Benchmarking State-of-the-Art Deep Learning Software Tools

Shaohuai Shi, Qiang Wang, Pengfei Xu, Xiaowen Chu
Department of Computer Science, Hong Kong Baptist University
{*csshshi, qiangwang, pengfeixu, chxw*}*@comp.hkbu.edu.hk*

***Abstract*—Deep learning has been shown as a successful machine learning method for a variety of tasks, and its popularity results in numerous open-source deep learning software tools coming to public. Training a deep network is usually a very time-consuming process. To address the huge computational challenge in deep learning, many tools exploit hardware features such as multi-core CPUs and many-core GPUs to shorten the training and inference time. However, different tools exhibit different features and running performance when they train different types of deep networks on different hardware platforms, making it difficult for end users to select an appropriate pair of software and hardware. In this paper, we present our attempt to benchmark several state-of-the-art GPU-accelerated deep learning software tools, including Caffe, CNTK, TensorFlow, and Torch. We focus on evaluating the running time performance (i.e., speed) of these tools with three popular types of neural networks on two representative CPU platforms and three representative GPU platforms. Our contribution is two-fold. First, for end users of deep learning software tools, our benchmarking results can serve as a reference to selecting appropriate hardware platforms and software tools. Second, for developers of deep learning software tools, our in-depth analysis points out possible future directions to further optimize the running performance.**

***Keywords*-Deep Learning; GPU; Feed-forward Neural Networks; Convolutional Neural Networks; Recurrent Neural Networks**

I. Introduction

In the past decade, deep learning has been successfully applied in diverse application domains including computer vision, image classification, speech recognition, natural language processing, etc. The success of deep learning is attributed to its high representational ability of input data, by using various layers of artificial neurons [1]. GPUs have played a key role in the success of deep learning by significantly reducing the training time [2]. In order to improve the efficiency in developing new deep neural networks, many open-source deep learning toolkits have been recently developed, including Caffe from UC Berkeley [3], CNTK from Microsoft [4], TensorFlow (TF) from Google [5], Torch [6], and many other tools like Theano [7], MXNet [8], etc. All these tools support multi-core CPUs and many-core GPUs for high-performance. One of the main tasks of deep learning is to learn a huge number of weights, which can be implemented by vector or matrix operations. TensorFlow uses Eigen [9] as accelerated matrix operation library, while Caffe, CNTK and Torch employ OpenBLAS [10] or cuBLAS [11] to speed up matrix related calculations. All the mentioned tools have recently started to use cuDNN [12] library to perform convolution operations on GPUs. However, because of the difference among optimization methods and memory management schemes by vendors, these tools exhibit different running time performance even when training the same neural network on the same hardware platform. Furthermore, the performance of a tool also varies a lot when training different types of networks, or using different types of hardware.

Given the diversity of deep learning tools and hardware platforms, it could be difficult for end users to choose an appropriate tool to carry out their deep learning tasks. In this paper, we present out attempt to benchmark three major types of deep neural networks, namely fully connected feed-forward neural networks (FCNs) [13], convolutional neural networks (CNNs) [14][15][16], and recurrent neural networks (RNNs) [17][18][19], on four state-of-the-art GPU-accelerated tools: Caffe, CNTK, TensorFlow and Torch. We compare their running performance on two representative CPUs (a desktop CPU and a server-grade CPU) and three representative NVIDIA GPUs (with architectures of Kepler, Maxwell, and Pascal).

For each type of networks, we choose a small-size network and a large-size network for evaluation.[1] Our major findings are summarized as follows[2]: (1) In general, the performance does not scale very well on many-core CPUs. In many cases, the performance of using 16 CPU cores is only slightly better than that of using 4 or 8 CPU cores. (2) We observe no obvious single winner on CPU-only platforms. The performance of each tool varies among different neural networks, CPU types, and different number of threads. E.g., Torch performs the best on FCNs; Caffe performs the best on AlexNet; Torch performs the best on ResNet-50; and CNTK and TF perform much better than Torch on RNNs. (3) All tools can achieve significant speedup by using contemporary GPUs. We observe 10-30X speedup by comparing the best GPU result to the best CPU result. With GPUs, Caffe, CNTK, and Torch have similar performance on FCNs and are faster than TF; for CNNs,

[1]Our source code and experimental data can be downloaded from http://www.comp.hkbu.edu.hk/~chxw/dlbench.html.

