Ordalia: Deep Learning Hyperparameter Search via Generalization Error Bounds Extrapolation

Benedetto J. Buratti
Computer Science
Brown University
Providence, Rhode Island, USA
benedetto_buratti@brown.edu

Eli Upfal
Computer Science
Brown University
Providence, Rhode Island, USA
eli_upfal@brown.edu

***Abstract*—We introduce *Ordalia*, a novel approach for speeding up deep learning hyperparameter optimization search through early-pruning of less promising configurations. Our method leverages empirical and theoretical results characterizing the shape of the generalization error curve for increasing training data size and number of epochs. We show that with relatively small computational resources one can estimate the dominant parameters of *neural networks*' learning curves to obtain consistently good evaluations of their learning process to reliably early-eliminate non-promising configurations. By iterating this process with increasing training resources *Ordalia* rapidly converges to a small candidate set that includes many of the most promising configurations. We compare the performance of *Ordalia* with Hyperband, the state-of-the-art model-free hyperparameter optimization algorithm, and show that *Ordalia* consistently outperforms it on a variety of deep learning tasks. *Ordalia* conservative use of computational resources and ability to evaluate *neural networks* learning progress leads to a much better exploration and coverage of the search space, which ultimately produces superior *neural network* configurations.**

***Index Terms*—Deep Learning, Hyperparameters Optimization, Multi-armed Bandits, Automated Machine Learning**

I. Introduction

Despite the great advancements in *Deep Learning*, finding a network's optimal *hyperparameters* is still a matter of tinkering. In the best cases, tuning a *neural network*'s *hyperparameters*[1] is a resource-intensive process that involves exhaustive evaluation of massive hyperparameters-grids, in the worst, a manual search for the best configuration. There exist many frameworks to tackle *neural networks*' hyperparameters optimization automatically, recently Hyperband [1] received considerable attention for its performances and simplicity. Hyperband outperforms more established Bayesian frameworks, e.g SMAC [2] or TPE [3], which do not scale well in *neural networks*' highly dimensional search space, as discussed by the authors in [4].

In this work, we introduce **Ordalia**, a novel approach for speeding up *neural networks* hyperparameter search through early-termination of less promising configurations. Our method leverages empirical and theoretical results from learning theory to characterize the shape of the generalization error bounds for increasingly larger training samples and epochs to evaluate the *neural networks* learning capabilities. *Ordalia* exploits the fact that *neural networks* validation learning curves have a sub-linear convergence to their *Bayes Error* with respect to the training sample size and number of epochs. By training on incrementally larger samples and epochs, *Ordalia* quickly tests a vast number of networks configurations and estimates their learning curves dominant parameters to get a projection of their final performances which it uses as a proxy-metric to evaluate their learning progresses and discard the non-promising ones.

Previous works [5]–[10] perform hyperparameter optimization based on empirical multi-fidelity estimate of their final performances. However, those works introduce a significant computational overhead due to their search-space modeling, which is particularly burdensome in the context of *neural networks* high-dimensional hyperparameter space. As suggested by the empirical analysis in [11], it is possible to reliably model *neural networks'* learning curves with power-laws, a phenomenon coherent with learning theory results on generalization error bounds [12]. The seminal work [13] was the first to propose the use of power-laws to model *neural networks* training/validation learning curves to predict their final performances. However recent work [14] showed that the large capacity of modern *neural networks* frequently leads them to overfit to the training sample, making it impossible to leverage training curves as originally proposed in [13].

Ordalia builds on those results and introduces a novel method to evaluate *neural networks* learning progress to perform model selection based on their intermediate performances. We show that with a relatively small-sized training set and number of epochs one can estimate the dominant parameters of the learning curve to obtain consistently sufficiently good evaluations to reliably eliminate sub-optimal configurations. By iterating this process with increasing training resources, the process rapidly converges to a small candidate set that includes many of the most promising configurations. In summary, the contributions of this paper are:

- We provide empirical evidence and theoretical motivation to explain why *neural networks* learning curves scale as a power-law with respect sample size/number of epochs
- We introduce *Ordalia*, a *neural network* early-termination strategy that leverages incremental training and empirical power-law projections to evaluate *neural networks* learning process and prune bad hyperparameters configurations.

[1] e.g. layers size, dropout rate, learning rate, batch size, number of epochs

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

- We test *Ordalia* on MNIST, CIFAR10 and CIFAR100 and show that it outperforms state-of-the-art *Hyperband*
- We test *Ordalia* on MNIST, CIFAR10, and CIFAR100 and show that it provides a better metric to evaluate and prune *neural networks* than train and validation error.

II. Background

During the last decade machine learning has become a ubiquitous and increasingly adopted technology embraced in a heterogeneous variety of fields, even far apart from computer science or statistics. Recently *Automated Machine Learning* (AutoML) frameworks [15]–[20] become a convenient way to automate end-to-end machine learning pipelines design in real-world problems. In order to extract actionable insights out of raw data, an AutoML system needs first to **(1)** *ingest raw data* from different formats, **(2)** *clean/prune* them by identifying corrupted records and **(3)** *pre-porcess* the attributes to create new features starting from the original ones. Then the systems needs to **(4)** *select the best machine learning model* for the given dataset and find its optimal parameters via **(5)** *Hyper-parameter Optimization*. Finally it needs to **(6)** *validate this pipeline* by evaluating the end-to-end solution against an hold-out dataset. **(7)** Once a model that satisfies the user desiderata is found, it is sent to *deployment*.

