SHADHO: Massively Scalable Hardware-Aware Distributed Hyperparameter Optimization

Jeffery Kinnison Nathaniel Kremer-Herman Douglas Thain Walter Scheirer
University of Notre Dame
{jkinniso,nkremerh,dthain,walter.scheirer}@nd.edu

Abstract

Computer vision is experiencing an AI renaissance, in which machine learning models are expediting important breakthroughs in academic research and commercial applications. Effectively training these models, however, is not trivial due in part to hyperparameters: user-configured values that control a model's ability to learn from data. Existing hyperparameter optimization methods are highly parallel but make no effort to balance the search across heterogeneous hardware or to prioritize searching high-impact spaces. In this paper, we introduce a framework for massively Scalable Hardware-Aware Distributed Hyperparameter Optimization (SHADHO). Our framework calculates the relative complexity of each search space and monitors performance on the learning task over all trials. These metrics are then used as heuristics to assign hyperparameters to distributed workers based on their hardware. We first demonstrate that our framework achieves double the throughput of a standard distributed hyperparameter optimization framework by optimizing SVM for MNIST using 150 distributed workers. We then conduct model search with SHADHO over the course of one week using 74 GPUs across two compute clusters to optimize U-Net for a cell segmentation task, discovering 515 models that achieve a lower validation loss than standard U-Net.

1. Introduction

Without question, advances in high-performance computing architectures have fueled the innovation and success of computationally expensive machine learning methods. New many-core architectures allow researchers to make full use of the large labeled datasets that are necessary to train effective models (i.e., solutions) for various learning problems. Modern machine learning tools backed by GPUs, clusters of traditional CPUs, and custom parallel hardware have enabled data-driven computer vision approaches in numerous domains, including healthcare [16], autonomous vehicles [9], human biometrics [34], and many more [17].

Despite these successes, selecting the correct model for particular data remains a difficult problem. Model performance is highly algorithm-specific, with different models producing wildly different results on the same dataset. However, model selection is not simply an algorithmic choice. Model searches must also account for hyperparameters: free parameters associated with a particular machine learning model that govern its ability to learn. These parameters are separate from the elementary parameters (i.e., weights) that are learned from the data, and are set before training takes place. Hyperparameters are often defined over nonlinear, non-convex spaces with many local minima, making optimization non-trivial. Choosing the best model for a particular learning task boils down to choosing the parametrized model that can accurately make predictions from new data. This is known as the *hyperparameter optimization problem*.

Steps have been made toward local [29, 35] and distributed [5, 11, 13, 38, 39, 40] hyperparameter search strategies. An example of the distributed hyperparameter optimization process is shown in Figure 1. Curiously, though, current distributed strategies do not take advantage of potential systems-based optimizations. Existing solutions do not account for the hardware being used, instead enforcing (in the case of cloud-based platforms [13, 39]) or assuming that all connected hardware is identical. A key problem that we and many other machine learning researchers have encountered is that available computing resources for distributed hyperparameter optimization are typically heterogeneous and often spread across different networks. This is the state of affairs for nearly all users outside of a handful of cloud-based providers, which have internal on-demand access to large homogeneous collections of fast hardware. In the former case, efficient hyperparameter searches must adjust the search strategy to make the best use of available hardware. Even in the latter case, though, for collections of homogeneous systems, decisions may be made about the level of parallelism to exploit during the search.

While adapting distributed hardware to a search is not possible, information about the search can be used to adapt

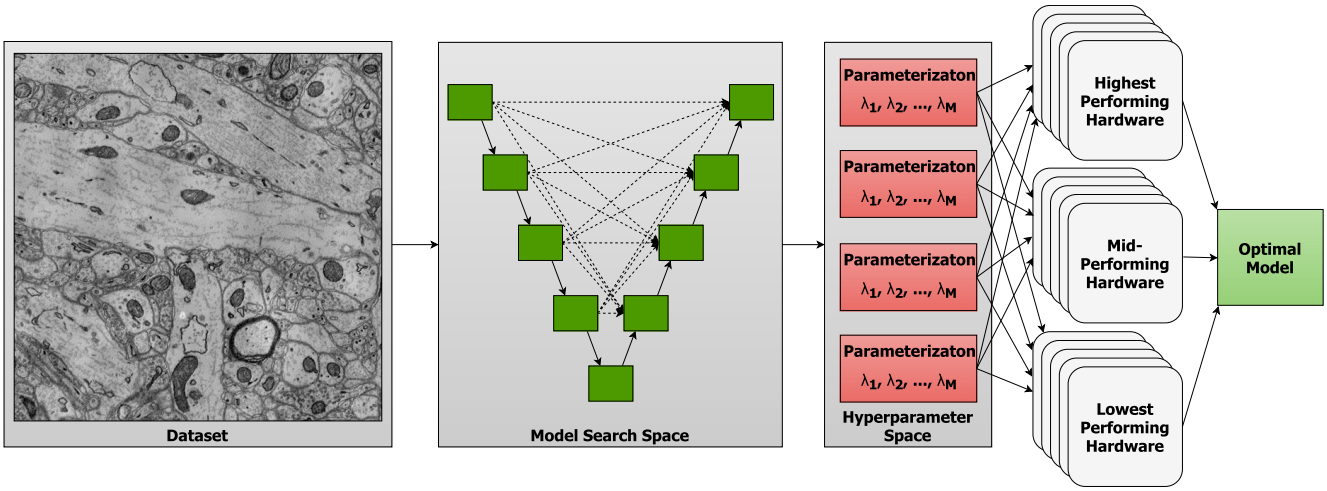

Figure 1. Example distributed hyperparameter optimization of U-Net [36] with different bypass connections for cell segmentation in images from electron microscopy [28]. During optimization, a set of bypass connections is selected, then the layers of the network are parametrized. The parametrized model is then sent to a remote worker for training and evaluation. This process is repeated many times, and the optimal parametrized model is returned. Existing distributed hyperparameter optimization frameworks assign models to the first available worker. SHADHO, by contrast, uses model structure and search performance to influence throughput by directing models to appropriate hardware.

