LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-677443

LBANN: Livermore Big Artificial Neural Network HPC Toolkit

B. Van Essen, H. Kim, R. Pearce, K. Boakye, B. Chen

September 23, 2015

Machine Learning in HPC Environments
Austin, TX, United States
November 15, 2015 through November 15, 2015

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

LBANN: Livermore Big Artificial Neural Network HPC Toolkit

Brian Van Essen Hyojin Kim Roger Pearce Kofi Boakye Barry Chen
Lawrence Livermore National Laboratory
Livermore, CA 94551
{vanessen1, kim63, pearce7, boakye1, chen52}@llnl.gov

ABSTRACT

Recent successes of deep learning have been largely driven by the ability to train large models on vast amounts of data. We believe that High Performance Computing (HPC) will play an increasingly important role in helping deep learning achieve the next level of innovation fueled by neural network models that are orders of magnitude larger and trained on commensurately more training data. We are targeting the unique capabilities of both current and upcoming HPC systems to train massive neural networks and are developing the Livermore Big Artificial Neural Network (LBANN) toolkit to exploit both model and data parallelism optimized for large scale HPC resources. This paper presents our preliminary results in scaling the size of model that can be trained with the LBANN toolkit.

1. INTRODUCTION

Deep learning systems have extended state-of-the-art performance on a broad range of applications including unsupervised feature learning [13], object recognition in images [11, 18], speech recognition [8, 2, 16, 7], and video action classification [10]. These recent successes in deep learning have been enabled by the ability to train deep and/or wide neural networks with many parameters on vast amounts of training data. From a statistical machine learning perspective, large and deep models make it possible to achieve very low bias, while the large amounts of training data serve to help minimize the variance. Other empirical studies have further corroborated the notion that bigger models trained on larger datasets can drastically improve classification accuracy [5, 12].

For our work, we are particularly motivated by the potential of deep unsupervised feature learning for automatically discovering useful features from vast amounts of unlabeled data. As new scientific instruments and sensors are developed and begin collecting data at rates that exceed the ability of humans to hand-annotate, unsupervised feature learning becomes more important. Furthermore, since such data may be in forms other than images, having flexible "universal function estimators" will become equally important. Hornik [9] proved "that standard [fully-connected] multilayer feed-forward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available."

For these reasons, we are developing the Livermore Big Artificial Neural Network (LBANN) toolkit for accelerating the training of large fully-connected neural networks. The LBANN toolkit takes advantage of the unique capabilities of modern HPC systems to enable efficiently scalable neural network training. LBANN uses the Elemental [15] high-performance, parallel, distributed matrix library for efficiently distributing the matrix-matrix and matrix-vector computations necessary for training neural networks. This paper describes our implementation of LBANN and demonstrates LBANN's strong and weak scaling efficiency on an autoencoding task.

2. RELATED WORK

The benefits of large models for both supervised and unsupervised learning tasks have prompted other notable efforts in large-scale neural network training. The "DistBelief" framework and accompanying "DownPour" solver, presented in [13, 6], enables model- and data-parallelism as well as asynchronous stochastic gradient descent (SGD). Model parallelism is achieved both within a machine via multithreading and across machines via message passing. Data parallelism is supported with multiple replicas of a model used to optimize a single objective. The authors demonstrate training of networks with over a billion parameters on a cluster with 1,000 machines (16,000 cores) in a few days.

More recent work by Coates et al. [4] also bears similarity to our efforts and has paved the path for large-scale unsupervised neural network training with HPC resources. Here the authors present a large-scale model-parallel training system for deep neural networks as developed for commercial off-the-shelf (COTS) HPC computing infrastructure. They use a cluster of GPU servers with Infiniband interconnects and MPI to train 1 billion parameter networks on just 3 machines (12 GPUs) in a couple of days. They also demonstrate scaling to networks with over 11 billion parameters trained using 16 machines (64 GPUs).

