Towards Scalable Parallel Training of Deep Neural Networks

Sam Adé Jacobs
Lawrence Livermore National Laboratory
jacobs32@llnl.gov

Nikoli Dryden
Department of Computer Science
University of Illinois at Urbana-Champaign
Lawrence Livermore National Laboratory
dryden2@illinois.edu

Roger Pearce
Lawrence Livermore National Laboratory
rpearce@llnl.gov

Brian Van Essen
Lawrence Livermore National Laboratory
vanessen1@llnl.gov

ABSTRACT

We propose a new framework for parallelizing deep neural network training that maximize the amount of data that is ingested by the training algorithm. Our proposed framework called Livermore Tournament Fast Batch Learning (LTFB) targets large-scale data problems. The LTFB approach creates a set of Deep Neural Network (DNN) models and trains each instance of these models independently and in parallel. Periodically, each model selects another model to pair with, exchanges models, and then run a local tournament against held-out tournament datasets. The winning model is will continue training on the local training datasets. This new approach maximizes computation and minimizes amount of synchronization required in training deep neural network, a major bottleneck in existing synchronous deep learning algorithms. We evaluate our proposed algorithm on two HPC machines at Lawrence Livermore National Laboratory including an early access IBM Power8+ with NVIDIA Tesla P100 GPUs machine. Experimental evaluations of the LTFB framework on two popular image classification benchmark: CIFAR10 [18] and ImageNet [19], show significant speed up compared to the sequential baseline.

KEYWORDS

deep learning, data parallelism, parallel computing

ACM Reference format:
Sam Adé Jacobs, Nikoli Dryden, Roger Pearce, and Brian Van Essen. 2017. Towards Scalable Parallel Training of Deep Neural Networks. In *Proceedings of Machine Learning in HPC Environments , Denver, CO, USA, November 12–17, 2017 (MLHPC '17),* 9 pages.
DOI: 10.1145/3146347.3146353

1 INTRODUCTION

The machine learning "revolution" has impacted several areas of computing, science, and engineering. Machine learning has been made popular in part, by the volumes and rates at which data are now captured, generated, processed, and analyzed. Deep learning in particular has attracted significant attention, and researchers have reported success of its applications in several domains including medical science [7], visual object recognition and image classification [3, 8, 16, 19, 23], speech processing [12, 13, 28], text processing [9] among others. Despite the remarkable progress in deep learning, substantial resources and time are still required in training computationally intensive deep learning applications.

In recent years, much attention has focused on parallel and distributed computing in solving deep learning problems at scale [9, 25–27]. For many data-intensive applications such as deep learning, parallel and distributed processing offers either improved time to solution or the opportunity to achieve a higher quality solution via more extensive training; both approaches enable larger problems to be solved than were feasible before. However, many of the current approaches in distributed deep neural network such as parameter server and asynchronous model update [9, 27] do not scale well to "wide" parallel data training algorithms or large data set. Some emerging research efforts [6, 11] are focusing on increasing the amount of data processed in each training step (i.e. mini-batch size), and have achieved good scaling, but only up to several thousand samples per training step.

The most common and existing neural network training technique uses gradient descent to optimize the parameters (i.e., weights, biases) of the network through error backpropagation [2]. Parallelizing gradient descent requires frequent synchronizations to produce a single gradient for all participants. These high-frequency synchronizations limit the scalability of gradient descent based training algorithms on large-scale HPC systems. Data parallelism is typically achieved by processing multiple input samples in a single step as a "mini-batch", which typically produces a better step in the descent. However, there are limits to the size of mini-batch, how effective the resulting step is, and how well the algorithm converges with fewer steps. Techniques that simply sum the parameters from independent models (via direct exchange or parameter servers) are examples of increasing the mini-batch size and exhibit these problems with convergence as the mini-batch scales to thousands of training examples per step. We made attempt in this work to address this problem by exploring a novel and new direction.

To address the aforementioned problem, we develop a new parallelization framework called Livermore Tournament Fast Batch Learning (LTFB) primarily targets at scaling deep neural network training for Big Data HPC. The main thrust of the proposed parallel algorithm is to minimize the amount of synchronization required

MLHPC '17, Denver, CO, USA
© 2017 ACM. 978-1-4503-5137-9/17/11…$15.00
DOI: 10.1145/3146347.3146353

for each step of the training algorithm and to develop a mechanism for merging or combining independently trained models. The LTFB framework creates a set of models, where each instance of the model is trained independently. Periodically, each model selects another model to pair with, synchronize, and then run a local tournament against held-out datasets. The winning model is then selected and continues training on the local dataset. This approach provides a mechanism for parallelizing the exploration of the initial state space as well as the state space after each tournament.

Our proposed framework leverages heterogeneity and large memory resources per compute node of future exascale systems. With such systems, larger models (per node) can be trained taking advantage of both memory and accelerators. At memory level, node-local NVRAM allow us to explore techniques for caching held-out datasets for periodically evaluating model performance or independently training instances of a model. Experimental evaluations of the LTFB framework on two popular image classification benchmark: CIFAR10 [18] and ImageNet [19], show significant speed up compared to the sequential baseline.

Contribution. We summarize the key contributions of our work as follows:

- Present a new multi-level tournament voting parallel training algorithm that uses scalable peer to peer communication.
- Our framework streamlines hyper parameter exploration by allowing diversity in each independently trained model, and minimizing the time spent training with suboptimal parameters.
- Demonstration of feasibility of the new approach on image classification tasks using HPC clusters.

Outline The rest of the paper is organized as follows. In Section 2, we provide a background to deep learning. In Sections 3, we discuss in details our proposed framework for parallel large-scale deep learning. We focus on problem definition, experimental setup, and experimental results in Section 4. We present related work in large-scale deep learning in Section 5. Finally, we conclude the paper in Section 6.

