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 deep learning end users, our benchmarking results can serve as a guide to selecting appropriate software tool and hardware platform. Second, for deep learning software developers, our in-depth analysis points out possible future directions to further optimize the training performance.**

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

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 from personal group [6], and some 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. TensoFlow 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 confused for 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, TensoFlow 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: (1) In general, all tools do not scale well on many-core CPUs. The performance using 16 CPU cores is only slightly better than using 4 CPU cores. (2) For FCNs and CNNs, all tools can achieve significant speedup by using contemporary GPUs. With GPUs, Caffe performs better on FCNs and CNNs while CNTK performs the best on RNNs [2]. (3) Among the three GPU platforms, GTX1080 always performs the best, and TensorFlow has the highest speedup on GTX1080 on CNNs. (4) The performance is also affected by settings of configuration file, such as CNTK can use tunes to consume more GPU memory to increase computing effeciency.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Section 3 introduces our benchmarking methodology. Experimental results are presented in Section 4, followed by our discussion in Section 5. We conclude the paper in Section 6.

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

[2]CNTK provides customized Brain Script APIs and many optional settings which can get much better efficiency

2. Background and Related Work

With the fast development of deep learning framework, there exists numerous deep neural networks including fully connected neural networks (FCNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), restricted boltzmann machine (RBM), etc. for different applications [19]. In this paper, we focus on analyzing three types of neural networks (i.e., FCNs, CNNs and RNNs). FCN is one of the deep neural networks that have the longest history dated back to 1980s when the backpropagation (BP) algorithm [20] was 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].

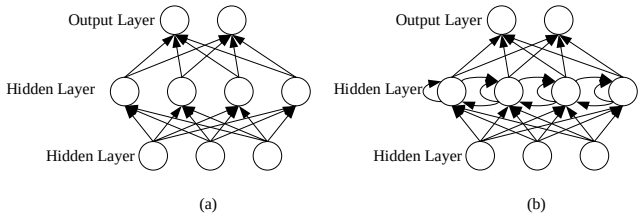


Figure 1. Fully Connected Network(a) and Recurrent Neural Network(b).

Fully connected network 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 which is 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].

The equation (1) shows that a layer with larger size of fully connected network easily leads to huge number of parameters to learn. 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). Starting from the LeNet architecture, CNNs have accomplished a lot of successful tasks including ImageNet classification [14], face recognition [22], and object detection [23], etc. An example of CNN is shown in Fig. 2 whose number of parameters is up to 61 millions [14].

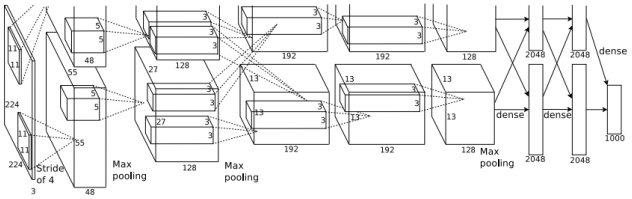


Figure 2. AlexNet [14]: ImageNet 2012 winner.

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

3. Experimental Methods

For each type of network, we set up a small size of network and a large size of network to do comparison. One important character to measure training time of models is the time of an iteration that takes a batch size, called mini-batch, of data. After maximum number of iterations or the convergence of learning, the training progress is 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.

All the toolkits provide a very flexible programming APIs or configuration options to do optimization. For example, in the settings of CNTK, we may specify m̈axTempMemSizeInSamplesForCNNöption in configuration file to control the size of temporary memory used by CNNs which may result in slightly worse effeciency but less memory requirement. TensorFlow and Torch, which are API based frameworks, have rich APIs for users to choose to do computations. In other words, there may exist different APIs doing the same operation. Therefore, the performances achieved in our experiments are based on our understanding of usage of these tools.

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
TensoFlow	0.9.0	Eigen 3.2.8	v4
Torch	423cfba	OpenBLASv0.2.18	v4

In terms of hardware consideration, we run experiments on single CPU, multiple CPUs and single GPU; and multiple generations of GPUs are used to generate differential results on GPU computing. The details of experimental setup are given as follows:

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. Firstly, for fully connected network, a 5 layers neural network (FCN-5) and an 8 layers neural network (FCN-8) are constructed with 3 and 6 hidden layers respectively. Secondly, we choose AlexNet [14] and ResNet-50 [30], both of which are CNNs, to do the testing. Lastly, because of the main computation complexity of RNNs is 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 networks configuration details can be found in Table 2.

