Usability Study of Distributed Deep Learning Frameworks For Convolutional Neural Networks

Jiayi Liu*
Jason.Liu@lge.com
LG Electronics
Santa Clara, CA 95054

Jayanta Dutta†, Nanxiang Li†
Jayanta.Dutta,Nanxiang.Li@us.bosch.com
Robert Bosch LLC.
Sunnyvale, CA 94085

Unmesh Kurup*, Mohak Shah*
Unmesh.Kurup,Mohak.Shah@lge.com
LG Electronics
Santa Clara, CA 95054

ABSTRACT

With the popularity of deep learning, the increasing complexity of deep learning models, and the availability of very large datasets, model training has become a time-consuming process. One way to make this process efficient is to distribute training across multiple GPUs and nodes, and many deep learning frameworks now support distributed training. In this paper, we survey the various distributed versions of popular deep learning frameworks. We analyze their performance from the practitioner's point of view and compare them on the basis of various indicators including time, memory, and accuracy. We show that there are measurable differences between various frameworks. We also provide a qualitative comparison that measures community popularity, functionality, compatibility, and ease-of-use. We believe this multi-faceted comparison will allow practitioners to make an informed choice on which framework to use when conducting distributed training of deep learning models. We also open source the benchmark pipeline so that newer frameworks (or newer versions of existing frameworks) may be compared as and when they become available.

CCS CONCEPTS

• **Software and its engineering** → **Software libraries and repositories**; • **Computing methodologies** → *Neural networks*; *Distributed programming languages*;

KEYWORDS

Deep Learning, Software Library, Distributed System, Benchmark

1 INTRODUCTION

Deep Neural Networks (DNNs) have achieved great success in many application domains including computer vision [13], natural language processing [5], and speech recognition [8]. And specifically in the computer vision domain, Convolutional Neural Networks (CNNs) have improved results on object recognition and object detection and enabled industrial applications such as disease diagnosis and autonomous driving [9, 18]. These improvements have, however, come at the cost of increasingly complex models with millions of parameters that need to be trained over ever larger datasets.

Training a CNN model is a time-consuming process. To speed up this process, three general directions can be pursued. First, specialized processors, (Graphics Processing Units (GPUs), TPUs etc.) and software libraries (CuDNN, fbfft) can be used. Second, additional computational resources can be deployed in a distributed paradigm. Third, better algorithmic methods that lead to faster convergence can be developed and applied. In this paper, we focus on the second direction, namely, that of speeding up development and deployment using a distributed approach. At the time of writing, the on-demand price of GPU instances is up to 3 US dollar per hour, which is significantly cheaper than the time cost of engineers.

Many of the popular open source Deep Learning (DL) frameworks now offer distributed versions that allow the user to train models that utilize multiple GPUs and even multiple nodes. We investigate how these different distributed versions stack up against each other. We evaluate these frameworks along two dimensions - quantitative and qualitative. We measure performance in terms of time, memory usage, and accuracy for Caffe2, Chainer, CNTK, MXNet, and Tensorflow as they scale across multiple GPUs and multiple nodes. We compare performance on two representative datasets (Cifar-10 and ImageNet) and various batch sizes to better understand how these frameworks perform under distributed loads. We also measure certain qualitative aspects of these frameworks to present a more rounded picture of their adoption and ease-of-use. These measures include popularity, functionality, and compatibility, as measured by Github stars, Application Programming Interfaces (APIs), and availability of pre-trained models among other attributes.

Previous work has evaluated the training and inference performance for multiple models and frameworks [3]. However, that work only analyzed single CPU/GPU performance. With the field evolving fast, and the availability of distributed frameworks, a new analysis is needed. More recent work [22] has analyzed the performance of distributed DL frameworks, but it does not provide an empirical evaluation. In this paper, we are addressing the problem of which DL framework to use from a practitioner's perspective. The key contributions of this paper include:

(1) An empirical evaluation of the performance of distributed DL frameworks for different use cases.

*This work was done while the authors were at Robert Bosch LLC.
†Corresponding authors.

KDD'18 Deep Learning Day, August 2018, London, UK

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.

(2) A usability analysis that combines the empirical evaluation with qualitative data that identifies the strength and weakness of the frameworks.
(3) Open source templates that the community can use to analyze the different frameworks for their particular problem sets.

2 DEEP LEARNING FRAMEWORKS

While there are a number of DL frameworks in use, in this paper we focus on those open-source packages that support distributed model training and development. We selected five popular frameworks and leave the rest for a future study.

- Caffe2[1] is a light-weight and modular DL framework open sourced by Facebook. It emphasizes model deployment for edge devices and model training at scale.
- Chainer[2] [20] is a flexible DL framework developed by Preferred Networks that provides an intuitive interface and high performance implementation. The distributed version is ChainerMN [2]. Rather than separate the definition of a computational graph from its use, Chainer uses a strategy called "Defined-by-Run" where the network is created when the forward computation takes place.
- Microsoft Cognitive Toolkit (CNTK)[3] [21] is a commercial-grade distributed deep learning toolkit developed at Microsoft. It also has advanced algorithms but these are not under open-source licenses.
- MXNet is a flexible and efficient library for deep learning, featuring high-level APIs. It is sponsored by Apache Incubator and selected by Amazon as its choice for DL.
- Tensorflow[4] is a general numerical computation library for data flow graphs. It was developed by Google Brain Team and is currently an open source project for machine learning and deep learning domains.