it to the hardware. Hyperparameters are ubiquitous across different classes of machine learning algorithms, and each algorithm can have a different set of hyperparameters distributed over different numeric and categorical spaces. For example, a Support Vector Machine (SVM) has one to four numeric parameters depending on the kernel function used (i.e., linear, radial basis function, sigmoid, and polynomial). Turning to deep learning, convolutional neural networks (CNNs) require tuning a number of hyperparameters within each convolutional layer, meaning that the search must expand with the size of the network. Information about the structure of the search, such as the hyperparameter count of each model, can inform how optimization proceeds.

A number of hyperparameter search strategies have been proposed to solve this optimization problem, ranging from naive approaches like random search [3], to more rigorous approaches like Bayesian optimization [6, 38], gradient-based learning [33], and bandit-based searches [31]. These strategies address the problem of choosing the next parametrization to test, but they operate under two major simplifying assumptions: 1) that hyperparameters have equal priority in the search, and 2) that hyperparameter search spaces are equally complex. In practice, neither of these assumptions are generally true.

One avenue for improving distributed hyperparameter optimization, then, is to account for the necessity and complexity of searching hyperparameter spaces to improve scheduling the search. Hyperparameter optimization is best represented as a bag of tasks (BoT) application, in which there are an effectively infinite number of tasks that must be mapped to a finite set of resources and no task is explicitly dependent upon another. Existing hyperparameter optimization software uses a naive first-come, first-serve (FCFS) approach to schedule hyperparameter evaluations. By incorporating information about the models and hyperparameters being searched, however, it is possible to schedule such that a greater proportion of searches are allocated to larger models with less certain performance.

In this paper, we present the Scalable Hardware-Aware Distributed Hyperparameter Optimization framework (SHADHO), a general-purpose hyperparameter optimization framework. For each model in the search, SHADHO approximates the complexity (aggregate size of the hyperparameter domains) and priority (variation in performance across different parametrizations). Models are ranked by these two heuristics, and high-complexity / high-priority models are assigned to more performant hardware. In this way, search throughput is increased across models with many hyperparameters and wider ranges of performance on the learning task, making SHADHO suitable for applications such as neural network architecture search. In summary, the contributions of this paper are as follows:

1. Two heuristics — complexity and priority — for ranking models in a model search / hyperparameter optimization process.
2. A description of the SHADHO framework.
3. A demonstration of what the increased throughput heuristic-based scheduling with SHADHO can offer over FCFS schemes.
4. An application of SHADHO to CNN optimization for membrane detection in microscopic images.

2. Related Work

Hyperparameter Optimization. Hyperparameter optimization methods typically address the problem of how to choose the next hyperparameter value to search. Manual tuning and grid search [15] are popular methods because they are easy to implement, however they rely upon domain knowledge and will skip over many values in continuous domains. Bergstra and Bengio [3] argued to replace these practices with random search because it is just as easy to implement and does not require discretizing the search space. This scheme will more thoroughly search the hyperparameter spaces, however it has no mechanism for narrowing the scope of the search. Random search is the basis for Hyperopt [4, 5], a widely-used open source hyperparameter optimization framework.

Guided approaches to hyperparameter optimization are also popular, notably genetic algorithms and Bayesian optimization strategies. Genetic algorithms [2, 19, 43] have historically been applied to hyperparameter optimization, however they are prohibitively expensive as the number of hyperparameters increases [6]. Sequential Model-based Bayesian Optimization [26], Tree-Structured Parzen Estimators [6], Gaussian process-based estimation [38], and Sequential Model-based Algorithm Configuration [24] are popular Bayesian optimization strategies, implemented in a number of open source and proprietary frameworks [5, 11, 13, 29, 38, 39, 40]. These methods use previous hyperparameter values and their corresponding evaluations as priors for approximating viable hyperparameter values, and as such can become stuck in local minima.

Methods beyond genetic algorithms and Bayesian optimization have also been explored to exploit different selection criteria. MacLaurin *et al.* [33] introduced a method that learns gradients with respect to hyperparameter values, allowing for fine-grained hyperparameter optimization but requiring an expensive gradient calculation step. Domhan *et al.* [14] introduced a method for extrapolating learning curves using Markov-Chain Monte Carlo inference to predict parametrized model performance and update the hyperparameter selection method using the predicted performance. The Hyperband method introduced by Li *et al.* [31] performs grid search using a budgeted successive halving method that assigns a cost to each search. Ilievski *et al.* [25] propose using radial basis function surrogates and dynamic coordinate search to select candidate hyperparameter values. Like Bayesian optimization, these each narrow the viable domain of each hyperparameter under optimization by examining previously tested values and applying an operation to prune the domain. As with Bayesian optimization, these methods run the risk of falling into local minima.