Hardware. We use two types of multi-core CPUs, say desktop level (i.e., Intel i7-3820 CPU @ 3.60GHz) and server level (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, its benchmark on RNNs will not be included. For CNTK, batch normalization (an operation of ResNet-50) is not supported on CPU version, therefore, result of ResNet-50 with CPU on CNTK will not be shown.

To make a total 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.

Caffe and Torch have close results on FCNs by using CPU-only platform, and both of them cannot run normally

TABLE 4. COMPARATIVE EXPERIMENT RESULTS (TIME PER MINI-BATCH IN SECOND).

			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	FCN-5	Caffe	0.919	0.495	**0.480**	-	0.769	0.446	0.354	**0.269**	0.287	0.688	0.020	**0.017**	0.028
		CNTK	2.351	1.239	0.961	**0.810**	2.311	1.229	0.827	0.546	**0.530**	0.549	0.043	**0.033**	0.052
		TF	7.205	4.904	2.626	**1.933**	7.449	5.203	2.803	1.574	0.857	**0.594**	0.071	**0.063**	0.098
		Torch	1.227	0.655	**0.661**	-	1.030	0.740	0.535	0.440	**0.425**	0.892	0.044	**0.039**	0.056
	FCN-8	Caffe	1.036	0.857	**0.572**	-	0.889	0.614	0.392	0.320	**0.317**	0.810	0.023	**0.020**	0.033
		CNTK	2.641	1.402	1.393	**0.919**	2.514	1.391	0.885	0.633	**0.580**	0.653	0.048	**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	0.046	**0.046**	0.057
CNN	AlexNet	Caffe	1.549	0.945	**0.879**	1.095	1.121	0.854	**0.644**	0.679	0.743	0.921	0.028	**0.023**	0.050
		CNTK	6.547	3.425	2.248	**2.166**	13.254	7.719	5.426	3.892	**2.938**	3.208	0.055	**0.042**	0.090
		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	Caffe	7.811	5.312	**4.057**	5.876	5.813	4.266	3.971	**3.705**	4.037	5.021	-	**0.215**	0.343
		CNTK	-	-	-	-	-	-	-	-	-	-	0.247	**0.209**	0.477
		TF	18.164	12.249	7.275	**5.685**	19.709	12.409	6.796	4.170	**2.784**	3.696	0.223	**0.087**	0.383
		Torch	12.101	**7.147**	-	-	10.275	6.971	5.145	4.043	**3.770**	4.428	0.215	**0.188**	0.435
RNN	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

when threads usage is set to be bigger than the number of CPU cores on desktop CPU. On the contrary, TensorFlow is slower, but its 8 threads version goes better than its smaller threads versions. TensorFlow performs the worst results on CPUs, and the best training speed of TensorFlow on FCN-8 with 8 CPU threads is at about 1.9 seconds, while Caffe only needs 0.48 second.

On the CPU parallel version, CNTK has a worst performance when training CNNs, but the speeds are doubled faster when the number of threads are doubled (the test machine is with 4 CPU cores). The running times of these tools are slightly shorter with the number of CPU cores being set larger. But the results of Torch with 4 and 8 CPUs are not filled on destktop CPU because the training time goes up to more than 400 seconds which are exceptional results. Caffe achieves a best time efficiency when the amount of CPU threads used are not bigger than the number of CPU cores, while TensorFlow is speedup even at 8 threads on 4 cores CPU. Table 4 indicates the results of CNNs on CPUs, in which the supported three tools have a similar speedup trend when increasing the number of CPU core usage with AlexNet. In ResNet-50, with the single CPU core version, TensorFlow has a dramatically slow training time which is up to 3 times than that of Caffe.

We also compare the parallelization speedup ability of different toolkits on RNNs under different CPU thread setting. CNTK and Torch have similar results, but CNTK is slightly better than Torch when the number of threads are set to larger. The figures show that the effeciency of TensorFlow increases with the number of threads setting larger. For example, performances of TensorFlow are about 2-3 times better than Torch when setting the same amount of threads with CPU cores.

In summary of CPU versions, the CPU of 16 cores has only slightly better effeciency than that of 4 cores, and it is hard to speedup if all CPUs are busy.

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 101 times than the 4 cores CPU version with TensorFlow, and approximate 70 times than 16 cores CPU by using CNTK. All the training speeds on GTX 1080 are faster than other cases on GTX 980 and K80. Fig. 3 shows more detailed comparison on GPUs with a range of mini-batch size.

