Software Frameworks for Parallel Deep Learning

James Fox
Georgia Institute of Technology
North Avenue
Atlanta, GA 30332
foxjas09@gmail.com

Yiming Zou
Indiana University Bloomington
107 S. Indiana Avenue
Bloomington, IN 47405-7000
yizou@iu.edu

ABSTRACT

The study and adoption of deep learning methods has led to significant progress in different application domains. As deep learning continues to show promise and its utilization matures, so does the infrastructure and software needed to support it. Various frameworks have been developed in recent years to facilitate both implementation and training of deep learning networks. As deep learning has also evolved to scale across multiple machines, there's a growing need for frameworks that can provide full parallel support. While deep learning frameworks restricted to running on a single machine have been studied and compared, frameworks which support parallel deep learning at scale are relatively less known and well-studied. This paper seeks to bridge that gap by surveying, summarizing, and comparing frameworks which currently support distributed execution, including but not limited to Tensorflow, CNTK, Deeplearning4j, MXNet, H2O, CaffeOnSpark, Theano, and Torch.

1. INTRODUCTION

Deep learning has been quite successful in improving predictive power in domains such as computer vision and natural language processing. As accuracies continue to increase, so do the complexity of network architectures and the size of the parameter space. Google's network for unsupervised learning of image features reached a billion parameters [17], and was increased to 11 billion parmeters in a separate experiment at Stanford [23]. In the NLP space, Digital Reasoning Systems trained a 160 billion parameter network [26] fairly recently. Handling problems of this size involves looking beyond the single machine, which Google first demonstrated through its distributed DistBelief framework [16].

The goal of this paper is to survey the landscape of deep learning frameworks with full support for parallelization. Three levels of parallelization exist on the hardware level: within a GPU, between GPUs on a single node, and between nodes. Two forms of parallelism also exist on the application level: model and data parallelism. Other aspects of frameworks include release date, core language, user-facing API, computation model, communication model, programming paradigm, fault tolerance, and visualization. This choice of criteria is explained in detail in Section 2. Tensorflow, CNTK, Deeplearning4j, MXNet, H2O, CaffeOnSpark, Theano, and Torch do not necessarily encompass the entire space of frameworks for deep learning. However, these were chosen by a combination of factors, including their being open-source, level of documentation, maturity as a complete product, and level of adoption by the community. These frameworks, as well as some others not included in the Section 2 chart, are examined in detail in Section 3. Additional comments regarding parallelism in deep learning, as well as scalability, are also discussed at the end of Section 3. Section 4 concludes the paper.

2. FRAMEWORK COMPARISON

The relevance of release date, core language, user-facing APIs are self-explanatory. Computation model specifies the nature of data consistency through execution, i.e. whether updates are synchronous or asynchronous. In context of optimization kernels like stochastic gradient descent (SGD), synchronous execution has better convergence guarantees by maintaining consistency or near-consistency with sequential execution. However, asynchronous SGD can exploit more parallelism and train faster, but with less guarantees of convergence speed. Note that frameworks like Tensorflow and MXNet simply leave this as a tradeoff choice for the user.

The communication model tries to categorize the nature of distributed execution by well-known paradigms.

Data and model parallelism are the two prevalent opportunities for parallelism in training deep learning networks at the distributed level. In data parallelism, copies of the model, or parameters, are each trained on its own subset of the training data, while updating the same global model. In model parallelism, the model itself is partitioned and trained in parallel.

Programming paradigm falls into the categories of imperative, declarative, or a mix of both. Conventionally, imperative programming specifies *how* a computation is done, where as declarative programming specifies *what* needs to be done. There is plenty of gray area, but the distinction is made in this paper based on whether the API exposes the user to computation details that require some understanding of the inner math of neural networks (imperative), or whether the abstraction is yet higher (declarative).

Fault tolerance is included for two reasons. Distributed execution tends to be more failure prone, especially at scale. Furthermore, any failures (not necessarily limited to distributed execution) that interrupt training part-way can be very costly, if all the progress made on the model is simply lost.