Tuning a neural network architecture for a particular learning problem presents a different set of challenges to standard hyperparameter tuning, including selecting neural network layers and connections between layers. The optimal neural network architecture is typically determined by iteratively building up the network and observing performance on the dataset. Historically, this iterative procedure has been carried out by trial-and-error [18, 20], in which one parameter from one layer is manually varied at a time. Several automated methods, including construction / pruning methods [23], particle swarm optimization [21], and genetic algorithms [2] select new layers based on observed performance. Zoph *et al.* [47, 48] developed a method that chooses optimal neural network models using reinforcement learning, however they report using several hundred GPUs to conduct this search in both published case studies. Neural networks are also being applied on a smaller scale for architecture selection [10, 12], utilizing a network trained to generate candidate architectures for evaluation.

All of these hyperparameter optimization and architecture search methods were created to decide **which** hyperparameter values to search while assuming all models and hyperparameters are equal. The question of **how** to conduct a hyperparameter search given a set of possibly heterogeneous distributed resources, particularly in terms of directing models to appropriate hardware and focusing on models with uncertain performance, is new to this problem.

Dynamic Distributed Task Scheduling. A number of heuristic-based distributed scheduling algorithms exist for general-purpose task scheduling, with a focus on optimizing for task dependency graphs. The two algorithms (HEFT and CPOP) presented in [41] focus on providing solutions to scheduling tasks in a directed acyclic graph (DAG) structure to prioritize critical tasks which many other tasks rely upon. HEFT uses a task complexity heuristic to schedule dependent tasks similar to the heuristic we propose. In [8], the authors built upon the HEFT algorithm presented in [41] to more explicitly consider the effect scheduling certain tasks before others will have on the performance of the DAG workflow. Biswas *et al.* [7] extended the HEFT algorithm with additional heuristics and a multi-queue scheduler to address the problem of scheduling across heterogeneous systems. Al Ebrahim and Ahmad [1] also extended HEFT to explicitly consider all dependencies and data transfer among a static set of tasks.

Existing heuristic scheduling solutions are not well-suited for solving the problem of scheduling a model search or hyperparameter optimization process because they assume that the task space is modeled as a dependency graph. Hyperparameter optimization is best modeled as a BoT application: mapping an effectively infinite set of independent tasks to a finite set of resources. Heuristic scheduling for BoT typically involves monitoring resource utilization [46, 45] to minimize the cost of running the tasks, or else scaling hardware to the set of tasks in the case of elastic cloud services [37]. This work, by contrast, schedules

based on properties of the tasks (i.e., model training sessions) themselves to match the available hardware, focusing on larger models with less certainty in their performance.

3. Heuristics for Hyperparameter Search

Fundamentally, hyperparameter optimization is the problem of selecting a model, $\mathcal{M}$, and parametrizing it with a set of hyperparameters, λ, such that $\mathcal{M}(\lambda)$ learns a desired set of patterns from a dataset with minimal error. Each hyperparameter $\lambda_i \in \lambda$ is defined over a distinct discrete or continuous domain, $s_i \in \mathbf{s}$, and hyperparameters are searched by drawing a value from each domain and evaluating the performance of $\mathcal{M}(\lambda)$.

Optimizing hyperparameter search over a particular set of hardware is the problem of mapping models to hardware with resources proportionate to the need to search a given model. To measure "need," two pieces of information must be known: 1) the number of searches necessary to completely search $\mathcal{M}(\lambda)$, and 2) the fitness of $\mathcal{M}$ to the learning problem. Both of these values must be approximated in general because hyperparameters are often defined over continuous domains, and model performance may only be determined experimentally. We define these approximations as the search *complexity* and *priority* of each model.

3.1. Complexity

The complexity of a model is determined by the aggregate size of its viable hyperparameter domains. Hyperparameter domains are defined over both continuous and discrete spaces, so any heuristic must be an necessarily approximation of the size of the search. Moreover, a complexity heuristic should maintain the order of spaces based on their size. Thus, for any hyperparameter domain $s_i \in \mathbf{s}$, we define the search complexity $C(s)$ as

$$C(s_i) = \begin{cases} 2 + \|b - a\| & \textit{if } s \textit{ is continuous} \\ 2 - \frac{1}{|s_i|} & \textit{if } s \textit{ is discrete} \end{cases} \quad (1)$$

where $[a, b]$ is the closed interval containing 99% of the probability distribution governing s. $C(s)$ enforces a strong ordering on spaces based on their size: continuous spaces are considered more complex to search than discrete spaces. Moreover, $C(s)$ maintains the order of continuous spaces relative to continuous spaces and discrete spaces relative to discrete spaces. Model complexity is then approximated as $C(\mathbf{s}) = \sum_{s_i \in \mathbf{s}} C(s_i)$.

Complexity offers a static measure of the need to search a particular space, gearing a search toward models with a large number of hyperparameters defined over a wide variety of spaces. Note that complexity is not necessarily an approximation of running time, rather it is a description of the size of the search spaces. In many cases, particularly when searching neural network architectures, there is a correspondence between hyperparameter count and running time, however this relationship does not generally hold.

3.2. Priority

To complement the static complexity heuristic, dynamic assessment of the need to search a model is also necessary. The priority heuristic accounts for model performance across different parametrizations, estimating the fitness of a model to learn from the data as a function of variation in performance.