Deep Learning for Java (DL4J)[5] is an open source project for JVM that is natively distributed by integrating with Hadoop and Spark. Although it also supports Python in the Skymind Intelligence Layer (SKIL), it targets a different community and we skip it in our analysis.

PyTorch[6] is another deep learning framework but its support for distributed computing is still in a early stage of development and is excluded from our analysis.

There is an open-source Theano-MPI project [14] that supports multi-GPU, multi-node training. But since the development of Theano has been stopped[7], we exclude it from the comparison.

3 COMPARISON OF USABILITY

In this section, we compare the feasibility and usability of the different frameworks. Distributed model training requires special skills, however we do see the benefits brought by the frameworks with easy user interfaces. In this work, we focus on the model development and deployment rather than support for extending and improving the framework itself (for example by adding new optimizers or loss functions). We first investigated the usability of high-level APIs. And after that, we focused on the model library and other extensions. We also comment on the package setup with an emphasis on the procedure for distributed use. Finally, the comparison in this section generally applies to the single GPU training, and it is a supplement to the previous benchmark studies.

3.1 APIs

APIs provide a high-level abstraction to the framework. Thus, a user-friendly API is crucial to efficiently developing new DL models. Also, consistent APIs simplify the search for the best model. We compared the high level API definitions of the frameworks along seven different categories:

- Activation functions control how information and gradient flow through the network layers. Besides the commonly used Rectified Linear Unit (ReLU), many others have been proposed, but their performances are not consistent for different models and datasets [15]. So a high-level API that enables practitioners to easily change the activation function is an important feature. Both Chainer and Tensorflow provide rich collections of activation functions for users to choose from. In Caffe2, activation functions are not listed separately, so we did not extract the exact number from the API documentation and listed it as None in Table 1. Similarly, CNTK supports 6 activation functions as part of BrainScript, but not independently in the main framework. So we also listed it as None.
- Initialization functions are crucial for effectively training DNNs and there are several theoretical approaches as well as best-practice heuristics in the literature [6]. A collection of such functions would also help users to fine-tune their model. Caffe2 lags behind the other frameworks in the number of such initialization functions that are available.
- Loss functions are critical to the optimization process. Chainer has the most extensive collection of loss functions among all the frameworks.
- Evaluation functions such as accuracy, while secondary to the loss function, are also an important part of the evaluating model performance. Tensorflow provides the largest number of evaluation functions among all frameworks.
- Neural Network (NN) functions are the basic building blocks of DNN models and a framework that provides more building blocks makes it easier to construct different type of architectures. However, because different frameworks have different abstractions, a straightforward comparison of the number of such building blocks should be taken with a grain of salt. For instance, Tensorflow has separate functions for convolutions in 1,2, and 3 dimensions while CNTK provides a single abstraction with the dimension as a parameter.
- Optimization functions are important for model training and critical for the convergence rate. There are many variants have been proposed [11, 16, 17] and CNTK leads the pack in the number of supported methods.

[1] https://caffe2.ai/
[2] https://chainer.org/
[3] https://docs.microsoft.com/en-us/cognitive-toolkit/
[4] https://www.tensorflow.org/
[5] https://deeplearning4j.org/
[6] http://pytorch.org/
[7] See https://groups.google.com/forum/#!topic/theano-users/7Poq8BZutbY

- Regularization functions also impact on the model performance (J. Dutta et al. in prep). However the explicit support for separate regularization functions is still limited across all frameworks.

In Table 1, we compare the numbers retrieved from different frameworks on Feb 7, 2018. We automatically extracted the numbers from the official framework websites (see `Comparison` folder in the open source project). The pages that we used to extract the numbers can be found at Comparison/url_table.json. We want to highlight that the numbers are from high level APIs and do not count the contributions scattered outside of the main document. Also sometimes the numbers are not fully comparable, as in the example with Tensorflow and CNTK above.

Overall, Tensorflow has the best coverage on high-level APIs. Chainer and MXNet have fewer NN layers available to use directly. Caffe2 and CNTK miss some aspects from the API level but are strong on available NN and optimizer APIs, respectively.

3.2 Model Library and Usability

We highlight several other important aspects to measure the usability of the frameworks and provide a brief summary in Table 2.

- Model library: A pre-built model library is a good starting point for many applications. We extracted the number of such pre-trained models from the official websites. Both CNTK and Tensorflow have the largest libraries that support models for different types of DL problems. In comparison, the other frameworks provide limited coverage.
- Documentation: Good documentation is crucial for users and developers. Currently, compared to other frameworks, Caffe2's documentation is still a work in progress.
- ONNX: Open Neural Network Exchange (ONNX) format helps users to easily reuse DL models across frameworks. Currently all compared frameworks support ONNX, except Tensorflow, which requires an independent backend (`onnx-tf` package).
- Visualization: A graph representation of a DNN helps users more easily debug their models. Tensorflow and CNTK uses Tensorboard to both visualize the NN architecture and investigate and monitor the values during the training process. The other frameworks also have their tools for NN visualization but are relatively weak in tracking values during the training stage.
- Deployment: Deploying models into the final production line is a crucial step for industrial applications. Tensorflow provides an interface for serving models for large-scale applications and a second lite-version for mobile deployment. Caffe2 is designed for mobile deployment and CNTK is best for corporate settings with Windows machines or Azure Cloud. Chainer and MXNet lack support for deploying models.

We measured the popularity of the frameworks by the number of stars, followers, and forks on Github (Table 3). Tensorflow is the most popular choice among all the frameworks.