Hyper-parameters Optimization: it is possible to model data with a wide variety of algorithms, each of which has a specific set of hyperparameters that needs to be tuned (e.g. SVM's γ and c have no meaningful interpretation for a Random Forest). We define as *Hyperparameters Space* $\mathcal{H}$ the space that contains all the possible hyperparameters combinations obtainable from a given hyperparameters domain. We then define as *loss* a function l that evaluates an algorithm's hyperparameter configuration h on a sample S returning a real value $l_S : h \to R$ (in practice the returned value can be any arbitrary metric that we want to use to evaluate the configurations). Finally we define h^* as the best possible configuration, that is $h^* = \arg\min_{h \in \mathcal{H}} l_S(h)$, and $y^* = l_S(h^*)$ as the loss that the configuration h^* obtains on the task using S. In other terms y^* is the best possible obtainable performance in the search space $\mathcal{H}$ for the given task, using a sample S. Hyperparameters spaces are massive, heterogeneous, conditional mathematical objects that do not allow an analytical treatment of l. For this very reason is hard to find the optimum h^* and the only way to achieve this goal is to compute $l_S(h)$ by evaluating many configurations performances and perform black-box optimization. There are two main paradigms to perform such optimization:

- **Model-Free:** the optimizer explores the search space $\mathcal{H}$ using a sampling and evaluation strategy that does not rely on any modeling assumptions of $l_S(\cdot)$. Examples in this category are Grid Search, Random Search and most of the Bandits-based strategies.
- **Model-Based:** the optimizer uses a surrogate model $\mathcal{M}_{l_S}$ of the unknown function $l_S(\cdot)$, built and updated based on past evaluations. This model is designed to be an analytically treatable surrogate of the function l_S, which is used to adaptively guide the search space exploration. Examples in this category are Cost-Based Selection, Bayesian Optimization, Simulated Annealing.

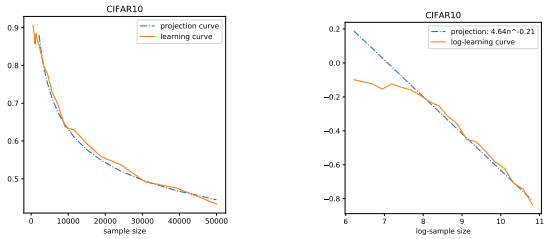

Fig. 1: **CIFAR10 learning curves**: we show a representative example of a CNN validation learning curve (blue) and its power-law modeling (orange). The first plot is in natural scale, while the second in log-log scale.

The two strategies are not in conflict with each other. We can use model-based techniques to achieve the global optimum h^* sampling n most promising configurations from $\mathcal{M}_{l_S}$ and use a model-free strategy to optimally allocate the budget by enforcing early-termination during the evaluation stage.

Non-Stochastic Infinite-armed Bandit Problem: in this paper we focus on a *Model-Free* optimization strategies, in a sense that we do not attempt to model the hyperparameter space. We frame *neural network* hyperparameter optimization as a *Non-Stochastic Infinite-armed Bandit Problem* (NIAB) as formalized in [1]. For each *neural network* configuration $h \in \mathcal{H}$ we associate a series of bounded loss functions $l_{S1}(h), l_{S2}(h), ...l_{Si}(h)$, where $l_{Si}(h)$ represent the validation error of a neural network with configuration $h \in \mathcal{H}$ using S_i resources. We assume the series to be monotonically decreasing and converging to a value $l_* = \lim_{k \to \infty} l_m$. While we make no assumptions on the loss functions l_{Si}, we do take advantage of the fact that *neural networks*' learning curves scales as power-laws with respect to training resources [11]. Hence, we model the losses-series convergence rate $l_* = \lim_{Si \to \infty} l_{Si}$ with empirical sublinear functions in the form $\hat{l}_*(h) = \hat{a} m^{-\hat{b}}$ (as shown in figure 1). In the next sections show in the next sections we show how we can use this phenomena to build a more reliable metric to evaluate *neural networks* early-performances than simply relying on point-wise training or validation error.

III. Related Work

In recent years an extensive literature on *automated machine learning* have been produced about hyperparameters optimization [15], [21]–[26]. The most similar framework to *Ordalia* is **Hyperband** [1] a bandits-algorithm that incrementally allocates a fixed budget to perform *neural network* hyperparameters tuning. The main drawback of such system is that it just uses an incremental number of epochs (not a combination of training sample and number of epochs) and does not make any assumptions on the networks convergence to their true error, which makes hard for *Hyperband* to distinguish between a complex estimator, that requires many epochs and training

samples to perform well, and a bad one. In [13], the authors investigate the relationship between training sample size and generalization error. They propose a technique to estimate *neural networks*' final performances by modeling their learning curves as $l_{validation}(h) = a+b/m^{\alpha}$ and $l_{train}(h) = a-c/m^{\beta}$ (where m is the training set size) based on partial training and validation results. To solve this system of equations and find a (the network final error) they assume that the validation and training error have the same convergence rates, hence $\alpha = \beta$ and $b = c$. While such assumption may appear conceptually appealing, it proved to be not realistic given that *neural networks* usually have such high-capacity to be often able to memorize the whole training set, even if randomly shuffled as showed in [14]. Other frameworks like *Auto-WEKA* [27] and the twin package **Auto-sklearn** ([16]) address hyperparameter optimization from the **model-based** perspective, by using *Sequential Model-based Algorithm Configuration* (SMAC). SMAC is a *Bayesian Optimization* (BO) algorithm that utilizes a random forest trained on previous runs logs to estimate new configurations potential improvement with respect the current best using *expected improvement* as acquisition function. Unfortunately, the sophistication introduced by this meta-modeling leads to a considerable overhead in the absence of any multi-fidelity strategy as shown in [1]. Other model-based hyperparameters optimization works [4], [6], [7] model the search space/learning curve to select the configurations to train based on multi-fidelity performances estimates. While those approaches introduce a computational overhead, mainly due to the Bayesian modeling, they can be considered as a valid model-based orthogonal approach to *Ordalia*. Finally, *Auto-Keras* [28] perform full neural architecture search in which the exploration process is performed by morphing the neural network (i.e., inserting a layer or adding a skip connection) guided by simulated annealing; also this approach can be considered orthogonal to *Ordalia*.