Fully connected networks testing results on GPUs are displayed in Fig. 3(a) and Fig. 3(b). Caffe has a best time efficiency when the mini-batch size is smaller than 256 on all the GPU cards, and TensorFlow has a significantly poor performance on these kinds of networks when the mini-batch size is small, while CNTK and Torch have a similar performance which is better than TensorFlow.

The benchmark results of AlexNet and ResNet-50, which are CNNs, on GPUs are shown in Fig. 3(c) and 3(d), respectively. In tools level analysis of these two figures, Caffe has a better performance than any others with all the mini-batch sizes, while CNTK displays the poorest performance on AlexNet. On different GPUs, GTX 1080 card always performs the best, 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. 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 and GTX 1080 cards. All the benchmark results can be generated with TensorFlow on all

GPU cards, and it acquires better performance on ResNet-50 than the other three tools. It seems that TensorFlow has a much better strategy of GPU memory utilization on running convolutional neural networks.

As for RNN, 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, specifically near 25% speed up for LSTM-32 compared with CNTK under 64 mini-batch setting. Regarding performance scaling with different mini-batch size, most of the toolkits consume more time with bigger mini-batch size.

5. Discussion

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 4 or 16 cores CPUs, so that the maximum speedup can reach 112x in TensorFlow with GTX 1080 card. In overall, the performance of GTX 1080 is better than GTX 980, and much better than Tesla K80 (with single GK210 chip) corresponding to GPU cores computing performances (i.e., GTX 1080 at 8.8 TFLOPS >GTX 980 at 4.9 TFLOPS >Telsa K80 at 2.8 TFLOPS).

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.

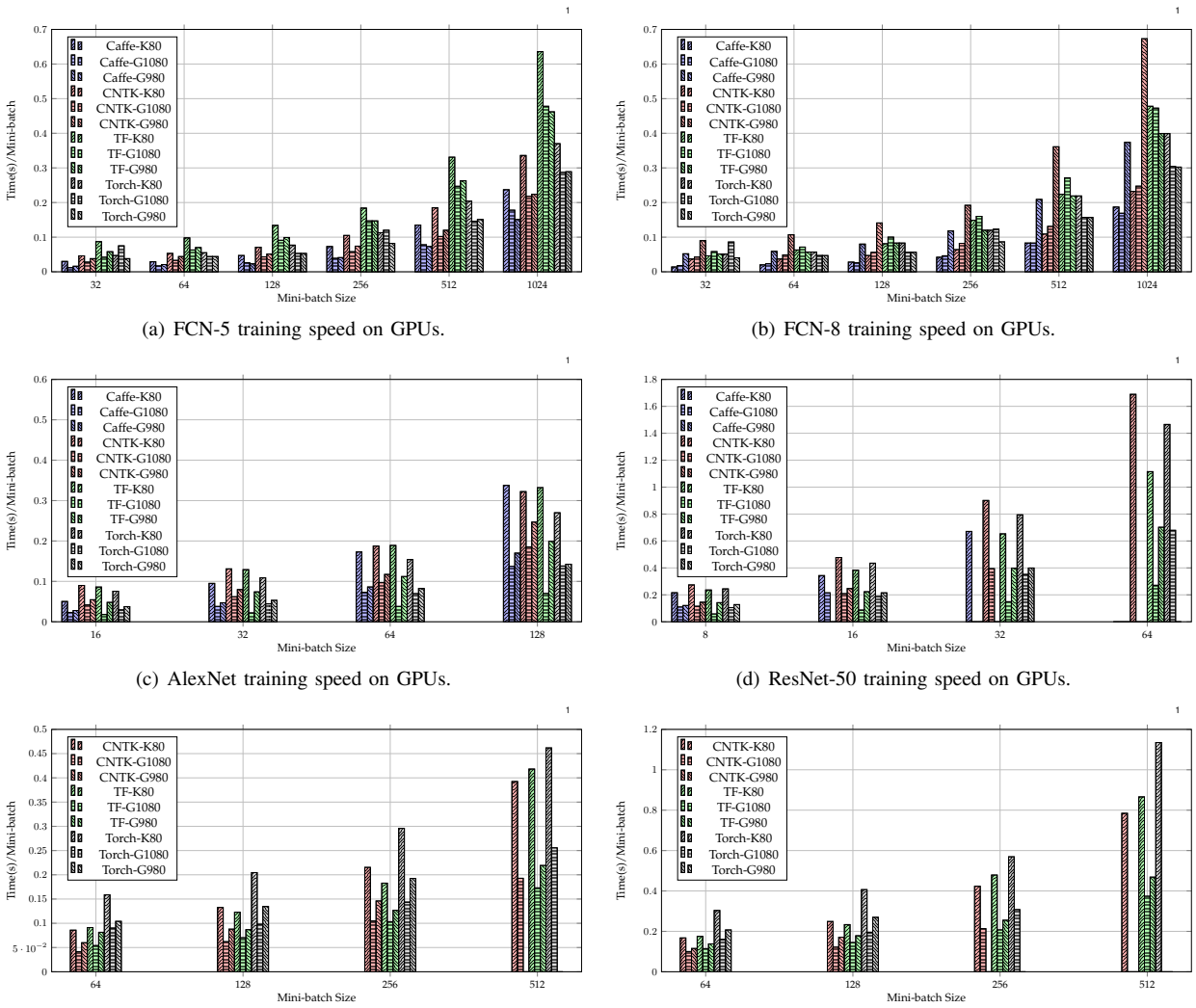
On FCNs, CNTK performs a little better than TensorFlow, but slightly worse than Caffe and Torch with one GPU. The results are matched with the official claims of CNTK [31]. It seems that CNTK has very outstanding performance in multiple GPUs scaling, but the scalability is not discussed in this paper. In general, training a network is a two-phase computations (i.e., feed forward and backward propagation). In feed forward phase, matrix multiplication is the main cost, and cuBLAS API: cublasSgemm is the optimazation solution to all the four tools. However, there is a trick calling parameter of cublasSgemm, which may result in different performance when doing matrix-matrix multiplication. With 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 effeciency computation of A times transposition of B by calling cuBLAS, it may be better if you 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 effeciency could be much better. To optimize the effeciency 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 call, the parameters may determine different kernels to use to do related computations. We found that on convolution operation, FFT is an optimized way compared to do convolution directly. After FFT of a matrix, convolution calculation can be considered as inner product operation which is much faster. TensorFlow acheive much better performance than the other three tools on ResNet-50, and FFT is one of the main factor that contributes to the efficiency. Therefore, FFT algorithms might should be better accelerated to improve the effeciency of CNNs training.