3.3 Framework Setup

In this section, we compare the ease of setup for the frameworks. A brief summary is listed in Table 4.

3.3.1 Single Node Setup. One advantage of using Python is the low maintenance on software packages. Python package management system, `pip`, simplifies the Python library setup process. Most of the DL frameworks are compatible with it, except Caffe2 at the moment. However we still encountered some problems:

- Ambiguous CPU and GPU packages: the default pip package of MXNet and Tensorflow are for CPU only. It is easy for new users to accidentally select the default package and use the CPU instead of the GPU for training.
- Non-compatible CUDA libraries: CUDA and CuDNN are the foundation of most of DL frameworks. However with the fast upgrade pace of both the frameworks and CUDA, these are occasionally mismatched and become incompatible. For instance, the pre-built Tensorflow version 1.5 package requires CUDA 9.0 whereas the documentation is still written for CUDA 8.0 [8].
- OS requirement: Most packages are prebuilt for Ubuntu, MacOS, and Windows. The support for Redhat/CentOS systems is missing at the moment.

Besides the above mentioned issues, most of the frameworks require additional setup for distributed computation. We will discuss them in the next section.

3.3.2 Multiple Node Setup. The setup of the distributed framework falls into three difficulty levels[9]:

- Caffe2 requires compilation with additional flag `-DUSE_REDIS=ON` for the distributed version. In addition, Caffe2 also requires either a Network File System (NFS) or a Redis server for the communication between nodes. In this study, we used a Redis server setup.
- Chainer provides a pip package for single GPU training. And for the distributed version, the ChainerMN package and Message Passing Interface (MPI) are required.
- CNTK is built for distributed deep learning. There is no additional setup needed, except MPI is required to run the distributed training.
- MXNet requires an additional compilation flag `USE_DIST_KVSTORE=1` to enable the distributed training. Also an additional package `dmlc` and its corresponding drivers are needed to launch the distributed job.
- Tensorflow is self-contained that no other package is needed.

3.3.3 Code Conversion from Single Node to Multiple Node. The level of difficulty to convert the existing code for multi-GPU and multi-node training is also important for practitioners. In this aspect, Chainer and CNTK require only minor changes to the codebase with high-level APIs to transfer existing code to the distributed version. Also leveraging MPI, it is very easy to use these frameworks at runtime. Both Caffe2 and MXNet requires minor changes to the existing code. In addition, Caffe2 requires jobs to be submitted

[8] https://www.tensorflow.org/install/install_linux retrieved on Jan 31, 2018

[9] The detailed installation setup can be found in our open-sourced experiment code under `installation/setup.sh`.

Table 1: Framework Functionality Comparison

Framework	Activation	Evaluation	Initializer	Loss	NN	Optimizer	Regularizer
Caffe2	None	None	2	6	64	9	2
Chainer	17	7	14	20	28	9	None
CNTK	None	3	10	8	20	17	None
MXNet	4	10	13	12	25	9	None
tensorflow	10	33	10	14	90	8	1

Table 2: Functionality

Framework	Model	ONNX	Visual.	Deployment
Caffe2	7	Native	`net_drawer`	iOS/Android
Chainer	13	Native	dot	None
CNTK	47	Native	Tensorboard	Windows/Azure
MXNet	9	Native	`plot_network`	None
Tensorflow	44	`onnx-tf`	Tensorboard	Server/mobile

Table 3: Framework Popularity Comparison

Framework	Star	Watcher	Fork	Created
caffe2	7225	521	1670	2015-06-25
chainer	3459	300	918	2015-06-05
cntk	13798	1333	3645	2015-11-26
mxnet	12995	1116	4781	2015-04-30
tensorflow	88544	7399	43157	2015-11-07

Table 4: Framework Setup Comparison

Framework	Single Node	Multi Node		
	Installation	Installation	Runtime	Code
Caffe2	source	source	scrip	Medium
Chainer	pip	pip	mpi	Easy
CNTK	pip	pip	mpi	Easy
MXNet	pip	source	dmlc	Medium
Tensorflow	pip	pip	script	Hard

individually on each node. And MXNet requires a driver code based on the `dmlc-core/tracker` library to start the training process via MPI, ssh, or other resource management tools. Tensorflow requires the most changes due to the parameter-server setup. And both the parameter servers and the workers need to be started individually, which is not user-friendly.

4 COMPARISON OF PERFORMANCE

4.1 Data and Model Selection

We choose use the most representative data and model to evaluate the framework performance rather than explore the wide spectrum of models, which is economically challenging. We choose the commonly-used Cifar-10 [12] and ImageNet [10] as the evaluating datasets. Although the images of Cifar-10 is very small, it still represents many vision-based use cases. For the model choice, we implement the ResNet model [7], which contains most of the building blocks in modern CNN architectures. In addition, it is flexible with the network depth that we can easily switch between model complexity. In this work, we focus on the most popular ResNet-50 model.

4.2 Experiment Setup

All experiments were performed on the Amazon Web Service (AWS)[10] P2.xlarge instances. The P2.xlarge instance is equipped with one NVIDIA Tesla K80 GPU and four AWS-Specific version of Intel's Broadwell processor, running at 2.7 GHz [11].

In the multi-node case, the network bandwidth (10 Gbps) is default for all cases. The data is pre-loaded in memory, so disk I/O is not a problem. In real applications, the disk I/O has critical impact on the training performance. But the problem can be partially resolved by using approaches like parallel data input pipelines, faster hard disks, etc.