IV. Ordalia

In this section, we discuss *Ordalia*, a multi-stage bandits-based early-termination-strategy for *neural network* hyperparameter optimization. At each stage *Ordalia* prunes sub-optimal networks by incrementally training them on increasingly large samples/number of epochs and evaluating their learning progress. We first introduce the theoretical framework behind *Ordalia*, then we use these results to give an algorithmic overview of *Ordalia*. Finally, we discuss the implementation details regarding the learning curves extrapolation.

A. Generalization Error Convergence Rates

As introduced in Section III previous work on *neural networks* model-free *early-termination* has been based on the idea of pruning bad configurations using partially trained networks. Those intermediate evaluations are computed from early iteration results of *stochastic gradient descend* **(SGD)** and only based on its latest point-wise evaluation. As presented in [1] this approach has the main drawback of penalizing slow-learners, e.g. *neural networks* with small learning rate, which in the early stages do not perform well and suffer the 'competition' from ultimately-worst, but faster-learning configurations. Even the model-based error prediction approach proposed in [29] is not feasible for *neural networks* hyperparameters optimization due to the high dimensionality of their search space, and consequent computational overhead as discussed in [4]. *Ordalia* is a model-free early-termination strategy that leverages generalization error bounds and makes them actionable for practical *neural networks* selection by modeling their learning curves .

Following [12], given an arbitrary function h in an hypothesis space $\mathcal{H}$[2], and a sample $S = \{z_1, z_2, \cdots, z_m\}$ from a distribution $\mathcal{D}$, we define the true loss of a function h with respect the distribution $\mathcal{D}$ as $L_h = L_{\mathcal{D}}[h(z)]$ and the empirical estimate of L_h on the sample $\hat{L}_h = \frac{1}{m}\sum_{i=1}^{m} h(z_i)$.

Then, with high probability (as $m \to \infty$), we have

$$|L_h - \hat{L}_h| \leq 2\widetilde{\mathcal{R}}(\mathcal{H}, \mathcal{S}) + 3\epsilon, \tag{1}$$

where $\widetilde{\mathcal{R}}(\mathcal{H}, \mathcal{S})$ is the empirical Rademacher Complexity. Now, using Massart's Lemma, we bound $\widetilde{\mathcal{R}}(\mathcal{H}, \mathcal{S})$ as follows:

$$\widetilde{\mathcal{R}}(\mathcal{F}, \mathcal{S}) \leq \frac{B\sqrt{2\ln|\mathcal{F}|}}{m} = \sqrt{\frac{2d\ln(m)}{m}} \tag{2}$$

where $B = \max_{f\in\mathcal{F}} \left(\sum_{i=1}^{m} f^2(z_i)\right)^{1/2} = \sqrt{m}$, and d is the VC-dimension of $\mathcal{H}$. Equations (1) and (2) provide an upper bound on the difference between the *generalization error* and its empirical estimate, and provides a useful metric to evaluate the overall learning progress of the networks based on the training resources and the observation that $|L_h - \hat{L_h}|$, converges as $O(1/\sqrt{m})$.

This theoretical result is supported by the extensive empirical evidence in [11] in which *neural networks* exhibit this kind of behavior for a wide variety of learning settings (e.g. image classification, machine translation). In figure 1 we provided a visual instance of this phenomena for CNN solving image-classification on **CIFAR10** and show the validation learning curve in standard and log-log scale with its quasi-perfectly fitting empirical power-law models. In the next subsection, we use those results in the description of *Ordalia*, which leverages *neural networks* convergence rates to perform hyperparameters optimization for neural networks.

B. Ordalia

Ordalia efficiently explores a given hyperparameter space $\mathcal{H}$ by incrementally allocating a given budget B, similarly to *Successive Halving*. It works over pruning rounds, at the end of which just the top performing fraction configurations are passed to the next stages. *Ordalia* takes as input (1) n *neural networks* sampled at random from the search space $\mathcal{H}$, (2) a training and (3) a validation set. It first splits the training set into monotonically increasing train sub-samples to which it associates a specific number of epochs. The numbers of epochs are also monotonically increasingly such that we have a series of tuples $(m_1, e_1), (m_2, e_2), ...(m_b, e_b)$ which

