Performance of Deep Learning Computation with TensorFlow Software Library in GPU-Capable Multi-Core Computing Platforms

Young Jong Mo[†], Joongheon Kim[‡], Jong-Kook Kim[*], Aziz Mohaisen[♮], and Woojoo Lee[§]
[†‡]School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea
[*]School of Electrical Engineering, Korea University, Seoul, Republic of Korea
[♮]Department of Computer Science and Engineering, State University of New York at Buffalo (SUNY Buffalo), NY, USA
[§]Department of Electronic Engineering, Myongji University, Yongin, Republic of Korea
E-mails: dudwhd93@naver.com, joongheon@cau.ac.kr, jongkook@korea.ac.kr, mohaisen@buffalo.edu, spacelee@mju.ac.kr

***Abstract*—In this paper we measure and verify the performance improvements in deep learning computation under the support of GPU-enabled multi-core parallel computing platforms. To measure the performance practically, we built our own computing platforms using a GPU hardware (1152 cores) and the TensorFlow software library. In order to evaluate the performance with GPU, we conducted the deep learning computation with various numbers of hidden layers in multilayer perceptron. As presented in the comparative performance results, utilizing GPU hardware improved the performance in terms of computation time (about 3 times or even more).**

I. Introduction

Recently, deep learning algorithms and their applications have received a lot of attention from both academia and the industry. Along with the advancing deep learning algorithms suited for real-world complex problems, computing platforms have evolved, too. For example, processing units (both of central processing unit (CPU) and graphics processing unit (GPU)) are becoming faster and are increasingly upgraded, every single day [4], [5]. To make use of such an evolution, various software packages and libraries have been implemented, including TensorFlow, a major software library implemented by Google Brain Team [1]–[3]. The evolution in both software and hardware is promising to make deep learning computations not only performance-relevant but feasible.

With the potential of applying deep learning on advanced hardware architectures, in this paper we sit out to answer the question of "*how much the multi-core GPU hardware improve the performance of deep learning computation in terms of computation speed*". To answer this question, we built our own computing platforms with an Intel i5 CPU and a NVIDIA GeForce GTX 1060 3GB (with 1152 cores). Using this platform, we ported and operated deep learning algorithms for the MNIST data set classification. We observe multifold performance improvements with GPU multi-core computing platforms while changing the number of layers in multilayer perceptron. Finally, we can observe the performance gap between (i) CPU-enabled and GPU-disabled computing and (ii) CPU-enabled and GPU-enabled computing. As presented in performance evaluation results (section III), the GPU-enabled platform achieves about 3 times better performance than the GPU-disabled platform.

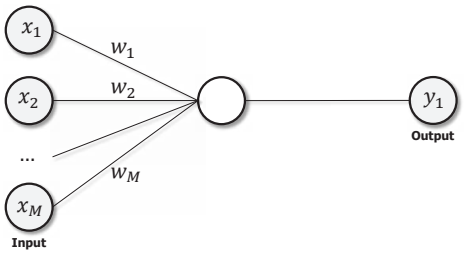

Fig. 1. Neuron – a computational element of artificial neural networks.

The rest of this paper is organized as follows. In section II, multilayer perceptron, a fundamental concept of deep learning technique, is introduced. In section III, the performance results measured at various layers in multilayer perceptron and various hardware configuration are presented. In section IV we sum up with concluding remarks.

II. Multilayer Perceptron

An artificial neural network (ANN) is defined as an interconnected network of simple computational elements, where the elements are named neurons. The architecture of ANN is as illustrated in Fig. 1. As presented in this figure, the input to a neuron consists of a number of values, namely $x_1, \cdots, x_M$, while the output is a single value y_1. Both the input and the output values are continuous values, usually in the range of $(0, 1)$. In this ANN, a neuron does the following: (i) it computes the weighted summation of its inputs $x_1, \cdots, x_M$, where the weights are $w_1, \cdots, w_M$, (ii) it subtracts a predefined threshold T, and (iii) it outputs the result into a nonlinear function f, e.g., a sigmoid function

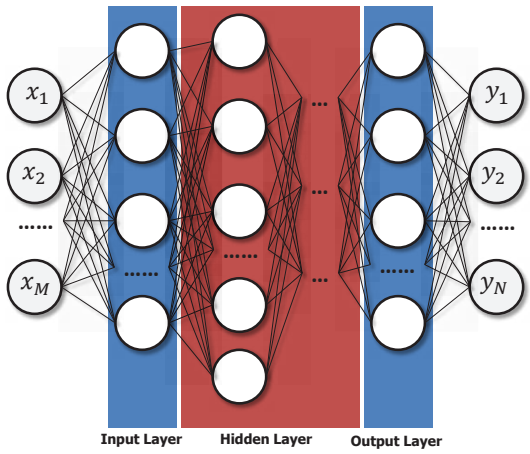

Fig. 2. A conceptual diagram for multilayer perceptron.

