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 time. However, different tools exhibit different features and running performance when training different types of deep networks on different hardware platforms, which makes it difficult for end users to select an appropriate pair of software and hardware. In this paper, we aim to make a comparative study of the state-of-the-art GPU-accelerated deep learning software tools, including Caffe, CNTK, TensorFlow, and Torch. We benchmark the running performance of these tools with three popular types of neural networks on two CPU platforms and three GPU platforms. Our contribution is two-fold. First, for end users of deep learning software tools, our benchmarking results can serve as a guide 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.**

***Index Terms*—Deep Learning; GPU; Feed-forward Neural Networks; Convolutional Neural Networks; Recurrent Neural Networks**

1. Introduction

In the past decade, deep learning has been successfully applied in diverse areas including computer vision, 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 neurals [1]. GPUs have played a key role in the success of deep learning by significantly reducing the training time [2]. In order to increase the efficiency in developing deep learning methods, there are a number of open-source deep learning toolkits including Caffe from UC Berkeley [3], CNTK from Microsoft [4], TensorFlow 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. One of the main tasks of deep learning is to learn a number of weights in each layer of network, which can be implemented by vector or matrix operations. TensorFlow uses Eigen [9] as accelerated matrix operation library, while Caffe, CNTK, Torch employ OpenBLAS [10] or cuBLAS [11] to speed up matrix related calculations. All the mentioned tools import cuDNN [12], which is a GPU-accelerated deep learning library, for their neural network computing. However, because of the difference of optimization methods by vendors, these tools exhibit different running performance even when training the same neural network on the same hardware platform. Furthermore, the performance of a tool also changes 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 benchmark three major types of deep neural networks (i.e., fully connected neural networks (FCNs) [13], convolutional neural networks (CNNs) [14][15][16], and recurrent neural networks (RNNs) [17][18][15]) on state-of-the-art GPU-accelerated tools (i.e., Caffe, CNTK, TensorFlow and Torch), and analyze their advantage and disadvantage on both CPUs and GPUs, in terms of running time performance.

For each type of networks, we benchmark the networks of both small size and large size.[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) On CPU-only platforms, the performance of each tool varies among different neural networks, CPU types, and different number of threads; and there is no obvious single winner. 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 see 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 obviously faster than TF; for CNNs, the performance really depends on the type of GPU, and 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,

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

[2]These findings are based on our own experimental platforms and only apply to the software versions specified in the paper.

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 2 presents the background and related work. Section 3 introduces our benchmarking platform and methodology. Experimental results are presented in Section 4, followed by our discussion in Section 5. We conclude the paper and discuss our future work in Section 6.

2. Background and Related Work

With the fast development of deep learning techniques, numerous deep neural networks including fully connected neural networks (FCNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), restricted boltzmann machine (RBM) have been developed for different applications [19]. In this paper, we focus on analyzing the running performance (or 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 [20] 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][16].

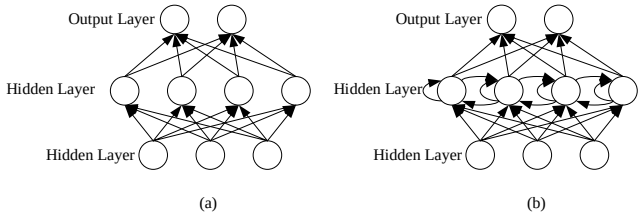


Figure 1. Example of (a) Fully Connected Network and (b) Recurrent Neural Network.

FCN is a feed-forward neural network, the first successful use of which is the ZIP codes recognition by Yann LeCun et al. in 1989 [13]. Assume that an FCN F has I input neurons and O output neurons, and there are H hidden layers, where the number of neurons is N_i (i = 1, 2, ..., H). The total number of weights in F is:

$$I \times N_1 + \sum_{i=1}^{H-2} N_i \times N_{i+1} + N_{H-1} \times O. \qquad (1)$$

To reduce the total number of parameters in each layer, CNNs build a convolutional layer by using a set of kernels, and the parameters of each kernel are shared across entire field (e.g., a channel of color image). RNNs allow cyclical connections of the units in the network [18][15][16]. Furthermore, Long-Short Term Memory(LSTM) [21][17] has been proposed to address vanished and exploding gradients on RNNs.