[2]The software tools are being upgraded frequently. The findings are based on our own experimental platforms and only apply to the software versions specified in the paper. We will regularly update our benchmarking results for new software versions on our website.

Torch consistently performs very well; but for RNNs, CNTK and TF perform much better than Torch. (4) Among the three GPU platforms, GTX1080 performs the best in most cases, due to its highest computational power. (5) The performance is also affected by the design of configuration files. E.g., CNTK allows the end users to fine-tune the system and trade off GPU memory for better computing efficiency.

The rest of the paper is organized as follows. Section II presents the background and related work. Section III introduces our benchmarking platform and methodology. Experimental results are presented in Section IV, followed by our discussion in Section V. We conclude the paper and discuss our future work in Section VI.

II. Background and Related Work

With the fast development of deep learning techniques, numerous deep neural networks including fully connected feed-forward neural networks (FCNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), restricted boltzmann machine (RBM) have been developed for different applications [20]. In this paper, we focus on analyzing the running time performance (i.e., speed) of three types of neural networks, namely FCNs, CNNs and RNNs. FCN has a long history dated back to 1980s when the backpropagation (BP) algorithm [21] was first developed. And for CNN and RNN, they have been revealed strong power on the applications of image recognition and natural language processing respectively [15][16]. FCN could result in large number of parameters. CNNs build a convolutional layer by using a set of kernels to reduce parameters, and RNNs allow cyclical connections of the units in the network. With the growing success of deep learning, there comes out many popular open-source software tools, among which Caffe, CNTK, TensorFlow and Torch are examples of the most active and popular ones. These tools all support multi-core CPUs and many-core GPUs.

Caffe is developed by Berkeley Vision and Learning Center (BVLC) and has become open source since 2014. The authors [3] claim that Caffe can process 40 million images per day with GPU-accelerated version on a single NVIDIA K40 or Titan GPU. After integrated with cuDNN, it achieves another 1.3x speedup on NVIDIA K40 card [12].

CNTK is a unified computational network toolkit developed by Microsoft Research, which supports many popular neural networks. At December 2015, it was officially reported that the performance of CNTK with multiple GPUs on a fully connected 4-layer neural network was much better than Caffe, TensorFlow, Theano and Torch [22].

TensorFlow is developed by Google which has integrated most common units in deep learning framework. It supports many up-to-date networks such as CNNs and RNNs with different settings. TensorFlow is designed for remarkable flexibility, portability, and high efficiency of equipped hardware.

Torch is a scientific computing framework which provides data structures for the most useful components in machine learning algorithms such as multi-dimensional tensors and mathematical operations over them.

To accelerate the training speed of deep neural networks, both CPU SSE techniques and floating-point SIMD models have been used to implement deep learning algorithms [23], which achieved 3x speedup over an optimized floating-point baseline. Andre Viebke et al. also exploited thread- and SIMD-parallelism of Intel Xeon Phi processors to speedup the training of CNNs [24]. Considering parallel algorithms for GPUs, Jeffrey Dean et al. [25] proposed a large scaled distribute deep networks and developed two algorithms (i.e., Downpour SGD and Sandblaster L-BFGS) that can be easily implemented on clusters with thousands of machines including GPU machines. Another way to accelerate training speed is to reduce the number of learning parameters. Song Han et al. [26] used the method of pruning redundant connections to reduce parameters without losing network representational ability, which could reduce the number of parameters of AlexNet from 61 millions to 6.7 millions. Bahrampour et al. [27] did the similar work with us, but they only used a single architecture of GPU (i.e., NVIDIA Maxwell Titan X) and old version softwares (e.g., cuDNN v2, v3). As a complement, we use three recent GPU architectures and benchmark on some new networks (e.g., ResNet-50) and software library (e.g., cuDNN v4).