Finally, UI/Visualization is a feature supported to very different degrees across the frameworks studied. The ability to monitor the progress of training and the internal state

Table 1: Open-source Frameworks

Platform	Tensorflow	CNTK	Deeplearning4j	MXNet	H2O	CaffeOnSpark
Release Date	2016	2016	2015	2015	2014	2016
Core Language	C++	C++	Java	C++	Java	C++, Scala
API	C++, Python	NDL	Java, Scala	C++, Python, R, Scala, Matlab, Javascript, Go, Julia	Java, R, Python, Scala, Javascript, web-UI	Python, Matlab, Scala
Computation Model	Sync or async	Sync	Sync	Sync or async	Async	Sync
Communication Model	Parameter server	MPI	Iterative MapReduce	Parameter server	Distributed fork-join	MPI Allreduce
Data Parallelism	✓	✓	✓	✓	✓	✓
Model Parallelism	✓	N/A	✗	✓	✗	✗
Programming Paradigm	Imperative	Imperative	Declarative	Both	Declarative	Declarative
Fault Tolerance	Checkpoint-and-recovery	Checkpoint-and-resume	Checkpoint-and-resume	Checkpoint-and-resume	N/A	N/A
Visualization	Graph-visualization (interactive), training monitoring	Graph visualization (static)	Training monitoring	None	None	Summary Statistics

of networks over time could be useful for debugging or hyperparameter tuning, and could be an interesting direction. Tensorflow and Deeplearning4j both support this kind of visualization.

3. FRAMEWORK DISCUSSION

3.1 Tensorflow

Tensorflow was released by Google Research as open source in November 2015, and included distributed support in 2016. The user-facing APIs are C++ and Python. Programming with Tensorflow leans more imperative. While plenty of abstraction power is expressed in its library, the user will probably also be working with computational primitive wrappers such as matrix operations, element-wise math operators, and looping control. In other words, the user is exposed to some of the internal workings of deep learning networks. Tensorflow treats networks as a directed graph of nodes encapsulating dataflow computation and required dependencies [12]. Each node, or computation, gets mapped to devices (CPUs or GPUs) according to some cost function. This partitions the overall graph into subgraphs, one per device. Cross-device edges are replaced to encode necessary synchronization between device pairs. Distributed execution appears to be a natural extension of this arrangement, except that TCP or Remote Direct Memory Access (RDMA) is used for inter-device communication on separate machines. This approach of mapping subgraphs onto devices also offers potential scalability, because each worker can schedule its own subgraph at runtime instead of relying on a centralized master [12]. Parallelism in Tensorflow can be expressed at several levels, notably both data parallelism and model parallelism. Data parallelism can happen both across and within workers, by training separate batches of data on model replications. Model parallelism is expressed through splitting one model, or its graph, across devices. Model updates can either be synchronous or asynchronous for parameter-optimizing algorithms such as SGD. For fault tolerance, Tensorflow provides checkpointing and recovery of data designated to be persistent, while the overall computation graph is restarted. In terms of other features, TensorBoard is a tool for interactive visualization of a user's network, and also provides time series data on various aspects of the learning network's state during training.

3.2 CNTK

Computational Network Toolkit (CNTK) was made open-source by Microsoft around January of 2016. It currently offers a high-level domain-specific language called NDK for implementing networks. The programming paradigm is expressive, complete with vector, matrix, and tensor operations [27] for specifying computations. CNTK generally treats a network as a directed graph, where nodes encapsulate operations and edges for data flow, similar to Tensorflow. There is also a special operator to handle feedback, supporting arbitrary recurrent neural networks [10]. MPI is used for distributed communication. Data parallelism is enabled in a synchronous manner [7], for SGD [9]. However, support for model parallelism is not explicitly mentioned. Fault tolerance involves checkpoint-and-restart [4].

3.3 Deeplearning4j

Deeplearning4j is a Java-based deep learning library built and supported by Skymind, a machine learning intelligence company, in 2014. It is an open source product designed for adoptability in industry, where Java is very common. The framework currently interfaces with both Java and Scala, with a Python SDK in-progress. Programming is primarily declarative, involving specifying network hyperparameters and layer information. Deeplearning4j integrates with Hadoop and Spark, or Akka and AWS for processing backends. Distributed execution provides data parallelism through the Iterative MapReduce model [5]. Each worker processes its own minibatch of training data, with workers periodically ”reducing” (averaging) their parameter data. Deeplearning4j hosts its own Java linear algebra library, which claims 2x or more speedup over Python's Numpy library on large matrix multiplies [8]. Fault tolerance is not mentioned, although Spark does have built-in fault tolerance mechanisms.

3.4 MXNet

MXNet became available in 2015 and was developed in collaboration across several institutions, including CMU, University of Washington, and Microsoft. It currently interfaces with C++, Python, R, Scala, Matlab, Javascript, Go, and Julia. MXNet supports both declarative and declarative expressions; declarative in declaring computation graphs with higher-level abstractions like convolutional layers, and imperative in the ability to direct tensor computation and control flow. Data parallelism is supported by default, and it also seems possible to build with model parallelism. Distributed execution in MXNet generally follows a parameter server model, with parallelism and data consistency managed at two levels: intra-worker and inter-worker [15]. Devices within a single worker machine maintain synchronous consistency on its parameters. Inter-worker data consistency can either be synchronous, where gradients over all workers are aggregated before proceeding, or asynchronous, where each worker independently updates parameters. This trade-off between performance and convergence speed is left as an option to the user. The actual handling of server updates and requests is pushed down to MXNet's dependency engine, which schedules all operations and performs resource management [15]. Fault tolerance on MXNet involves checkpoint-and-resume, which must be user-initiated.

