HyperSpace: Distributed Bayesian Hyperparameter Optimization

M. Todd Young, Jacob Hinkle, Arvind Ramanathan
Computational Science & Engineering Division
Oak Ridge National Laboratory
Oak Ridge, USA
youngmt1,hinklejd,ramanathana@ornl.gov

Ramakrishnan Kannan
Computer Science & Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, USA
kannanr@ornl.gov

***Abstract*—As machine learning models continue to increase in complexity, so does the potential number of free model parameters commonly known as hyperparameters. While there has been considerable progress toward finding optimal configurations of these hyperparameters, many optimization procedures are treated as black boxes. We believe optimization methods should not only return a set of optimized hyperparameters, but also give insight into the effects of model hyperparameter settings. To this end, we present *HyperSpace*, a parallel implementation of Bayesian sequential model-based optimization. HyperSpace leverages high performance computing (HPC) resources to better understand unknown, potentially non-convex hyperparameter search spaces. We show that it is possible to learn the dependencies between model hyperparameters through the optimization process. By partitioning large search spaces and running many optimization procedures in parallel, we also show that it is possible to discover families of good hyperparameter settings over a variety of models including unsupervised clustering, regression, and classification tasks.**

***Index Terms*—Bayesian optimization, SMBO, parallel computing, HPC**

I. Introduction

Machine learning (ML) models often contain numerous hyperparameters, free parameters that must be set before the models can be trained. Optimal settings for these hyperparameters are rarely known a priori, though their settings often dictate our algorithms' ability to learn from data. The challenge of hyperparameter tuning for ML algorithms can be attributed to several factors: (1) heterogeneity in the types of hyperparameters, (2) the potentially complex interactions between hyperparameters, and (3) the computational expense inherit to hyperparameter optimization. Though there has been considerable progress in hyperparameter optimization, optimization in this space remains hard.

As Snoek, Larochelle, and Adams [20] note, hyperparameters are often considered nuisances. However, this belies the fact that hyperparameters are design choices made by ML practitioners. When models consist of many hyperparameters, the differences between models presented across the literature could boil down to small changes in model architectures. If it is reasonable to expect that ML practitioners should be able to defend their choice of one model over others, it is also reasonable to expect that their choices for hyperparameter settings be defensible. It could even be the case that hyperparameter optimization could change a seemingly poor model choice into the best performing model for a given problem. We believe that careful study of our models' hyperparameters allows us to make more informed choices when modeling. It pays to know how our models' behavior changes as we vary our hyperparameters.

Some ML models such as non-negative matrix factorization (NMF) have relatively few hyperparameters. However, emerging deep learning (DL) algorithms often consist of significantly more tunable parameters. As the number of model hyperparameters increases, their optimization becomes significantly more challenging as we face a combinatorial increase in potential model configurations. Similarly, there is an increased chance that our models' hyperparameters interact in complex ways. Modeling these interactions in high-dimensional search spaces quickly becomes challenging and can often defy our intuition. Even experts find it difficult to manually configure hyperparameters [20].

While optimizing model performance with respect to hyperparameters is an obvious goal, we posit that hyperparameter optimization techniques must provide insight into how ML models' behavior changes across a range of possible model configurations. We argue that simply finding an optimal setting for our hyperparameters over a given training and validation set is rather uninformative. We may wonder, how much would our models' performance change if we just slightly perturbed our model hyperparameters? Would a small change in our hyperparameters mean a significant change in our models' behavior? After running through some fixed number of optimization iterations, do we believe that a particular setting of our hyperparameters is unique in that it enables our models to generalize well, or could there be several settings of hyperparameters that perform reasonably well? If we found that there are several good settings of hyperparameters, would they have anything in common? We believe these are important questions and that their answers could lead us to a better understanding of the ML models we employ.

To this end, we present *HyperSpace*, a distributed Bayesian optimization approach that allows one to utilize high performance computing (HPC) resources to (1) find hyperparameter settings that lead to more performant ML models, and (2) highlight regions in the hyperparameter search space where

our algorithms perform well *and* where they perform poorly. Our contributions are the following:

- We present a distributed Bayesian optimization algorithm that can scale across both HPC and cloud resources.
- Our approach can sample high-dimensional spaces to identify dependencies between hyperparameters.
- Finally, it can highlight areas in the hyperparameter search space where machine learning models perform well *and* where they perform poorly.

We demonstrate HyperSpace's ability to optimize hyperparameters for a variety of ML algorithms, including NMF, gradient boosted regression trees, and a convolutional neural network (CNN). We compare HyperSpace against the following optimization algorithms: `Spearmint` [9], `Hyperband` [13], `Scikit-Optimize` [8] and `Random Search` and show that the optimal settings obtained by HyperSpace are indeed better than current approaches. Finally, we believe that the availability of such a toolkit will be beneficial for the broader community and maintain an open source repository for HyperSpace at https://github.com/yngtodd/hyperspace.

II. Related Work

Approaches for automatically configuring hyperparameters broadly fit within two categories: model-based and model-free algorithms. Model-based approaches aim to create a surrogate model of some unknown function that would otherwise be to expensive to query in full [15]. Model-free approaches, on the other hand, are often simpler to implement but ignore the underlying structure of the problem we care about [2].