III. Experimental Methods

For each type of neural network, we set up a small-size network and a large-size network to make comparison. One popular and effective way to evaluate the training performance is to measure the time duration of an iteration that processes a mini-batch of input data. In practice, after a certain round of iterations or the convergence of learning, the training progress will be terminated. Therefore, we benchmark the networks by using a range of mini-batch sizes for each network on these tools. For each mini-batch size, we run numerous iterations and evaluate their average speed. The methods of time measurement for each tool are as follows:

- Caffe: use "caffe train" command to train a specified network, and then calculate the average time difference between two consecutive iterations.
- CNTK: similar to Caffe, but we need to exclude the first epoch which may include the time of disk I/O.
- TensorFlow: use "datetime" function to calculate the average iteration time in source scripts.
- Torch: the same as TensorFlow.

All the toolkits provide very flexible programming APIs or configuration options to do performance optimization. For example, in CNTK, we may specify "maxTempMemSizeInSamplesForCNN" option in configuration file to control the size of temporary memory used by CNNs which may result

in slightly worse efficiency but less memory requirement. TensorFlow and Torch, which are API based frameworks, have rich APIs for users to choose for computations. In other words, there may exist different APIs performing the same operations. As a result, we need to point it out that the performances reported in this paper are based on our understanding of usage of these tools and are not necessarily the best that can be achieved.

The software versions and related libraries are shown in Table I.

TABLE I: THE SOFTWARES USED FOR EXPERIMENTS.

Software	Version	CPU BLAS LIB	cuDNN
Caffe	1.0.0-rc3	OpenBLASv0.2.18	v4
CNTK	1.5rc	OpenBLASv0.2.18	v4
TensorFlow	0.9.0	Eigen 3.2.8	v4
Torch	423cfba	OpenBLASv0.2.18	v4

TABLE II: THE EXPERIMENTAL SETUP OF NEURAL NETWORKS.

Networks		Input	Output	Layers	Parameters
FCN	FCN-5	26,752	26,752	5	55 millions
	FCN-8	26,752	26,752	8	58 millions
CNN	AlexNet	150,528	1,000	4	61 millions
	ResNet-50	150,528	1,000	50	3.8 billions
RNN	LSTM-32	10,000	10,000	2	13 millions
	LSTM-64	10,000	10,000	2	13 millions

TABLE III: THE EXPERIMENTAL SETUP OF HARDWARE.

Computational Unit	Cores	Memory	OS	CUDA
Intel CPU i7-3820	4	64 GB	Ubuntu 14.04	-
Intel CPU E5-2630x2	16	128 GB	CentOS 7.2	-
NVIDIA GTX 980	2048	4 GB	Ubuntu 14.04	7.5
NVIDIA GTX 1080	2560	8 GB	Ubuntu 14.04	8.0
NVIDIA Telsa K80	2496	12 GB	CentOS 7.2	7.5

Neural networks. For fully connected network, a 5-layer neural network (FCN-5) and an 8-layer neural network (FCN-8) are constructed. For CNNs, we choose the classical AlexNet [14] and the recently proposed ResNet-50 [28]. For RNNs, considering that the main computation complexity is related to the length of input sequence, we select 2 LSTM [17] layers for testing, with input length of 32 (LSTM-32) and 64 (LSTM-64) respectively. The network configuration details can be found in Table II.

Hardware. We test two types of multi-core CPUs, one PC with a quad-core desktop CPU (i.e., Intel i7-3820 CPU @ 3.60GHz) and one server with dual 8-core CPUs (i.e., Intel Xeon CPU E5-2630 v3 @ 2.40GHz), by setting different number of threads. We also test three generations of GPU cards, NVIDIA GTX 980 with Maxwell architecture, GTX 1080 with Pascal architecture, and Telsa K80 with Kepler architecture. Notice that we only use one of the two GK210 chips of K80 GPU in this study. We leave it as a future work to benchmark the performance of multiple GPUs. The details of our hardware configurations are shown in Table III.

IV. RESULTS

Since the official Caffe does not implement LSTM neuron operation, we do not benchmark Caffe on RNNs. For CNTK, batch normalization (an operation of ResNet-50) is not supported on CPU version, therefore, we do not benchmark CPU-only CNTK on ResNet-50. For TensorFlow on CNNs with GTX 1080, it is not well supported with cuDNN-v4, so we do not benchmark this case either. For more detailed discussion and future updates, please refer to our online report [29].

