A Fast Deep Learning System Using GPU

Zhilu Chen[+], Jing Wang[*], Haibo He[*], Xinming Huang[+]
[+]Department of Electrical and Computer Engineering, Worcester Polytechnic Institute
[*]Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island

***Abstract*—The invention of deep belief network (DBN) provides a powerful tool for data modeling. The key advantage of DBN is that it is driven by training data only, which can alleviate researchers from the routine of devising explicit models or features for data with complicated distributions. However, as the dimensionality and quantity of data increase, the computing load of training a DBN increases rapidly. Prospectively, the remarkable computing power provided by modern GPU devices can reduce the training time of DBN significantly. As highly efficient computational libraries become available, it provides additional support for GPU based parallel computing. Moreover, GPU server is more affordable and accessible compared with computer cluster or supercomputer. In this paper, we implement a variant of the DBNs, called folded-DBN, on NVIDA's Tesla K20 GPU. In our simulations, two sets of database are used to train the folded-DBNs on both CPU and GPU platforms. Comparing execution time of the fine-tuning process, the GPU implementation results 7 to 11 times speedup over the CPU platform.**

***Index Terms*—Restricted Boltzmann machine, Deep learning, Deep belief network, GPU, Parallel computing**

I. Introduction

In recent years, with the exponentially increase of the data availability in many complex systems, Big Data research has attracted significant growing attention in the community. Although existing data mining and machine learning techniques have demonstrated great potential for data analysis, many new and emerging topics such as imbalanced learning [1], dynamic stream data learning [2], real-time data analysis, among others, still present significant challenges to the community. Inspired by the hierarchical information processing mechanism of human brain, deep network with multiple hidden layers could provide significant learning capability with data through its hierarchical structure [3] [4]. In such deep networks, large among of free parameters brought by multiple hidden layers provide such structures sufficient capacity to model data with complicated, highly-nonlinear structures. However, it shows that the efficiency of back propagation (BP) decreases greatly when it is used to train a deep network, which limits the application of deep network in modeling real data. The milestone for deep network is marked by the invention of efficient training method for DBN. The training method is composed of layer-wise greedy learning as pre-training and BP as fine-tuning [5], which greatly reduces the training time for a DBN. It is expected that DBN can be trained sufficiently with as much data as possible so as to cover the entire input space. Unfortunately, the dimensionality of data increases much faster than the evolving computing techniques. For instance, it will take more than 10 hours on a desktop computer to train the DBN used in [5], where the dimensionality and quantity of the image data still stay in the magnitude of 10^2 and 10^5, respectively. In practice, we often have to deal with millions of images, each with dimensionality of 10^6 or higher. Therefore, there is a critical need to accelerate the computation to reduce the DBN training time significantly.

However, as the performance of a single-core processor almost hits the limit, researchers turn to multi-core system for parallel processing using computer clusters [6] [7] and graphic processing units (GPUs). For GPU, Compute Unified Device Architecture (CUDA) programming [8] makes it possible to accelerate the program by dividing the program into the small tasks and running them simultaneously on a large number of cores available on the GPU. There are a number of GPU-accelerated libraries available, making CUDA programming much more efficient. For example, the NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS) library [9] is a complete standard BLAS library that delivers 6x to 17x faster speedup than the latest MKL BLAS [10]. In this paper, we first implement the DBN on both CPU and GPU using Matlab and cuBLAS, respectively. Second, we implement a modified DBN, called folded-DBN, which reduces the fine-tuning time using the symmetric property of the weights of conventional DBN [11]. Third, the training procedures of DBN are highly serial and independent, which makes it difficult to convert into parallel form. Therefore only the matrix operations are parallelized by cuBLAS. Moreover, for a fair comparison, only the execution time is recorded. The simulation results show that a speedup of 7 times to 11 times is gained by using GPU acceleration.