Sequential model-based optimization (SMBO) is a class of optimization algorithms that iterate between building a model of some unknown objective function and using the information from that model to query the next point in the domain of that function. They are commonly used when the objective function is very expensive to evaluate. A Bayesian approach to SMBO was first introduced by Mockus [15]. The authors Bergstra, Bardenet, Bengio, and Kégl [3] offer a detailed overview of applying SMBO for machine learning algorithms.

Snoek, Larochelle, and Adams [20] first proposed a Bayesian SMBO approach to optimize ML hyperparameters using Gaussian processes (GP), showing that GPs outperform human experts in configuring model hyperparameters. GPs are now commonly used as models in model-based approaches. They are an appealing model choice as they formalize the optimization problem as learning a distribution of functions consistent with our data. We give a brief introduction to GPs in Section III. For an excellent, full treatment of GPs, see [18].

Klein, Falkner, Bartels, Hennig, and Hutter [10] worked to speed up SMBO by exploring model configurations on subsets of the training set and extrapolating their performance on the full dataset.

In response to the Bayesian SMBO procedures, Hyperband framed hyperparameter optimization as a multi-armed bandit problem. This approach is "adaptive in computation, allocating more resources to promising hyperparameter configurations while quickly eliminating poor ones" [13]. In contrast to SMBO, Hyperband does not model the objective function being optimized; rather it randomly samples from the hyperparameter search space and progressively prunes poor performing settings after a fixed budget of computation.

Where Hyperband prunes search spaces found during the optimization procedure, JobPruner [19] uses information from previous experiments in order to prune non-promising spaces in the future. When combined with Bayesian optimization, this approach can lead to more efficient computation as future experiments require fewer resources. Search space pruning for HPC applications was also explored outside of ML/DL algorithms in [14].

While there has been considerable progress in hyperparmater optimization research, Li, Jamieson, DeSalvo, Rostamizadeh, and Talwalkar [13] has shown that random search remains a competitive approach that works well in practice. However, random search bears a little to no insight as to why certain settings of hyperparameters might be useful. While random search is a simple strategy to implement, it hardly informs us about the behaviour of the models. In light of iterative nature of machine learning, we would ideally have hyperparameter optimization approaches that will inform future iterations, and this is something that random search cannot do.

III. Methods

In this section, we describe the design and development of HyperSpace, our distributed hyperparameter optimization software. First, we provide an overview of the Bayesian optimization strategy and next, we describe our parallel implementation of the Bayesian SMBO approach.

A. Bayesian Optimization

Following the work of [5, 20], we approach the problem of configuring our models' hyperparameters using Bayesian optimization. Given a computationally challenging objective function $f : X \to \mathbb{R}^+$ over a compact hyperparameter domain X[1], Bayesian optimization is an approach to minimizing f without using gradient information, which we will briefly describe here.

First, we sample a small, predefined number of $n = 10$ points $x_i \in X$ uniformly randomly, and compute the function values at those locations, $f(x_1), \ldots, f(x_n)$. We then model f by fitting a probabilistic model for the function. Here we assume that f is drawn from a Gaussian process (GP). A GP defines a prior distribution over the universe of possible functions which can be transformed into a posterior distribution over functions after observing some data [18]. This posterior provides a probability distribution over functions given only a finite set of points $x_1, \ldots, x_n$. The Gaussian process assumes *a priori* that the probability $p(f(x_1), \ldots, f(x_n))$ is jointly multivariate Gaussian distributed, specified by the mean function $\mu(x)$ and covariance function $\kappa(x_i, x_j)$, where κ is a positive definite kernel function such as the commonly

[1]Typically X is a high dimensional rectangle with finite-length sides corresponding to acceptable ranges for each hyperparameter.

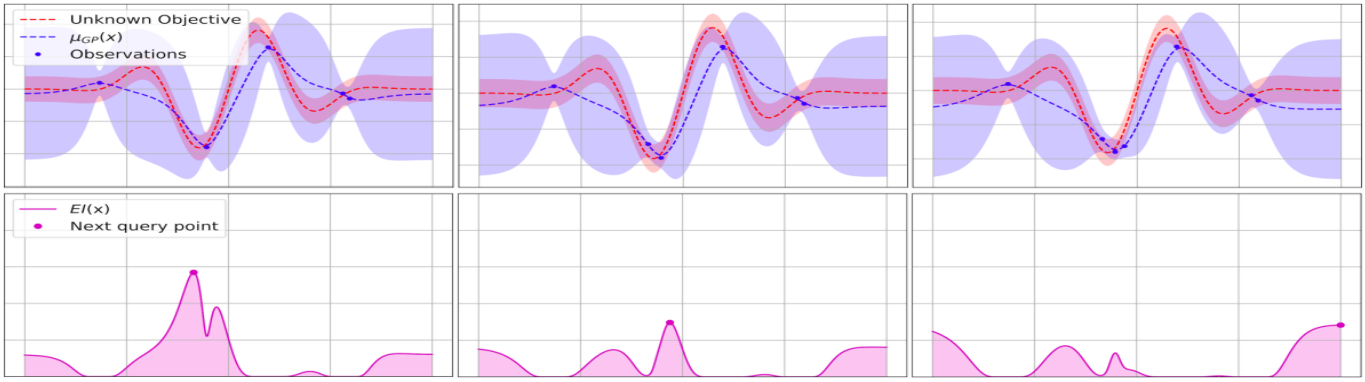

Fig. 1. Bayesian optimization of a one dimensional objective function. From left to right shows three successive steps of the Bayesian optimization process. Here the x-axis represents the domain of the objective function, and the y-axis represents $f(x)$. At each step, the top figure shows the true, unknown objective function we would like to model in red, and the Gaussian process model of the objective in purple. The bottom figure shows the expected improvement acquisition function. The maximum of the expected improvement function at each step of the optimization indicates where in the objective function's domain should the Gaussian process should query next.