We used Ubuntu 16.04 (ami-1ee65166) as the base OS and the following versions of associated software: Nvidia driver: 8.0.61-1, CUDA: 8.0, CuDNN: 6.0; Caffe2: 0.8.0; CNTK: 2.3.1; MXNet: 1.0.0; Tensorflow: 1.4. We used OpenMPI 3.0.0 except for CNTK where we chose 1.10.3 due to compatibility reasons.

Except for Tensorflow, we use the latest versions of all other packages. Tensorflow 1.5 uses CUDA 9 by default. As we are using CUDA 8 for the other software frameworks, we choose an earlier Tensorflow version (1.4) instead.

For the EC2 instances, we choose the instances on the spot and no dedicated host has been used. Persistence mode is turned on for all GPUs before measurements.

We use the images per second as our measure of framework speed. For each speed measurement, we typically run 64-256 iterations to get enough time interval measurements. And we skip the first 10 iterations in our calculation because the initialization of

[10] https://aws.amazon.com/

[11] https://aws.amazon.com/blogs/aws/new-p2-instance-type-for-amazon-ec2-up-to-16-gpus/ retrieved at Feb 8, 2018.

the model typically has an impact on the first few iterations (see Figure 1).

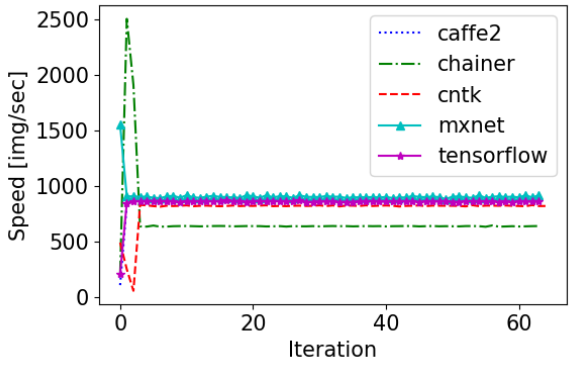

Figure 1: GPU training speed at each iteration for Cifar-10 and ResNet-50 with batch size of 256. Notice how the initial runs take longer.

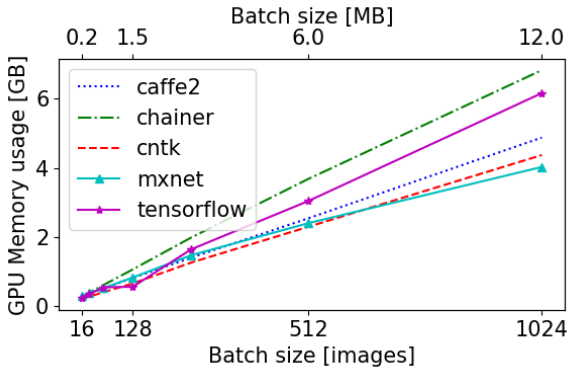

Figure 2: Memory usage for ResNet-50 on Cifar-10 dataset with single GPU.

4.3 Memory

Memory usage is an important measure of the performance of frameworks. The maximum size of a mini-batch is bounded by the available GPU memory. And a larger mini-batch size can lead to both higher throughput and model convergence [23].

We compared the ResNet-50 model [7] on Cifar-10 dataset [12] in Figure 2. The model contains 11.7 million weights to be trained. We used the memory usage from `nvidia-smi` and for Tensorflow, we set `gpu_opentions.allow_growth` to `True` to prohibit the Tensorflow script from taking the full GPU memory.

Our results show that MXNet has better memory management comparing to other frameworks, and Chainer has the largest memory consumption. And the difference among the frameworks is significant for large batch size training cases.

4.4 Inference

We compared the performance of the frameworks of model inference on synthetic images. The images are created as random noise with specific image dimensions and batch sizes. The inference step only computes the forward pass of the DNN and its performance is critical during deployment.

We measured the inference speed for single CPU and single GPU cases, see Figure 3 For single CPU case, we force the script to use only one thread to avoid exploiting the benefits of a multi-core system.

We didn't measure the performance of multi-GPU and multi-node cases because the inference step can be naively parallelized. And the speed is linearly proportional to the number of processors.

(1) The CPU case is insensitive to the number of batches due to the lack of processing power. Overall CNTK has the best inference performance, and Caffe2 and Tensorflow show better performance than the rest two.
(2) The GPU case shows that larger batch sizes are preferred as the GPU processor threads are efficient in processing the forward pass. The performance of the frameworks in the single GPU case are consistent with previous studies [22]. CNTK stands out in two datasets when the optimally large batch size is chosen. Tensorflow and MXNet also have good performance in training. Interestingly when the large batch size is chose, the performance of Caffe2 is worse than expected. And Chainer has poor performance for the single CPU/GPU tests.
(3) ImageNet images are larger than the Cifar-10 images, this results in a lower inference speed in terms of the number of images processed, because the network is wider in each layer and more numerical calculations are needed.

4.5 Single GPU Training

The single GPU training case measures the total time for the forward and the backward steps. Its performance is directly related to the training time for a NN model. Also, it is the baseline for multi-node training in the next section.

We first tested on synthetic Cifar-10 and ImageNet datasets with different batch sizes, see Figure 4. For the small image size case, Caffe2, CNTK, and Tensorflow perform well. The performance of MXNet is problematic as discussed in Sec 4.3. And for the large image size case, MXNet and CNTK are better. The result from ImangeNet case is consistent with the previous study [19]. This indicates that even for a similar CNN model, i.e. ResNet, the performance varies due to the image size difference.