This work was extended by us in [14] to demonstrate even larger scaling. Using a similar framework, we have trained

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MLHPC '15 Austin, Texas USA
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

the largest unsupervised neural network on 99.7 million images from the YFCC100M dataset [19], effectively repeating the work in [4] with 10× more training data and 1.5× more network parameters.

3. AUTOENCODER MODELS

For most machine learning tasks, the choice of input feature representation is as important (and sometimes more important) as the learning algorithm used. For this reason, much effort has gone into discovering or designing good features in a number of application domains, including audio, video, and images. Recent work in unsupervised feature learning [13] has demonstrated the benefit of autoencoder neural networks for learning good feature representations, in particular for the case of images. An autoencoder seeks to learn an efficient, distributed representation for a set of data by first defining a reconstruction $r(x) = g(h(x))$ of some input x from the composition of an encoder, $h(\cdot)$, and a decoder, $g(\cdot)$. In its neural network construction, $r(x)$ is the output of a one-hidden layer network with $h(x)$ being the non-linear output of the hidden layer. Training proceeds via standard back-propagation to minimize the average of the reconstruction errors, $L(r(x), x)) = ||r(x) - x||^2$. The learned features of one autoencoder can be used as the input for another autoencoder, as part of a procedure referred to as *greedy layer-wise training* [3]. This results in a stacked autoencoder that can learn highly nonlinear transformations for hierarchical—and often interpretable—feature representations (e.g., edges, curves, and faces in image data).

The use of autoencoders is appealing for multiple reasons. First, as an unsupervised approach, autoencoders do not rely on labeled data, which is typically in short supply in the specific application areas of interest to us. The complexity of our data and restrictions to access often make large-scale labeling approaches such as Amazon Mechanical Turk unfeasible. Second, the interpretability of the learned features helps in the interaction between the machine learning system and the human analyst, who is often a subject matter expert, but not an experienced machine learning practitioner. Lastly, the back-propagation training of autoencoders provides a very efficient way to learn these unsupervised feature representations.

4. LBANN

We are developing the Livermore Big Artificial Neural Network (LBANN) Toolkit to train very large models on very large data sets. It is a distributed memory algorithm that is optimized for the distinct characteristics of high performance computing systems. The toolkit is designed as a MPI+Threads framework, exploiting distributed processing and communication, as well as node-local thread-level parallelism through its BLAS library. The training algorithm's current implementation is primarily optimized for exploiting model parallelism, with very large distributed neural network models, and modest data parallelism. The key characteristics of upcoming HPC systems are:

- low latency interconnect
- high power CPUs
- node-local storage
- tightly-coupled GPUs
- highly-optimized communication libraries

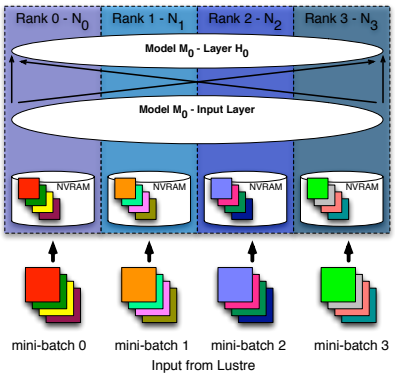
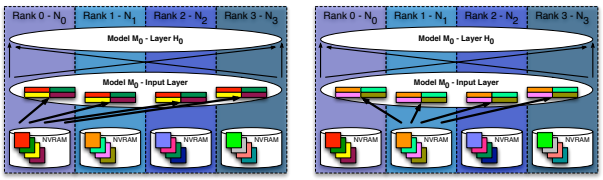


Figure 1: LBANN Distributed neural network model with parallel fetch from Lustre PFS

These characteristics make scientific HPC systems well suited for large scale, distributed mathematics that allow us to parallelize the training of a single model over many nodes. Another characteristic of our scientific supercomputing environment that impacts distributed algorithm design is dedicated job scheduling. Exclusive access to our nodes, as well as highly reliable links allows us to avoid many of the implementation overheads that were necessary in prior work, such as the Google DownPour system [6]. Specifically, multi-tenant, scale-out compute environments require significant effort and code devoted to recovering individual tasks.