used squared exponential kernel, the rational quadratic kernel, and the Matérn kernel [16]. As a default, HyperSpace uses a Matérn kernel though it is possible to specify the other options mentioned. Modeling f using a Gaussian process gives a posterior predictive mean function $\mu(x)$ and a posterior predictive marginal variance function $\sigma^2(x)$, both of which are defined over the entire domain X and are computed in closed form.

We then must decide where to sample our next point, x_{n+1}, from X as we seek the location of the function's minimum. This is controlled by a proxy optimization of an acquisition function, $u : X \to \mathbb{R}^+$:

$$x_{n+1} = \arg\max_x \; u(x)$$

We use the expected improvement algorithm [15] and define

$$u_{EI}(x) = E\left[\max\left(0, \;\; f(x^*) - f(x)\right)\right] \tag{1}$$

where $f(x^*)$ is the minimal observed value of f thus far and $E[\cdot]$ denotes expectation over the random variable $f(x)$. We therefore receive a reward equal to the "improvement" $f(x^*) - f(x)$ when $f(x)$ is less than $f(x^*)$, and no reward otherwise. Given the GP's predictive mean and variance functions, the right hand side of Eq. 1 can be re-expressed as

$$u_{EI}(x) = (f(x^*) - \mu(x))\Phi\left(Z\right) + \phi(Z),$$

where Φ is the standard normal cumulative distribution function, ϕ is its derivative, and $Z = \frac{f(x^*) - \mu(x)}{\sigma(x)}$ [5].

The expected improvement balances two competing aims: the need to explore the domain of our objective function and the desire to exploit points on the domain which may be the global minimum of the function. A natural question to ask is, when we find some minimum, is it the global minimum or some local minimum? Should we continue our search for the global minimum or be satisfied with the lowest minimum found? Balancing these two aims is encoded in the two components of the expected improvement above. First, we can maximize our expected improvement by reducing our Gaussian processes' mean function $\mu(x)$. Secondly, we can maximize the expected improvement by increasing our predictive marginal variance $\sigma^2(x)$. In other words, the expected improvement is maximized when we exploit points in the domain of f where $f(x)$ is minimal and when we explore points in the domain where we are most uncertain. Figure 1 shows three successive steps of the Bayesian optimization using the expected improvement acquisition function

B. *Parallelism*

HyperSpace approaches parallelism by concurrently running many Gaussian processes over our large hyperparameter search space. This is done by partitioning the hyperparameter search space, allowing us to model many regions of potential algorithm configurations. We emphasize the partitioning the search space since we are interested in how the changes of this optimization space affects our learner's performance. This differs from the parallel approach presented in [20], where the authors parallelize a single Gaussian process. By running many GPs in parallel, we are able to compare our models' performance over various areas in the large hyperparameter search space as each GP grants a unique perspective over the space. More critically, partitioning the search space means that a more detailed search is performed as many more models are explored across a diverse set of search spaces. This addresses concerns of exponential scaling acknowledged in early papers on Bayesian optimization theory [21, 6], increasing the odds of finding near optimal values.

The hyperparameter search space is partitioned as follows. For each hyperparameter, we define a lower and upper bound believed to be limits of good potential hyperparameter settings.

While Gaussian processes are defined over continuous domains, HyperSpace can optimize over categorical and integer valued spaces. This is done by first one-hot encoding the search dimensions and embedding them in $\mathbb{R}$. At each modeling step, we then take the $\arg\max$ of the one-hot encoded dimension as we maximize the acquisition function.