2 BACKGROUND AND PRELIMINARIES

In this section we provide some background and preliminaries discussion on the subject of deep learning and exascale system architecture: two research areas of relevance to this paper.

2.1 Deep Learning

In broad terms, deep learning training algorithm can be summarized in two steps; forward propagation (forwardprop) and back propagation (backprop). Training a neural network involves multiple passes on the data in both forward and backward directions until a desire objective is achieved. In forward propagation, the training algorithm makes a forward pass of the neural network to make inference or extract underlying structure of the input data. This is typically a series of transformation implemented as matrix multiplication (and similar linear algebra) across layers in the network.

The back propagation phase involves calculating error loss (using a defined objective function) and applying an update to the neural network model parameters. Commonly used algorithm for the back propagation phase is the stochastic gradient descent (SGD) optimization algorithm. SGD algorithms (a variant shown in Algorithm 1 [2]) are inherently sequential and constitute major bottleneck to efficient parallelization of deep learning algorithm. Several research efforts have focused on how to deal with this bottleneck.

Algorithm 1 Back propagation with SGD

```
Input: Initial model parameter $\theta$, learning rate $\epsilon$
Output: Updated model parameter $\theta$
1: while stopping criterion not met do
2:     Sample a mini-batch of m training set and target examples $\mathbf{x}^1, .., \mathbf{x}^m$ and $y^1, .., y^m$
3:     Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +(1/m)\triangledown_{\theta}\sigma_i L(f(\mathbf{x}^i;\theta), \mathbf{y}^i)$
4:     Apply update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
5: end while
6: return $\theta$
```

Two modes of parallelism are typically considered in parallelizing deep learning algorithms: data parallelism and model parallelism. In model parallelism, the parameters (i.e. weights) of a single model are distributed across the computing elements. Updates to each part of the model are made by their individual computing element, respectively. In data parallelism, each mini-batch of input data is partitioned across the computing elements and processed concurrently, while the model parameters (weights) are replicated. Updates to the model are then aggregated across all samples, and distributed to each processing element.

2.2 Next-Generation HPC Systems

US Department of Energy (DOE) has projected 2017 timeframe to deliver its next-generation CORAL supercomputers. These systems will exhibit the following characteristics: high-bandwidth and low latency interconnect, high power CPUs, node-local non-volatile memory (NVRAM), tightly-coupled GPUs, and highly-optimized communication libraries, global files system with high bandwidth, and low power consumption – features that will allow HPC-enabled deep learning [31]. When properly tuned, deep learning will benefit greatly from all the unique capabilities of these future systems. For example, while hundreds (possibly thousands) of GPU accelerators speed up parallel training, high-bandwidth interconnects efficiently move parameter updates, and node-local NVRAM provides low-latency caching for massive training data sets.

3 A SCALABLE FRAMEWORK FOR PARALLEL TRAINING OF DNN

In this section we discuss the LTFB framework, highlight its benefits, and provide implementation details of the underlying algorithm.

3.1 Multi-level Tournament Voting Framework

In this work, we develop a new data parallelization framework for scalable deep neural network training that maximize the amount of data that is ingested by the training algorithm. This framework also supports hierarchical parallelism, allowing parallel trainers to exploit both data and model parallelism internally. Figure 1 shows the high-level layout and general communication structure for the

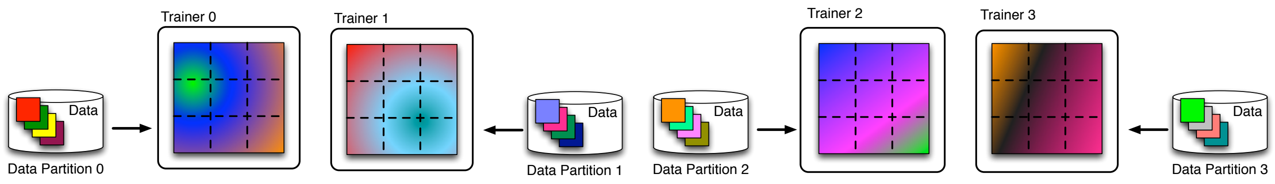


Figure 1: LTFB Architecture: training data is partitioned among p (4) trainers. Each trainer has an instance of the model and a partition of the training data set. For this example, each trainer shown is internally parallelized over 9 HPC nodes. Multi-color patterns represent the trainer's computed gradient for its weight matrices.

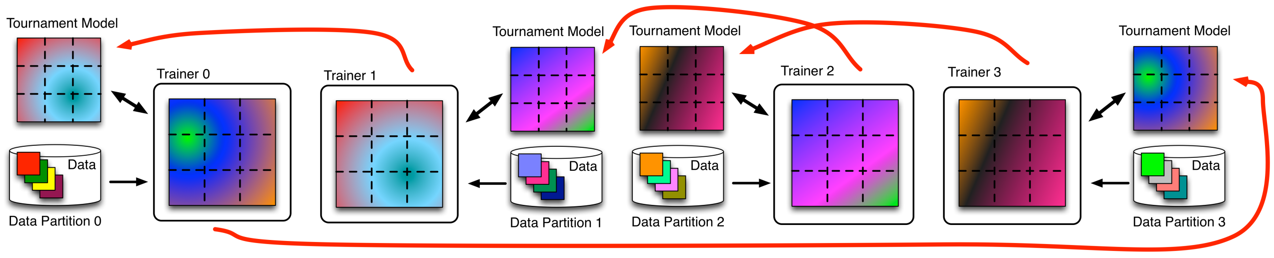


Figure 2: LTFB Architecture: models are exchanged between trainers. Each trainer evaluates the state of its current model and model received from another trainer against their private holdout tournament dataset. The winning model is kept for next round of training and loser is discarded.

