x} - \mathbf{x_i}\|) + p(\mathbf{x}) \qquad (2)$$

Here $\phi(r) = r^3$ denotes the cubic spline RBF, $\|\cdot\|$ is the Euclidean norm, and $p(\mathbf{x}) = \mathbf{b}^\top \mathbf{x} + a$ is the polynomial tail, with $\mathbf{b} = [b_1, \ldots, b_d]^\top \in \mathbb{R}^D$, and $a \in \mathbb{R}$. The parameters λ_i, $i = 1, \ldots, n$, b_k, $k = 1, \ldots, d$, and a are the interpolation model parameters, determined by solving the linear system of equations (Gutmann 2001):

$$\begin{bmatrix} \mathbf{\Phi} & \mathbf{P} \\ \mathbf{P}^\top & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{\lambda} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix} \qquad (3)$$

Here elements in the matrix $\boldsymbol{\Phi} \in \mathbb{R}^{n \times n}$ are defined as $\Phi_{i,j} = \phi(\|\mathbf{x_i} - \mathbf{x_j}\|), i, j = 1, \ldots, n$, $\mathbf{0}$ is a zero matrix of compatible dimension, and

$$
\mathbf{P} = \begin{bmatrix} \mathbf{x_1}^\top & 1 \\ \vdots & \\ \mathbf{x_n}^\top & 1 \end{bmatrix} \lambda = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix} \mathbf{c} = \begin{bmatrix} b_1 \\ \vdots \\ b_k \\ a \end{bmatrix} \mathbf{F} = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \quad (4)
$$

Searching the hyperparameter space

The training and evaluation of a deep neural network (DNN) can be regarded as a function f that maps the hyperparameter configuration used to train the network to the validation error obtained at the end. Optimizing $f(\mathbf{x})$ with respect to a hyperparameter configuration $\mathbf{x}$ as in Eq. (1) is a global optimization problem since the function $f(\mathbf{x})$ is typically highly multimodal. We employ the RBF interpolation to approximate the expensive function $f(\mathbf{x})$. We build the new *mixed integer* global optimization HORD algorithm with some features from the continuous global Dynamic coordinate search (DYCORS-LMSRBF) (Regis and Shoemaker 2013) to search the surrogate model for promising hyperparameter configurations.

As in the LMSRBF (Regis and Shoemaker 2007), the DYCORS-LMSRBF algorithm starts by fitting an initial surrogate model S_{n_0} using $A_{n_0} = \{(\mathbf{x}_i, f_i))\}_{i=1}^{n_0}$. We set $n_0 = 2(D + 1)$, and use the Latin hypercube sampling method to sample $\mathbf{x}_{1:n_0}$ hyperparameter configurations and obtain their respective validation errors $f_{1:n_0}$.

Next, while $n < N_{\max}$ a user-defined maximal evaluation number, the candidate point generation algorithm (Alg. 1) populates the candidate point set $\Omega_n = t_{n,1:m}$, with $m = 100D$ candidate points, and selects for evaluation the most promising point as $\mathbf{x}_{n+1}$. After $N_{\max}$ iterations, the algorithm returns the hyperparameter configuration $\mathbf{x}_{\text{best}}$ that yielded the lowest validation error $f(\mathbf{x}_{\text{best}})$. A formal algorithm description is given in Alg. 1.

Candidate hyperparameter generation and selection

We describe how to generate and select candidate hyperparameters in the following section.

The set of candidate hyperparameters Ω_n, at iteration n (Step 7 in Alg. 1) is populated by points generated by adding noise δ_i to some or all of the coordinates[1] of the current $\mathbf{x}_{\text{best}}$. In HORD, the expected number of coordinates perturbed is monotonically decreasing since perturbing *all* coordinates of the current $\mathbf{x}_{\text{best}}$ will result in a point much further from $\mathbf{x}_{\text{best}}$, a significant problem when the dimension D becomes large, *e.g.*, $D > 10$. Note that, during one iteration, each candidate point is generated by perturbing a potentially different subset of coordinates.