Priority is calculated using the method described by Bergstra and Bengio [3] for approximating hyperparameter importance to a model's performance on a dataset. For a given model, a Gaussian process with RBF kernel is fit to a dataset consisting of the tested hyperparameter values and their resulting loss values. The learned length scale l of the RBF kernel that maximizes the log marginal likelihood is extracted after fitting as an approximation of the sensitivity of the RBF kernel to changes in the hyperparameter values. This process is repeated 50 times to obtain a sample of length scales, L, and the priority is approximated as

$$P(L) = \min(L)^{-1} - \max(L)^{-1} \quad (2)$$

In essence, $P(L)$ approximates the covariance between parametrizations and their resulting performance to give an estimate of the intrinsic fitness of a model for the learning task. Models with high parameterization covariance (low $P(L)$) are less pressing to search because their fitness is known; those with low covariance (high $P(L)$) have a wide range of observed performance and thus should be searched more thoroughly to determine their fitness. Note that a model with "consistent performance" will receive a low priority regardless of whether it performs well or poorly, as priority is a measure of variation.

Like complexity, models with a larger priority are preferred in the search because there is less certainty about their fitness to the learning task. This lack of certainty indicates that the hyperparameter spaces should be explored more thoroughly to better understand how the model performs under different parameterizations. Models with low priority should continue to be searched, given that the performance of the model over the entire hyperparameter domain cannot be known, however the number of searches allocated should be scaled back.

4. SHADHO Framework

To create a heuristic scheduler for hyperparameter optimization, we implemented complexity and priority in the SHADHO framework. SHADHO is an open-source Python package built on top of the scientific Python stack [27, 35] and the Work Queue framework [44] for hyperparameter generation and distributed task management, respectively.

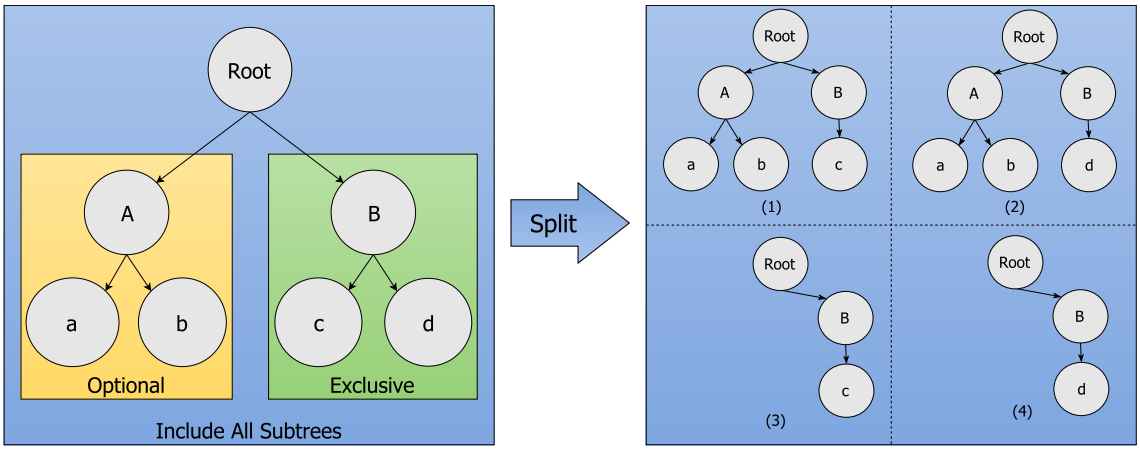

Figure 2. Splitting a specification tree into a set of disjoint trees. (Left) The specification tree contains two subtrees: an optional subtree A and an exclusive subtree B. The root level indicates that a subset of both subtrees should be included if possible. (Right) The trees created by splitting the specification tree. Because A is flagged as optional, trees (1) and (2) include both children of A, while A is excluded entirely from trees (3) and (4). Additionally, each of the four trees contains only one child of B. In terms of hyperparameter search, A corresponds to an operation that is global across models, such as a preprocessing step, and B corresponds to disjoint models.

4.1. Defining Search Spaces

Prior to a search, models are defined as a tree with hyperparameter domains as the leaves. In this tree, individual models are demarcated by tagging subtrees as "exclusive" (only one path below may be followed at a time) or "optional" (the subtree is either included or excluded entirely). SHADHO splits this tree into a forest at runtime based on the exclusive and optional tags, with each tree in the forest corresponding to a single model, as shown in Figure 2.

Search space definition semantics were created based on the best practices outlined by Bergstra *et al.* [4]. As a result, the API will be familiar to users of Hyperopt, a standard open-source distributed hyperparameter optimization framework. Examples of defining search spaces, including the usage of "exclusive" and "optional" flags, can be found in the SHADHO documentation (see supp. material).

4.2. Hardware Awareness with Compute Classes

To simplify hardware-aware scheduling, SHADHO groups connected workers based on common hardware resources. These *compute classes* can be grouped by GPU model, number of cores, memory size, and other arbitrary user-defined features. These are then ranked in order of performance to create a hierarchy by which models may be assigned to hardware.

4.3. Using Heuristics in SHADHO

SHADHO uses an implementation of the complexity and priority heuristics defined in Equations 1 and 2 to rank each tree (model) in the forest. The models are then ranked by their complexity and priority, with equal precedence given to both heuristics. The ranking is used to weight models such that higher-ranked models are scheduled more often and assigned to higher-performing hardware. Considering both heuristics as equal balances the search between emphasizing models with more potential parametrizations and those with greater performance variation across parametrizations.

While SHADHO uses both heuristics by default, one or both may be deactivated to accommodate the model search and available resources. For example, if a large model cannot feasibly run on one particular compute class, the search can use complexity only to ensure that the model is only trained and evaluated on nodes of that compute class. Similarly, exploratory studies with a large number of models may benefit from using priority only to direct the search based on model performance alone. For evaluation purposes, SHADHO may also operate in heuristic-free mode (SHADHO-FCFS), in which models are scheduled FCFS. SHADHO-FCFS is equivalent to other distributed hyperparameter optimization frameworks in that it makes no decisions about task scheduling.