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. However, we found that the order of launched kernels during one training procedure is not consistent. As for CNTK, some specific kernels for training LSTM designed by itself are also very different from those of Torch and TensorFlow. As for CNTK, its brain scripts are very flexible to customize networks. The R̈ecurrentLSTMPStackF̈unction used in configuration file may have some operating optimization which contributes RNNs on CNTK achieve high effeciency calculations.

6. Conclusion

This is an evaluation of GPU-accelerated tools on three types of popular deep learning methods utilizing different hardware platforms. According to benchmark results, when there exists GPUs, we found that Caffe performs better on both fully connected networks and convolutional neural networks, and CNTK is outstanding on recurrent neural networks. The GPU memory is one of the key component for running big size networks in many tools, such as Caffe, CNTK and Torch cannot run ResNet-50 with 64 mini-batch size or more on GTX 980 card which has 4GB memory,



(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.

while TensorFlow and Torch has a better capability in managing GPU memory and it runs smoothly in all of configured cases. cuBLAS is a high-performance BLAS library, but the parameters of APIs are very important to achieve good results. FFT is a better choice to calculate the convolution operation in some circumstances.

On the CPU-only machines, Caffe has better results in CPU parallelization, and TensorFlow also makes better use of CPU resources. On the CPU parallel mechanism, it is better that threads allocated are equal to the number of CPU cores.

GTX 1080, which has higher base clock at 1733 MHz and more CUDA cores, gets better results in most cases. However, there is larger memory (12 GB) on Tesla K80 card which allows applications run with larger size of networks and greater mini-batch size. In addition, each K80 card is equipped with 2 GPU chips, which may result in better performance if the programs are parallelized.

Last but not least, these platforms are very flexible and the runtime performance depends on the configuration files. Our results only reflect the performace that we can achieve by the selected configurations, which may not be the best.

Limitations: We do not test the scalability across multiple GPUs and multiple machines, in which way it might not enhance main features of some tools, such as CNTK is the one that supports parallelization across both multiple GPUs and machines while others are not.

7. Future Work

To make a better performance comparison, more hardware platforms such as AMD GPUs, Intel Xeon Phi could be included. For those very time-consuming tasks, distributed version is the key to go. Ditributed design might be a very challege work and we will research this work in the future.

8. Acknowledgements

We would like to thank the CNTK team for their valuable feedback and providing the 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.

9. Appendix

9.1. Revision History

- 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 S̈impleNetworkBuilderẗo customized brain scripts.