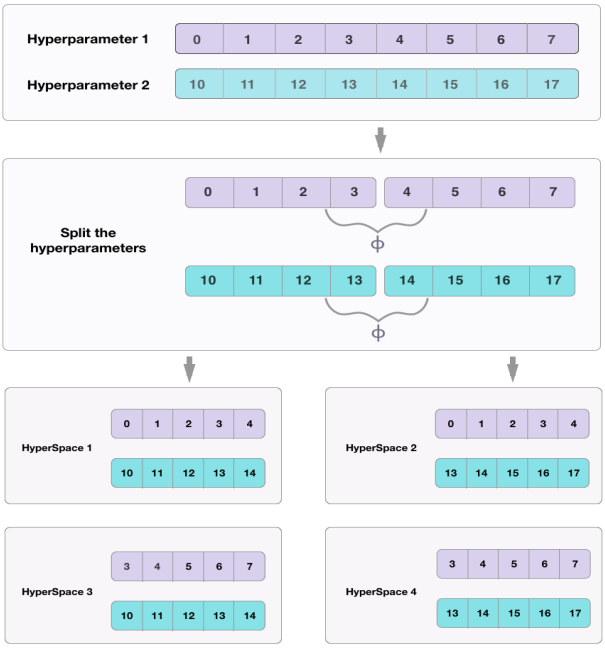

Fig. 2. Partitioning two hyperparameters with HyperSpace. Two hyperparameters, both integer valued, are each represented as a vector of eight elements. Each hyperparameter is divided into two equal subintervals, each consisting of four elements. Using $\phi = 0.25$, a single element from the lower and upper subintervals are appended to each subinterval respectively, making each subinterval of size five. The lower subinterval's maximum is now the minimum of the upper subinterval, and vice versa. Finally, all possible hyperparameter subintervals are combined to create the distributed search spaces of HyperSpace.

We then partition each hyperparameter bound into two near equally sized subspaces with a degree of overlap, $\phi \in [0, 1]$. When $\phi = 0$, there is no overlap between the subspaces.When $\phi = 1$, there is perfect overlap and two copies of the original space are created. For all experiments, the default value of $\phi = 0.5$ was used, allowing each of the parallel Gaussian processes to sample from one half of their neighboring spaces.

The overlap, ϕ then allows each of the Gaussian processes in our optimization loop to sample from a shared range. While each Gaussian process gains a unique perspective by optimizing over some small subspace, the overlap parameter allows each of the optimizers a partial view into the space of their adjacent neighbors. This is similar to the considerations of [11] for sensor placement when using Gaussian processes for spatial learning, namely that the placement of our sensors (optimizers in our case) will affect how well we learn.

After dividing the original search space, we then create all possible combinations of these subintervals, what we call 'hyperspaces'. A total of $2^H - S$ subspaces are created where H is the number of hyperparameters whose interval bounds exceed some minimal length, α, and S is the number of hyperparameters whose interval bounds length is less than α. Each of these hyperspaces is subsequently distributed across an HPC system. At each node, we then run the Bayesian optimization loop as previously described for some predetermined m iterations in parallel. Figure 2 gives an example of partitioning the search space for two hyperparameters. For a compact representation of HyperSpace, see Algorithm 1.

By leveraging HPC resources, HyperSpace is able to run $(2^H - S) * m$ iterations of Bayesian optimization in the time that it takes traditional SMBO algorithms to run m iterations. This is a great advantage of our approach; sampling our objective many more times than traditional SMBO approaches allows us to gain a more complete picture our models under various configurations.

Algorithm 1: HyperSpace

Input: Hyperparameter intervals, minimal subinterval length α, overlap ϕ
Output: Optimization results over 2^N hyperspaces

```
for each hyperparameter interval do
    if interval length ≥ α then
        partition interval into two nearly equal
          subintervals with overlap φ;
    else
        do not split;
combine all possible subintervals to form hyperspaces;
for each hyperspace do in parallel
    run Bayesian optimization;
return optimization results for each hyperspace
```

IV. Results

In this section, we demonstrate HyperSpace on real world datasets such as RCV1, Boston Housing and MNIST dataset over different ML approaches such as regularized NMF, Gradient Boosted Regressor and a convolutional neural network (CNN) respectively. We compare our algorithm against baselines `Spearmint`, `Hyperband`, `Scikit-Optimize` and `Random Search`. In each case, we show that HypeSpace outperforms the different baselines.

A. Regularized NMF

We begin with a simple latent factor model, namely NMF for determining clusters in sparse text documents of the RCV1 dataset [12]. NMF is a multivariate analysis of a data matrix $\mathbf{X}$ that is factorized into two matrices $\mathbf{W}$ and $\mathbf{H}$, under the constraint that none of the three matrices have any negative elements. NMF is typically used for clustering data.

We used an off-the-shelf NMF implementation based on elastic net regularization from the `Scikit-Learn` that minimizes the following optimization function

$$\underset{\mathbf{W}\geq 0,\mathbf{H}\geq 0}{\arg\min}\ 0.5\|X - WH\|_F^2 + \alpha\beta\|vec(W)\|_1 + \alpha\beta\|vec(H)\|_1 + 0.5\alpha(1-\beta)\|W\|_F^2 + 0.5\alpha(1-\beta)\|H\|_F^2 \quad (2)$$

In the above equation α is an importance factor and $\beta \in [0, 1]$ determines the ratio of the ℓ_1 over ℓ_2. The matrix $X \in$