(a) Processing mini-batch 0 (b) Processing mini-batch 1

Figure 2: Distributing mini-batches

4.1 Distributed Matrices + Model Parallelism

The core data structure for LBANN neural network models is the distributed matrix structure that allows the neural network model to span multiple HPC nodes. LBANN is built upon the open source Elemental parallel, distributed, matrix math library by Poulson et al.[15]. Elemental provides a C++ framework for controlling data distribution and uses MPI for parallelization and communication. This framework provides us with an efficient mechanism for exploiting model parallelism when training the neural network. As the model is distributed across the system (Figure 1), the aggregate CPU and memory resources used to train a single model increase. Section 5.2 provides preliminary results for strong scaling the model.

4.2 Data Parallelism

The new generation of data-intensive HPC systems are incorporating node-local Non-Volatile Random Access Memory (NVRAM) into the compute node architecture. We use the node-local NVRAM to provide local data staging, improved bandwidth, and reduced latency for processing the input data set. Figure 1 illustrates how data is loaded in parallel from the parallel file system to node-local NVRAM, and Figures 2(a) and 2(b) illustrates how data is iteratively served from NVRAM into the model for each mini-batch.

Section 5.3 and Figure 8 compares the performance of using node-local NVRAM versus directly fetching hot data from the parallel file system, when it is cached in OST and router nodes.

4.3 Challenges to scaling

Seide et al. [17] performed both an analytical and empirical analysis of model and data parallelism for dense, deep neural networks targeting speech encoding. They observed significant challenges in scaling their training efficiently beyond a small number of GPUs within a single host. One primary challenge noted was the limited bandwidth within the host, compared to the arithmetic intensity of the NVIDIA GPUs. Additionally, they note that while large scale CPU-only systems will have more favorable arithmetic intensity to bandwidth ratios, their relative performance to GPUs makes them unfavorable. It is important to recognize that the context of this evaluation was a relatively small model, with only 46M parameters and an estimated size of 0.17GB (assuming single precision floating point representation). As we experiment with model sizes that are orders of magnitude larger than this, the scaling characteristics are much more favorable, especially for future HPC architectures.

5. EXPERIMENTS

To test the scaling characteristics of LBANN over several axis we trained a single layer autoencoder using part of the ImageNet 2012 (ILSVRC2012) training data set. The goal for our experiments was not to achieve the lowest reconstruction nor classification performance. Rather the goal was to test the strong and weak scaling properties of LBANN. Specifically, most of these experiments were designed to test the amount of wall clock time required to finish a fixed amount of training. We conducted these tests on the LLNL Catalyst HPC system [1], which consists of 324 nodes with 24 Xeon EP X5660 cores, 128 GB DRAM, and 800GB of node-local NVRAM. Catalyst has a aggregate bandwidth of 24-32 GB/s to a Lustre parallel file system.

The neural network topology that we used was $\sim 197K - X - \sim 197K$, where X is the number of neurons in a fully connected hidden layer. The input and output layers are full sized images from ILSVRC2012 and so are actually $256 \times 256 \times 3 = 196,608$. Our test suite includes $50K$, $100K$, $400K$ neuron models, which corresponds to approximately $9.8B$, $19.7B$, $78.6B$ parameters. Using double precision floating point values, the total size for each model is proportional to the size of the autoencoder's weight matrix ($73GB$, $147GB$, $293GB$), for layer sizes $50K$, $100K$, $400K$, respectively. For each test we specify how many MPI ranks (processes) execute on each node, and the remaining available cores are used by the multi-threaded BLAS library. In all cases, LBANN effectively uses the 48 Hyper-Threaded cores per node. Unless otherwise specified all tests had an Elemental block size of 256, 12 ranks per node, and was trained on the first 1200 images of ILSVRC2012.

