1 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, MXNet, Deeplearning4j, and H2O.

2 Introduction

- Application domain examples: vision, speech recognition, NLP
- Architecture types: CNNs, RNNs, fully-connected, auto-encoders, etc.
- Overview of what deep learning involves? E.g. training data, objective function, optimization, tricks for avoiding overfitting; network-agnostic issues.
- Big data and progress of deep learning implementations (CPU to GPU to distributed)
- Motivation for distributed deep learning (ref. Jeff Dean paper; seminal?)
- Introduce paper motivation; distributed not well-studied
- Mention all the frameworks chosen (including non-distributed); emphasis on distributed.
- "The rest of the paper is organized as follows ..."

3 Overview of Deep Learning Frameworks

Applications: LeNet, AlexNet, Stacked autoencoders, LSTM.

Results:

Torch is fastest even on every condition. TensorFlow does well on CPU (except on Stacked autoencoders) but does worst on GPU. Theano does pretty good with GPU, even on some special applications with CPU. On LSTM, Theano is better on GPU while Torch is better on CPU.

Comments:

This comparison is based on single node, including several mainly used frameworks (Torch, TensorFlow, Theano, Caffe, etc). According to the results of the four applications, Torch does the best while Theano follows. TensorFlow, which attracts most attention, doesn't perform well on GPU.

If we want to get further results, we should compare them on distributed clusters and use the latest version. This paper is submitted on March 2016, and during the four months, I do believe the frameworks especially TensorFlow evolves a lot.

4 Comparison Table

platform	TensorFlow	deeplearning4j	MXNet
Single Node	√		
Single Node	√		
Distributed	√		
CPU	√		
GPU	√		
Core	C++		
API	Python		
Popular	313 contributors		
Computation Model			
Programming Model			
Programming Expressiveness			

Platform	H2O	GraphLab	CaffeOnSpark
Single Node	√	√	
Distributed	√	√	
CPU	√	√	
GPU	√	√	
Core	Java	C++	
API	R, Python, etc	Python	
Popular	51 contributors	40 contributors	
Computation Model			
Programming Model			
Programming Expressiveness			

5 Framework Discussion

5.1 MXNet

MXNet is a distributed deep learning framework that became available in 2015. It was developed with collaborators from 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. Results from MXNet's own scaling benchmarking, using Googlenet, show good scaling from 1 to 10 machines [1].

5.2 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. Websites provides clear documentation of available features and API, which range from range from a menu of optimization algorithms to built-in vectorization libraries. Seems to have active community.

5.3 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 [1]. MPI is used to synchronize executor's respective the parameters' gradients in an Allreduce-

like fashion, per training batch [2]. 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.

6 Future Work

7 Conclusion

References

[1] "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).

[2] "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).

[3] 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.