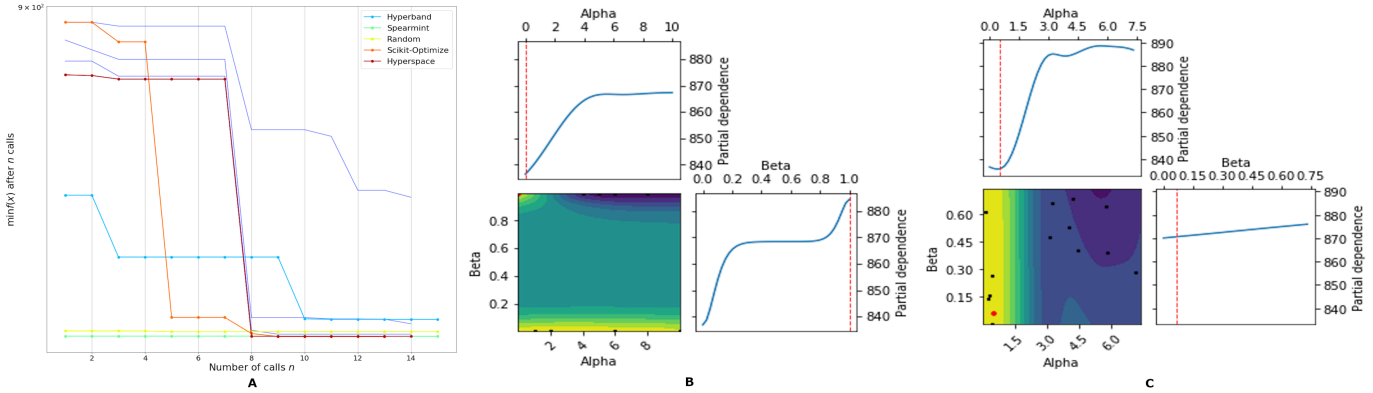

Fig. 3. Optimization over regularized NMF where the search space is two dimensional. Bounds for the values of α, β were chosen as specified in the Table I. When the search space is low dimensional, all optimization algorithms converge to similar points as shown in plot A. Plot B shows the partial dependence of our objective function with respect to settings of α and β for the worst performing model found by HyperSpace. Plot C shows the partial dependence of our objective function with respect to settings of α and β for the best performing model found by HyperSpace.

$\mathbb{R}^{800,000\times47,236}$ is frequency based bag of words matrix from the RCV1 dataset with 800,000 samples and 47,236 features. We chose a low rank factor k=103 to represent the number of different categories in the dataset. The NMF algorithm yielded $W \in \mathbb{R}^{800,000\times103}$ and $H \in \mathbb{R}^{103\times47,236}$.

We benchmark HyperSpace against each of the optimization procedures using NMF to illustrate how the various approaches behave when the search space is low dimensional.

TABLE I
ORIGINAL SEARCH SPACE FOR NMF

hyperparameter	lower bound	upper bound
importance factor: α	0	10
ℓ_1 to ℓ_2 ratio: β	0.0	1.0

Figure 3.A shows the convergence for each of the optimization approaches. All algorithms except Hyberband converge to similar minima. As the search space consists of only two dimensions, we expected that each of the algorithms would yield good results. As we later show, when the dimensions of the hyperparameter search space increases, results from each of the optimization algorithms begins to diverge.

As HyperSpace runs parallel optimization procedures over subsets of the overall search space, we can compare the results of each parallel run. Figure 3.B and Figure 3.C show the partial dependence plots of each of our hyperparameters for the worst and best parallel runs respectively. The diagonal of each partial dependence plot shows how various settings of each hyperparmater affects the minimization of the objective function when holding all other hyperparameters constant. The off diagonal plots show two-way interactions between the hyperparameters. Yellow regions in the off-diagonal plots indicate lower values. One thing to note here is that the partial dependence plot corresponding to the best hyperspace (Figure 3.C) shows a larger space of potential minimal values compared with the plot for the worst performing hyperspace (center). Points in this plot show where the GP sampled within each region. The optimal setting found by HyperSpace is indicated by the red point.

These partial dependence plots become very valuable as they allow us to inspect how choices of our hyperparameter search bounds affects the optimization procedure. They give us insight into interactions between our hyperparameters in various regions of the search space. Comparing how these partial dependencies behave across subsets of the search space allows us make more informed decision about the space of all possible models for a given problem.

B. Gradient Boosted Regression

Gradient boosting machines, first introduced by [4], remain one of the most widely used algorithms in ML competitions such as Kaggle. While effective learners, they have many free hyperparameters. As is the case with many ML models, finding reasonable settings for these hyperparameters requires experience and intuition.

Here we optimize the hyperparameters of a gradient boosted machine in order to predict Boston housing prices. The Boston housing data [7] is a small dataset consisting of 506 samples and 13 features and is readily available through `scikit-learn`'s application programming interface [17]. Whereas our NMF model had only two hyperparameters, the gradient boosting machine has seven hyperparameters. With the number of hyperparameters being more than half the number of features in our data, the choices of our hyperparameters are potentially more important the representation of our data. Table II gives the bounds for each of our hyperparameters.