The probability of perturbing a coordinate (Step 6 in Alg. 1) in HORD declines with the number of iterations and

---

[1]A coordinate in the hyperparameter space, is one specific hyperparameter (*e.g.*, learning rate).

Algorithm 1 **H**yperparameter **O**ptimization using **R**BF-based surrogate and **DYCORS** (HORD)

---

input $n_0 = 2(D + 1)$, $m = 100D$ and $N_{\max}$.
output optimal hyperparameters $\mathbf{x}_{\text{best}}$.
1: Use Latin hypercube sampling to sample n_0 points and set $\mathcal{I} = \{\mathbf{x}_i\}_{i=1}^{n_0}$.
2: Evaluating $f(x)$ for points in $\mathcal{I}$ gives $\mathcal{A}_{n_0} = \{(\mathbf{x}_i, f(\mathbf{x}_i))\}_{i=1}^{n_0}$.
3: **while** $n < N_{\max}$ **do**
4:     Use $\mathcal{A}_n$ to fit or update the surrogate model $S_n(\mathbf{x})$ (Eq. 2).
5:     Set $\mathbf{x}_{\text{best}} = \arg\min\{f(\mathbf{x}_i) : i = 1, \ldots, n\}$.
6:     Compute φ_n (Eq. 5), i.e, the probability of perturbing a coordinate.
7:     Populate Ω_n with m candidate points, $\mathbf{t}_{n,1:m}$, where for each candidate $\mathbf{y}_j \in \mathbf{t}_{n,1:m}$, (a) Set $\mathbf{y}_j = \mathbf{x}_{\text{best}}$, (b) Select the coordinates of $\mathbf{y}_j$ to be perturbed with probability φ_n and (c) Add δ_i sampled from $\mathcal{N}(0, \sigma_n^2)$ to the coordinates of $\mathbf{y}_j$ selected in (b) and round to nearest integer if required.
8:     Calculate $V_n^{ev}(\mathbf{t}_{n,1:m})$ (Eq. 6), $V_n^{dm}(\mathbf{t}_{n,1:m})$ (Eq. 7), and the final weighted score $W_n(\mathbf{t}_{n,1:m})$ (Eq. 8).
9:     Set $\mathbf{x}^* = \arg\min\{W_n(\mathbf{t}_{n,1:m})\}$.
10:    Evaluate $f(\mathbf{x}^*)$.
11:    Adjust the variance σ_n^2 (see text).
12:    Update $\mathcal{A}_{n+1} = \{\mathcal{A}_n \cup (\mathbf{x}^*, f(\mathbf{x}^*))\}$.
13: **end while**
14: Return $\mathbf{x}_{\text{best}}$.

---

is given by:

$$
\varphi_n = \varphi_0 \left[1 - \frac{\ln(n - n_0 + 1)}{\ln(N_{\max} - n_0)} \right], \quad n_0 <= n < N_{\max}
$$
(5)

where φ_0 is set to $\min(20/D, 1)$ so that the average number of coordinates perturbed is always less than 20. HORD combines features from DYCORS (Regis and Shoemaker 2013) and SO-MI (Mueller, Shoemaker, and Piche 2013) to generate candidates. DYCORS is a continuous optimization RBF Surrogate method that uses Eq. 5 for controlling perturbations. SO-MI is the first mixed integer implementation of a RBF surrogate method.

The perturbation δ_i is sampled from $\mathcal{N}(0, \sigma_n^2)$. Initially, the variance $\sigma_{n_0}^2$ is set to 0.2, and after each $\mathcal{T}_{\text{fail}} = \max(5, D)$ consecutive iterations with no improvement over the current x_{best}, is set to $\sigma_{n+1}^2 = \min(\sigma_n^2/2, 0.005)$. The variance is doubled (capped at the initial value of 0.2), after 3 consecutive iterations with improvement. The adjustment of the variance (Step 11 in Alg. 1) is in place to facilitate the convergence of the optimization algorithm. Once the set of candidate points Ω has been populated with $m = 100D$ points, we estimate their corresponding validation errors by computing the surrogate values $S(\mathbf{t}_{1:m})$ (for clarity, we leave out n from notation). Here $\mathbf{t}$ denotes a candidate point.