4.4. Distributed Task Management

Distributed task management and execution in SHADHO are handled by the Work Queue execution engine [44]. Work Queue uses a master-worker scheme consisting of a centralized master and distributed worker processes. The master coordinates the workers that may be running on a variety of machines in clusters, clouds, or grids. Work Queue workers are persistent processes submitted to a batch job system that communicate with the master to request work. If tasks are available, the master dispatches them as shown in Figure 3. The worker operates entirely within a sandboxed environment on its host system that includes cached versions of files used across tasks.

Workers and tasks support specifying hardware resource

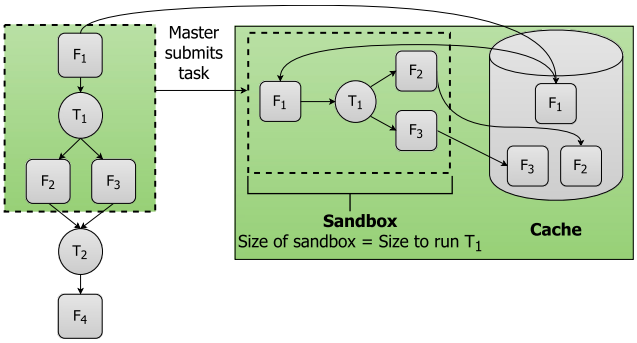

Figure 3. Work Queue Master-Worker architecture. Tasks are dispatched by the master process with the necessary input files (F_1), command to run (T_1), and the expected output files (F_2 and F_3) to an available worker process. Each worker may handle multiple tasks that all share a common data cache in each worker.

requirements. For the worker process, the resource requirements may consist of any or all of: cores, memory, disk, and GPUs. This ensures that the batch system managing the workers will only land each worker process on a machine with at least the minimum requested resources. A worker can also advertise an arbitrary feature to the master (i.e., the CPU model). For tasks, the resource specifications ensure the master only dispatches tasks to workers that can handle them. Task resource specification includes the same resources as the worker (cores, memory, disk, and GPUs), but it may also request any arbitrary feature. SHADHO uses Work Queue's hardware requirements specification to group workers into compute classes.

5. Experiments

To demonstrate the effectiveness of heuristic-based hardware assignment for hyperparameter search, we apply SHADHO to two computer vision problems: Support Vector Machine (SVM) optimization for handwritten character identification, and fully-convolutional neural network (FCNN) optimization for cell segmentation in electron microscopy images. SVM optimization allows us to quantify improvements in performance compared to software with FCFS scheduling. FCNN optimization demonstrates SHADHO's applicability to difficult learning problems.

5.1. SVM Optimization

Our first experiment, optimizing SVM for handwritten digit recognition using the MNIST dataset [30], is a benchmark for determining the throughput of hyperparameter optimization software. While MNIST classification is a solved problem, it is a useful benchmark for demonstrating increases to throughput enabled by SHADHO because each individual sets of hyperparameters may be tested quickly, but performance and running times vary across SVM kernels. Figure 4 presents the four SVM kernels and the spaces searched for each of their hyperparameters.

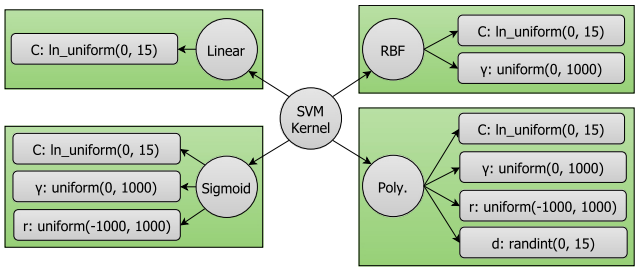

Figure 4. Specification for searching four SVM kernel functions, with four distinct models. Each kernel contains a different number of hyperparameters, giving the trees a strong complexity-based ordering. C: soft-margin constant, defined uniformly over $[0, 15]$, and scaled logarithmically. γ: kernel coefficient, defined uniformly over $(0, 10^3]$. r: an additional coefficient, defined over $[-10^3, 10^3]$. d: polynomial degree, a random integer from 1 – 15.

In this experiment, we used the `scikit-learn` [35] SVM implementation trained using one-vs-rest classification with one estimator per class (10 total) parallelized over the number of available cores on a worker. Tests were distributed over 150 workers: 50 4-core machines, 50 8-core machines, and 50 16-core machines. In this case, a 4-core worker was able to train 4 estimators in parallel, while a 16-core worker trained all 10 simultaneously.

We performed random search using SHADHO, SHADHO-FCFS, and Hyperopt [5], a standard hyperparameter optimization package, and report the performance in terms of the number of hyperparameters tested per hour. To standardize each trial, SHADHO performed a search for one hour, assigning searches to workers based on their complexity and priority. The same set of parameters were then tested using SHADHO-FCFS and Hyperopt, both of which perform FCFS scheduling.