II. Restricted Boltzmann Machine and deep belief network

RBM, as a generative model, could learn and represent any unknown distribution that is hard to be expressed explicitly. Though RBM is proposed by Hinton [12] and Smolensky [13] in 1986, separately, it is not widely used in data modeling until three decades later when efficient training algorithm for RBM is invented by Hinton [5] and the computing power is greatly improved. As illustrated in Fig.1, a single RBM contains two layers: the visible layer, which corresponds to the observation; and the hidden layer, which corresponds to the mutual impact between components of observations. The energy function of a joint configuration $(\mathbf{v}, \mathbf{h})$ is defined as

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T\mathbf{W}\mathbf{v} \quad (1)$$

978-1-4799-3432-4/14/$31.00 ©2014 IEEE

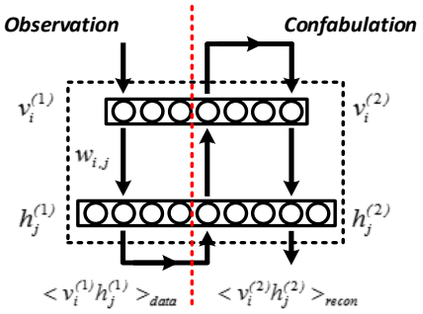

Figure 1. The procedure to train a RBM: (1) update the state of each hidden unit based on samples from observation; (2) update the real value of each visible unit based on updated states of hidden units; (3) update the state of each hidden unit one more time based on updated value of visible units; (4) update the weights $w_{i,j}$, b_j and c_i (b_j and c_i are not depicted in figure)

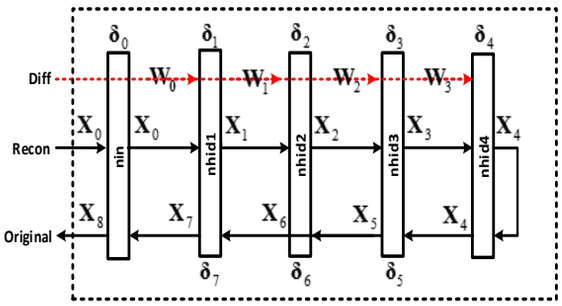

Figure 2. A folded structure of deep belief network, the solid lines represent the direction that data forward propagates and the dashed lines represent the direction that error backward propagates

The goal of training is to minimize the energy function defined in Eq.1, which is accomplished through alternating Gibbs sampling between visible layer and hidden layer. However, given the limited number of weights between visible units and hidden units within a single RBM, a single RBM is insufficient to model data with complicated, highly-nonlinear structure, such as speech or image. The real power of RBM is DBN constructed by stacked RBMs [5]. The training method for DBN contains two processes: layer-wise greedy unsupervised learning as pre-training process plus a supervised fine-tuning process. In pre-training process, the output of lower level RBM is taken as the input of higher level RBM and only one RBM is trained at a time. In fine-tuning process, DBN is treated as regular MLP thus could be trained using most of optimization algorithms, such as BP, conjugate gradient descent (CGD) or Levenberg-Marquardt (LM). The advantage of this training strategy is that, the layer-wise greedy learning as pre-training is very computationally efficient and could provide a very good start point, rather than random guess, for fine-tuning process such that both the total training time and the final training error are largely reduced [5] [11]. For more details about the theory and implementation of RBM and DBN, readers could refer to [5] [14] [15] [16].

To further reduce the training time required by fine-tuning process, we adopt a folded structure of DBN [11] in our GPU implementation. The folded-DBN reduces the training time for fining-tuning process by reducing the weights to be tuned, while it is still as competitive as conventional DBN in data representation. The structure of folded DBN is illustrated in Fig.2. Note that in Fig.2, in contrary to update all the weights, W_0-W_7, in conventional DBN, only half of the weights, W_0-W_3 need to be updated in folded-DBN.

[Algorithm I: The folded-DBN]
Initialize weights
$\{\mathbf{W}_0, \ldots, \mathbf{W}_{(L-1)/2}\}$
$\{\mathbf{X}_1, \ldots, \mathbf{X}_m, \ldots, \mathbf{X}_M\}$, $\mathbf{X}_m \in \Re^{n \times 1}$

while$(epoch < 200 \ and \ MSE^* > 10^{-8})$
for $m = 1 : M$

Forward propagation:
$\mathbf{X}_{l,m} = \sigma(\mathbf{W}_{l-1} \cdot \mathbf{X}_{l-1,m} + \mathbf{Wb}_{l,m} \cdot 1)$
where, $l = 1, \ldots, (L-1)/2$
$\mathbf{X}_{(L+1)/2,m} = \mathbf{W}_{(L-1)/2,m} \cdot \mathbf{X}_{(L-1)/2,m} + \mathbf{Wb}_{(L+1)/2,m} \cdot 1$
$\mathbf{X}_{l,m} = \sigma(\mathbf{W}^T_{l-(L+1)/2} \cdot \mathbf{X}_{l-1,m} + \mathbf{Wb}_{l,m} \cdot 1)$
where $l = (L+1)/2, \ldots, L$