5.1 Single-layer autoencoders

Unsupervised feature learning relies on greedy, layer-wise training to teach each layer of the autoencoder how to best represent the input data. Quality of the autoencoder's learning is measured with reconstruction error, comparing the original image to the output of the autoencoder layer. Traditionally, for image processing tasks, early layers use local receptive fields that limit a neuron's scope of view and teach the layer to learn Gabor filters. When using a fully connected, dense layer, neurons are able to learn much more complex representations of the image. Figure 3 shows how well single layer autoencoders are able to reconstruct a training image after 100 epochs when using hidden layers of 10K, 50K, and 400K neurons. Note that the mini batch size is 40 images and the learning rate is 0.0001. Figure 4 plots the reconstruction error versus the number of training epochs.

(a) 10K (b) 50K (c) 400K (d) Original

Figure 3: Reconstructed images using a single-layer autoencoder with X neurons after 100 epochs.

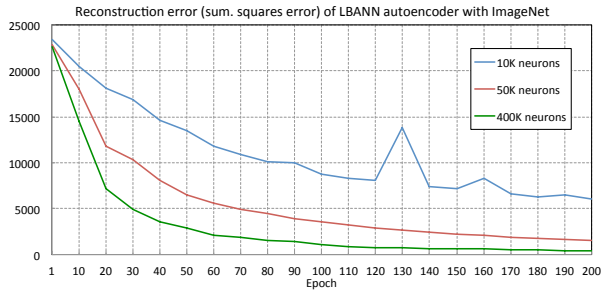


Figure 4: Reconstruction error of a single-layer autoencoder with ImageNet.

5.2 Exploiting model parallelism

As a single model is distributed over multiple nodes there are two ways to improve performance, solve a fixed sized model faster (strong scaling) and solve a larger problem in "fixed" amount of time (weak scaling). To explore how well LBANN strong scales we used a single layer autoencoder with 50K hidden units. The tests were run with 8 - 128 nodes, 12 ranks per node, and mini-batch sizes from 8 - 1024 images. Figure 5 shows the runtime on the y-axis as the number of nodes is increased. LBANN is able to achieve good strong scaling for the 50K neuron model from 8 - 32 nodes, but improvements taper off at 64 to 128 nodes as I/O begins to dominate the communication/computation performance tradeoff. As part of future studies we will profile the performance of LBANN with even larger models and more nodes.

The next experiment demonstrates how LBANN weak scales, solving a larger problem with more resources. For this experiment we doubled the size of the of the model and number of nodes at each point tested. Ideally, the time to train should remain constant as the problem size increases. Figure 6 shows that the performance is relatively flat, indicating efficient weak scaling.

5.3 Exploiting data parallelism

To process larger data sets, it is necessary to ingest and train on data faster. Currently, LBANN uses two techniques that provide modest amounts of data parallelism: large mini-batches, and parallel data movement. Increasing the mini-batch has two effects on performance, first it

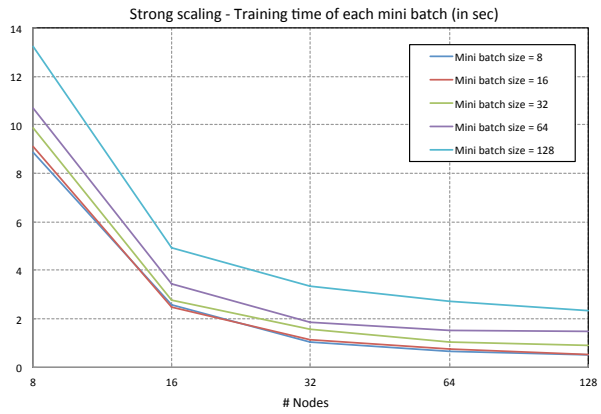


Figure 5: Strong scaling 50K neuron hidden layer: per mini-batch training time vs. number of nodes.