The convergence plot for all of the optimization procedures in Figure 4.A shows the log mean absolute error over five fold cross-validation for our gradient boosting regressor. We find that random search converges to its lowest point with the least number of iterations (ten) and outperforms `Hyperband` on this task. Random search also finds nearly the same minimum compared with the `Scikit-Optimize` baseline (the Bayesian optimization over the global search space). Spearmint outperforms random search, `Scikit-Optimize`, and `Hyperband`, but requires 41 iterations in order to converge to its lowest point. HyperSpace converges to a lower

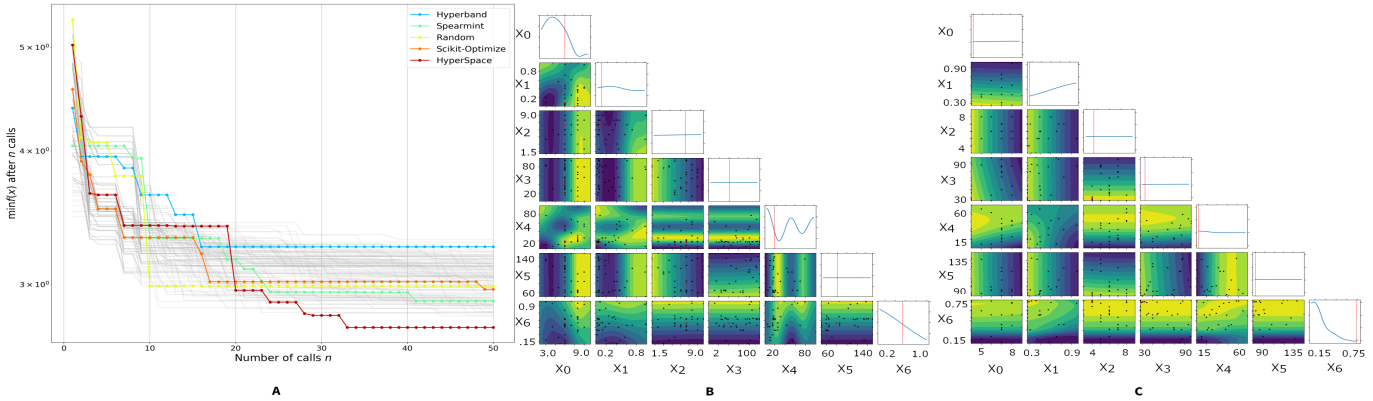

Fig. 4. Optimization of gradient boosting regressors. Plot A shows the convergence rates of all optimization algorithms. The many thin grey lines represent all other parallel runs of HyperSpace. In this seven dimensional search space, HyperSpace returns a model with the lowest mean absolute error over five-fold cross validation. Plot B gives the partial dependence plot of the model hyperparameters when optimizing over the large search space with `Scikit-Optimize`. Plot C shows the partial dependence of the hyperparameters over the best space found by HyperSpace. The hyperparameters corresponding to axis labels can be found in Table II.

overall minimum compared to all other methods, doing so in 33 iterations.

Note that HyperSpace finds the best model configuration in a region that does not initially seem promising. While pruning criteria similar to `Hyperband` can save computational expense, there is a risk that we may terminate early optimization runs that ultimately perform best.

The thin grey lines designate all other parallel optimizations run by HyperSpace, we find that there are many possible configurations of hyperparameters that lead to good solutions. Many of these optimization runs outperform `Hyperband`, `Scikit-Optimize`, and random search. Several solutions outperform Spearmint, the best performing baseline. While much of the research in hyperparameter optimization has focused on finding a single best configuration of hyperparameters, we find that there are many possible solutions for this optimization procedure.

TABLE II
SEARCH SPACE FOR THE GRADIENT BOOSTED REGRESSOR

hyperparameter	label	lower bound	upper bound
max depth	X_0	2	10
learning rate	X_1	1×10^{-5}	1
max features	X_2	1	10
min. samples per split	X_3	2	100
min. samples per leaf	X_4	1	100
number estimators	X_5	50	150
subsample	X_6	0.1	1.0

Figure 4.B and Figure 4.C show the partial dependence plots over the large search space using `Scikit-Optimize` and the best performing region of HyperSpace respectively. The diagonal of each plot gives importance of each hyperparameter after averaging out the effects of all other hyperparameters. All off-diagonal plots show the two-way interactions between each of the hyperparameters.

Consider Figure 4.B corresponding to `Scikit-Optimize` optimization over the larger search space. The first plot along the diagonal of the plot shows the dependence of our objective function to the setting of our gradient boosted regressor's max depth parameter. What we find, holding all other hyperparameters constant, is that our objective function is minimized when max depth is set to eight. However, considering the influence of all other variables, the `Scikit-Optimize` baseline optimization sets the best choice for the hyperparameter to be six. This is indicated by the dotted red line within each of the plots.

Figure 4.C shows the partial dependencies of our hyperparameters found by our best performing optimization in HyperSpace. We find that by optimizing over this particular region in the search space, varying five of the seven hyperparameters has little effect on the objective function, holding all other hyperparameters constant. For instance, when optimizing over this subspace, we now find that the setting for max depth to be less important, holding all other hyperparameters constant. Note that, though this is a restricted area of the domain, the setting for max depth could still fall within the interval $[4, 10]$. Recall that, in the larger search space, the objective function decreased as we increased the setting for max depth. This is no longer the case in this subspace when optimizing over the smaller subspace!