We compute the distances from the previously evaluated points $\mathbf{x}_{1:n}$ for each $\mathbf{t} \in \Omega$ with $\Delta(\mathbf{t}) = \min \|\mathbf{t} - \mathbf{x}_{1:n}\|$. Here, $\|\cdot\|$ is the Euclidean norm. We also compute $\Delta^{\max} = \max\{\Delta(\mathbf{t}_{1:m})\}$ and $\Delta^{\min} = \min\{\Delta(\mathbf{t}_{1:m})\}$.

Finally, for each $\mathbf{t} \in \Omega$ we compute the score for the two criteria: V^{ev} for the surrogate estimated value:

$$V^{ev}(\mathbf{t}) = \begin{cases} \frac{S(t)-s^{\min}}{s^{\max}-s^{\min}}, & \text{if } s^{\max} \neq s^{\min}; \\ 1, & \text{otherwise.} \end{cases} \quad (6)$$

And the distance metric V^{dm}:

$$V^{dm}(\mathbf{t}) = \begin{cases} \frac{\Delta^{\max}-\Delta(t)}{\Delta^{\max}-\Delta^{\min}}, & \text{if } \Delta^{\max} \neq \Delta^{min}; \\ 1, & \text{otherwise.} \end{cases} \quad (7)$$

We use V^{ev} and V^{dm} to compute the final weighted score:

$$W(\mathbf{t}) = wV^{ev}(\mathbf{t}) + (1-w)V^{dm}(\mathbf{t}) \quad (8)$$

The final weighted score (Eq. 8) is the acquisition function used in HORD to select a new evaluation point. Here, w is a cyclic weight for balancing between global and local search. We cycle the following weights: $0.3, 0.5, 0.8, 0.95$ in sequential manner. We select the point with the lowest weighted score W to be the next evaluated point $\mathbf{x}_{n+1}$.

Muller and Shoemaker (2014) compare the above candidate approach with instead searching with a genetic algorithm. Krityakierne, Akhtar, and Shoemaker (2016) use multi-objective methods with RBF surrogate to select the next point for evaluation for single objective optimization of $f(x)$.

Experimental Setup

DNN problems We test HORD on four DNN hyperparameter optimization problems with 6, 8, 15 and 19 hyperparameters. Our first DNN problem consists of 4 continuous and 2 integer hyperparameters of a Multi-layer Perceptron (MLP) network applied to classifying grayscale images of handwritten digits from the popular benchmark dataset MNIST. This problem is also referred as **6-MLP** in subsequent discussions.

The MLP network consists of two hidden layers with *ReLU* activation between them and *SoftMax* at the end. As learning algorithm we use Stochastic Gradient Descent (SGD) to compute θ in Eq. 1. The hyperparameters (denoted as x in Eq. 1) we optimize with HORD and other algorithms include hyperparameters of the learning algorithm, the layer weight initialization hyperparameters, and network structure hyperparameters. Full details of the used datasets, the hyperparameters being optimized and their respective value ranges, as well as the values used as initial starting point (always required by SMAC) are provided in the supplementary materials.

The second DNN problem (referred as **8-CNN** in subsequent discussions), has 4 continuous and 4 integer hyperparameters of a more complex Convolutional Neural Network (CNN). The CNN consists of two convolutional blocks, each containing one convolutional layer with batch normalization, followed by *ReLU* activation and 3×3 max-pooling. Following the convolutional blocks, are two fully-connected layers with *LeakyReLU* activation, and *SoftMax* layer at the end.[2] We again use the MNIST dataset, and follow the same

---

[2]LeakyReLU is defined as $f(x) = \max(0, x) + \alpha * \min(0, x)$, where α is a hyperparameter.

problem setup as in the first DNN problem by optimizing similar hyperparameters (full details in supplementary).

The third DNN problem incorporates a higher number of hyperparameters. We increase the number of hyperparameters to 15, 10 continuous and 5 integer, and use the same CNN and setup as in the second experiment. This test problem is referred as **15-CNN** in subsequent discussion.