[2] $\mathcal{H}$ is the hyperparameters space in our context

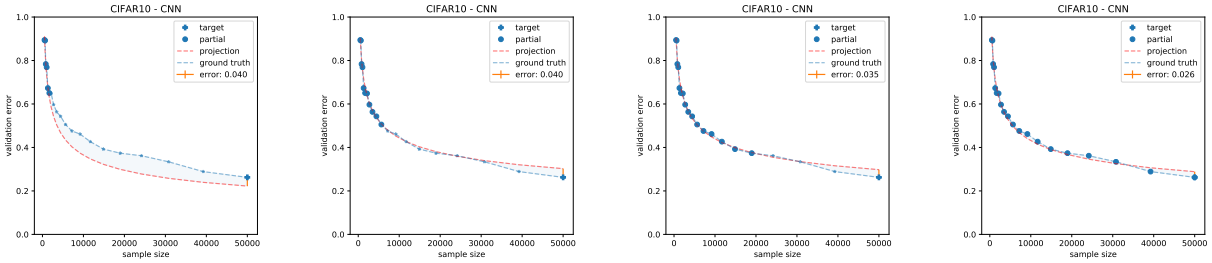

Fig. 2: **Ordalia's Projection:** progression of Ordalia's projections, with 5, 10, 15, 20 partial-evaluations. The dashed blue line is the actual learning curve, the red one Ordalia's projection at that round, the blue points are the partial errors.

Algorithm 1: Ordalia-Termination (OT). *Ordalia* splits the training resources into exponentially larger sets and samples n configurations at random from the search space. Then at each round the algorithm computes the Ordalias's projection for each of the surviving configuration up to that round and then ranks them accordingly: at round i just the top-$\frac{\eta_{i-1}}{\eta_i}$ are retained for the successive rounds. During the last round all the resources η are used and the top performing configuration returned.

Input: Training Resources η, validation set $\mathcal{D}_{val}$, search space $\mathcal{H}$
Output: h^* (best architecture)

Split η into exponentially larger sets $\eta_1, \ldots, \eta_N$
$\mathcal{H}$ = get_rnd_network_architecture(n);
for $i \in 1 \ldots N$ **do**
  $top = \frac{\eta_{i-1}}{\eta_i} \cdot |\mathcal{H}|$
  $\hat{l}_\eta(h) \leftarrow [OP(h, \eta)\ \forall h \in \mathcal{H}]$
  $\mathcal{H} = \left[\mathcal{H}[i]\ \forall i \in argsort(\hat{l}_\eta(h))[0 : top]\right]$
  if $len(\mathcal{H}) == 1$ **then**
    $h^* = \mathcal{H}[1]$
    break
  end
end
return h^*

determines how many training sample m_i and for how many epoch e_i the *neural networks* should train at each pruning round i. Each time a *neural network* is trained using (m_i, e_i) resources, its validation error stored and used to extrapolate the dominant parameters $\hat{a}, \hat{b}$ of its validation learning curve $\hat{l}_\eta(\cdot) = \hat{a}\eta^{-\hat{b}}$, where η is the product of the training sample size and number of epochs $\eta = me$[3]. Once the dominant parameters $\hat{a}$ and $\hat{b}$ are obtained, *Ordalia* computes the *neural network*'s performance projections and uses them as a metric to evaluate their learning progresses and decide which configuration to early-terminate. At each pruning round i, *Ordalia* applies the following criterion:

Pruning Criterion: *at pruning round i, all survived neural networks are trained using η_i resources.* Ordalia *then computes and uses the partial validation error to compute the networks final performance projection. Once all configurations performance estimates are computed, the algorithms ranks them accordingly, and retains the top $(\eta_i)/(\eta_{i+1})$[4] networks for the next pruning round and terminates the other ones. Once just k configurations are left,* Ordalia *trains them using all the available resources and returns them ranked based on the final validation error.*

In Algorithm 1 and figure 2 we provide an algorithmic outline and a visual intuition of *Ordalia* running time behaviour. While the performances of *neural networks* in the early stages are unrepresentative in terms of *absolute* values, *Ordalia* uses them to judge the overall learning process of the networks up to iteration i and prune bad performers without actually needing to training them on the whole training sample. One of the main innovations introduced by *Ordalia* is the empirical extrapolation of learning curves' dominant parameters to compute this proxy score of the networks final performance. Other algorithms such as *Hyperband* utilize just the partial validation error as a ranking measure, however, this has the drawback to prune resources-hungry configurations (slow-learners) together with the bad ones. Indeed resources-hungry configurations are likely to perform poorly in early-stages while improving and ultimately taking over in the later stages. By extrapolating the learning curve parameters, *Ordalia* takes into account the full learning dynamic of the network distinguishing slow-learners from bad ones.

C. Learning Curves Early-Extrapolation

To evaluate the learning process of a *neural network* configuration we extrapolate its dominant parameters from its validation learning curve as shown in Algorithm 2. In order to do so we map the validation errors and ηs into the log-log space. In this space the original model $\hat{l}_\eta(\cdot) = \hat{a}\eta^{-\hat{b}}$ is represented as $\log \hat{l}_\eta(\cdot) = \log \hat{a} - \log(\hat{b})\eta$, hence we can efficiently fit a regression line and obtain the intercept and slope that represent the power-law multiplicative constant in the log-space[5] $\log(\hat{a})$

[3] we interpret η_i as the cardinality of a sample generated by the concatenation of another sample of size m_i with itself for e_i times.

[4] this particular ratio is used to maintain each round computational cost constant

[5] hence, further exponentiation is required

Algorithm 2: Ordalia-Projection (OP). For a given *neural network* configuration h *Ordalia* evaluates its overall learning process by extrapolating $l(h)$'s dominant parameters $\hat{a}$ and $\hat{b}$ in the log-log space, and then computing the projection $\hat{a}\eta_B^{-\hat{b}}$.