$S(t)$, which is defined as follows:

$$S(t) = \frac{1}{1 + e^{-t}}. \tag{1}$$

Therefore, the neuron in an ANN computes the following:

$$y_1 = f\left(\sum_{i=1}^{M} w_i x_i - T\right). \tag{2}$$

The outputs of some neurons are connected to the inputs of other neurons. In a multi-layer perceptron architecture, neurons are grouped into distinct layers as presented in Fig. 2. The outputs of each of those layer are connected to inputs in the following layers. The inputs of the first layer (input layer) are the inputs to the network, whereas the outputs of the last layer form the output of the network.

By optimizing the weights and thresholds for all nodes in the multilayer perceptron networks, the network can represent a wide range of classification functions. Optimizing the weights can be conducted by a supervised learning process, where the network learns from a large number of samples. The samples are usually provided one at a time. For each sample, the actual vector is computed and compared to the desired output. After this procedure, weights and thresholds are adjusted, proportional to their contribution to the error made at the respective output. One of the most widely used methods to achieve that is the back-propagation method (which works in an iterative manner), where the errors are propagated into the lower layers, to be used for the adaptation of weights. The error in this context is defined as the difference between the desired output and the actual output of the ANN.

When the number of layers in multilayer perceptron is large, the ANN setup is called deep-learning computation. For a long time, deep learning, while feasible as a concept, was not practically possible for its high computational requirements and the lack of the proper hardware to match such requirements. More recently, however, and thanks to the advanced hardware technologies such as general purpose GPU (GPGPU) multi-core computing, deep learning has been realized.

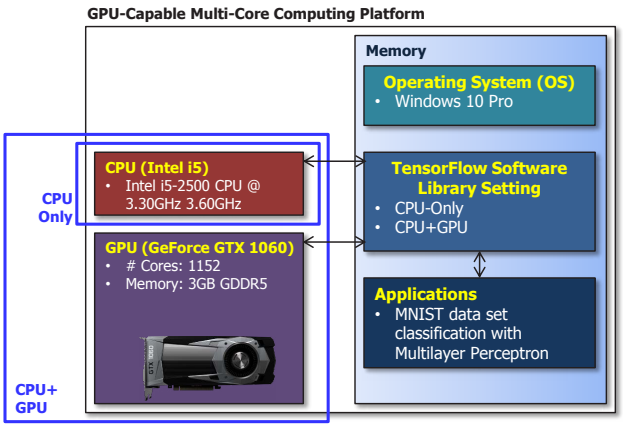

Fig. 3. A reference multi-core system model

TABLE I
SPECIFICATION OF MULTI-CORE COMPUTING PLATFORMS.

System	Specification
CPU	Intel(R) Core(TM) i5-2500 CPU@3.30GHz • RAM: 8GB
GPU	NVIDIA GeForce GTX 1060 3GB • Built-In Memory: 3GB • NVIDIA CUDA Cores: 1152 • Memory: 3GB GDDR5 • Memory Speed: 8Gbps
Software	OS: Windows 10 Pro (64 bits) • TensorFlow Version: tensorflow-gpu-1.0.1 • CUDA Version: cuda_8.0.61 • CuDNN Version: cuDNN v5.1 (Jan 20, 2017)

More details on multilayer perceptron are in [6].

III. A Performance Comparison Study

This performance comparison study is present in this section. Namely, in the following, we outline the GPU-capable computing platform specification (section III-A) and the detailed performance comparison results (section III-B).

A. GPU-Capable Computing Platform Specification

The multi-core computing platform used in this study for the performance evaluation of deep learning is illustrated in Fig. 3. In addition, the detailed specifications of each of the components listed in this figure are shown in Table I.

B. Performance Comparison Details

To evaluate the performance boast due advanced hardware, we used the MNIST data set as an input to the multilayer perceptron based learning in order to conduct classification. In order to observe the performance difference between computations with CPU (referred to as CPU-only) and and computations with CPU and GPU (referred to as CPU+GPU), the number of layers is fixed to 4, 6, 8, and 10.