3.5 H2O

H2O is the open-source product of H2O.ai, a company focused on machine learning solutions. H2O is unique among the other tools discussed here in that it is a complete data processing platform, with its own parallel processing engine (improving on MapReduce) with a general machine learning library. The discussion will be limited to H2O's deep learning component, available since 2014. H2O is Java-based at its core, but also offers API support for Java, R, Python, Scala, Javascript, as well as a web-UI interface [13]. Programming for deep learning appears declarative, as model-building involves specifying hyperparameters and high-level layer information. Distributed execution for deep learning follows the characteristics of H2O's processing engine, which is in-memory and can be summarized as a distributed fork-join model (targeting finer-grained parallelism) [20]. Data parallelism follows the ”HogWild!” [22] approach for parallelizing SGD. Multiple cores handle subsets of training data and update shared parameters asynchronously. Scaling up to multi-node, each node operates in parallel on a copy of the global parameters, while parameters are averaged for a global update, per training iteration [13]. There does not seem to be explicit support for model parallelism. Fault tolerance involves user-initiated checkpoint-and-resume. H2O's web tool can be used to build models and manage workflows, as well as some basic summary statistics, e.g. confusion matrix from training and validation.

3.6 CaffeOnSpark

CaffeOnSpark is a Spark deep learning package released open-source in early 2016 by Yahoo's Big ML team. It serves as a distributed implementation of Caffe [19], a framework for convolutional deep learning released by UC Berkeley's computer vision community in 2014. The language interface for CaffeOnSpark is Scala (following Spark), while Caffe itself offers Python and Matlab API. Programming is highly declarative; creating a deep learning network involves specifying layers and hyperparameters, which are compiled down to a configuration file that Caffe then uses. During distributed runtime, Spark launches ”executors,” each responsible for a partition of HDFS-based training data and trains the data by running multiple Caffe threads mapped to GPUs [6]. MPI is used to synchronize executor's respective the parameters' gradients in an Allreduce-like fashion, per training batch [1]. In terms of other notable features, Caffe itself hosts a repository of pre-trained models of some popular convolutional networks such as AlexNet or GoogleNet. It also integrates support for data preprocessing, including building LMDB databases from raw data for higher-throughput, concurrent reading. It does not seem that CaffeOnSpark presently offers any fault tolerance other than what comes with Spark.

3.7 Additional Frameworks

Theano [25] and Torch [11] are two relatively popular frameworks in Python and Lua, respectively. Neither framework currently supports distributed training on more than one machine. Theano on MPI [21] describes an external effort for a distributed version of Theano.

Caffe, besides SparkOnCaffe, also has some other external implementations. There was a recent distributed version, FireCaffe [18], through work done at UC Berkeley. Operating on the premise that deep learning scalability is dominated by communication overhead, FireCaffe uses a reduction-tree communication model, which is meant to be asymptotically faster than global synchronization via parameter server model. This work does emphasize having proper hardware, namely fast interconnects for communication between devices. Experiments were ran on a 128-GPU cluster with Infiniband/Cray interconnect. DIGITS [2], while not distributed, is an NVIDIA integration with Caffe that provides scalability across NVIDIA-GPUs. As a framework, or an extension thereof, one distinguishing feature is that programming is entirely through a GUI interface, making for highly declarative programming.

3.8 Relevant Research

Distributed frameworks exist under the basic premise that easy scaling up to more machines, thereby allowing more parallelism, is important for scaling up to bigger problems. While this is generally true, there are also other important factors, namely the underlying hardware and application characteristics. Whereas Google's billion-parameter model was trained using thousands of machines for several days, Stanford demonstrated training a 11 billion-parameter version of a similar model using a tiny fraction of that number in

3 days [14]. The underlying hardware was a HPC cluster of 16 machines, with 4 NVIDIA GPUs each. Communication happened via MPI on top of fast Infiniband interconnects. The application characteristics, namely a convolutional neural network (with auto-encoding) with 200x200 images as input, limited the amount of model parallelism that could be extracted from mapping partitions of the images to different GPUs. Therefore, it seems that 16 machines, or 64 GPUs, was optimal enough.