Input: Network h, partial training resources $\{\eta_1, \ldots, \eta_i\}$, max resources η, breaking point μ, to be returned k

Output: Projection

$partial\_score \leftarrow h.train(\eta_i)$
$h.scores.append(partial\_score)$;
if $i \leq \mu$ **then**
  $log\_scores = \log(h.scores)$
  $log\_\eta = \log(\eta_1, \ldots, \eta_i)$
  $\log(\hat{a}), \hat{b} = LinearRegression(log\_\eta, log\_scores)$
  $l_\eta(h) = \hat{a}\eta_B^{-\hat{b}}$
else
  $l_\eta(h) = partial\_score$
return $l_\eta(h)$

and exponent $\hat{b}$, respectively. Given that the partial errors obtained in the early pruning rounds are computed on relatively small sample sizes, they are less reliable than the later ones, and for this reason when *Ordalia* computes the dominant parameters in the log-log space, it weights each partial validation point using ηs' standard deviations: $\sqrt{\eta_1}, \sqrt{\eta_2}, ...\sqrt{\eta_i}$.

In figure 1 we show that there are two major and distinct learning phase. This phenomena, clearly visible in the log-log plots and also reported also in [11], suggests that in the initial stages the training sample/epoch number is too small and not representative of the whole distribution, hence the architecture performs as random guesser. In this phase, called *small-sample learning region*, it is hard to extrapolate any information regarding the learning process and a larger sample size/epoch number is required. Then the trained architecture's h generalization error begins to converge to its *irreducible error* as a power-law with respect η. In this second stage, called *power-law learning region*, it is possible to empirically model the networks learning process via $\hat{l}_\eta(\cdot) = \hat{a}\eta^{-\hat{b}}$. [6].

The transition point between the *small-sample learning region* and *power-law learning region* is called the *breaking point*. Each algorithm has its own breaking point that depends on the complexity of the model and on complexity of the task. *Ordalia* starts extrapolating the bounds constants just after the breaking point μ[7] and uses validation error in the initial coarse pruning rounds, as reported in the Algorithm 2. In case of pathologically configured models, which never reach a breaking point and perform as random guessers, only the validation error is used and no attempt to model the learning curve is performed.

[6]Eventually the trained network hits its irreducible error (Bayes error), where more data, or resources, do not improve the performances

[7]which is computed checking the derivatives of the errors progression

	MNIST	**CIFAR**s
Batch Sz.	16, 32, 64,128	32, 64,128
Learning Rate	0.00005, 0.0001, 0.0002, 0.0005, 0.001	0.00005, 0.0001, 0.0005, 0.001
Optimizer	RMSprop, SGD, Nadam, Adagrad, Adadelta, Adam, Adamax	RMSprop, SGD, Adadelta, Adam
Dropout	0.1, 0.2, 0.5	0.1, 0.2, 0.5
FF Size	128, 256, 512	128, 256, 512
CNN Size	N.A.	16, 32,64

TABLE I: **Search Space:** on the left we report the hyperparameters space of the 3-layer FNNs used on MNIST, for a total of 1260 hyperparameter combinations. On the right, the hyperparameters space for the 3-layer CNNs used on CIFAR10 and CIFAR100, for a total of 1296 configurations.

V. Experiments

A. Experimental Setup

For our experiments we used image classification datasets **MNIST**, **CIFAR10** and **CIFAR100**. The architectures that we used in our experiments are 3-layers FNN for MNIST and a 3-layers CNN. The neural network implementation has been done using *Keras* using *Tensorflow* as backend. The hyperparameters search has been executed over the space defined by table I.

Hardware Environment: the experiments have been executed on *Google Cloud Platform* (GCP) with N1-highmem machines, with 4-core *Intel Haswell* vCPUs with 26 GB memory, running Debian 4.9 and Python 3.6.3. The *neural networks* training has been performed on a NVIDIA Tesla V100 GPUs.

B. Ordalia v Hyperband

We now discuss the experimental results obtained during the comparison of *Ordalia* against *Hyperband* on *MNIST, CIFAR10* and *CIFAR100*. On **MNIST**, we executed 100 hyperparameter optimization runs using a 3-layered FNN with a *running time cap* of 12000 seconds for each experiment. During those runs, *Ordalia* was able to test an average of 270 different hyperparameters configurations, while *Hyperband* just 67. Instead on **CIFAR10** and **CIFAR100**, we executed 100 hyperparameters optimization runs using a 3-layered CNN with a *running time cap* of 15000 seconds for each experiment. In those cases, *Ordalia* was able to test on average respectively 195 and 200 configurations, while *Hyperband* just 32. The experiments total running time for each dataset is roughly 17 CPU-days, for a total of 51 days.

The plots in figure 3 show that given a *running time cap*, *Ordalia* achieve consistently better final performance by testing much more hyperparameters configurations than *Hyperband*. In particular in the first column in figure 3 it is possible to see that *Ordalia* is able to perform on average **~8x** more partial evaluations than *Hyperband*. The second column displays the validation error against the number of partial evaluations over the three datasets and it shows how *Ordalia* achieves a much better coverage of the search space by taking a conservative