C. Convolutional Neural Network

Finally, we tested the various optimization algorithms on a two layer CNN trained on the MNIST dataset. Our objective function here is the loss on a withheld validation set that consisted of 15% of the full dataset. As with most neural network architectures, the CNN contains many hyperparameters that must be set before the training the model. These include the settings for kernel filter sizes, the stride length of the kernel filters, the number of hidden units per layer, etc. Typically, researchers and practitioners have some intuition as to which hyperparameters may be most important given their task and their dataset. Since we are considering a vision problem, we emphasize on optimizing kernel filter sizes and their strides. We have also included two separate measures of

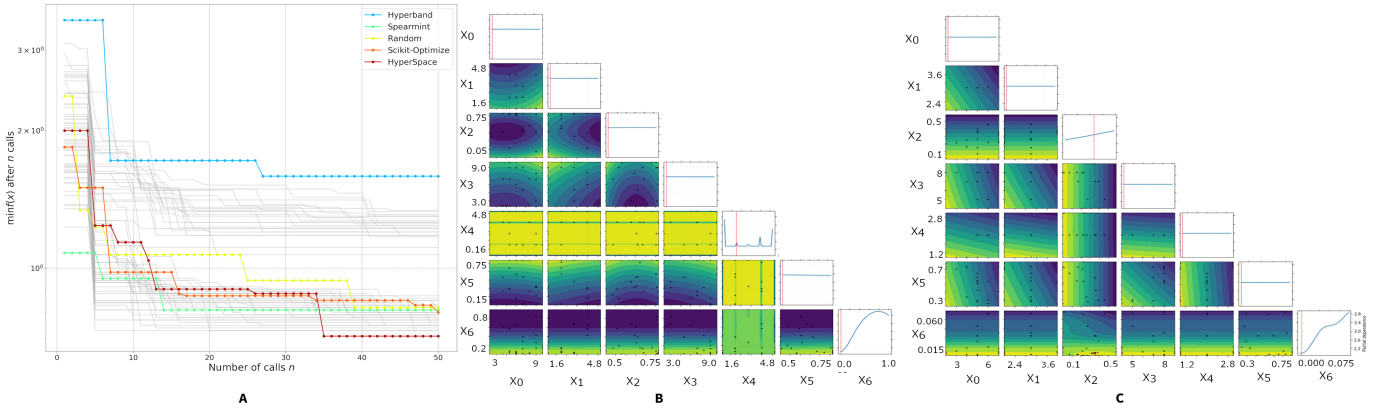

Fig. 5. Optimizing a convolutional neural network. Plot A gives the convergence rates for each of the algorithms. The many grey lines designate the parallel optimizations run by HyperSpace. Here HyperSpace finds the best performing model and also finds several other model configurations that perform reasonably well. Plot B gives shows the objective function's partial dependence to the model hyperparameters when optimizing over the large search space with `Scikit-Optimize`. Plot C shows the partial dependence with respect to hyperparameters for the best parallel SMBO model found by HyperSpace. The hyperparameters corresponding to axis labels can be found in Table III

dropout to act as regularization terms for the network. See Table III for the specification of the lower and upper bounds our model's seven hyperparameters. This defines the large search space which we would like to optimize over. Notice that the bounds were chosen to be wide, so that we do not impose too much of our own assumptions on the possible settings of the hyperparameters.

Comparing the convergence rates of the optimization runs in Figure 5.A, we find that the `Spearmint`, `Scikit-Optimize`, and random search all converge to nearly identical values in the later stages of the procedures. These baselines all outperform `Hyperband`. HyperSpace again finds a region in the domain of f where the minimum found is lower than all baselines. The many thin grey lines in the convergence plot show all other parallel runs of HyperSpace. Note that many of the parallel runs that converged to poor values HyperSpace, but there are also many runs that converged near to the best value found by HyperSpace.

TABLE III
SEARCH SPACE FOR CONVOLUTIONAL NET

hyperparameter	label	lower bound	upper bound
kernel 1 size	X_0	2	10
stride 1	X_1	1	5
dropout 1	X_2	0.0	0.8
kernel 2 size	X_3	2	10
stride 2	X_4	1	5
dropout 2	X_5	0.0	0.8
learning rate	X_6	1×10^{-3}	0.1

Figure 5.B shows the partial dependence plot when optimizing over the entire search space with `Scikit-Optimize`. Here there are large regions where two-way interactions between hyperparameters give poor results, as indicated by the dark blue regions in the off-diagonal of the plot. When compared with the partial dependence plot from the best space found by HyperSpace in Figure 5C, there is far less area in the hyperaparmeter search space where the two-way interactions result in poor performing models. By splitting the hyperparameter search space, HyperSpace is able to find better configurations of the CNN compared with Bayesian SMBO approaches over the entire search space.

V. CONCLUSION

We have developed a distributed Bayesian SMBO algorithm, *HyperSpace*, for optimizing ML model hyperparameters. We evaluated its performance on a variety of ML algorithms such as NMF, Gradient Boosted Regressors and a CNN on image data. In each of these cases, HyperSpace consistently outperformed commonly used hyperparameter optimization approaches. We showed that it is not only possible to find optimal configurations of our model hyperparameters, but that in many cases there can be several reasonably good settings for hyperparameters. We believe that future research into the statistical nature of interactions between hyperparameters will continue to improve the optimization procedures. While we are currently able to explore only two-way interactions, it would be interesting to study higher order interactions.