LTFB framework. The main thrust of the proposed framework is to minimize the amount of synchronization required for each step of the training algorithm and to develop a mechanism for merging or combining independently trained models.

The LTFB framework is described in Algorithm 2, and illustrated Figures 1 and 2. Our multi-level tournament training algorithm creates a set of trainers, where each trainer has an instance of the model that is trained independently. Periodically, each trainer randomly selects another trainer to pair with, exchanges models, and then run a local tournament against held-out datasets. The model with the highest accuracy on the held-out dataset wins the tournament and continues training on the local dataset. There is a communication graph that manages the frequency of pair-wise communication and synchronization among trainers. The communication graph is an abstraction loosely modeled after an Erdos-Renyi graph. It controls the amount of cross section bandwidth required as well as how far each trainer's model is allow to separate from the others. A simplified version of the communication graph would be random partners or nearest partners (based on some "distance metric") or a combination of both.

The LTFB framework exhibits two major characteristics: (1) it reduces the frequency of synchronization by allowing each model to process multiple mini-batches or epochs independently; (2) it provides an approach for combining independently trained models such that each model is able to contribute some of the knowledge learned from their training set. The pair-wise exchange and knowledge gained from model sharing is crucial for faster convergence and faster time to accuracy. In addition, LTFB framework streamlines data-parallel training and hyperparameter tuning, as shown in Sections 3.2, and 3.3.

Algorithm 2 LTFB Algorithm

```
Require: training dataset $D_{train}$, tournament dataset $D_{tour}$, validation dataset $D_{val}$, number of parallel trainers $P$
1:  for all trainer $p \in P$ par do
2:      while stopping criterion not met do
3:          for $N$ epochs do
4:              Train model on local dataset $d^p_{train} \in D_{train}$
5:          end for
6:          Send locally trained model to partner
7:          Receive locally trained model from partner
8:          Evaluate both local and neighbor's models on locally hold-out tournament dataset $d^p_{tour} \in D_{tour}$
9:          Evaluate winning model on validation dataset $D_{val}$
10:     end while
11: end for
```

3.2 Data Parallelism

Data parallelism is enabled by multiple trainers having independent and parallel access to a subset of the large-scale dataset. This provides scalability that streamlines the processing of these large dataset across multiple computing nodes with local memory. It enables us to deal with much larger streams of data and larger parameter sets. This advantage is especially needed in deep learning problems with big data that does not fit on a single node.

Our framework also enables flexibility on how the data is partitioned and ingested. The amount of data per trainer is a parameter to the algorithm that could be varied by the user for specific goal. A wide spectrum of data partitioning from fully partitioned training data to fully overlap training data is possible. A fully partitioned of the dataset among processing element allow for faster processing time per epoch, while full overlap (data replication) allow for increased accuracy within a budgeted time.

3.3 Hyperparameter Tuning

Hyperparameter tuning is a necessary but tedious task in machine learning in general and deep learning in particular. Hyperparameter tuning is typically done in a trial and error fashion. It is time consuming and computationally intensive to figure out what hyperparameters work for what network architectures. It is even more tedious getting working hyperparameters for a given network to generalize for a different dataset.

Our framework helps streamlines this process by allowing diversity in each independently trained model, and minimizing the time spent training with sub-optimal parameters. To accelerate hyperparameter search, each independent model in the LTFB framework would be instantiated with different hyperparameters. When they compete at tournaments the winning model retains its metadata and hyperparameters, migrating them to the new trainer.

3.4 Implementation Details

Our implementation of LTFB is built around the Caffe deep learning framework [15]. Caffe was our choice in part because (1) its popularity among deep learning researchers and users (2) support for different deep learning network architecture and datasets and (3) its C++ implementation. In implementing our framework, we configure an MPI wrapper to run multiple instances of Caffe in separate MPI processes. Each trainer in the LTFB framework is a Caffe network as an MPI process.

Each trainer in the LTFB framework is fed (a partition of) the training and tournament data. At a predetermined point (multiple mini batches or epoch), all trainers randomly pair up with another trainer and exchange their currently trained models. After exchange, each trainer compares the two models against their holdout tournament dataset, keeps the "better" model and discards the "poor" model. The expectation is that the "better" model(s) as the tournaments progress, will spread through the system and eventually emerge as the overall "winner(s)". In this respect LTFB exhibits behavior somewhat like a genetic algorithm, with the tournament establishing the fitness function. Unlike a genetic algorithm there is no analogue for directly merging the two models.

Our framework supports heterogeneous Caffe architectures. It also leverages the richness and diversity of network architectures and advanced optimization techniques available in Caffe. By simply leveraging what Caffe provides, our framework does not require that all trainers have the same number of and dimensions of layers. This allows us to train many (different) and diverse architecture in parallel and have them compete with each other.

4 EXPERIMENTAL STUDY

4.1 Experimental Setup

4.1.1 Machine Architectures. All experiments were run on Surface and Ray clusters at Lawrence Livermore National Laboratory [21]. Surface consists of 156 compute nodes. Surface nodes are Intel (Sandy Bridge) with 16 cores and 256GB memory per node. Each node also has 2 Tesla K40 GPUs. Ray is one of early access SIERRA (CORAL) systems[20]. Ray consists of 54 IBM Power8+ computer nodes. Each compute node contains 20 CPU cores, 4 GPUs NVIDIA Tesla P100 (Pascal), and 256GB of memory. Ray has 1.6TB NVMe flash storage, 1.6TB global parallel file system and Mellanox EDR 100Gbps Infiniband network interfaces. In all of our experiments involving GPUs, 1 GPU per trainer was used. For the rest of discussion in this section, GPU and LTFB trainer or task are used interchangeably.