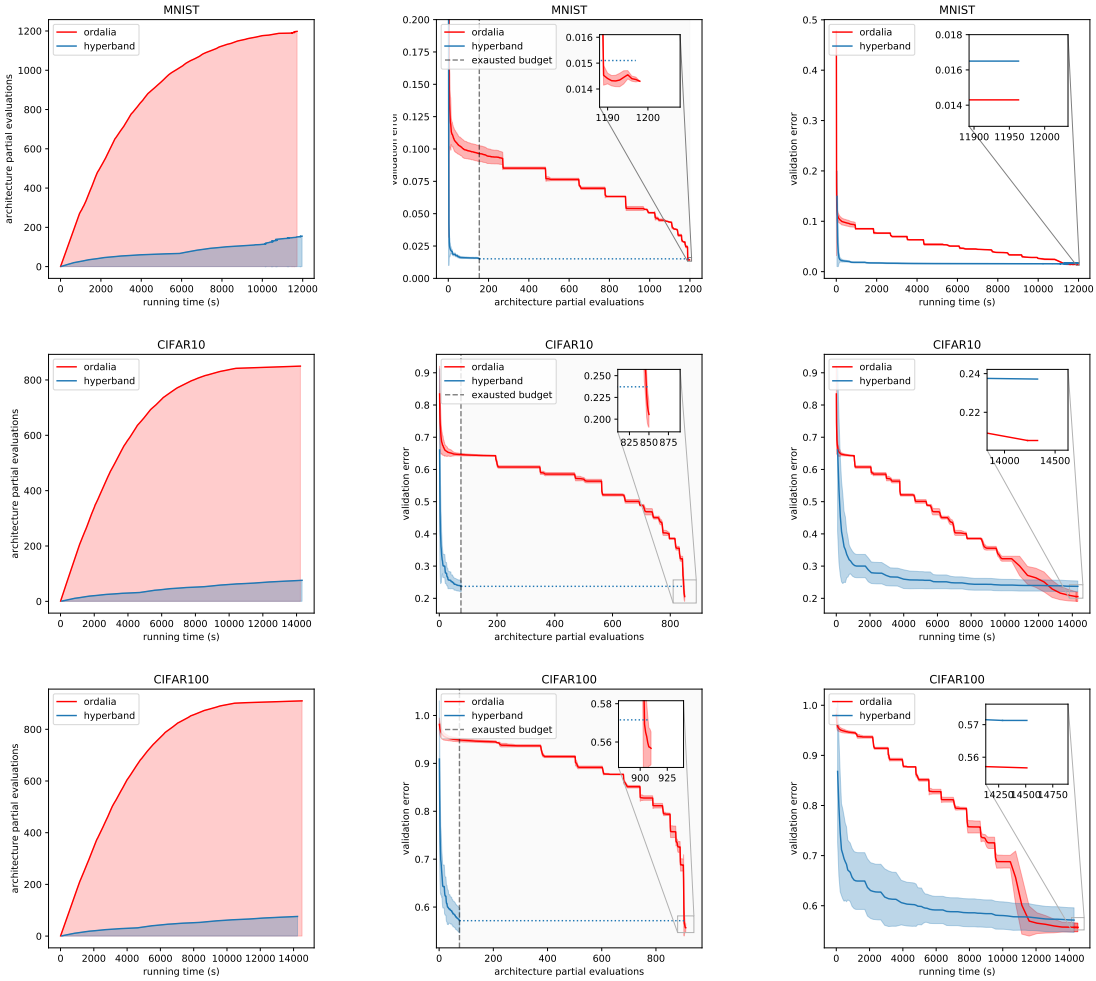

Fig. 3: ***Ordalia* v *Hyperband***: each row shows the experiments' results obtained on MNIST, CIFAR10 and CIFAR100, respectively. The left column compares the number of partial evaluated configurations by the two strategies at any given time. In the central column the two strategies global validation error is plotted against the number of partial evaluations for the three datasets. The black dashed line indicates at which point of the search space when *Hyperband* exhausted its budget. Finally in the last column the global validation error is plotted against the running time.

approach in the evaluation of configurations. The vertical dashed line represents the number of configurations after which *Hyperband* exhausts its budget while *Ordalia* still has plenty of resources to explore the search space. *Ordalia* red curve has a monotonically decreasing step-wise shape and each step represents a new round using more training data and epochs: this phenomenon reflects exactly the design behind *Ordalia*.

Finally, the third column shows that, while *Hyperband* intermediate results converge faster to its final performance by immediately training the networks on the whole training set, for this same very reason, it wastes too many resources on bad configurations, quickly exhausting its budget during the exploration and returning worst configurations.

In every tested dataset, *Ordalia's* conservative use of resources leads to a better coverage of the search space, which bundled with an early-on detection of top configurations via projections, leads ultimately to a better selection process. On average, *Ordalia* top-network accuracy outperforms *Hyperband's* one by **0.22%** on MNIST, **3.18%**, on CIFAR10 and **1.46%** on CIFAR100.

C. Pruning Criteria Comparison

We now discuss the impact of different pruning criteria on the final performances of the model selection process. At each pruning round the *neural networks* are evaluated, ranked and pruned based on the four criteria: **Ordalia's projections**, **validation-error**, **train-error** and **random** (baseline). For this experiment we used *recall* as metric to evaluate the ability of those strategies to retain top-performing configurations until the last round. We tested the four criteria on a series of 100 experiments over **MNIST**, **CIFAR10** and **CIFAR100** datasets. For each experiment we sampled 400 configurations from the hyperparameters grid defined in table I (roughly ~1200 architectures). The total running time for each dataset