Taking the hardware concept further, NVIDIA now has a state-of-the-art server (DGX-1) consisting of 8 Tesla P100 GPUs (over 28,000 CUDA cores), optimized for deep learning [3]. The network hardware is Infiniband, with a direct-connect topology for inter-GPU communication. It is certainly possible that such specialized hardware would sufficiently handle certain deep learning problems, at-scale, without the need of a second machine.

On the application side of things, it's also not necessarily the case that greater model size correlates with higher model accuracy. In fact, a fairly state-of-the-art convolutional neural network like GoogleNet [24] achieves similar accuracy to other networks that use far more parameters.

4. CONCLUSIONS

Tensorflow, CNTK, Deeplearning4j, MXNet, H2O, and CaffeOnSpark were the distributed frameworks chosen for comparison, as a result of factors such as being open source and their level of adoption by the community. They were compared according to the following aspects: release date, core language, API, computation and communication models, data and model parallelism, programming paradigm, fault tolerance, and visualization. These findings are summarized in Table 1. It is interesting that while all the distributed frameworks mentioned support data parallelism, only two (Tensorflow and MXNet) explicitly support model parallelism, suggesting that it is harder to generalize for deep learning problems as opposed to data parallelism. Well-known frameworks like Theano, Torch, and Caffe do have in-house support for distributed training, yet there are efforts by 3rd parties to support it (as is the case for Theano and Caffe). Finally, it is worth noting that while distributed frameworks are designed to scale up to many (heterogeneous) machines, some deep learning problems can be solved efficiently and accurately without needing many machines, given the right utilization of specialized hardware.

5. REFERENCES

[1] Caffeonspark open sourced for distributed deep... — hadoop at yahoo. `http://yahoohadoop.tumblr.com/post/139916563586/caffeonspark-open-sourced-for-distributed-deep`. (Accessed on 07/23/2016).

[2] Digits: Deep learning gpu training system — parallel forall. `https://devblogs.nvidia.com/parallelforall/digits-deep-learning-gpu-training-system/`. (Accessed on 07/28/2016).

[3] Gp100 pascal whitepaper. `https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf`. (Accessed on 07/27/2016).

[4] An introduction to computational networks and the computational network toolkit. `http://www.zhihenghuang.com/publications/CNTKBook-Draft0.7-2015-03-19.pdf`. (Accessed on 07/27/2016).

[5] Iterative reduce with dl4j on hadoop and spark - deeplearning4j: Open-source, distributed deep learning for the jvm. `http://deeplearning4j.org/iterativereduce`. (Accessed on 07/26/2016).

[6] Large scale distributed deep learning on hadoop... — hadoop at yahoo. `http://yahoohadoop.tumblr.com/post/129872361846/large-scale-distributed-deep-learning-on-hadoop`. (Accessed on 07/23/2016).

[7] Multiple gpus and machines microsoft/cntk wiki. `https://github.com/Microsoft/CNTK/wiki/Multiple-GPUs-and-machines`. (Accessed on 07/27/2016).

[8] Nd4j's benchmarking tool n-dimensional scientific computing for java. `http://nd4j.org/benchmarking`. (Accessed on 07/26/2016).

[9] Powerpoint presentation. `http://research.microsoft.com/en-us/um/people/dongyu/CNTK-Tutorial-NIPS2015.pdf`. (Accessed on 07/27/2016).

[10] Tensorflow meets microsofts cntk — the escience cloud. `https://esciencegroup.com/2016/02/08/tensorflow-meets-microsofts-cntk/`. (Accessed on 07/27/2016).

[11] Torch — scientific computing for luajit. `http://torch.ch/`. (Accessed on 07/28/2016).

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[13] A. Candel, V. Parmar, E. LeDell, and A. Arora. Deep learning with h2o, 2015.

[14] B. Catanzaro. Deep learning with cots hpc systems.

[15] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[16] 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.

[17] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014.

[18] F. N. Iandola, K. Ashraf, M. W. Moskewicz, and K. Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. *arXiv preprint arXiv:1511.00175*, 2015.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[20] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin. A survey of open source tools for machine learning with big data in the hadoop ecosystem. *Journal of Big Data*, 2(1):1–36, 2015.

[21] H. Ma, F. Mao, and G. W. Taylor. Theano-mpi: a theano-based distributed training framework. *CoRR*, abs/1605.08325, 2016.

[22] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[23] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[25] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[26] A. Trask, D. Gilmore, and M. Russell. Modeling order in neural word embeddings at scale.

[27] D. Yu, K. Yao, and Y. Zhang. The computational network toolkit [best of the web]. *IEEE Signal Processing Magazine*, 32(6):123–126, 2015.