Finally, we design the fourth DNN problem on the more challenging dataset, CIFAR-10, and in even higher dimensional hyperparameter space by optimizing 19 hyperparameters, 14 continuous and 5 integer. This hyperparameter optimization problem is referred as **19-CNN** in subsequent discussions. We optimize the hyperparameters of the same CNN network from Problem 15-CNN, except that we include four dropout layers, two in each convolutional block and two after each fully-connected layer. We add the dropout rate of these four layers to the hyperparameters being optimized.

Baseline algorithms We compare HORD against Gaussian processes with expected improvement (GP-EI) (Snoek, Larochelle, and Adams 2012), Gaussian processes with predictive entropy search (GP-PES) (Hernández-Lobato, Hoffman, and Ghahramani 2014), the Tree Parzen Estimator (TPE) (Bergstra et al. 2011), and to the Sequential Model-based Algorithm Configuration (SMAC) (Hutter, Hoos, and Leyton-Brown 2011) algorithms.

Evaluation budget, trials and initial points DNN hyperparameter evaluation is typically computationally expensive, so it is desirable to find good hyperparameter values within a very limited evaluation budget. Accordingly, we limit the number of optimization iterations to 200 function evaluations, where one function evaluations involves one full training and evaluation of DNN. We run at least five trials of each experiment using different random seeds.

The SMAC algorithm starts the optimization from a manually set Initial Starting Point (ISP) (also called default hyperparameter configuration). The advantage of using manually set ISP as oppose to random samples becomes evident when optimizing high number of hyperparameters. Unfortunately, both GP-based algorithms, as well as TPE, cannot directly employ a manually set ISP to guide the search, thus they are in slight disadvantage.

On the other hand, HORD fits the initial surrogate model on n_0 Latin hypercube samples, but can also include manually added points that guide the search in a better region of the hyperparameter space. We denote this variant of HORD as HORD-ISP and test HORD-ISP on the 15-CNN and 19-CNN problems only. We set the ISP following common guidelines for setting the hyperparameters of a CNN network (exact values in supplementary). We supply the same ISP to both HORD-ISP and SMAC for the 15-CNN and 19-CNN problems. HORD-ISP performed the same as HORD on the 6 and 8 hyperparameter cases.

Implementation details We implement the HORD algorithm with the open-source surrogate optimization toolbox

pySOT (Eriksson, Bindel, and Shoemaker 2015). HORD has a number of algorithm parameters including n_0 the number of points in the initial Latin hypercube samples, m the number of candidate points, w the cycling weights used in Eq. 8 for computing candidate point score and the parameters controlling the changes in perturbation variances when there is no improvement. We have used the default values of these algorithm parameters as given in the pySOT software toolbox (since they were used in earlier RBF surrogate algorithms and tested on numerous benchmark optimization problems) and did not attempt to adjust them to improve performance. We use the Spearmint library[3] to obtain results for the GP-based algorithms. We use the HyperOpt library[4] to obtain results for the TPE method. We use the public implementation of SMAC. The HORD implementation, the networks being optimized, and other code necessary for reproducing the experiments are available at bit.ly/hord-aaai.

Numerical Results and Discussion

Algorithm Comparison Methodology   A purpose to the algorithm comparison we have done is to assess which algorithms are more likely to be efficient and how does this efficiency vary with the number of hyperparameters and across different problems. We also would like to identify some of the possible causes for this variation. Table 1 shows the mean and standard deviation for the best test set error obtained by each of the methods after two hundred evaluations for all of the test problems. We see here that for 3 out of 4 problems HORD performs better than other algorithms as per the test set error. However, we analyze in detail, the comparative performance of algorithms in subsequent discussions via validation error, since it is the objective function in Eq. 1.