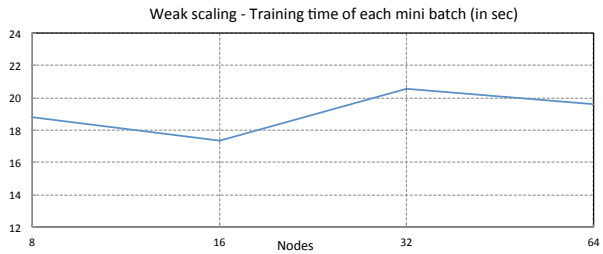


Figure 6: Weak scaling the number of neurons from 50K to 400K on 8 nodes to 64 nodes, respectively.

allows more data to be processed in parallel, and second it reduces the number of synchronization steps. Increasing the amount of work is important for this kind of distributed algorithm because it improves the efficiency of the blocked matrix multiplications. Reducing the number of synchronization steps helps minimize the amount of communication. Figure 7 shows how the performance improves as we increase the size of the mini-batch.

The second part of the experiment is to parallelize the data movement. LBANN is designed so that each node handles a single mini-batch: reading it from NVRAM or parallel file system (PFS), copying it into local memory, and then distributing it to all of the nodes in the job. LBANN parallelize the most time consuming part of this sequence, reading the mini-batch from NVRAM or PFS. As many MPI ranks as possible in the LBANN job will read in mini-batches concurrently and then distribute them round-robin as indicated in Figure 2. Figure 8 illustrates how the I/O time is effectively amortized away as more ranks are allowed to fetch data in parallel. Tests were conducted with the training data residing in node-local NVRAM and read directly from the Lustre parallel file system (PFS). For the node-local NVRAM, the page cache was flushed between each test; however, for the Lustre PFS the data was being cached in DRAM within the Lustre system.

5.4 Tuning Elemental Library

The last set of experiments vary the block size of the distributed matrix within the Elemental library and the number of ranks per node versus the amount of thread-level parallelism available to the Intel BLAS library. The first experiment, shown in Figure 9 plots the runtime versus Elemental libraries' block size. Note that for the smallest block size (32) the performance is about 11% slower than the average of the larger block sizes. For our test neural network of 400K hidden units, there is not significant advantage to varying the block size beyond 64. Further exploration of this parameter will be the subject of future work.

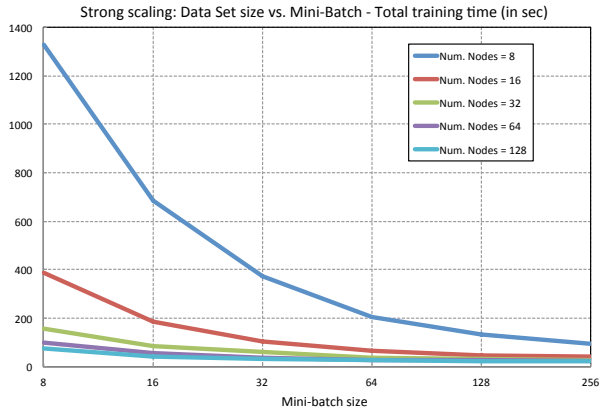


Figure 7: Strong scaling mini-batch size to exploit data parallelism (50K neuron hidden layer): total training time vs. size of mini-batch.

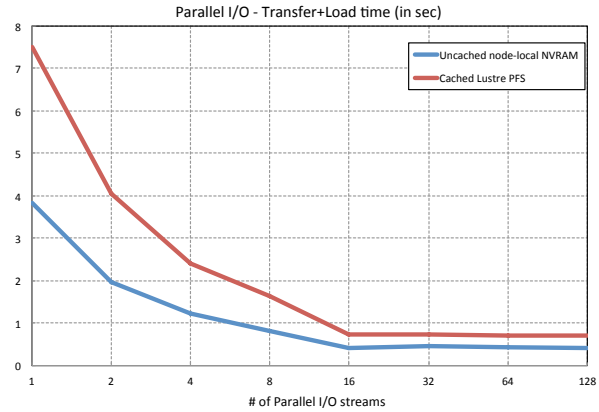


Figure 8: # parallel I/O streams fetching each mini-batch versus the total I/O time to load all images into memory. All tests were executed on 32 nodes and had a hidden layer with 50K neurons.