RNN allows cyclical connections of the units in the network, as illustrated in Figure 1(b). RNN can link the entire historical input sequence to each output and find the relationship between the contextual features of inputs and the output. With this characteristic, RNN can maintain the information given by the former inputs similar with memory during the training period of one single sample. Furthermore, to address the training difficulty of vanished and exploding gradients, Long-short Term Memory(LSTM) [17] has been proposed to record and discard the information properly. RNN with LSTM units has been proved most successful in handling tasks of speech recognition and natural language processing [15][16].

Eq. (1) shows that fully-connected layers easily lead to huge amount of parameters to learn. To reduce the total number of parameters in each layer, CNNs use convolutional layers in which a set of kernels are designed and the parameters of each kernel are shared across entire field. Starting from the LeNet architecture, CNNs have accomplished a lot of successful tasks including ImageNet classification [14], face recognition [22], and object detection [23]. An example of CNN is shown in Fig. 2 whose number of parameters is up to 61 millions [14].

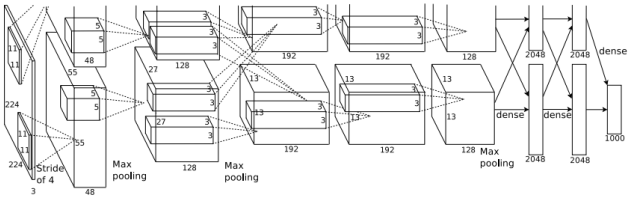


Figure 2. AlexNet [14]: ImageNet 2012 winner.

With the growing success of deep learning, there comes out many popular open source GPU-accelerated software tools, among which Caffe, CNTK, TensorFlow and Torch are examples of the most active and popular ones.

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 speedup about 1.3x on NVIDIA K40 card [12].

CNTK is a unified computational network toolkit developed by Microsoft Research, which supports most popular networks. At December 2015, the official reported a performance result benchmarking on a fully connected 4-layer neural network compared to Caffe, TensorFlow, Theano and Torch, and the result shows that CNTK with multiple GPUs on single node or cross multiple machines achieve a much better speed (about 1.5x speedup) than the other compared toolkits. However, CNTK does not support concatenation operation in some CNNs (e.g., GoogLeNet [24]).

TensorFlow is developed by Google which has integrated most common units in deep learning framework using data flow graphs. It supports many up-to-date networks such as CNNs, 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 CPUs SSE techniques and float points SIMD models are used to implement deep learning algorithms [25], which achieve 3x speedup over optimized floating-point baseline. Andre Viebke et al. also exploit thread- and SIMD-parallelism Intel Xeon Phi processors to speedup training of CNNs [26].Considering parallel algorithms for GPU, Jeffrey Dean et al. [27] proposed a large scaled distribute deep networks and developed two algorithms (i.e., Downpour SGD and Sandblaster L-BFGS) that can be easily running on computing 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. [28] use 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. [29] 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). We use three major architectures of GPU and benchmark on some new networks (e.g., ResNet-50) and softwares (e.g., cuDNN v4), and we also go insight into the source codes to analyze performance.

3. Experimental Methods

For each type of neural network, we set up a small size of network and a large size of 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 our experiments 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 1.

TABLE 1. 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 2. 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 3. 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 [30]. 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 2.

Hardware. We use two types of multi-core CPUs, one quad-core desktop CPU (i.e., Intel i7-3820 CPU @ 3.60GHz) and one 8-core server CPU (i.e., Intel Xeon CPU E5-2630 v3 @ 2.40GHz), to test the performance of tools with different number of threads; and three generations of GPU cards, NVIDIA GTX 980 with Maxwell architecture, GTX 1080 with Pascal architecture, and Telsa K80 with Kepler architecture, are used to compare the performance on different GPU platforms. Notice that we only use one of the two GK210 chips of K80 GPU in this study. In order to avoid host memory dependency of neural network size, the two test machines are equipped with 64GB memory

and 128GB memory respectively. The details of hardware configurations are shown in Table 3.

4. 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.

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 are shown in Table 4. The performances of different mini-batch sizes on different GPUs are shown in Figure 3.

4.1. 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 also 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, except TensorFlow which relies on more threads to saturate the system.