To make a full comparison of all toolkits, networks and hardware, we choose a proper mini-batch size for each type of network so that both CPUs and GPUs have their best performance. In our cases, we choose mini-batch sizes of 64, 16 and 128 for FCNs, CNNs and RNNs respectively, and the comparative results with CPU platforms are shown in Fig. 1. The performances of different mini-batch sizes on different GPUs are shown in Fig. 2.

A. CPU Results

On FCNs, Torch has in general the best performance on CPU-only platform. Its performance at 4 threads is even better than other tools with more threads. Both CNTK and Caffe perform slightly worse than Torch. TensorFlow's performance on desktop CPU is not good. It requires 32 threads to achieve similar performance with other software tools.

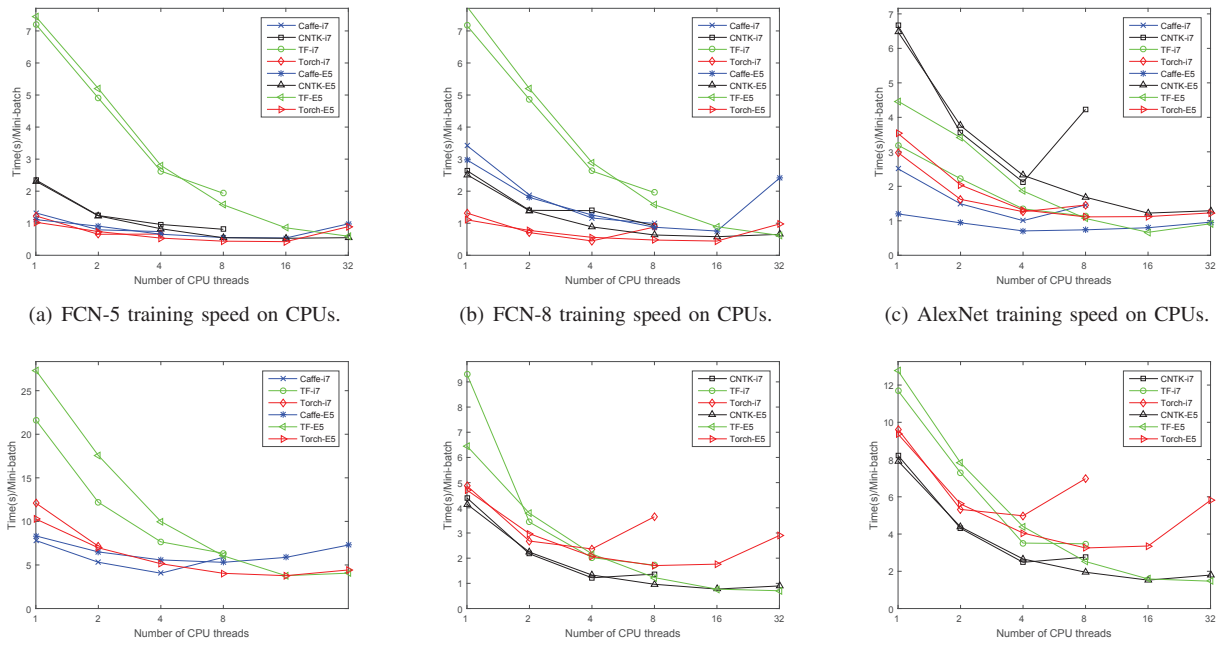
On AlexNet with CPU-only computing resources, Caffe performs the best on the quad-core desktop CPU with 4 threads, while TensorFlow performs the best on the server CPU with 16 threads. On the more complicated ResNet-50, Torch performs the best on both CPUs. Caffe's performance on desktop CPU is very close to Torch, but its performance on server CPU becomes 40% slower than Torch and TensorFlow.

On RNNs, CNTK performs consistently very well. It is almost twice as fast as Torch. TensorFlow's performance is slower than CNTK on desktop CPU, but becomes slightly better than CNTK on server CPU when using 32 threads.

In general, the speedup achieved by multi-threading on multi-core CPUs are diminishing with more and more threads. In most cases, using 16 threads can only slightly reduce the training time as compared to using 4 or 8 threads.