4.1.2 Deep Network Architecture and Datasets. CIFAR10 and ImageNet datasets were used to evaluate the LTFB framework and compared to sequential Caffe as baseline. CIFAR10 consists of 60,000 32×32 pixel color images with 40,000 of the images used for training, 10,000 as tournament holdout dataset and 10,000 images as test set. The ImageNet database consist of 1.2 million 256 × 256 pixel color training images, 64,000 tournament holdout dataset, and 50,000 validation (test) images. The deep learning task is to classify the object in each image to the 10 and 1000 classes for CIFAR10 and ImageNet test label respectively.

We leverage existing deep learning network architectures in Caffe without modification for the supervised classification tasks. The network architecture for the CIFAR10 problem was three convolution layers interspersed with RELU activations and pooling layers and then followed by two fully connected layers and a softmax layer. The GoogleNet [30] architecture was used for the ImageNet experiment. For all the experiments, the same hyper parameters (learning rate, optimizer, convolutional filter size etc) were used both for LTFB and sequential Caffe. These parameters are as set in the original Caffe implementation [15] for each benchmark.

4.2 Experimental Results

Our experiments were broadly divided into small scale and large scale experiments depending on the data size. CIFAR10 and ImageNet dataset are considered small and large scale respectively.

4.2.1 CIFAR10: Small Scale Experiments. Our first set of experiments were run on Surface cluster using the CIFAR10 dataset, and a fully partitioned data distribution. This set of experiments gave us initial empirical understanding as to the working of LTFB framework and its performance in comparison to the sequential Caffe baseline.

Figure 3 shows performance of LTFB and Caffe measured as number of iterations to accuracy. The result shows that LTFB outperforms Caffe in time to accuracy by a 3 to 4 points higher accuracy.

Figures 4 and 5 provide more insight to understand why LTFB may be better (higher accuracy) than the sequential algorithm. The underlying explanation for the better performance is that of knowledge transfer, strength, and diversity. LTFB algorithm exhibits these three attributes leading to better performance in comparison to sequential algorithm.

Figure 4 shows that without model exchange or knowledge transfer, LTFB will perform poorly and learn less than the sequential algorithm, when training with a fully partitioned data set. LTFB framework benefits from exchanging model periodically through tournament voting. Figure 5 is a testament to the strength and diversity attribute of LTFB. LTFB by design inherently favors stronger or better models. The figure shows LTFB is suboptimal if the trainers were to exchange models without tournament (that is to pick a winning model blindly or at random). The figure also show that for a four-way trainer, running different sequential algorithm four times with four different seeds (as constituted in a four-way parallel LTFB) will not achieve the same result as would have with LTFB.

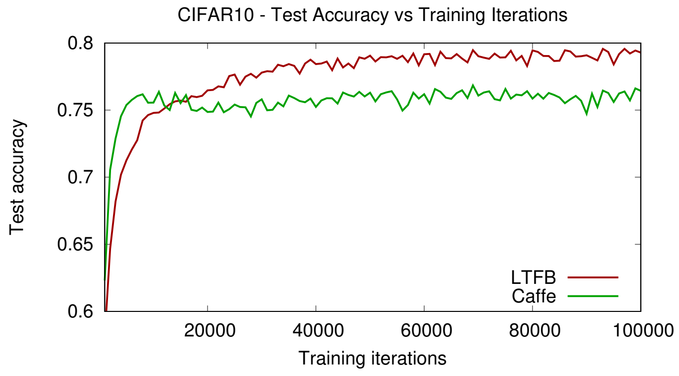


Figure 3: Caffe versus LTFB on CIFAR10 dataset, LTFB outperforms Caffe in time to accuracy by a 3 to 4 points higher accuracy.

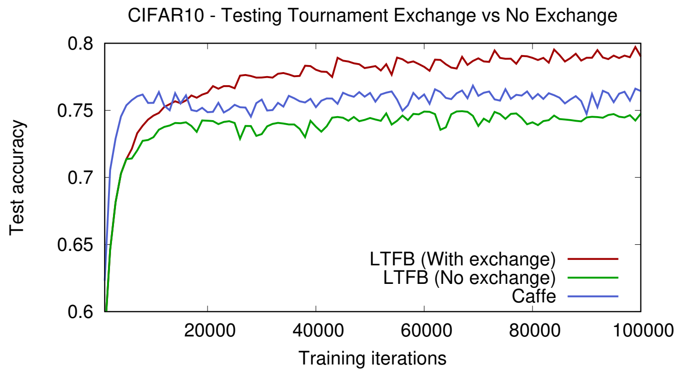


Figure 4: Comparing LTFB with and without model exchange or tournament voting. Results shows that knowledge sharing from tournament voting is critical to achieving higher accuracy.

4.2.2 ImageNet: Large Scale Experiments. As a follow up to CIFAR10 experiment, we set up large-scale experiments on all of 1.2 million ImageNet dataset to run on Ray. Our first experiments with large ImageNet dataset explores how varying data partition impacts iterations to accuracy. We explored three level of data partitioning: (1) non-overlapping partition in which the full data set is

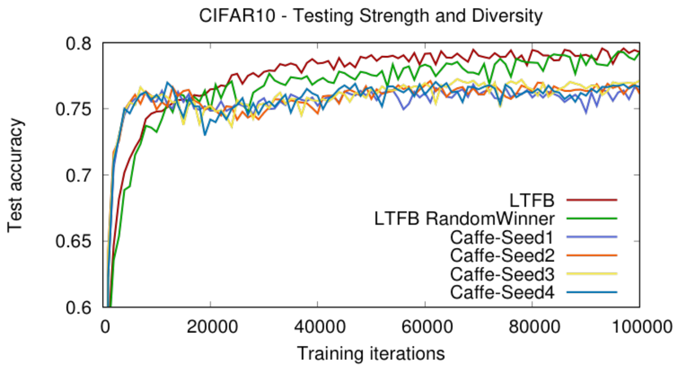


Figure 5: Results show that LTFB is biased for strength and diversity. We test for strength bias by picking random winner instead of trainer with better model (higher accuracy). We test for diversity bias by running Caffe with seeds that are set of 4 seeds used in LTFB.