Overall, CNTK has good performance, but for problems that the end users face, a detailed performance analysis is needed to choose the right framework.

4.6 Scalability

The scalability study compared the training speed of different frameworks for single-GPU, multi-GPU, and multi-node cases. The performance of the single-GPU case in the previous section is the baseline performance of different models. And we used synthetic Cifar-10 dataset preloaded before the training stage for all the experiments.

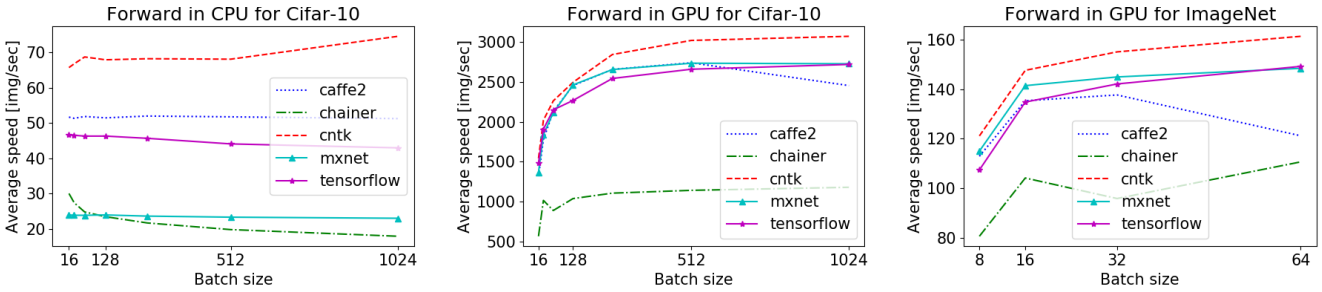

Figure 3: Inference speed of ResNet-50 model using CPU/GPU for Cifar-10 and ImangeNet datasets.

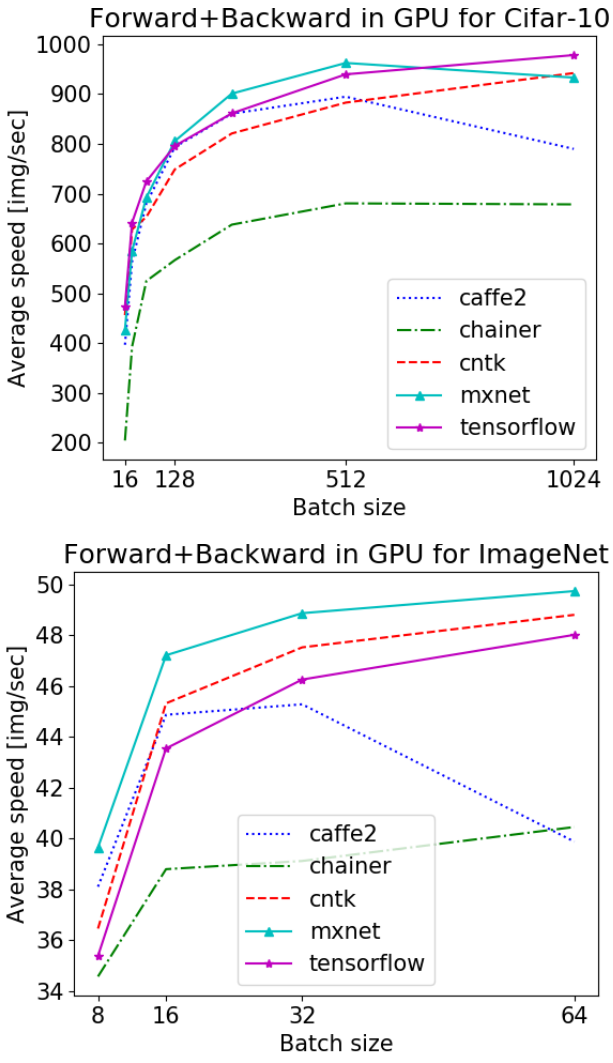

Figure 4: Training speed of ResNet-50 model using single GPU for Cifar-10 and ImangeNet datasets.

The distributed training typically uses either synchronous or asynchronous stochastic gradient descent method to update the weights of a NN model. The synchronous method requires collecting the weight updates from all workers before proceeding to the next iteration, which results in idle workers waiting for the updates from other workers. In contrast, the asynchronous method updates weights based on the update from the current worker's result and may end in a less optimal model[4]. In this work, we only focus on the training speed.

4.6.1 Multi-GPU training. We first present the result from multi-GPU training on Cifar-10 using the ResNet-50 model with a batch size of 1024, see Figure 5. Due to the benefits from high intra-node communication speed and a limited number of GPUs, the synchronous method is preferred in the multi-GPU case. Both CNTK and MXNet showed good scalability up to 8 GPUs. The synchronous method with Tensorflow requires a hand-coded weight averaging function or a parameter-server setup[12], which is different from the approach for other frameworks and is thus excluded from this comparison. The Chainer implementation does not include NVIDIA Collective Communications Library (NCCL), which results in a significant bottleneck in GPU communication. Thus its performance decreases after two GPUs as the idle time increases in tandem with the number of GPU. Caffe2 also shows good scalability but the initial single GPU speed is slightly slower and requires further improvement.

Overall, MXNet and CNTK are good for the multi-GPU model training. And Chainer is not recommended for multi-GPU training because it has not included the critical NCCL component.