In the view that each optimization run is a chance to collect more information and update our beliefs about our models, HyperSpace would naturally fit in future work for meta-learning. HyperSpace could be treated as a data collecting algorithm for future meta-learning studies of ML architectures. We imagine that HyperSpace could be integrated into projects such as WorkWays [1], where researchers continually interact with their scientific tools, updating their knowledge about their experimental procedures. Similarly, HyperSpace could be incorporated into projects like JobPruner [19], where HyperSpace would optimize some process, and JobPruner would select the next episode of Bayesian optimization search spaces.

The implementation of HyperSpace lends itself naturally for HPC environments and the cloud. By splitting the range of hyperparameters into overlapping blocks, we are able to search many more configurations in the time it takes traditional SMBO algorithms to run. Our approach acknowledges

the concerns of the exponential scaling nature of Bayesian optimization [21, 6], and attempts to give ML/DL practitioners a reasonable method for exploring hyperparameter search spaces. This scaling remains an issue, however, when hyperparameter search spaces increase beyond the scale of current HPC resources. In the future, we hope to explore approaches that will more effectively handle the issue of scaling. Perhaps methods of search space pruning could prove beneficial, even if it entails increasing our uncertainty in the optimization process.

We used the `Scikit-Optimize` implementation of SMBO within Hyperspace; however, based on recent advances in learning generative models for the validation error as a function of training set size (e.g., FABOLAS [10]), the next iteration will include faster approaches. We will also explore combining the adaptive pruning approach of Hyperband with the distribributed SMBO approach of HyperSpace. This combination would allow us to continue to explore many more regions in the search space compared with traditional SMBO approaches but also allowing us to save computation time by terminating searches poor performing regions.

Acknowledgment

This work has been supported in part by the Joint Design of Advanced Computing Solutions for Cancer (JDACS4C) program established by the U.S. Department of Energy (DOE) and the National Cancer Institute (NCI) of the National Institutes of Health. This work was performed under the auspices of the U.S. Department of Energy by Argonne National Laboratory under Contract DE-AC02-06-CH11357, Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, Los Alamos National Laboratory under Contract DE-AC5206NA25396, and Oak Ridge National Laboratory under Contract DE-AC05-00OR22725. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory.

The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of the manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

References

[1] N. H. Anh, A. David, K. Timoleon, J. Andrew, and G. Graham. Workways: interacting with scientific workflows. *Concurrency and Computation: Practice and Experience*, 27(16):4377–4397. doi: 10.1002/cpe.3525.

[2] Z. Z. B. *Random Search Algorithms*. American Cancer Society, 2011. ISBN 9780470400531. doi: 10.1002/9780470400531.eorms0704.

[3] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[4] J. H. Friedman. Stochastic gradient boosting. *Comput. Stat. Data Anal.*, 38(4):367–378, Feb. 2002. ISSN 0167-9473.

[5] M. A. Gelbart, J. Snoek, and R. P. Adams. Bayesian optimization with unknown constraints. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, pages 250–259, Arlington, Virginia, United States, 2914. AUAI Press. ISBN 978-0-9749039-1-0.

[6] S. Grünewälder, J.-Y. Audibert, M. Opper, and J. Shawe-Taylor. Regret Bounds for Gaussian Process Bandit Problems. In *AISTATS 2010 - Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 273–280, Chia Laguna Resort, Sardinia, Italy, May 2010.

[7] D. Harrison and D. Rubinfeld. Hedonic Housing Prices and the Demand for Clean Air. *Journal of Environmental Economics and Management*, 5:81–102, 1978.

[8] T. Head, MechCoder, G. Louppe, I. Shcherbatyi, fcharras, Z. Vincius, cmmalone, C. Schrder, nel215, N. Campos, T. Young, S. Cereda, T. Fan, rene rex, K. K. Shi, J. Schwabedal, carlosdanielcsantos, Hvass-Labs, M. Pak, SoManyUsernamesTaken, F. Callaway, L. Estve, L. Besson, M. Cherti, K. Pfannschmidt, F. Linzberger, C. Cauet, A. Gut, A. Mueller, and A. Fabisch. scikit-optimize/scikit-optimize: v0.5.2, Mar. 2018.

[9] R. P. A. Jasper Snoek, Hugo Larochelle. Spearmint. https://github.com/JasperSnoek/spearmint, 2012.

[10] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*, 2016.

[11] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *J. Mach. Learn. Res.*, 9:235–284, June 2008. ISSN 1532-4435.

[12] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

[13] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *International Conference on Learning Representations (ICLR2017)*. CBLS, April 2017.

[14] P. Luszczek, M. Gates, J. Kurzak, A. Danalis, and J. Dongarra. Search space generation and pruning system for autotuners. In *30th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Chicago,

IL, 05-2016 2016. IEEE, IEEE.

[15] J. Mockus. On bayesian methods for seeking the extremum. In *Proceedings of the IFIP Technical Conference*, pages 400–404, 1974. ISBN 3-540-07165-2.

[16] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[18] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Jan. 2006.

[19] B. Silva, M. A. Netto, and R. L. Cunha. JobPruner: A machine learning assistant for exploring parameter spaces in HPC applications. *Future Generation Computer Systems*, 83:144 – 157, 2018. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2018.02.002.

[20] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[21] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger. Gaussian process bandits without regret: An experimental design approach. *CoRR*, abs/0912.3995, 2009.