Table 1. SVM Optimization Metrics

Method	Avg. Throughput (tasks/hr)
SHADHO	463.03
SHADHO-FCFS	227.28
Hyperopt [5]	252.42

The average throughput per trial over 48 trials is shown in Table 1. On average, SHADHO achieved a $1.8\times$ increase in throughput over Hyperopt and a $2\times$ increase over SHADHO-FCFS, with all three using approximately the same workers in each trial. As shown in Figure 5, the distribution of SHADHO's throughput across the 48 trials was much smaller than that of Hyperopt or SHADHO-FCFS, indicating that heuristic-based hardware-aware task scheduling leads to more consistent performance than FCFS scheduling. We attempted to mitigate the effects of distributing trials across a network with dynamic usage patterns by running the three methods back-to-back in each trial. In general, this led to similar worker connectivity patterns, however sharp drops in connectivity can be seen

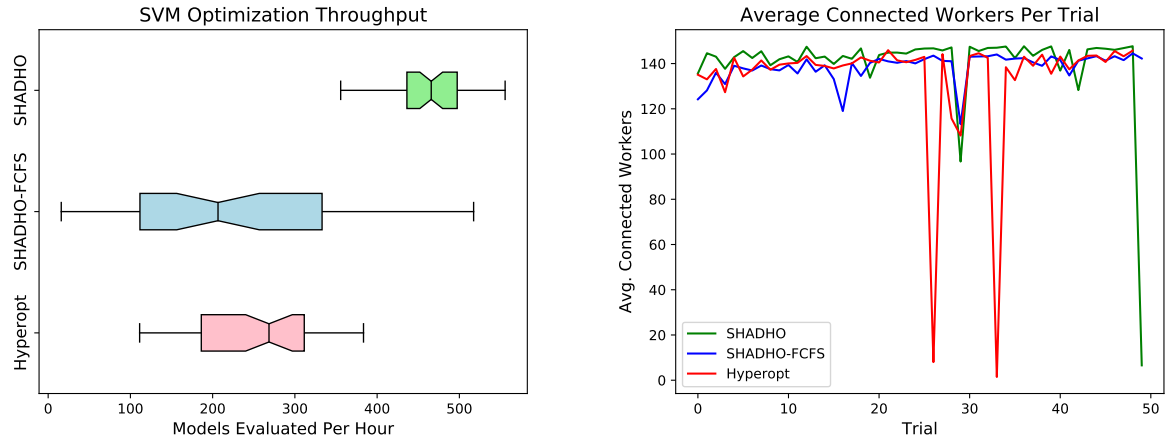

Figure 5. The throughput in tasks completed per hour over all SVM optimization trials (left) and average connected workers over the first hour of each trial (right) for SHADHO (green), SHADHO-FCFS (blue), and Hyperopt [5] (red). SHADHO had the highest throughput on average, $1.8\times$ that of Hyperopt and $2.0\times$ that of SHADHO-FCFS. SHADHO also had the smallest throughput distribution, indicating that heuristic-based distribution leads to more stable performance. In most cases, the difference in average number of connected workers between SHADHO, SHADHO-FCFS and Hyperopt in a given trial is extremely small, indicating that the three methods were operating with approximately the same set of distributed workers in every case. Thus, the difference in throughput is attributable to heuristic-based task scheduling except in extreme cases of network load (e.g., trials 26, 29, 33, and 48).

in Figure 5. In each of these cases, the limiting factor on throughput was network load — we intentionally ran these experiments in a production environment with other users.

It should be noted that both SHADHO-FCFS and Hyperopt were able to match the throughput of SHADHO during a small number of trials. This is not unexpected: a FCFS scheduling scheme, which is effectively random task scheduling, will on occasion achieve much higher throughput than average due to task order. This is not reliable, though, as is evident from the size of each distribution. Both of the FCFS-based methods have wide throughput distributions, indicating extremely unreliable performance. SHADHO, on the other hand, boasts a smaller throughput distribution and thus more consistent performance.

5.2. U-Net Optimization

In the neuroscientific domain, neuronal cell segmentation by membrane detection in electron microscopic (EM) images is a difficult problem. In general, EM data is noisy with staining and imaging artifacts obscuring features, and few annotated datasets exist to train a pixel-wise classifier [22, 32]. We selected the EM dataset introduced by Kasthuri *et al.* [28] because it is one of the few with comprehensive ground-truth annotations created by a domain expert that is not of trivial size. We selected a $253 \times 2048 \times 2048$ voxel subset of the data for training and evaluation. The goal of this experiment was to train a pixel-wise classifier to correctly segment cells in the volume.

We used SHADHO to optimize U-Net [36], a state-of-the-art fully-convolutional neural network for microscopic image segmentation. In addition to a forward downsampling / upsampling path, U-Net includes bypass connections that feed the output of early convolutional layers on the downsampling path to the corresponding convolutional unit on the upsampling path. In this experiment, we conducted model search and hyperparameter optimization over a number of different U-Net models with varying bypass connections[1]. Table 2 lists the hyperparameter values searched, with all hyperparameters except learning rate randomly sampled for each convolutional layer. In all cases, models were trained to optimize cross entropy on augmented tiles from the first 80% of the dataset and evaluated on the last 20% of the dataset. The ground truth was inverted to emphasize cell membranes over cell interiors. Models were trained for 150 epochs with a batch size of 100 128×128-pixel images and early stopping for models with plateauing performance.

Table 2. U-Net Hyperparameter Search Spaces

Parameter	**Values**
Min. Kernels	16, 32, 64, 128
Kernel Size	1, 3, 5, 7, 9
Activations	sigmoid, tanh, relu, elu, PReLU, LeakyReLU, ThresholdedReLU
Initializers	zeros, ones, glorot_normal, he_normal
Regularizers	l1, l2, l1_l2
Dropout Rate	uniform distribution over $[0, 1]$
Learning Rate	uniform distribution over $[10^{-4}, 1]$

Over the course of one week, we evaluated 1075 U-Net

[1]Details about the bypass connections may be found in supp. mat.