wholly partitioned among the processing elements without overlap in training data processed by each process (2) data partitioning with some amount of overlap in training data for each process and (3) full replication of data among processing elements. Figures 6 and 7 show the results of varying data partition for fixed 4 trainers and 16 trainers respectively. While LTFB outperforms sequential Caffe baseline in all scenarios, the results show that having more training data samples outperforms a reduced data sample. Note that as data is replicated across trainers, data augmentation techniques at each trainer effectively expands the size of the data set.

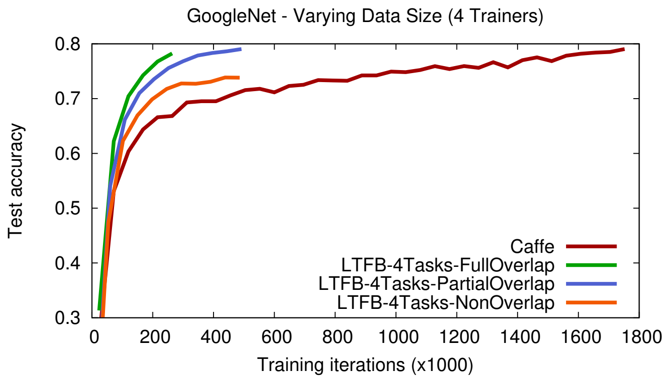


Figure 6: Caffe versus LTFB on ImageNet dataset, varying data overlap for LTFB with 4 trainers (GPUs), more data improve iterations to accuracy

Given the results from varying the amount of training samples processed by each GPU at a fixed number of GPUs or LTFB tasks,

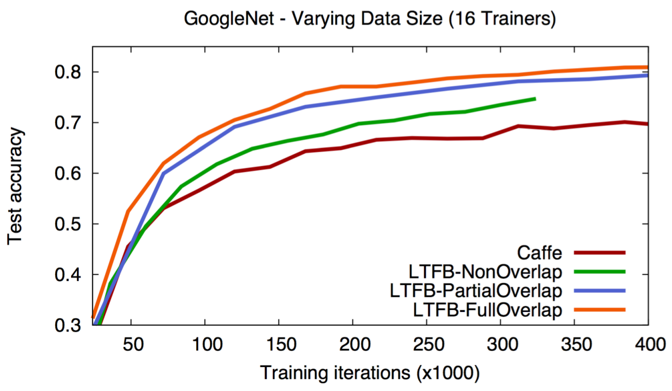


Figure 7: Caffe versus LTFB on ImageNet dataset, varying data overlap for LTFB with 16 trainers (GPUs), more data improve iterations to accuracy

we carried out set of scaling experiments with partial overlap and fully data replication schemes. A reasonable use case for the full data replication scaling experiment involves improving training accuracy within a time budget by adding more GPUs and increasing the available data augmentation. This scenario fits the HPC Big Data paradigm in which “infinite stream" of data is a possibility. Figures 8 and 9 show results of these experiments. The results show that increased concurrency in LTFB pays off especially at higher accuracy levels. By adding more GPUs we increase the diversity in the model set, thus increasing exploration of the solution space, and amplifying the effective size of the data set. Moreso, by biasing toward stronger models, every round of tournament voting explores multiple paths from the strongest models. Note that missing columns for 4 and 16 tasks in Figure 9 are results not available (could not reach the next accuracy point) within alloted time.

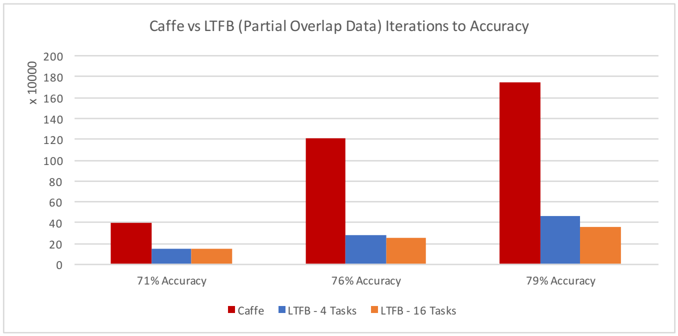


Figure 8: Scaling of Caffe versus LTFB on partial overlapping ImageNet dataset, improve iterations to accuracy by adding more GPUs.

4.2.3 *Hyperparameter Tuning.* Previous experiments demonstrate that increasing concurrency by adding more trainers improves time and reduces iterations to a higher accuracy. Another

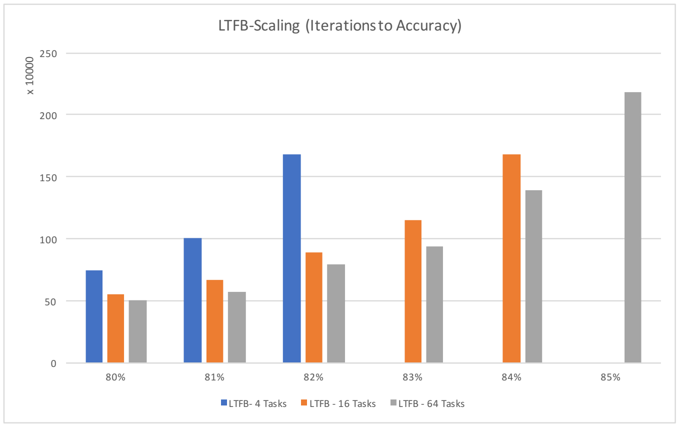


Figure 9: Scaling of LTFB on replicated ImageNet dataset. Increase concurrency in LTFB pays off at higher accuracy levels, every round of voting explores multiple paths from strongest models.

well known technique for improving iterations to accuracy is hyperparameter tuning. The LTFB framework streamlines hyperparameter exploration by allowing multiple trainers and models with different hyperparameters to run concurrently and compete periodically to dynamically select the best hyperparameter for the current state of the training.