B. GPU Results

The FCNs testing results on GPUs are displayed in Fig. 2(a) and Fig. 2(b). Caffe and CNTK have similar results, which are better than that of TensorFlow and Torch. TensorFlow has a poorer performance on these kinds of networks, which is about 2 times slower than Caffe and CNTK.



(a) FCN-5 training speed on CPUs. (b) FCN-8 training speed on CPUs. (c) AlexNet training speed on CPUs.

(d) ResNet-50 training speed on CPUs. (e) LSTM-32 training speed on CPUs. (f) LSTM-64 training speed on CPUs.

Figure 1: Training time on CPUs in second. (The mini-batch sizes for FCNs, CNNs and RNNs are 64, 16 and 128 respectively. And the legend with "-i7" means results on desktop CPU of i7, and "-E5" means results on server CPU of E5-2630).

The CNNs' results (i.e., AlexNet and ResNet-50) on GPUs are shown in Fig. 2(c) and 2(d). When training the AlexNet on GTX 980 and K80 cards, Caffe has a better performance than any others with all the mini-batch sizes. However, CNTK is on par with TensorFlow and Torch on ResNet-50, and slightly better than Caffe on GTX 980 and K80 cards.

As for RNNs, we have different settings of input sequence lengths which are 32 and 64, and mini-batch size varying from 64 to 256. According to our results demonstrated in Fig. 2(e) and Fig. 2(f), CNTK achieves the best performance for all available settings.

On different GPUs, GTX 1080 card performs the best in most cases, and the average running time on GTX 1080 is around 1.5x shorter than on GTX 980. However, when running much larger network (e.g., ResNet-50), with the mini-batch size getting larger, much more GPU memory is needed to do the computation, which results in crash for some tools on GPUs with not enough memory. For example, with mini-batch size of 16, Caffe cannot run on GTX 980. The programs with CNTK are also crashed at not less than 64 mini-batch size on both GTX 980 and GTX 1080 cards. Torch can run all the configured mini-batch size except 64 on GTX 980 card. All the benchmark results can be generated with TensorFlow on all GPU cards except the case of ResNet-50 on GTX 980 card with the mini-batch size of 64. It seems that TensorFlow and Torch have a better GPU memory management strategy on running convolutional neural networks.