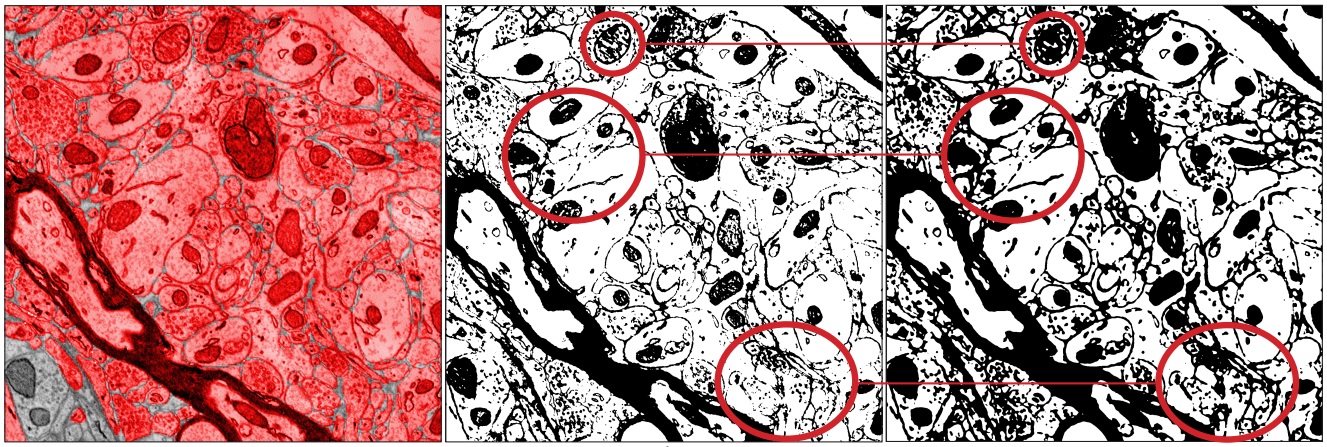

Figure 6. An image from the validation set created from the EM cell segmentation dataset of Kasthuri *et al.* [28] with ground-truth overlaid in red (a), and predictions obtained from the standard U-Net model (b) and an optimized model found by SHADHO (c). The SHADHO model includes less noise in its predictions and more clearly separates the individual cells. Both segmentations classify dark features in the cell interiors as background, and this is a difficult challenge to overcome due to their similarity to the background of the ground-truth. Circled: Examples of merge errors and misclassifications improved upon by our discovered model. Best viewed in color.

models with different parametrizations using 16 NVIDIA Titan X Pascal, 50 NVIDIA Tesla K80, and 8 NVIDIA GTX 1080ti GPUs. We compared these against a standard U-Net model as described by Ronneberger *et al.* [36] trained using a cross entropy loss function and the Adam optimizer with a learning rate of 10^{-4}. The published U-Net achieved a validation loss of 1.07 after training, and we discovered 515 models with a lower loss. An example of the difference in classification ability between one of our models and the published U-Net model is shown in Figure 6. As can be seen, the U-Net model discovered by SHADHO increases the separability between cells in the image and incurred fewer merge errors and misclassifications than the standard U-Net model. Based on these results, we conclude that SHADHO was able to find higher-quality models than the hand-tuned published U-Net model.

6. Conclusions

Hyperparameter optimization is a crucial step in the machine learning process, and evaluating as many parametrizations as possible increases the chance of finding a high-quality model. SHADHO increases the throughput of hyperparameter optimization by determining the complexity and priority of searching each model and adjusting the proportion of searches allocated to each by assigning models to appropriate hardware. In the case of SVM kernel optimization, SHADHO increased the throughput of model evaluations by a factor of 2.0 over FCFS distributed hyperparameter optimization. Moreover, when applied to U-Net for microscopic image segmentation, SHADHO discovered a large number of models with lower validation error than the standard U-Net, indicating that SHADHO can improve upon hand-tuned models.

The current implementation of SHADHO only makes use of a random hyperparameter search strategy. Recent studies have introduced promising search strategies involving Bayesian optimization and bandit-based search, and future versions of SHADHO will incorporate these and forthcoming developments. Additionally, the problem of automated neural network architecture construction has been broached in recent deep learning work [42, 47]. One of the prime benefits of SHADHO is the ability to dynamically reallocate work to appropriate hardware. Neural network architecture search involves testing networks of varying complexities and running times, making SHADHO a suitable framework for exploring and scaling their architectures.

The ability to effectively allocate hyperparameters to hardware is central to SHADHO's operation. We will continue to explore other methods for approximating complexity and priority, including methods for understanding running time across heterogeneous hardware (complexity) and comparative performance metrics between models and search spaces (priority). SHADHO will allow us to explore these methods at a massive scale and advance both machine learning and distributed computing.

7. Acknowledgements

This research was supported in part by the Notre Dame Center for Research Computing through access to distributed computing resources. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. Funding was provided under IARPA contract #D16PC00002.