4.2. GPU Results

The FCNs testing results on GPUs are displayed in Fig. 3(a) and Fig. 3(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.

The CNNs' results (i.e., AlexNet and ResNet-50) on GPUs are shown in Fig. 3(c) and 3(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. Furthermore, TensorFlow has a much better speedup on GTX 1080 card when running CNNs, and it performs the best both on AlexNet and ResNet-50.

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 Figure 3(e) and Figure 3(f), CNTK achieves the best performance for all available settings.

On different GPUs, GTX 1080 card performs the best in most of cases, and the average running time on GTX 1080 is around 1.5x shorter than on GTX 980 which is about 2 times faster than running on Tesla K80 card. Specially, TensorFlow has a very outstanding speedup on GTX 1080 card. 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 of some tools on smaller memory GPUs. 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.

In summary of GPU versions, most of the training speeds on GTX 1080 are faster than on GTX 980 and K80. On FCNs, Caffe and CNTK may be better choices, and Caffe and Torch perform better on AlexNet, while CNTK achieves better performance on ResNet-50 and RNNs. When there exists GTX 1080 with CUDA-8.0, TensorFlow performs the best on CNNs.

To make a simple comparison between GPUs and CPUs parallelization, the numbers in Table 4 show that GPU version has an obviously high performance than CPU versions, and the maximum speedup on GTX 1080 reaches to 81 times than the 4 cores CPU version with TensorFlow, and approximate 35 times than 16 cores CPU by using CNTK.

5. Discussion

Considering parallelization on CPUs, the number of computation threads are recommended to be not larger than the number of physical CPU cores. Because if there are lots of calculation tasks in all CPUs, it is difficult for the system to specify idle CPU to do scheduling, which could easily lead to bad performance such as Caffe, CNTK and Torch at 8 threads on 4-core CPUs. On single core CPU version, the performance of TensorFlow is much worse than others, the reason of which is because the BLAS library (i.e., Eigen-3.2) that TensorFlow (v0.9.0) used does not support AVX (x86_64) [9], while OpenBLAS which is used by other three

TABLE 4. COMPARATIVE EXPERIMENT RESULTS (TIME PER MINI-BATCH IN SECOND, AND THE MINI-BATCH SIZES FOR FCNS, CNNS AND RNNS ARE 64, 16 AND 128 RESPECTIVELY).

		Desktop CPU (Threads used)				Server CPU (Threads used)						GPU		
		1	2	4	8	1	2	4	8	16	32	G.980	G.1080	T.K80
FCN-5	Caffe	1.322	0.795	0.739	**0.622**	1.112	0.908	0.656	0.553	**0.539**	0.981	0.042	**0.033**	0.051
	CNTK	2.351	1.240	0.962	**0.810**	2.311	1.229	0.828	0.547	**0.530**	0.549	0.044	**0.033**	0.053
	TF	7.206	4.905	2.626	**1.934**	7.449	5.203	2.804	1.574	0.857	**0.595**	0.070	**0.063**	0.098
	Torch	1.227	0.655	**0.661**	-	1.030	0.741	0.536	0.440	**0.425**	0.892	**0.044**	0.046	0.055
FCN-8	Caffe	3.423	1.883	1.162	**0.993**	2.976	1.805	1.257	0.875	**0.756**	2.405	0.048	**0.038**	0.057
	CNTK	2.641	1.402	1.393	**0.919**	2.514	1.391	0.885	0.633	**0.580**	0.653	0.049	**0.037**	0.059
	TF	7.167	4.863	2.630	**1.955**	7.760	5.198	2.896	1.577	0.892	**0.620**	0.071	**0.063**	0.107
	Torch	1.317	0.707	**0.448**	0.881	1.106	0.774	0.560	0.475	**0.444**	0.976	**0.047**	0.048	0.057
AlexNet	Caffe	1.609	0.999	**0.885**	1.074	1.199	0.942	**0.707**	0.738	0.799	0.960	0.033	**0.026**	0.054
	CNTK	6.541	3.426	2.140	**2.063**	6.476	3.760	2.319	1.684	**1.223**	1.292	0.054	**0.040**	0.091
	TF	3.988	3.127	1.833	**1.462**	4.465	3.471	1.747	1.003	**0.607**	0.835	0.048	**0.018**	0.086
	Torch	4.554	2.483	**2.087**	3.938	3.450	1.878	1.250	1.076	**1.033**	1.076	0.038	**0.029**	0.076
ResNet-50	Caffe	11.529	7.641	8.739	**7.230**	8.332	6.499	5.580	**5.303**	5.870	7.283	-	**0.307**	0.503
	CNTK	-	-	-	-	-	-	-	-	-	-	0.245	**0.207**	0.475
	TF	26.707	16.105	10.093	**8.187**	27.339	17.560	9.989	6.048	**3.773**	4.060	0.346	**0.184**	0.486
	Torch	12.101	**7.147**	-	-	10.275	6.971	5.145	4.043	**3.770**	4.428	0.215	**0.188**	0.435
LSTM-32	CNTK	4.393	2.173	**1.220**	1.369	4.144	2.241	1.331	0.964	**0.773**	0.897	0.088	**0.062**	0.133
	TF	9.306	3.432	2.021	**1.723**	6.453	3.783	2.168	1.229	0.770	**0.706**	0.087	**0.070**	0.123
	Torch	4.872	2.680	**2.366**	3.645	4.704	2.972	2.067	**1.706**	1.763	2.901	0.135	**0.098**	0.205
LSTM-64	CNTK	8.218	4.307	**2.483**	2.762	7.920	4.384	2.662	1.949	**1.527**	1.798	0.171	**0.122**	0.249
	TF	11.699	7.292	3.516	**3.477**	12.760	7.823	4.402	2.525	1.590	**1.469**	0.178	**0.144**	0.234
	Torch	9.623	5.324	**4.980**	6.976	9.365	5.614	4.054	**3.252**	3.358	5.815	0.269	**0.194**	0.407