Figure 1 plots the mean value of the best solution found so far as a function of the number of expensive evaluations of $f(\mathbf{x})$ in Eq. 1. The average is over five trials. Since we are minimizing, lowest curves are better. We can see from the horizontal dotted line in Figure 1, that for the 19-CNN problem, HORD reaches in only 54 evaluations the mean best validation error achieved by SMAC after 200 evaluations, *i.e*, HORD required only 27% of evaluations required by SMAC to get the same answer; 27% and 54 are shown in lower right corner of Table 2. Table 2 reports this percentage for HORD in comparison to all other algorithms and for all DNN hyperparameter optimization problems tested. All the percentages are significantly less than 100%, implying that in every problem, HORD obtained an equivalent mean answer to all other methods in less time.

Since HORD and all baseline algorithms are stochastic, it is also important to do a more stringent test statistical test to understand that if and after how many function evaluations (as a percentage of 200) does HORD achieves best validation error (in multiple trials) that is statistically better than the best validation error (multiple trials) achieved by other algorithms after 200 evaluations. We employ the rank sum test for this analysis and report the percentages in Table 3.
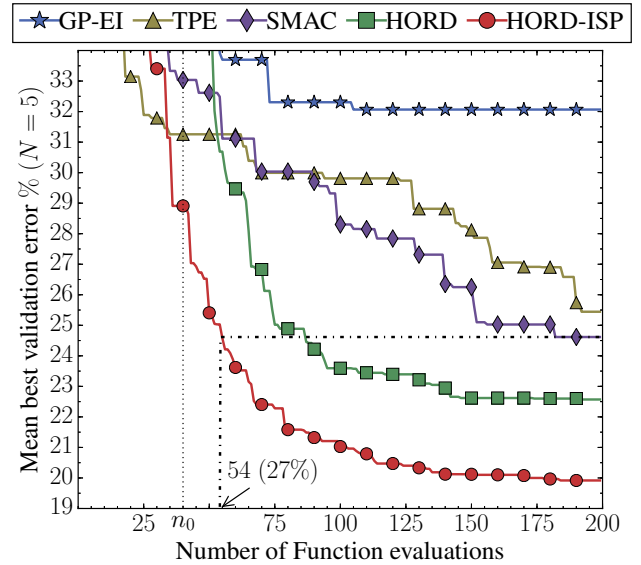
Figure 1: Efficiency comparison of HORD and HORD-ISP with baselines for optimizing a CNN with 19 hyperparameters on the CIFAR-10 dataset (19-CNN). We plot validation error curves of the compared methods against number of the function evaluations (averaged over 5 trials). HORD and HORD-ISP show to be significantly more efficient than other methods. HORD-ISP only takes 54 function evaluations to achieve the lowest validation error that the best baseline (SMAC) achieves after 200 evaluations.

For 15-CNN and 19-CNN problems the comparative analysis reported in Table 2 and Table 3 is against HORD-ISP.

To evaluate the performance of deep learning algorithms with optimized hyperparameters, we report the mean and standard deviation of best test set error for all algorithms and DNN problems after 200 evaluations in Table 1. Results indicate that HORD and HORD-ISP achieve the best results on average for all problems.

Results for Optimization of MLP and CNN network hyperparameters   For the 6-MLP problem we see that the average performance of HORD is better than all other algorithms after 200 evaluations (see Column 2 of Table 2). Furthermore, Table 3 (Column 2) clearly indicates that HORD found significantly better hyperparameters than the tree-based algorithms, SMAC and TPE, and GP-PES within 70% of evaluations required by other algorithms after 200 evaluations. The performances of HORD and GP-EI, however, are not statistically distinguishable (the p-value of rank sum test at 200 function evaluations is 0.54). The performance of the GP-EI algorithm is expected since GP-based methods are well suited to low-dimensional optimization. However, HORD also performs almost exactly the same, while using about 1,000 times less computing resources to propose new hyperparameter configurations. See supplementary for comparison of algorithm running times.

In the 8-CNN DNN case we again see that average performance of HORD is better than other algorithms after

Table 1: We report the mean and standard deviation (in parenthesis) of test set error obtained with the best found hyperparameter configuration after 200 iterations in at least 5 trials with different random seeds. The algorithm with the lowest test error is shown in bold.