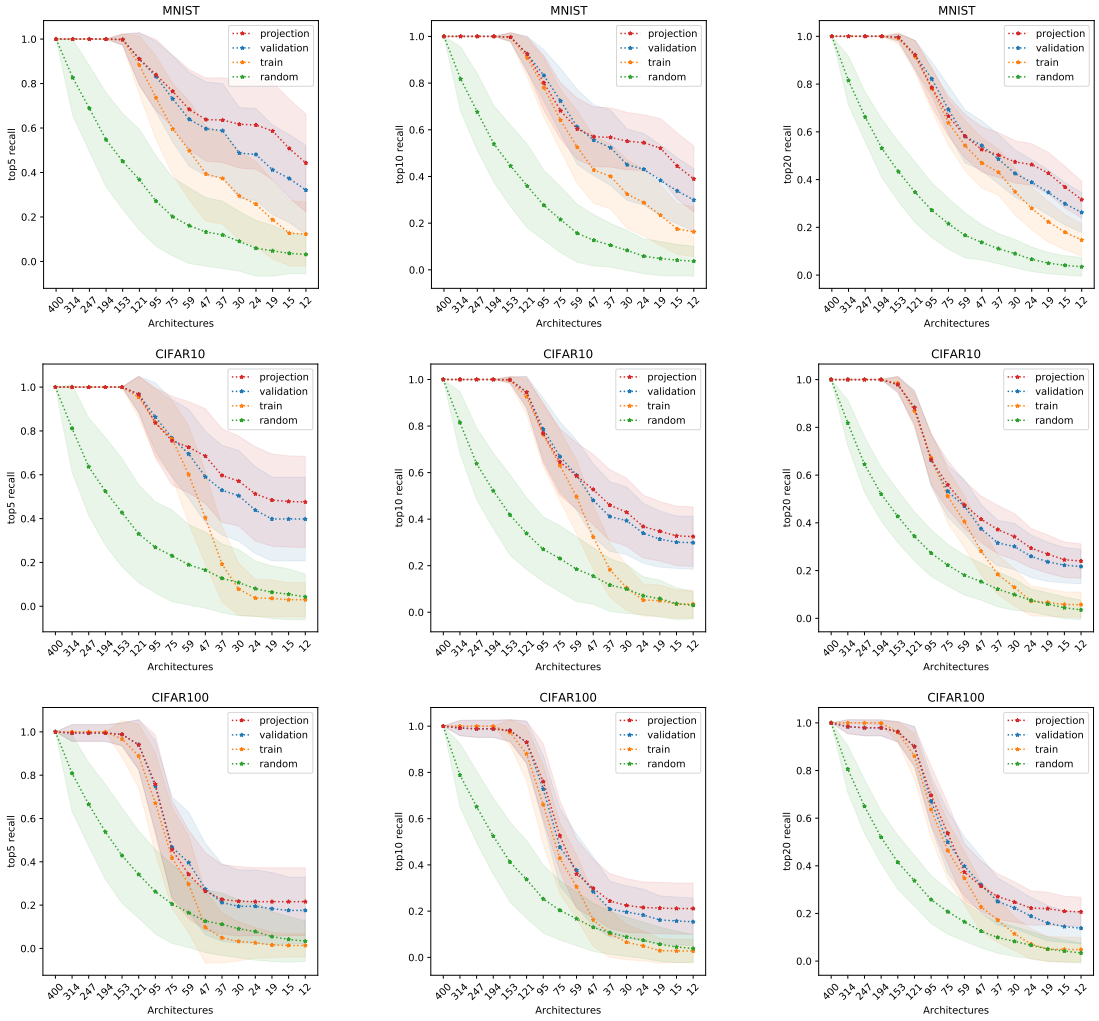

Fig. 4: **Pruning Criteria Comparison:** each row shows the experimental results performed on MNIST, CIFAR10 and CIFAR100. Each column reports the recall values at each pruning round for different *top-k* values, from left to right *top-5*, *top-10* and *top-20*. Ordalia's projections determine a substantial improvement in terms of recall in the mid-range pruning rounds, where enough intermediate evaluations are available.

experiments has been roughly 30 CPU-days, for a total of 90 days over the three datasets. In figure 4 we show the results of those experiments which indicates that *Ordalia* retains *top-k* performing configurations better than the other criteria. On the *abscissa* we report the number of "surviving" architectures at each pruning round and on the *ordinate* the strategies recall (percentage of retained top-k configurations at that round). In each row we report the recall plots for each of the tested dataset, while for each column we report different values of k that we used to compute the recall (from the left to right *top-5*, *top-10* and *top-20*).

It is immediate to notice that for each of the performed experiments **projection** and **validation error** set themselves apart from the other two criteria, being able on average to retain much more *top-k* configurations. There are virtually no differences between *OP* and *validation error* in the earlier rounds. This is due to the *small-sample learning region* problem discussed in section IV-C, which makes hard to get reliable projections, hence during those rounds *Ordalia* relies on the validation score to prune the networks. However, once enough partial evaluations points are available, *Ordalia* starts to evaluate the networks based on Algorithm 2. This determines better configurations pruning, especially in mid-pruning rounds, that ultimately leads to an higher final recall of top performing networks. On *MNIST* Ordalia's Projections generate the largest improvements over the other techniques, especially for the top-5 recall experiments, where there is a gap of $\sim 10\%$ between the projections and the validation error.

