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 becoming distributed, there's a growing need for frameworks that can support execution beyond a single machine. While deep learning frameworks restricted to running on a single machine have been studied and compared, frameworks which support deep learning distributed across multiple machines 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 Tensorflow, Deeplearning4j, MXNet, H2O, and CaffeOnSpark.

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 their architectures and the size of the parameter space, and this trend is likely to continue. Google's network for unsupervised learning of image features reached a billion parameters [1], and that number increased to 11 billion in a separate experiment at Stanford [2]. In the natural language processing space, Digital Reasoning Systems has recently outdone those sizes by training a 160 billion parameter network [3]. Handling problems of this size requires scaling up to thousands of cores across many machines, which Google first demonstrated on its distributed DistBelief framework [4].

The goal of this paper is to survey the landscape of deep learning frameworks with support for distributed execution, and compare them according to a consistent set of characteristics. The characteristics compared across frameworks include release date, core language, user-facing API, computation model, communication model, data parallelism, model parallelism, programming paradigm, fault tolerance, and visualization. This choice of criteria is explained in Section 2. Tensorflow, Deeplearning4j, MXNet, H2O, and CaffeOnSpark were chosen by a combination of factors, including their being open-source, level of documentation, maturity as a product, and adoption by the community. Frameworks currently without distributed support won't be the focus of this discussion, but well-known ones such as Theano, Torch, Caffe (without Spark) have been studied [5].

The rest of the paper is organized as follows: Section 2 compares distributed frameworks according to aforementioned criteria, and also makes not of some non-distributed frameworks. Section 3 discusses each distributed framework in detail. Section 4 outlines future directions of work. Section 5 concludes the paper.

2 Framework Comparison

Platform	Tensorflow	Deeplearning4j	MXNet	H2O	CaffeOnSpark
Release Date	2016 (distributed)	2015	2015	2014 (deep learning)	2016
Core Language	C++	Java	C++	Java	C++ (Caffe) Scala (Spark)
API	C++, Python	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 or async	Async	Sync
Communication Model	Parameter server	Iterative MapReduce	Parameter server	Distributed fork-join	MPI Allreduce
Data Parallelism	✓	✓	✓	✓	✓
Model Parallelism	✓	✗	✓	✗	✗
Programming Paradigm	Imperative	Declarative	Both	Declarative	Declarative
Fault Tolerance	Checkpoint-and-recovery	Checkpoint-and-resume	Checkpoint-and-resume	N/A	N/A
Visualization	Graph-visualization, training monitoring	Training monitoring	None	None	Summary Statistics

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 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.

The scope of deep learning functionality was a possible point of discussion. However, initial surveying suggested that at this point most deep learning libraries provide all the standard network architectures, training and optimization techniques. More differences are likely to be found in programmability or performance.

3 Framework Discussion

3.1 Tensorflow

Tensorflow was released by Google Research for open source in November 2015, and updated to include distributed support in 2016. The user-facing APIs are C++ and Python. Programming with Tensorflow leans more imperative than with some of the other frameworks discussed. While plenty of abstraction power is expressed in its library, the user will probably also be working with computational wrappers of primitives 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 sees constructed networks as a directed graph of nodes encapsulating dataflow computation and required dependencies [8]. 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 scalability potential, as each worker can schedule its own subgraph at runtime instead of relying on a centralized master [8]. Parallelism in Tensorflow can be expressed at several levels, notable 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 stochastic gradient descent (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 Deeplearning4j

Deeplearning4j is a Java-based deep learning library built and supported by Skymind, a machine learning intelligence company founded 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 (**ref?**). Each worker processes its own minibatch of training data, with workers periodically "reducing" (averaging) their parameter data. Formal benchmarking in terms of scaling was not found, but benchmarking on their custom Java linear algebra library show 2x or more speedup over Numpy on large matrix multiplies. Deeplearing4j's website provides clear documentation of available features and API, which range from range from a menu of optimization algorithms to built-in vectorization libraries. Fault tolerance is not mentioned, although Spark does have built-in fault tolerance mechanisms.

3.3 MXNet

MXNet is a distributed deep learning framework that became available in 2015. It 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; symbolic in declaring computation graphs with higher-level abstractions like convolutional layers, and imperative in the ability to direct tensor computation and control flow [1]. 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. 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. Fault tolerance on MXNet involves checkpoint-and-resume, which must be user-initiated.

3.4 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 and other tools. 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 [9]. 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) [10]. Data parallelism is enabled following the "Hog-Wild!" [11] approach for parallelizing SGD. In implementation, 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 [9]. 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, and also provides some basic summary statistics, e.g. confusion matrix from training and validation.

3.5 CaffeOnSpark

CaffeOnSpark is a Spark deep learning package released as open-source in early 2016 by Yahoo's Big ML team. It serves as a distributed implementation of Caffe, 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 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 [7]. In terms of some 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 from surveyed literature that

CaffeOnSpark offers any fault tolerance other than what comes with Spark.

4 Future Work

5 Conclusion

References

[1] 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*, pp. 647–655, 2014.

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

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

[4] 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*, pp. 1223–1231, 2012.

[5] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of caffe, neon, theano, and torch for deep learning," *CoRR*, vol. abs/1511.06435, 2015.

[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] "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).

[8] 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*, vol. abs/1603.04467, 2016.

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

[10] 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*, vol. 2, no. 1, pp. 1–36, 2015.

[11] 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*, pp. 693–701, 2011.