References

[1] S. Al Ebrahim and I. Ahmad. Task scheduling for heterogeneous computing systems. *The Journal of Supercomputing*, 73(6):2313–2338, 2017. 3

[2] P. Benardos and G.-C. Vosniakos. Optimizing feedforward artificial neural network architecture. *Engineering Applications of Artificial Intelligence*, 20(3):365–382, 2007. 3

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012. 2, 3, 4

[4] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015. 3, 5

[5] J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pages 13–20. Citeseer, 2013. 1, 3, 6, 7

[6] 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. 2, 3

[7] T. Biswas, P. Kuila, and A. K. Ray. Multi-level queue for task scheduling in heterogeneous distributed computing system. In *Advanced Computing and Communication Systems (ICACCS), 2017 4th International Conference on*, pages 1–6. IEEE, 2017. 3

[8] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 27–34, Feb 2010. 3

[9] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. 1

[10] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017. 3

[11] M. Claesen, J. Simm, D. Popovic, and B. Moor. Hyperparameter tuning in Python using optunity. In *Proceedings of the International Workshop on Technical Computing for Machine Learning and Mathematical Engineering*, 2014. 1, 3

[12] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang. Adanet: Adaptive structural learning of artificial neural networks. *arXiv preprint arXiv:1607.01097*, 2016. 3

[13] I. Dewancker, M. McCourt, S. Clark, P. Hayes, A. Johnson, and G. Ke. Evaluation system for a bayesian optimization service. *arXiv preprint arXiv:1605.06170*, 2016. 1, 3

[14] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, pages 3460–3468, 2015. 3

[15] K.-B. Duan and S. S. Keerthi. Which is the best multiclass svm method? an empirical study. In *International Workshop on Multiple Classifier Systems*, pages 278–285. Springer, 2005. 3

[16] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017. 1

[17] R. Fergus. Deep learning for computer vision, 2013. Tutorial Presented at NIPS 2013. 1

[18] F. Fernandes and L. Lona. Neural network applications in polymerization processes. *Brazilian Journal of Chemical Engineering*, 22(3):401–418, 2005. 3

[19] F. Friedrichs and C. Igel. Evolutionary tuning of multiple svm parameters. *Neurocomputing*, 64:107–117, 2005. 3

[20] R. Furtuna, S. Curteanu, and M. Cazacu. Optimization methodology applied to feed-forward artificial neural network parameters. *International Journal of Quantum Chemistry*, 111(3):539–553, 2011. 3

[21] B. A. Garro and R. A. Vázquez. Designing artificial neural networks using particle swarm optimization algorithms. *Computational intelligence and neuroscience*, 2015:61, 2015. 3

[22] M. Helmstaedter. Cellular-resolution connectomics: challenges of dense neural circuit reconstruction. *Nature methods*, 10(6):501–507, 2013. 7

[23] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 3

[24] F. Hutter, H. Hoos, and K. Leyton-Brown. An evaluation of sequential model-based optimization for expensive black-box functions. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 1209–1216. ACM, 2013. 3

[25] I. Ilievski, T. Akhtar, J. Feng, and C. A. Shoemaker. Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates. In *AAAI*, pages 822–829, 2017. 3

[26] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998. 3

[27] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2017-02-23]. 4

[28] N. Kasthuri, K. J. Hayworth, D. R. Berger, R. L. Schalek, J. A. Conchello, S. Knowles-Barley, D. Lee, A. Vázquez-Reina, V. Kaynig, T. R. Jones, et al. Saturated reconstruction of a volume of neocortex. *Cell*, 162(3):648–661, 2015. 2, 7, 8

[29] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 17:1–5, 2016. 1, 3

[30] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998. 6

[31] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Efficient hyperparameter optimization and infinitely many armed bandits. In *ICML 2016 workshop on AutoML (AutoML 2016)*, 2016. 2, 3

[32] J. W. Lichtman, H. Pfister, and N. Shavit. The big data challenges of connectomics. *Nature neuroscience*, 17(11):1448–1454, 2014. 7

[33] D. Maclaurin, D. K. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, pages 2113–2122, 2015. 2, 3

[34] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015. 1

[35] 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. 1, 4, 6

[36] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015. 2, 7, 8

[37] J. a. N. Silva, L. Veiga, and P. Ferreira. Heuristic for resources allocation on utility computing infrastructures. In *Proceedings of the 6th International Workshop on Middleware for Grid Computing*, MGC '08, pages 9:1–9:6, New York, NY, USA, 2008. ACM. 3

[38] 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. 1, 2, 3

[39] A. Team. Aetros, 2016. 1, 3

[40] T. H. Team. H2o: Scalable machine learning. version 3.1.0.99999, 2015. 1, 3

[41] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, Mar 2002. 3

[42] F. Valdez, P. Melin, and O. Castillo. Modular neural networks architecture optimization with a new nature inspired method using a fuzzy combination of particle swarm optimization and genetic algorithms. *Information Sciences*, 270:143–153, 2014. 8

[43] A. Vose, J. Balma, G. Wenes, and R. Sukumar. Deep neural network hyperparameter optimization with genetic algorithms. 2017. 3

[44] L. Yi, C. Moretti, S. Emrich, K. Judd, and D. Thain. Harnessing parallelism in multicore clusters with the all-pairs and wavefront abstractions. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 1–10. ACM, 2009. 4, 5

[45] Y. Zhang, J. Sun, and Z. Wu. An heuristic for bag-of-tasks scheduling problems with resource demands and budget constraints to minimize makespan on hybrid clouds. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pages 39–44, Aug 2017. 3

[46] Y. Zhang, J. Sun, and J. Zhu. An effective heuristic for due-date-constrained bag-of-tasks scheduling problem for total cost minimization on hybrid clouds. In *2016 International Conference on Progress in Informatics and Computing (PIC)*, pages 479–486, Dec 2016. 3

[47] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 3, 8

[48] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017. 3