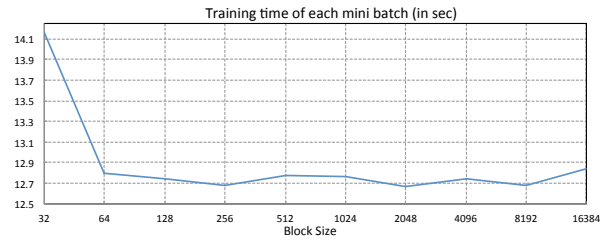


Figure 9: Tuning Elemental's distributed block size.

While the Elemental library package is parallelized via MPI, the Intel BLAS library that we use for on-node matrix manipulation is multi-threaded. As a result, LBANN is an MPI+Threads implementation. This last experiment trades off the balance of independent ranks per node versus available threads. As noted earlier there are 24 cores per node

and 48 threads available with Hyper-Threading. Figure 10 shows that as long as the number of ranks per node evenly divides the number of Hyper-Threaded cores, then LBANN achieves good utilization of all of the compute resources. Only the case with 18 ranks per node performs significantly worse; its runtime is 32% higher than the average runtime for the other configurations.

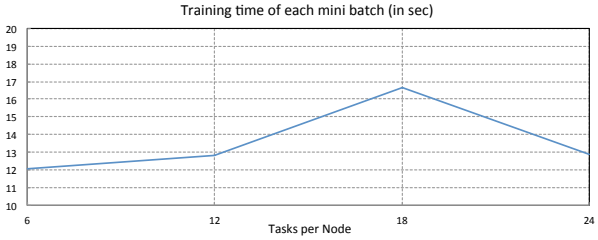


Figure 10: MPI ranks per node versus the number of threads available to Intel BLAS library.

6. CONCLUSIONS

We are developing the Livermore Big Artificial Neural Network toolkit to provide a vehicle for training very large Deep Neural Network models on very large data sets. We have targeted HPC systems, building on the Elemental library for distributed dense linear algebra. Our first steps on this project have been to add support for model parallel training of very large autoencoders. In this work we have demonstrated that the fundamental framework for LBANN provides a scalable platform, providing both good strong and weak scaling. Additionally, we have performed preliminary exploration of parallel I/O techniques and tuning the Elemental and Intel BLAS libraries. Future work will include continuing our scaling studies to larger models, and deeper models. Additionally, we are currently working on support for additional network layers, supervised training, and integrating support for distributed GPUs.

7. REFERENCES

[1] Catalyst at LLNL. https://www.llnl.gov/news/lawrence-livermore-intel-cray-produce-big-data-machine-serve-catalyst-next-generation-hpc.

[2] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, October 2014.

[3] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[4] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew. Deep learning with cots hpc systems. In *Proceedings of the 30th international conference on machine learning*, pages 1337–1345, 2013.

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

[6] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[7] A. Y. Hannun, C. Case, J. Casper, B. C. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[8] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. K. ngsbury. Deep neural networks for acoustic modeling in speech recogniti on. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012.

[9] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[10] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networ ks. In *CVPR*, 2014.

[11] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012.

[12] Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Ng. On optimization methods for deep learning. In *ICML*, 2011.

[13] Q. V. Le, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. In *In International Conference on Machine Learning, 2012. 103*.

[14] K. Ni, R. Pearce, K. Boakye, B. Van Essen, D. Borth, B. Chen, and E. Wang. Large-scale deep learning on the yfcc100m dataset. *arXiv preprint arXiv:1502.03409*, 2015.

[15] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Trans. Math. Softw.*, 39(2):13:1–13:24, Feb. 2013.

[16] T. N. Sainath, B. Kingsbury, A. Mohamed, G. E. Dahl, G. Saon, H. Soltau, T. Beran, A. Y. Aravkin, and B. Ramabhadran. Improvements to deep convolutional neural networks for LVCSR. *CoRR*, abs/1309.1501, 2013.

[17] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. On parallelizability of stochastic gradient descent for speech DNNS. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 235–239. IEEE, 2014.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[19] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.

Prepared by LLNL under Contract DE-AC52-07NA27344.