We conduct an experiment on CIFAR10 dataset using the same setup as the previously discussed experiments but varying base learning rate among the 4 LTFB trainers. We fixed all the hyperparameter provided by Caffe [15] for CIFAR10 image classification task, and only vary the base learning rate for each trainer. Base learning rate of each trainer was computed as a multiplier of its MPI Rank. The original base learning rate of sequential Caffe was fixed at 0.001 [15], thus the learning rate for each of four LTFB trainer was computed as:

$$lr = 0.001 * (MPIRank + 1)$$

The four LTFB trainers compete at intervals with the weak model taking both the model and solver state of the wining model. The experimental result is shown in Figure 10. We observed that LTFB with varying base learning rate outperforms both sequential Caffe and LTFB with fixed learning rate. The result shows that LTFB benefits from hyperparameter exploration to improve iterations to accuracy.

The hyperparameter experiment on CIFAR10 dataset demonstrate that higher accuracy could be achieved with further hyperparameters tuning of a given "mini-batch/learning rate pair" that is known to work. There are other scenarios in which the users have to explore (sometime in a trial by error fashion) what should be the appropriate hyperparameter for a given network architecture. We conduct a new experiment to demonstrate how the LTFB framework will address this problem. A new experiment on ImageNet was conducted to explore "unknown mini-batch/learning rate pair" scenario. A new mini-batch size of 64 and a set of 16 base learning rates ranging from 0.005 to 0.085 (i.e. 0.005, 0.01, 0.015,0.02,. . . ,0.085) were chosen. Each of the 16 base learning rates was given to an

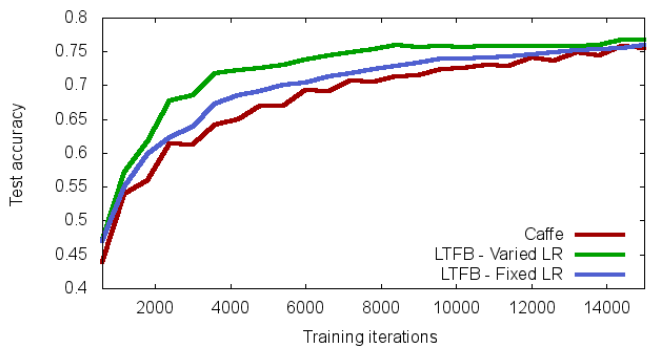
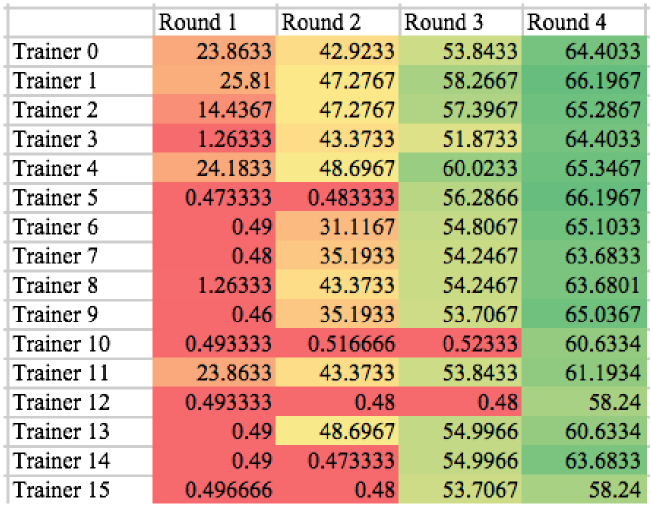


Figure 10: Caffe versus LTFB CIFAR10 dataset with fixed and varying learning rate

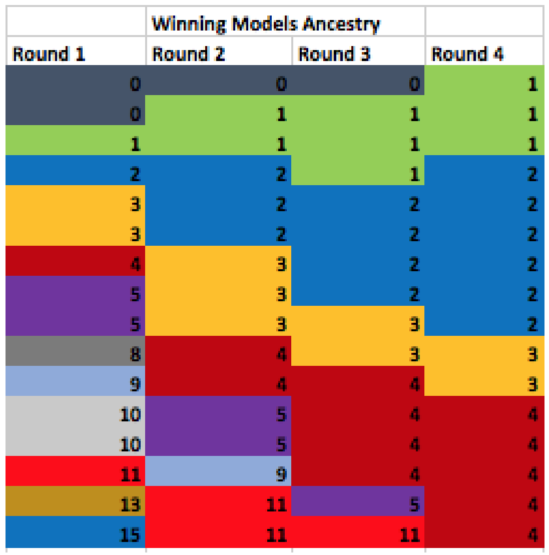
LTFB trainer, the median of the set (0.04) was given to the sequential Caffe (a blind or random selection for sequential Caffe, in practice, users will try out multiple values). What should be appropriate base learning rate is unknown beforehand. The sequential Caffe model did not learn for the given mini-batch size/base learning rate pair; 0.5% top-5 average accuracy during the allotted time of 4 epochs, whereas LTFB made substantial progress reaching 66% top-5 accuracy within the same time period. As shown in Figure 11, LTFB starts out with almost 50% "bad" models but then proceed to acquire "better" models as the training and tournament exchanges proceed. Set of trainers that start poorly (trainers with bad learning rate) improve over time due to influences from stronger models.