4.6.2 Multi-node training. The multi-node training case is an extension to the multi-GPU case that the DL model needs to communicate with the same model trained on other nodes. In this experiment, each node contains only one GPU. Thus, the inter-node communication becomes crucial. Also, AWS P2.xlarge case has no reserved network bandwidth (about 10 Gbps). A better internet interface could increase the model training performance. We encourage the readers to use our open-source code to measure the performance on their infrastructure.

In the experiments, we tested ResNet-50 model on the synthetic Cifar-10 dataset. For Caffe2, we used a Redis server to communicate the weight updates. Caffe2 also requires that at the runtime, we start the jobs on each node individually. For Chainer and CNTK,

[12]https://www.tensorflow.org/deploy/distributed

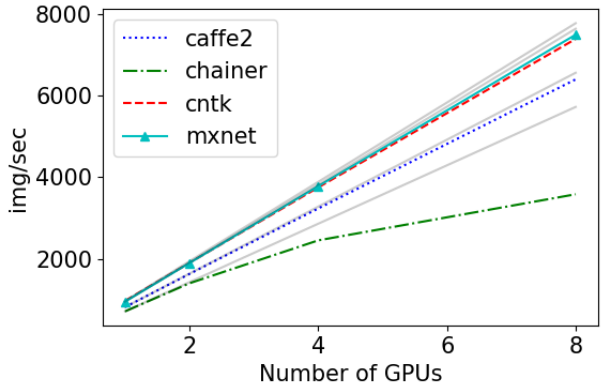

Figure 5: Multi-GPU Performance with sync update

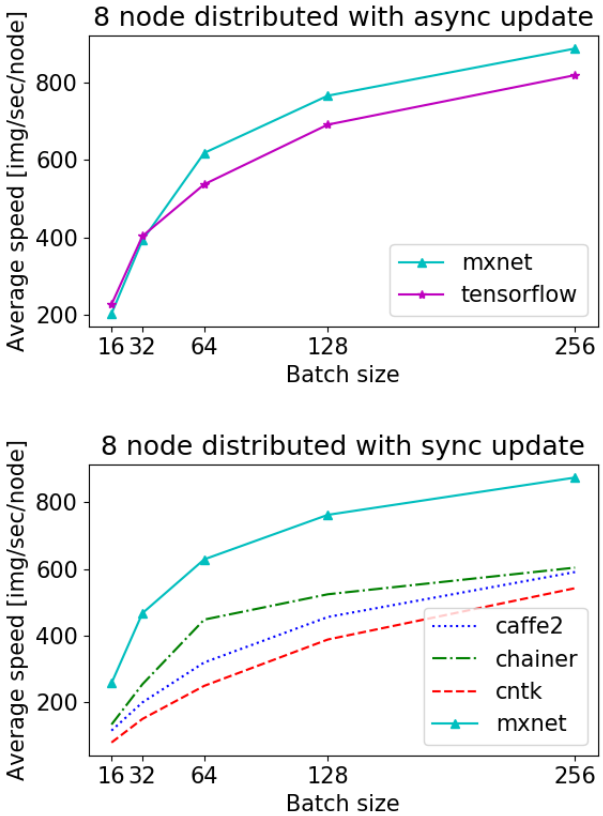

Figure 6: Performance of distributed training with 8 nodes for ResNet-50 on Cifar-10 dataset.

MPI implementation is used and we submit the job once with corresponding resource requirements in the MPI arguments. MXNet provides an additional script to initialize the training. And under the hood, we choose the ssh driver to start the jobs on each node

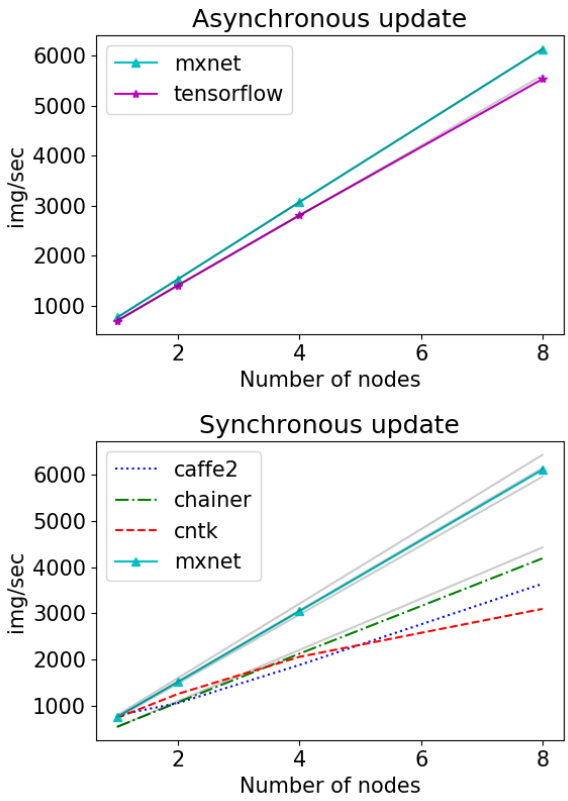

Figure 7: Scalability in performance of distributed training for ResNet-50 on Cifar-10 dataset with batch size 128. Grey solid lines are the linear scalability limits.

via ssh. Tensorflow uses the parameter-server architecture to manage the communication of weight updates [1]. And it requires two different tasks, worker and server, to be started separately. We used one server and one worker on each node. Other configurations are also possible but we do not explore them in this paper.