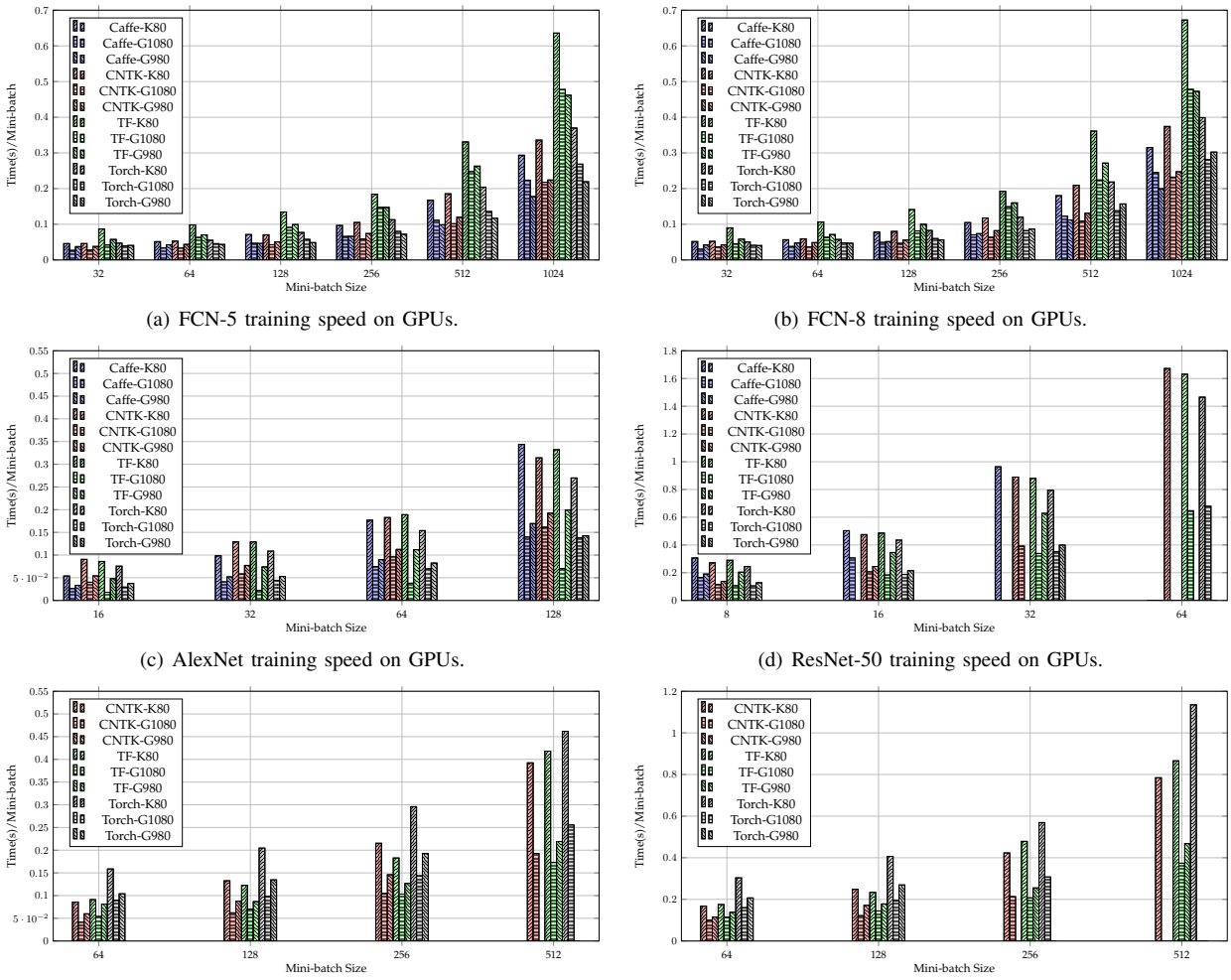
tools supports [10]. Fortunately, Eigen-3.3 adds support for AVX (x86_64) and it would be applied to newer version of TensorFlow.