	Round 1	Round 2	Round 3	Round 4
Trainer 0	23.8633	42.9233	53.8433	64.4033
Trainer 1	25.81	47.2767	58.2667	66.1967
Trainer 2	14.4367	47.2767	57.3967	65.2867
Trainer 3	1.26333	43.3733	51.8733	64.4033
Trainer 4	24.1833	48.6967	60.0233	65.3467
Trainer 5	0.473333	0.483333	56.2866	66.1967
Trainer 6	0.49	31.1167	54.8067	65.1033
Trainer 7	0.48	35.1933	54.2467	63.6833
Trainer 8	1.26333	43.3733	54.2467	63.6801
Trainer 9	0.46	35.1933	53.7067	65.0367
Trainer 10	0.493333	0.516666	0.52333	60.6334
Trainer 11	23.8633	43.3733	53.8433	61.1934
Trainer 12	0.493333	0.48	0.48	58.24
Trainer 13	0.49	48.6967	54.9966	60.6334
Trainer 14	0.49	0.473333	54.9966	63.6833
Trainer 15	0.496666	0.48	53.7067	58.24

Figure 11: Heatmap plot of hyperparameter tuning in scenario with mini-batch size/learning rate mismatch. LTFB tournament help to dynamically select the best (appropriate) hyperparameter for trainers that start out with bad learning rate.

Figure 12 is a visual representation of ancestry history of winning models sorted and grouped by model *id* and color. It is a count of the number of times each model or its descendant won the tournament at each round. The figure shows that lower rank trainers, that is, trainers[0 – 4], have appropriate base learning rate resulting in stronger models. Models produced by these trainers are are picked up by the other trainers. trainer4 in particular appears to be the "overall best" up to the point of evaluation, having won all of the four tournaments and models originating from it having the best overall validation accuracy. From Figure 11, trainer1 and trainer5 have models with highest validation accuracies, these models originated from trainer4 as shown in Figure 13.



	Winning Models Ancestry		
Round 1	Round 2	Round 3	Round 4
0	0	0	1
0	1	1	1
1	1	1	1
2	2	1	2
3	2	2	2
3	2	2	2
4	3	2	2
5	3	2	2
5	3	3	2
8	4	3	3
9	4	4	3
10	5	4	4
10	5	4	4
11	9	4	4
13	11	5	4
15	11	11	4

Figure 12: LTFB tournament history, taken into consideration the ancestry of winning model at each tournament round. Lower rank trainers produce (directly or otherwise) most of the winning models.

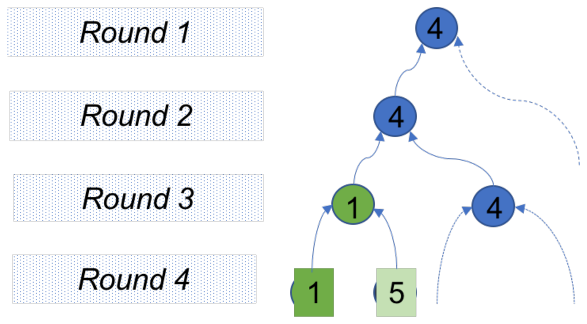


Figure 13: Partial ancestry trace of winning models. Trainers 1 and 5 winning models can be traced to trainer 4.

5 RELATED WORK

Advances in data acquistion, algorithms and computing power have prompted renewed interest in large-scale deep learning. It has been said that the long training time is a key challenge at the root of the development of new DNN architectures, and that accelerating DNN training solutions would solve a major pain point for many organizations with interest in deep learning [14]. However, most of existing work in parallelizing deep learning have focused on distributed clusters and heterogeneous cloud computing environments [1],[29],[22],[9]. Our work address the need for scaling deep learning training in HPC environment. HPC environments are unique in the nature of computing, memory and network bandwidth resources. Our work, while focusing on HPC environments, is not bounded to it. As an algorithmic abstraction, it could extend easily to other computing environments.

Closely related research work on large-scale data-parallel deep learning is data parallelism with mini-batch summation coordination through centralized parameter server or reduction tree, synchronously or asynchronously. Hogwild [27], Tensorflow[1], FireCaffe [14] and their variants [5] are examples of framework that exhibit data parallelism with mini-batch summation. Scalability and parallelism using the mini-batch summation approach is bounded by aggregate mini-batch size [11]. Other research efforts found that aggregate mini-batch size could only increase to a modest extent until it negatively affects training or testing accuracy or gets less than linear returns in terms of optimization performance [2, 11, 17].

Gradient quantization as a technique to reduce data communication in deep neural network training have been explored. Selected work in this area include: one-bit quantization [28], threshold quantization [29], and adaptive quantization [10]. We consider research efforts in mini-batch summation and gradient quantization as complementary to the approach describe in this paper. Each model within the LTFB framework could be trained leveraging techniques such as mini-batch summation with multiple GPUs and/or gradient quantization.

Researchers have also explored hyperparameter tuning and model ensemble as training techniques for DNN. Young et.al. [32] proposed an HPC-based framework for optimizing hyper parameters of DNN using genetic algorithm. Unlike LTFB, the proposed technique employed the master-slave architecture. Gene selection, crossover, and mutation are done by the single master node, while the slave nodes evaluate the fitness functions of the hyperparameters received from the master node. The technique showed some promising qualitative results on CIFAR10 dataset but lacked scalability evaluation. Similarly, Breuel and Shafait [4] described a simple genetic algorithm aimed at tuning learning and network size by combining ensemble of neural network models that are trained in parallel with different rates and network sizes. The propose technique, while similar to our work in regards to hyperparameter tuning, lacks in-depth evaluation. Experimental evaluation was done on a small scale MNIST data set. LTFB provide a unified framework for large-scale DNN training with support that extends beyond hyperparameter tuning. Lastly, whereas LTFB replaces “weaker" models with “stronger" models through tournament voting, some recent work have explored other ensemble techniques such as model averaging [33] and model combination through parameter sharing and ensemble-aware loss training [24]. LTFB could leverage some of these techniques to increase diversity of ensemble models and reduce model sizes while retaining the scalability of pair-wise tournament voting.