Data Set	MNIST	MNIST	MNIST	CIFAR-10
Problem	6-MLP	8-CNN	15-CNN	19-CNN
GP-EI	1.94(.11)	**0.77**(**.07**)	0.99(.11)	37.19(4.1)
GP-PES	1.94(.07)	0.87(.04)	1.06(.07)	—
TPE	2.00(.079)	0.96(.07)	0.97(.03)	27.13(3.2)
SMAC	2.13(.11)	0.85(.07)	1.10(.07)	29.74(2.1)
HORD	**1.87**(**.06**)	0.84(.04)	0.94(.07)	23.23(1.9)
HORD-ISP	—	—	**0.82**(**.05**)	**20.54**(**1.2**)

Table 2: The percentage of function evaluations (actual function evaluations are also reported in parenthesis) required by HORD (6-MLP, 8-CNN) and HORD-ISP (15-CNN, 19-CNN) to reach the mean best validation error achieved by other algorithms after 200 function evaluations.

Data Set	MNIST	MNIST	MNIST	CIFAR-10
Problem	6-MLP	8-CNN	15-CNN	19-CNN
GP-EI	78%(155)	73%(145)	18%(36)	17%(33)
GP-PES	16%(32)	50%(100)	10%(20)	—
TPE	38%(75)	50%(100)	29%(58)	25%(49)
SMAC	20%(39)	28%(55)	20%(40)	27%(54)

200 function evaluations (see Column 3 of Table 2). Furthermore, performance of GP-EI is statistically comparable to HORD (p-values of rank sum tests are 0.391) after 200 function evaluations. However, HORD statistically significantly outperforms the tree-based algorithms (see Column 3 of Table 3) and GP-PES, which shows that our algorithm can perform well on a more complex network and with slightly higher number of integer hyperparameters.

Results for Optimization of CNN hyperparameters on MNIST and CIFAR-10   The 15-CNN and 19-CNN experiments confirm that higher dimensional spaces are where HORD-ISP truly shines (See Columns 4 and 5 of Table 2 and Table 3). As expected, the performance of the GP-based algorithms (see Column 5 of Table 3) degrades significantly in high dimensional search space. While, HORD-ISP (with the same ISP as SMAC) continues to perform well and even outperforms the tree-based algorithms, SMAC and TPE, which are designed specifically for optimizing high number of hyperparameters.

An analysis in Figure 2, indicates HORD and HORD-ISP are finding better solutions in most iterations than other algorithms for 19-CNN. We observe that after n_0 points are used to build the surrogate, HORD and HORD-ISP propose hyperparameter values that result in validation errors mostly in the range $20-30\%$. While, all the other algorithms are essentially performing a random search over the whole hyperparameter space. Furthermore, HORD and HORD-ISP after just few function evaluations from n_0, reach validation errors lower than the final lowest validation errors of all the other algorithms. We only compare HORD-ISP against GP,
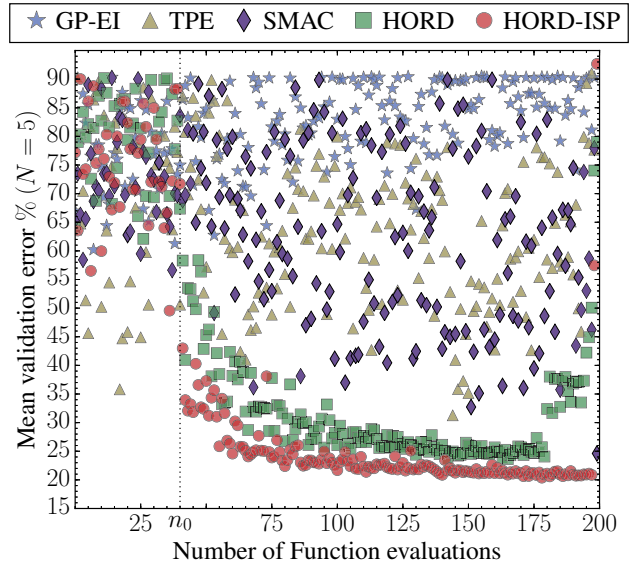

Figure 2: Mean validation error *v.s.* number of function evaluations of different methods for optimizing 19 hyperparameters of CNN on CIFAR-10. One dot represents validation error of an algorithm at the corresponding evaluation instance. After n_0 evaluations, the searching of HORD and HORD-ISP starts to focus on the hyperparameters with smaller validation error ($\leq 35\%$), in stark contrast with other methods.