With GPU computing resources, all the deep learning tools mentioned achieve very high speedups compared to their CPU-only versions because of high parallelization on lots of CUDA cores. The theoretical performance of GTX 1080 is up to 8873 GFLOPS which is much higher than CPUs. In overall, the performance of GTX 1080 is better than GTX 980, and much better than Tesla K80 (with single GK210 chip).

On FCNs, Caffe and CNTK performs a little better than TensorFlow and Torch with one GPU. Our results match with the official claims of CNTK [31]. 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 all the four tools. However, there is a tricky calling parameter of cublasSgemm, which may result in very different performance when doing matrix-matrix multiplication. With the same matrix sizes, if we set the second parameter to CUBLAS_OP_T of cublasSgemm API, the kernel actually used by the API is different and it results in 3 times slower compared to CUBLAS_OP_N in some cases (e.g., $C = A \times B^T$, where $A \in R^{1024\times26752}$ and $B \in R^{2048\times26752}$). 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 and then call A times B API. 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 better 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 determine different kernels to use. We found that on convolution operation, FFT is an optimized way compared to performing convolution directly. After FFT of a matrix, convolution calculation can be transformed into inner product operation which is much faster. TensorFlow acheives much better performance than the other three tools on ResNet-50 on GTX 1080, and FFT is one of the main factor that contributes to its high efficiency.

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,



(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 3. GPU training speed comparison on Deep Learning Tools.

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.

6. Conclusion and Future Work

This work aims to evaluate the running 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 include more deep learning software tools (such as MXNet, Paddle) and hardware platforms (such as AMD's GPU, and Intel Xeon Phi) into this benchmarking study. Secondly, we plan to benchmark the performance of multi-GPUs within a server and across a high-performance cluster.

7. Acknowledgements

We would like to thank the CNTK team for their valuable feedback on this work and for providing the CNTK configuration files.

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

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

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

[22] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[23] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *Journal of Machine Learning Research*, vol. 17, pp. 1–32, 2016.

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

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

[26] 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.

[27] 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.

[28] 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.

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

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

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

8. Appendix

8.1. Revision History

This version (v4):

- Remedy the bug of ResNet-50 configuration in TensorFlow.
- Change time measurement method of Caffe from "caffe time" to "caffe train".
- Add an option of "prefetch=true" to configuration file of CNNs in CNTK.

Version 3 (v3):

- Correct the CUDA version on Table 3 and re-test the experiments.
- Revise minor difference of network configuration and delete some extra operations like dropout on AlexNet.
- On RNN of CNTK, we remove an extra LSTM classification task which is not included in other tools and change the configuration file with "SimpleNetworkBuilder" to customized brain scripts.