VI. Conclusions and Future Work

This paper leveraged empirical evidence and learning theory results to propose a novel technique based on incremental training and power-law projections to perform early-stage pruning of non-promising *neural network* configurations. While

the proposed technique outperforms *state-of-the-art* strategies, our research raised new important questions regarding *neural networks*. The *first problem* is to build a comprehensive theoretical framework to explain why neural networks learning curves scale as power-laws with respect increasing training set sizes. While recent results in learning theory and deep learning [30]–[32] provided us with a first intuition on this point, more investigation is required to fully understand why such consistent behavior takes place. The *second open question* arose during our experiments and it is related to the minimum amount of resources, in terms of epochs and sample size, that a *neural network* architecture requires to actually start learning. We are interested in investigating the reasons why after a specific amount of training resources, neural networks stops behaving as a random guesser and abruptly starts learning from data. This problem is crucial in the context of *hyperparameter optimization* because the ability to detect the *"small-sample learning region"* deeply impacts the goodness of the projections. The *last open problem* to investigate is the relationship between train and validation error. Our empirical investigation showed that for the same datasets, different networks have strikingly different train and validation learning curves. For some architectures, we found a positive correlation between the two, while for others a negative or a zero one. However, regardless of the correlation of the two curves, the models show good generalization capabilities which stress the need for a deeper understanding of what overfitting means in the context of *neural networks*.

ACKNOWLEDGEMENTS

The project is supported, in part, by DARPA/Army grant W911NF-16-1-0553, DARPA/AF Award FA8750-19-2-1006, and NSF Award IIS-1016648.

REFERENCES

[1] L. Li *et al.*, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *arXiv preprint arXiv:1603.06560*, 2016.

[2] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523.

[3] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2546–2554. [Online]. Available: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

[4] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," *ArXiv*, vol. abs/1807.01774, 2018.

[5] M. Poloczek, J. Wang, and P. Frazier, "Multi-information source optimization," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4288–4298. [Online]. Available: http://papers.nips.cc/paper/7016-multi-information-source-optimization.pdf

[6] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," in *ICLR*, 2017.

[7] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian hyperparameter optimization on large datasets," 2017.

[8] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Poczos, "Multi-fidelity bayesian optimisation with continuous approximations," 03 2017.

[9] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task bayesian optimization," in *NIPS*, 2013.

[10] J. S. K. Swersky and R. P. Adams, "Freeze-thaw bayesian optimization," *ArXiv*, vol. abs/1406.3896, 2014.

[11] N. A. G. D. H. J. H. K. M. M. A. P. Y. Y. Y. Z. Joel Hestness, Sharan Narang, "Deep learning scaling is predictable, empirically," *arXiv:1712.00409*, 2017.

[12] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. New York, NY, USA: Cambridge University Press, 2005.

[13] C. Cortes, L. D. Jackel, S. A. Solla, V. Vapnik, and J. S. Denker, "Learning curves: Asymptotic values and rate of convergence," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan-Kaufmann, 1994, pp. 327–334. [Online]. Available: http://papers.nips.cc/paper/803-learning-curves-asymptotic-values-and-rate-of-convergence.pdf

[14] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," 11 2016.

[15] E. Z. P. E. E. U. T. K. Zeyuan Shang, Benedetto J. Buratti. (2019) Democratizing data science through interactivecuration of ml pipelines.

[16] M. Feurer *et al.*, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2962–2970.

[17] R. S. Olson *et al.*, *EvoApplications 2016*, 2016, ch. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pp. 123–137. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-31204-0_9

[18] H2O.ai, *H2O Driverless AI*, 2019. [Online]. Available: https://github.com/h2oai/h2o-3

[19] C. Binnig, B. Buratti, Y. Chung, C. Cousins, T. Kraska, Z. Shang, E. Upfal, R. Zeleznik, and E. Zgraggen, "Towards interactive curation & automatic tuning of ml pipelines," in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, ser. DEEM'18. New York, NY, USA: ACM, 2018, pp. 1:1–1:4. [Online]. Available: http://doi.acm.org/10.1145/3209889.3209891

[20] C. Binnig, F. Basık, B. Buratti, U. Cetintemel, Y. Chung, A. Crotty, C. Cousins, D. Ebert, P. Eichmann, A. Galakatos, B. Hättasch, A. Ilkhechi, T. Kraska, Z. Shang, I. Tromba, A. Usta, P. Utama, E. Upfal, L. Wang, N. Weir, R. Zeleznik, and E. Zgraggen, "Towards interactive data exploration," in *Real-Time Business Intelligence and Analytics*, M. Castellanos, P. K. Chrysanthis, and K. Pelechrinis, Eds. Cham: Springer International Publishing, 2019, pp. 177–190.

[21] E. R. Sparks *et al.*, "Automating model search for large scale machine learning," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 368–380.

[22] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[23] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, no. 1, p. 18, 2016.

[24] T. Li *et al.*, "Ease. ml: towards multi-tenant resource sharing for machine learning workloads," *Proceedings of the VLDB Endowment*, vol. 11, no. 5, pp. 607–620, 2018.

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

[26] D. Golovin *et al.*, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1487–1495.

[27] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 847–855.

[28] H. Jin, Q. Song, and X. Hu. (2018) Auto-keras: Efficient neural architecture search with network morphism.

[29] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," in *ICLR*, 2017.

[30] D. Jakubovitz, R. Giryes, and M. R. D. Rodrigues, "Generalization error in deep learning," *CoRR*, vol. abs/1808.01174, 2018. [Online]. Available: http://arxiv.org/abs/1808.01174

[31] S. L. Smith, P. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *CoRR*, vol. abs/1711.00489, 2017. [Online]. Available: http://arxiv.org/abs/1711.00489

[32] K. Kawaguchi, L. Pack Kaelbling, and Y. Bengio, "Generalization in Deep Learning," *arXiv e-prints*, p. arXiv:1710.05468, Oct 2017.