TPE and SMAC for 19-CNN. HORD-ISP obtains significantly betters results than other algorithms after 200 evaluations, within 33% of 200 evaluations (see Column 5 of Table 3).

Discussion

The differences in results among the methods compared to HOD-ISP are dramatically different for the 15 and 19 dimensional hyperparameter problems. For example on 19-CNN problem, HORD-ISP obtains in 33 function evaluations the same average validation answer that GP-EI required 200 evaluations to obtain, meaning that HORD-ISP is about 6 (=200/33) times faster than GP-EI. Even compared to SMAC, which is designed to work better in higher dimensions, HORD is about 3.7 (=200/54) times faster based on average times (Table 2). When we use the more stringent statistical test (using 95% significance level) given in Table 3, we see that for example HORD-ISP is over 3 (=200/66) times faster than the best of the other algorithms, which is SMAC (which also has an initial starting point, ISP). HORD without an initial guess was also much better than GP-EI or GP-PES in higher dimensions (see Figure 2) when neither algorithm had an initial guess, both in terms of speed and in the quality of the answer HORD and HORD-ISP obtained after 200 evaluations. For the lowest dimension problems as shown in Table 2, the speed up of HORD compared to other methods is equal to or greater than 1.29 (=200/155) in comparison to the other methods. As table 3 shows some of the comparisons of HORD to the two GP methods on the lower

Table 3: We report the minimum percentage of function evaluations (actual function evaluations are also reported in parenthesis) required by HORD (6-MLP, 8-CNN) and HORD-ISP (15-CNN, 19-CNN) to obtain significantly (via Rank Sum Test) better best validation error than best error achieved by other algorithms after 200 function evaluations. If the difference in algorithms is insignificant at 200 function evaluations, we report it as a "–". All statistical tests are performed at 5 percent significance level.

| Data Set | MNIST | MNIST | MNIST | CIFAR-10 |
Problem	6-MLP	8-CNN	15-CNN	19-CNN
GP-EI	—	—	29%(58)	18%(35)
GP-PES	38%(75)	—	14%(27)	—
TPE	67%(134)	77%(154)	62%(123)	32%(64)
SMAC	39%(77)	72%(144)	29%(58)	33%(66)

dimensional problems are not statistically significant at the 95% level because of variability among trials of both methods. However, HORD (low dimension) and HORD-ISP (in higher dimensions) does get the best average answer over all dimensions and it is the only algorithm that performed well over all the dimensions.

To understand the differences in performance between the algorithms, we should consider the influence of surrogate type and the procedure for selecting the next point $\mathbf{x}$ where the algorithm evaluates the expensive function $f(\mathbf{x})$. With regard to surrogate type, comparisons have been made to the widely used EGO algorithm (Jones, Schonlau, and Welch 1998) that has a Gaussian process (GP) surrogate and uses maximizing Expected Improvement (EI) to search on the surrogate. EGO functions poorly at higher dimensions (Regis and Shoemaker 2013). With a deterministic RBF surrogate optimization method DYCORS was the best on all three multimodal problems in this same paper. The previous comparisons are based on numbers of function evaluation. In the current study, non-evaluation time was also seen to be much longer for the Gaussian Process based methods, presumably because they require a relatively large amount of computing to find the kriging surface and the covariance matrix, and that amount of computing increases rapidly as the dimension increases.