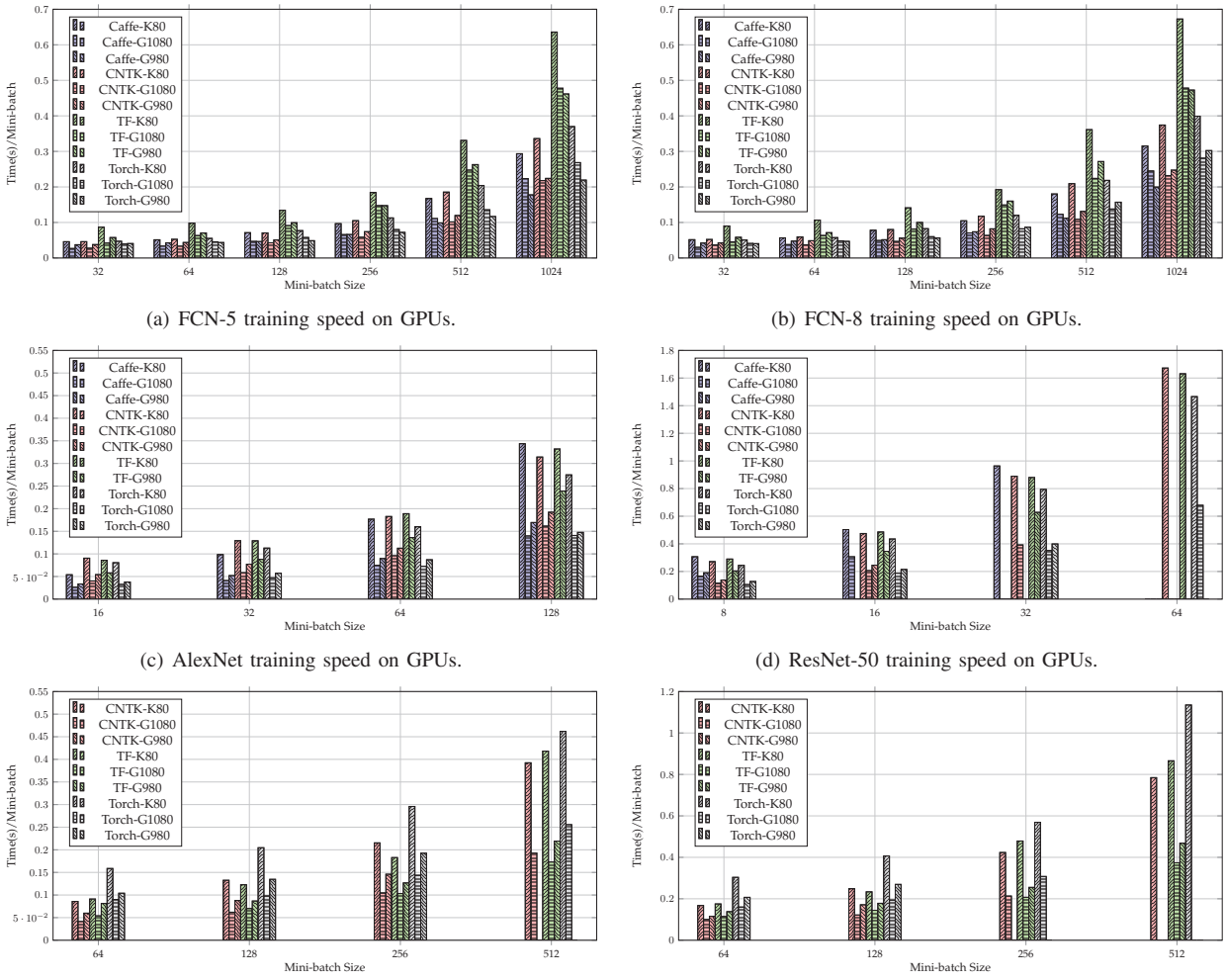
To make a simple comparison between GPUs and CPUs parallelization, Fig. 1 and Fig. 2 show that GPU version has an obviously high performance than CPU versions.

V. Discussion

Considering parallelization on CPUs, the number of computation threads are recommended to be not larger than the number of physical CPU cores. On single core CPU version, the performance of TensorFlow is much worse than others, simply because the current BLAS library (i.e., Eigen-3.2) used by TensorFlow v0.9.0 does not support AVX (x86_64) [9], while OpenBLAS which is used by other three tools supports [10]. Fortunately, Eigen-3.3 adds support for AVX (x86_64) and it would be applied to newer version of TensorFlow.

With GPU computing resources, all the deep learning tools mentioned achieve very high speedups compared to their CPU-only versions. This is not surprising because the performance of matrix multiplication and FFT on the tested GPUs are significantly better than on CPUs.

On FCNs, Caffe and CNTK performs a little better than TensorFlow and Torch with one GPU. In general, training a network involves a two-phase computation (i.e., feed-forward and backward propagation). In feed-forward phase, matrix multiplications are the most time-consuming operations, and cuBLAS API: cublasSgemm is adopted by



(a) FCN-5 training speed on GPUs.

(b) FCN-8 training speed on GPUs.

(c) AlexNet training speed on GPUs.

(d) ResNet-50 training speed on GPUs.

(e) LSTM-32 training speed on GPUs.

(f) LSTM-64 training speed on GPUs.

Figure 2: GPU training speed comparison on Deep Learning Tools.

all the four tools. However, if we want to multiply matrix A and the transpose of matrix B, we can set the second parameter of cublasSgemm API to CUBLAS_OP_T, which will apply in-place matrix transpose and result in up to 3 times slower performance as compared to matrix multiply without transpose (e.g., $C = A \times B^T$, where $A \in R^{1024 \times 26752}$ and $B \in R^{2048 \times 26752}$). This is because in-place matrix transpose is very time-consuming. CNTK and TensorFlow construct its data structure to call cublasSgemm use CUBLAS_OP_N, while Caffe and Torch use CUBLAS_OP_T. In the phase of backward propagation, it needs to use matrix multiplication to calculate the gradients and use element-wise matrix operation to update the parameters. When it comes to low efficiency computation of A times transposition of B by calling cuBLAS, it may be better if we transpose B first (out-place if the GPU has enough memory) and then apply matrix multiply. Furthermore, the cublasSgemm API provides the full support to backward propagation because it adds a scaled (parameter beta as scalar) matrix after matrix multiplication. So if we merged gradients calculation and update operation into a single GPU kernel, the calculation efficiency could be improved. To optimize the efficiency of FCNs, it is possible to use cublasSgemm API without transpose and use cublasSgemm to calculate gradients and do update operations at the same time.