Similar to multi-GPU training, asynchronous and synchronous updates are used in the multi-node model training. MXNet and Tensorflow support both updates, however, we didn't test the Tensorflow version for synchronized update, because its `SyncReplicasOptimizer` is not mature to use at the moment [13]. The MPI-based frameworks typically provide the synchronous update method.

We first examine the impact of the batch size for 8 nodes (Figure 6). Because of slower communication among nodes (as opposed to between GPUs on the same node), the overhead of the weight update becomes critical. As expected, as the batch size becomes larger, the ratio of overhead time and computation time become smaller and the efficiency of the system is better [22].

Finally, we compared the scalability of the frameworks for 1, 2, 4, and 8 node configurations. The experiments measured the training speed for the ResNet-50 model on the Cifar-10 dataset with a batch

[13] See open issues with `SyncReplicasOptimizer` on Github.

size of 128. The overall performance, in terms of images per second, is reported in Figure 7.

For the asynchronous update, Tensorflow and MXNet showed good scalability. Other frameworks do not support it in the current form. This is due to the nature of asynchronous update that no need to wait for the peer workers. However, we expect to see the network bottleneck with larger scale experiments.

Compared to the asynchronous update, the synchronous update runs have slightly worse scalability due to the bottleneck that the update happens only when all workers have finished their computation for each iteration. Interestingly, in the multi-node case, MXNet and Chainer show the best scalability among all frameworks. For Chainer, as opposed to its performance in the multi-GPU version, ChainerMN uses NCCL which significantly improves the scalability. CNTK performance is sub-linear compared to the performance of Chainer. However CNTK also provided non-commercial licensed 1-bit SGD, which accelerate the synchronization [17]. But we do not include it in the analysis due to the limited usability. Caffe2 is also sub-linear, this is caused by the single-node setup of Redis server. Further study of the configuration of Redis is needed to fully exploit the computational efficiency of Caffe2.

Overall, Tensorflow and MXNet work well with the asynchronous update, and Chainer and MXNet are good with the synchronous update. And as expected, the asynchronous update has better scalability. We did not explore the convergence and optimization aspects for the different update methods.

4.7 Real Data performance

In this section, we examined the performance of different frameworks with the real data, including speed and convergence. We set up the same ResNet-50 model for the Cifar-10 dataset. We used the full training dataset to train the model and reported both the training and testing accuracy at each epoch along with the time spent on the training and inference steps. We used the same initialization[14] and the SGD update with a learning rate of 0.1 and a batch size of 128. The dataset is preloaded in memory before the training. We repeated the same training procedure 5 times and plotted all the results in Figure 8.

In Figure 8, the first figure shows the training accuracy. Tensorflow has the highest convergence rate. Caffe2 and CNTK have a larger variance in training accuracy compared to the other frameworks. This is related to how the accuracy is calculated, for CNTK and Tensorflow we used the full dataset, whereas for the others we used the built-in function to report the performance.

The second figure shows the test accuracy. Tensorflow and Caffe2 have high accuracy which follow similar performance in training accuracy. MXNet's accuracy is worse than models trained with others, although their training accuracy is comparable. Just as it was the case in training accuracy, CNTK has larger variance in the testing accuracy. Meanwhile, Caffe2, which had a larger variance in training accuracy, has a stable test accuracy.

The third figure shows the training time spent on each epoch. MXNET has the best performance persumablly due to the incomplete accuracy compuation. Caffe2 is also fast partially because it the data is already formatted in LMDB format. CNTK and Tensorflow are slower than the expected training speed, as the inference speed is lower and we reported the accuracy of the full dataset.

Overall, we found that different frameworks have different results. Users need to be cautious when implementing and reusing models for different frameworks and should check the performance and accuracy metrics against their datasets.

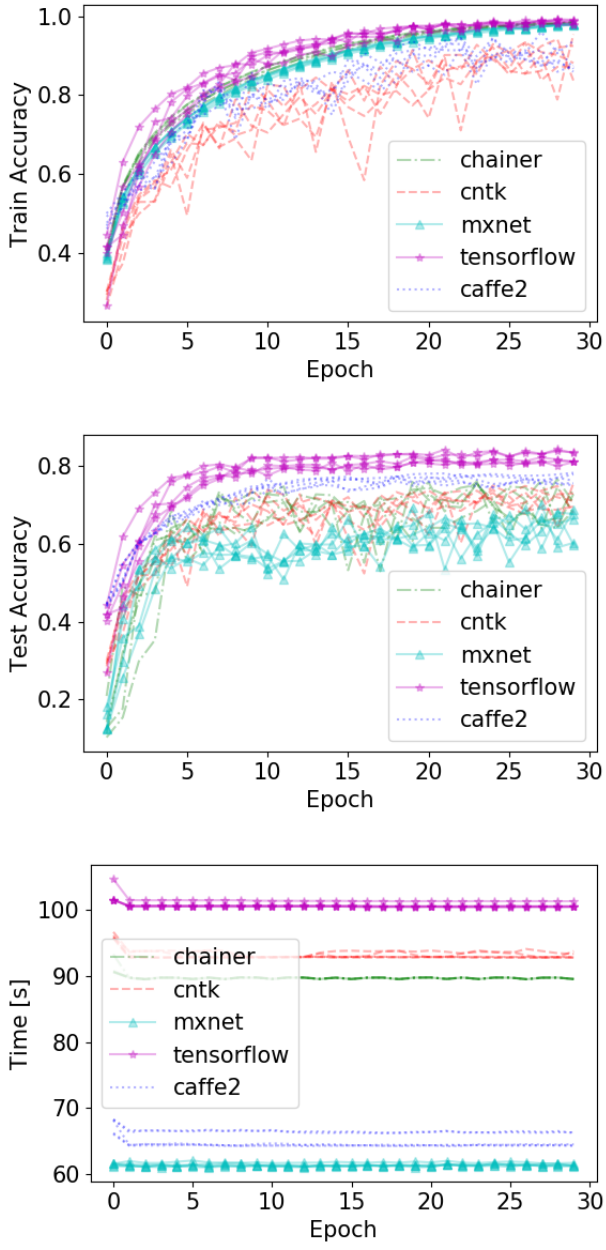

Figure 8: Convergence of ResNet-50 on CIFAR-10 for DL frameworks with single GPU.