However, there are also major differences among algorithms in the procedure for selecting the next expensive evaluation point based on the surrogate that are possibly as important as the type of surrogate used. All of the algorithms compared randomly generated candidate points and select the one point to expensively evaluate with a sorting criterion based on the surrogate and location of previously evaluated points. GP-EI, GP-PES, TPE and SMAC all distribute these candidate points uniformly over the domain. By contrast HORD uses the strategy in LMSRBF (Regis and Shoemaker 2007) to create candidate points by generating them as perturbations (of length that is a normal random variable) around the current best solution. In (Regis and Shoemaker 2007) the uniform distribution and the distribution of normally random perturbations around the current best solution were compared for an RBF surrogate optimization and

the LMSRBF gave better results on a range of problems. HORD uses the DYCORS strategy to only perturb a faction of the dimensions, a strategy not used by the other 4 baseline methods. This is a strategy that greatly improves efficiency of higher dimensional problems in other applications as shown in (Regis and Shoemaker 2013). So these earlier papers indicate that the combination of methods used in HORD to generate trial points and to search on them as a weighted average of surrogate value and distance has been a very successful method on other kinds of continuous global optimization problems. So it is not surprising these methods also work well in HORD for the mixed integer problems of multimodal machine learning.

Conclusion

We introduce a new hyperparameter optimization algorithm HORD in this paper. Our results show that HORD (and its variant, HORD-ISP) can be significantly faster (e.g up to 6 times faster than the best of other methods on our numerical tests) on higher dimensional problems. HORD is more efficient than previous algorithms because it uses a deterministic Radial Basis surrogate and because it in each iteration generates candidate points in a different way that places more of them closer to the current best solution and reduces the expected number of dimensions that are perturbed as the number of iterations increase. We also present and test HORD-ISP, a variant of HORD with initial guess of the solution. We demonstrated the method is very effective for optimizing hyperparameters of deep learning algorithms.

In future, we plan to extend HORD to incorporate parallelization and mixture surrogate models for improving algorithm efficiency. The potential for improving efficiency of RBF surrogate based algorithms via parallelization is depicted in (Krityakierne, Akhtar, and Shoemaker 2016) and via mixture surrogate models is depicted in (Muller and Shoemaker 2014).

Acknowledgements

References

Bergstra, J.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *NIPS*.

Bergstra, J.; Yamins, D.; and Cox, D. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*.

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

Eggensperger, K.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2015. Efficient benchmarking of hyperparameter optimizers via surrogates. In *AAAI*, 1114–1120.

Eriksson, D.; Bindel, D.; and Shoemaker, C. 2015. Surrogate optimization toolbox (pysot). https://github.com/dme65/pySOT.

Gutmann, H.-M. 2001. A radial basis function method for global optimization. *Journal of Global Optimization*.

Hernández-Lobato, J. M.; Hoffman, M. W.; and Ghahramani, Z. 2014. Predictive entropy search for efficient global optimization of black-box functions. In *NIPS*.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*. Springer.

Jones, D. R.; Schonlau, M.; and Welch, W. J. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization* 13(4):455–492.

Krityakierne, T.; Akhtar, T.; and Shoemaker, C. A. 2016. Sop: parallel surrogate global optimization with pareto center selection for computationally expensive single objective problems. *Journal of Global Optimization* 66(3):417–437.

Mockus, J.; Tiesis, V.; and Zilinskas, A. 1978. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*.

Mueller, J.; Shoemaker, C. A.; and Piche, R. 2013. SO-MI: A Surrogate Model Algorithm for Computationally Expensive Nonlinear Mixed-Integer, Black-Box Global Optimization Problems. *Computers and Operations Research*.

Muller, J., and Shoemaker, C. 2014. Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms forcomputationally expensive black-box global optimization problems. *Journal of Global Optimization* 60(2):123–144.

Powell, M. J. 1990. *The theory of radial basis function approximation*. University of Cambridge.

Regis, R. G., and Shoemaker, C. A. 2007. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*.

Regis, R. G., and Shoemaker, C. A. 2013. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*.

Snoek, J.; Rippel, O.; Swersky, K.; Kiros, J. R.; Satish, N.; Sundaram, N.; Patwary, M. M. A.; Prabhat; and Adams, R. P. 2015. Scalable bayesian optimization using deep neural networks. In *ICML*.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *NIPS*.

Swersky, K.; Snoek, J.; and Adams, R. P. 2014. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*.