On CNNs, all the toolkits use cuDNN library to do convolution operations. Though the same API calls are used, the parameters may result in different GPU kernels to use. We found that FFT is a better solution for convolution operations compared to performing convolution directly. After FFT of a matrix, convolution calculation can be transformed into inner product operation which is much faster.

On RNNs with LSTM, CNTK performs better than TensorFlow and Torch, both of which achieve similar performance. To launch training procedure of LSTM, Torch executes lots of Pointwise kernels of basic operations such

as multiplication, addition, sigmoid, etc. on tensors designed by itself. Regarding the workload of kernels, Torch gives more than 50 blocks of which size is batch size. In this way Torch can somehow fully utilize the computation resources of GPU. We know that TensorFlow organizes all the computations as graph operations [5], which is also indicated by the kernels launched during the training procedure to some extend. There are a large number of TensorAssignOp kernels with different types of operations also including scalar_sum, scalar_product etc. on tensors. As for CNTK, some specific kernels for training LSTM designed by itself are also very different from those of Torch and TensorFlow. Its brain scripts are very flexible to customize the neural networks.

VI. Conclusion and Future Work

This work aims to evaluate the running time performance of a set of modern deep learning software tools and see how they perform on different types of neural networks and different hardware platforms. Our experimental results show that all tested tools can make good use of GPUs to achieve significant speedup over their CPU counterparts. However, there is no single software tool that can consistently outperform others, which implies that there exist some opportunities to further optimize the performance.

We have two directions of future work on the agenda. First of all, we plan to benchmark more deep learning software tools (such as MXNet, Paddle) and hardware platforms (such as AMD's GPU, and Intel Xeon Phi). Secondly, we plan to benchmark the performance of multi-GPUs within a server and across a high-performance cluster.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] L. Deng, "Three classes of deep learning architectures and their applications: a tutorial survey," *APSIPA transactions on signal and information processing*, 2012.

[3] 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*, 2014, pp. 675–678.

[4] D. Yu, A. Eversole, M. Seltzer, K. Yao, Z. Huang, B. Guenter, O. Kuchaiev, Y. Zhang, F. Seide, H. Wang *et al.*, "An introduction to computational networks and the computational network toolkit," Technical report, Tech. Rep. MSR, Microsoft Research, 2014, 2014. research. microsoft. com/apps/pubs, Tech. Rep., 2014.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems, 2015," *Software available from tensorflow. org*, vol. 1, 2015.

[6] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[7] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, 2016.

[8] 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.

[9] "Eigen," http://eigen.tuxfamily.org/index.php, accessed: 2016-07-03.

[10] "Openblas," http://www.openblas.net/, accessed: 2016-07-12.

[11] C. Toolkit, "4.0 cublas library," *Nvidia Corporation*, 2011.

[12] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[15] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1422–1432.

[16] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[17] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[18] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks." in *ICML*, vol. 14, 2014, pp. 1764–1772.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

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

[21] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks, 1989. IJCNN., International Joint Conference on*. IEEE, 1989, pp. 593–605.

[22] X. Huang, "Microsoft computational network toolkit offers most efficient distributed deep learning computational performance," https://goo.gl/9UUwVn, 2015, accessed: 2016-07-12.

[23] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[24] A. Viebke and S. Pllana, "The potential of the intel (r) xeon phi for supervised deep learning," in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*. IEEE, 2015, pp. 758–765.

[25] 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*, 2012, pp. 1223–1231.

[26] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[27] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of deep learning software frameworks," *arXiv preprint arXiv:1511.06435*, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[29] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," *arXiv preprint arXiv:1608.07249*, 2016.