6 CONCLUSIONS AND FUTURE WORK

We propose LTFB as a scalable parallel training algorithm that combines gradient-less optimizations to overcome the limitations of gradient descent algorithms scaling up parallel training algorithms to HPC class system. Preliminary scaling evaluation on image classification tasks show promising results for further exploration. Future work in this regard will include extension of our framework to extremely large (infinite streams of scientific) data sets, typically not attempted in industrial or academic domains. In the pursuit of this goal, we will fine-tune LTFB framework by exploring different ways to optimize data ingestion, computation, inter- and intra-node communication. Algorithmic fine-tuning will include topics such as advanced diversity in individual models and varying the mini-batch sizes as the tournaments progress.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-700919). This work was partially funded under ARL contract L20263 and LDRD 17-SI-003. Experiments were performed at the Livermore Computing facility resources.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[2] Ian Goodfellow Yoshua Bengio and Aaron Courville. 2016. Deep Learning. (2016). http://www.deeplearningbook.org MIT Press.

[3] Yoshua Bengio. 2012. Deep learning of representations for unsupervised and transfer learning. *Unsupervised and Transfer Learning Challenges in Machine Learning* 7 (2012), 19.

[4] Thomas Breuel and Faisal Shafait. 2010. AutoMLP: Simple, Effective, Fully Automated Learning Rate and Size Adjustment. In *The Learning Workshop*. Online. Extended Abstract.

[5] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. 2016. Revisiting Distributed Synchronous SGD. *CoRR* abs/1604.00981 (2016). http://arxiv.org/abs/1604.00981

[6] M. Cho, U. Finkler, S. Kumar, D. Kung, V. Saxena, and D. Sreedhar. 2017. PowerAI DDL. *ArXiv e-prints* (Aug. 2017). arXiv:cs.DC/1708.02188

[7] Dan Claudiu Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. 2013. Mitosis Detection in Breast Cancer Histology Images with Deep Neural Networks. In *Proceedings of the 14th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*.

[8] Adam Coates, Andrew Y Ng, and Honglak Lee. 2011. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*. 215–223.

[9] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*. 1223–1231.

[10] Nikoli Dryden, Tim Moon, Sam Ade Jacobs, and Brian Van Essen. 2016. Communication Quantization for Data-parallel Training of Deep Neural Networks. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, 8.

[11] Priya Goyal, Piotr Dollar, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* (2017).

[12] Awni Y. Hannun, Carl Case, Jared Casper, Bryan C. Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates,

and Andrew Y. Ng. 2014. Deep Speech: Scaling up end-to-end speech recognition. *CoRR* abs/1412.5567 (2014). http://arxiv.org/abs/1412.5567

[13] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Ki ngsbury. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine* 29, 6 (November 2012), 82–97.

[14] Forrest N Iandola, Khalid Ashraf, Mattthew W Moskewicz, and Kurt Keutzer. 2015. FireCaffe: Near-linear acceleration of deep neural network training on compute clusters. *arXiv preprint arXiv:1511.00175* (2015).

[15] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[16] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale Video Classification with Convolutional Neural Networks. In *CVPR*.

[17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv preprint arXiv:1609.04836* (2016).

[18] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n. d.]. CIFAR-10 (Canadian Institute for Advanced Research). ([n. d.]). http://www.cs.toronto.edu/~kriz/cifar.html

[19] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*. 1097–1105.

[20] Lawrence Livermore National Laboratory. 2016. Sierra. https://asc.llnl.gov/coral-info. (2016).

[21] Lawrence Livermore National Laboratory. 2017. Livermore Computing. https://hpc.llnl.gov/hardware/platforms. (2017).

[22] Quoc V. Le, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. 2012. Building high-level features using large scale unsupervised learning. In *In International Conference on Machine Learning, 2012. 103*.

[23] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 609–616.

[24] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David J. Crandall, and Dhruv Batra. 2015. Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks. *arXiv* (2015). http://arxiv.org/abs/1511.06314

[25] Karl Ni, Roger Pearce, Kofi Boakye, Brian Van Essen, Damian Borth, Barry Chen, and Eric Wang. 2015. Large-Scale Deep Learning on the YFCC100M Dataset. *arXiv preprint arXiv:1502.03409* (2015).

[26] Rajat Raina, Anand Madhavan, and Andrew Y Ng. 2009. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 873–880.

[27] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*. 693–701.

[28] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs.. In *INTERSPEECH*. 1058–1062.

[29] Nikko Strom. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In *INTERSPEECH*, Vol. 7. 10.

[30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*. http://arxiv.org/abs/1409.4842

[31] Brian Van Essen, Hyojin Kim, Roger Pearce, Kofi Boakye, and Barry Chen. 2015. LBANN: Livermore big artificial neural network HPC toolkit. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, 5.

[32] Steven R. Young, Derek C. Rose, Thomas P. Karnowski, Seung-Hwan Lim, and Robert M. Patton. 2015. Optimizing Deep Learning Hyper-parameters Through an Evolutionary Algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments (MLHPC ’15)*. ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.1145/2834892.2834896

[33] Sixin Zhang, Anna Choromanska, and Yann LeCun. 2014. Deep learning with Elastic Averaging SGD. *CoRR* abs/1412.6651 (2014). http://arxiv.org/abs/1412.6651