[14] MXNet is slightly different as it uses the normal distribution without truncating for the initialization

5 DISCUSSION AND CONCLUSION

In this work, we first compared the usability of different DL frameworks. From the API perspective, Tensorflow provides the richest library compared to all others. Chainer and MXNet also provide large collections of DL functions to choose from. Caffe2 on the other hand lags behind in terms of functionality and the documentation.

CNTK, Chainer, and MXNet are good choices to get started easily in training and using distributed models. The installation process is straightforward and high-level APIs are sufficient to build models easily. More importantly, within those frameworks, it is simple to convert existing code into the multi-GPU/-node version. Among the three, CNTK has a large pre-built DNN library, which provides an additional advantage.

In terms of speed, MXNet CNTK, and Tensorflow are overall much better than the others. However, drawing a general conclusion in terms of speed is difficult. We found that the results are sensitive to different choices and problem sets, such as whether the network is used for training or inference, the image and batch sizes, and if the computing is on the CPU or the GPU. We provide the source code used in this paper to the public to facilitate the community to better measure the performance of the framework for their targeted problems.

For distributed training, the asynchronous update approach shows better scalability than the synchronous update. We found that the NCCL library has a critical impact on the training performance for Chainer. And that Chainer stands out for its scalability for multi-node training with the synchronous update.

We also tested the performance of different frameworks on the real Cifar-10 dataset. We found that Tensorflow converged to the best training and test accuracy within 30 epochs, however, it is also the slowest in terms of images/second. Tensorflow has reasonable speed and convergence. For the rest of frameworks, we found that their results have very large variance.

At the end, we found the CNTK framework to be overall the framework of choice for a broad range of applications especially with respect to usability and performance. However, users need to be cautious to choose a framework for performance critical tasks.

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 Mane, 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 Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. (mar 2016). arXiv:1603.04467 http://arxiv.org/abs/1603.04467

[2] Takuya Akiba, Keisuke Fukuda, and Shuji Suzuki. 2017. ChainerMN: scalable distributed deep learning framework. *arXiv preprint arXiv:1710.11351* (2017).

[3] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. 2016. Comparative Study of Deep Learning Software Frameworks. In *ICLR workshop poster*. arXiv:1511.06435 http://arxiv.org/abs/1511.06435

[4] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting Distributed Synchronous SGD. (2016). https://arxiv.org/pdf/1604.00981.pdf

[5] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12, Aug (2011), 2493–2537.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. (2015), 1026–1034. https://www.cv-foundation.org/openaccess/content

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[8] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[9] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y Ng. 2015. An Empirical Evaluation of Deep Learning on Highway Driving. (apr 2015). arXiv:1504.01716 http://arxiv.org/abs/1504.01716

[10] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[11] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference for Learning Representations*. arXiv:1412.6980 http://arxiv.org/abs/1412.6980

[12] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. (2009).

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[14] He Ma, Fei Mao, and Graham W. Taylor. 2016. Theano-MPI: a Theano-based Distributed Training Framework. *arXiv preprint arXiv:1605.08325* (2016).

[15] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for Activation Functions. (oct 2017). arXiv:1710.05941 http://arxiv.org/abs/1710.05941

[16] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (oct 1986), 533–536. https://doi.org/10.1038/323533a0

[17] 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 *Fifteenth Annual Conference of the International Speech Communication Association*. http://www.isca-speech.org/archive/archive

[18] Dinggang Shen, Guorong Wu, and Heung-Il Suk. 2017. Deep Learning in Medical Image Analysis. *Annual Review of Biomedical Engineering* 19, 1 (jun 2017), 221–248. https://doi.org/10.1146/annurev-bioeng-071516-044442

[19] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking State-of-the-Art Deep Learning Software Tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 99–104. https://doi.org/10.1109/CCBD.2016.029

[20] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, Vol. 5.

[21] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, et al. 2014. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112* (2014).

[22] Shang Xuan Zou, Chun Yen Chen, Jui Lin Wu, Chun Nan Chou, Chia Chin Tsao, Kuan Chieh Tung, Ting Wei Lin, Cheng Lung Sung, and Edward Y. Chang. 2017. Distributed training large-scale deep architectures. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 10604 LNAI. 18–32. https://doi.org/10.1007/978-3-319-69179-4_2 arXiv:1709.06622

[23] Shang-Xuan Zou, Chun-Yen Chen, Jui-Lin Wu, Chun-Nan Chou, Chia-Chin Tsao, Kuan-Chieh Tung, Ting-Wei Lin, Cheng-Lung Sung, and Edward Y Chang. 2017. Distributed training large-scale deep architectures. In *International Conference on Advanced Data Mining and Applications*. Springer, 18–32.