To measure the multilayer perceptron computation speed across the two hardware settings, we conducted 50 experiment iterations. For each iteration, we measured the computation

TABLE II
AVERAGE TIME FOR MULTILAYER PERCEPTRON COMPUTATION

Numbers of Layers	CPU-only (Unit: Sec.)	CPU+GPU (Unit: Sec.)	Performance Gap
4	83.7599790	28.83337065	2.90
6	104.9254136	32.12257901	3.27
8	145.0726396	39.76726586	3.65
10	151.6537407	44.31995752	3.42

TABLE III
VARIANCES IN MULTILAYER PERCEPTRON COMPUTATION

Numbers of Layers	CPU-only (Unit: Sec.)	CPU+GPU (Unit: Sec.)	Performance Gap
4 (min)	82.0042253	28.41898632	2.89
6 (min)	103.0235872	31.82259274	3.24
8 (min)	139.7174885	39.22879481	3.56
10 (min)	150.0382178	43.60450292	3.44
4 (Max)	85.9771371	29.34874034	2.93
6 (Max)	116.2098830	33.40200472	3.48
8 (Max)	150.8112738	40.87909937	3.69
10 (Max)	154.2534342	46.08502030	3.35

speed with various numbers of layers in the CPU-only mode and CPU+GPU settings. The measurement results are listed in Table II, Table III, and Table IV.

In Table II, the average computation speeds of 50 experiment iterations with various numbers of layers in the CPU-only mode and the CPU+GPU mode are presented. In Table II, the performance gap, which captures the performance difference between the CPU-only mode and the CPU+GPU mode is presented. This gap is calculated as follows:

$$\text{Performance Gap} = \frac{\alpha}{\beta}, \tag{3}$$

where α and β are the computation speed with CPU-only and the computation speed with CPU+GPU, respectively.

As observed in the average computation speed comparison in Table II, we notice that CPU+GPU is $2.90 - 3.65$ times faster than that of the CPU-only in executing the same task.

In Table III, the fastest and slowest computation speed cases among the 50 experiment iterations are obtained with various numbers of layers in the CPU-only mode and CPU+GPU mode, respectively. In Table III, the performance gap between the two cases of hardware setups is calculated as before, where the gap is shown to be $2.89 - 3.56$ and $2.93 - 3.69$ times.

As in Table II and Table III, the performance gap is smallest when the number of layers is minimum (i.e., 4 in this study). This is, the gap grows as the number of layers grows.

In Table IV, the differences between the fastest and slowest computation speed cases among the 50 experiment iterations are obtained with various numbers of layers in the CPU-only mode and the CPU+GPU mode. As observed in Table IV, the CPU-only mode computation shows a larger variance ($3.97 - 13.19$) than the CPU+GPU mode computation (which had $0.93 - 2.48$).

TABLE IV
STABILITY ON PERFORMANCE

Numbers of Layers	CPU-only (Unit: Sec.)	CPU+GPU (Unit: Sec.)
4	3.97291184	0.92975402
6	13.18629575	1.579411983
8	11.09378529	1.650304556
10	4.215216398	2.480517387

IV. CONCLUDING REMARKS AND FUTURE WORK

This paper verifies the performance improvements with GPU-enabled platforms for deep learning computation in terms of computation speed. In order to verify the performance with real-world platforms, we implemented the MNIST data classification using the multilayer perceptron. In addition, we used TensorFlow software library and GPU-enabled platforms (with 1152 cores) to understand the performance boast as compared to CPU-only computations. The performance evaluation was conducted while changing the number of layers, i.e., 4, 6, 8, and 10, and as presented in performance evaluation results, the GPU-enabled platform delivers about 3 times better performance (in term of speed of computations) than that of the CPU-only settings. Moreover, the performance with GPU-enabled platform shows lower variances among the 50 iterations, which verifies that the GPU-enabled computing is more stable.

As a future research direction, we will consider various algorithms and applications of deep learning in order to measure the performance improvements with GPU-enabled computing platforms.

ACKNOWLEDGEMENT

Joongheon Kim is a corresponding author of this paper. This work was supported by National Research Foundation of Korea (NRF Korea) under Grant 2016R1C1B1015406.

REFERENCES

[1] P. Louridas and C. Ebert, "Machine Learning," *IEEE Software*, 33(5):110–115, September-October 2016.

[2] S. Lim and J. H. Yang, "Driver State Estimation by Convolutional Neural Network using Multimodal Sensor Data," *IET Electronics Letters*, vol. 52, no. 17, pp. 1495–1497, August 2016.

[3] L. Sánchez, J. Otero, I. Couso, and C. Blanco, "Battery Diagnosis for Electrical Vehicles through Semi-Physical Fuzzy Models," in *Proc. of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Vancouver, Canada, 24 - 29 July 2016.

[4] H. Jeon, W. H. Lee, and S. W. Chung, "Load Unbalancing Strategy for Multi-Core Embedded Processors," *IEEE Transactions on Computers*, vol. 59, no. 10, pp. 1434-1440, October 2010.

[5] Q. Xu, H. Jeon, and M. Annavaram, "Graph processing on GPUs: Where are the bottlenecks?," in *Proc. of the IEEE International Symposium on Workload Characterization (IISWC)*, Raleigh, NC, October 2014.

[6] S. K. Pal and S. Mitra, "Multilayer Perceptron, Fuzzy Sets, and Classification," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 683–697, September 1992.