Backpropagation:
$\mathbf{diff}_m = \mathbf{X}_{0,m} - \mathbf{X}_{L,m}$
$\boldsymbol{\delta}_{L,m} = \mathbf{diff}_m \cdot (1 - \mathbf{X}_{L,m}) \cdot \mathbf{X}_{L,m}$
$\boldsymbol{\delta}_{l,m} = \mathbf{W}_l \cdot \boldsymbol{\delta}_{l,m} \cdot (1 - \mathbf{X}_{L-l,m}) \cdot \mathbf{X}_{L-l,m}$
where $l = 0, \ldots, (L-1)/2$
$\boldsymbol{\delta}_{l,m} = \mathbf{W}^T_{L-l} \cdot \boldsymbol{\delta}_{L+1-l,m} \cdot (1 - \mathbf{X}_{L-l,m}) \cdot \mathbf{X}_{L-l,m}$
where $l = (L+1)/2, \ldots, L$

Weights update:
$\Delta\mathbf{W}_l = \eta \cdot \mathbf{X}_{L-l,m} \cdot \boldsymbol{\delta}^T_{l,m}$, $\mathbf{W}_l \leftarrow \mathbf{W}_l + \Delta\mathbf{W}_l$
where $l = 0, \ldots, (L-1)/2$
$\Delta\mathbf{Wb}_l = \eta \cdot \boldsymbol{\delta}^T_{L-l,m}$, $\mathbf{Wb}_l \leftarrow \mathbf{Wb}_l + \Delta\mathbf{Wb}_l$
where $l = 0, \ldots, L$
end
$epoch \leftarrow epoch + 1$
end

---

*MSE refers to mean square error in the following discussion

III. Learning Deep Belief Network using GPU

The training algorithm of folded-DBN can be described as the following: The higher level RBMs are pre-trained; the fine-tuning of DBN are accelerated through cuBLAS on GPU since it involves a large amount of matrix operations. The CUDA C program consists of three main parts: initialization, computation, and finalization. The following chart of our CUDA C program is illustrated in Fig.3.

In the initialization phase, raw data is read from MAT file using MATLAB API. Each matrix is stored in a one-dimensional array along with its width and height. It is copied to GPU device memory and resides in the global memory until the whole program ends. The occupied space in host memory is released upon the transferring is completed. In the computation part, the iteration continues until the mean square error converges or it reaches the maximum training epoch, such that the weights of DBN are well tuned. In each

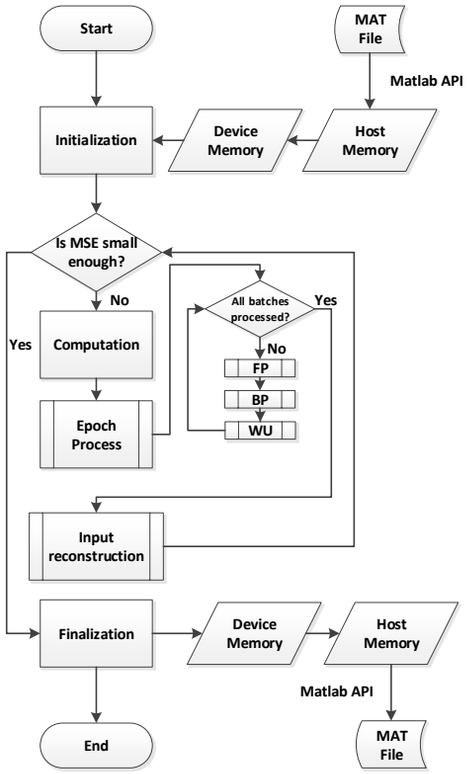

Figure 3. The flowchart of GPU implementation of folded-DBN

epoch, the input training data is divided into several batches to avoid memory overflow. For instance, if the size of original input data is 1000×100 and divided into 10 data chunks, then the size of each data chunk would be 100×100. There are three processing steps for each training epoch. The first step is forward propagation. The nonlinear sigmoid function is performed in each hidden layer and output layer, except for the middle layer. The second step is the error backward propagation. Note that the middle layer is different from all the other layers because there is no nonlinear operation is performed. The overloaded functions are used to perform the linear and nonlinear operations in each layer. The third step is weights update. In the finalization part of the CUDA C program, all the necessary parameters such as weights are saved and transferred from GPU device memory to the host memory on the CPU.

IV. Simulations

In our simulation, MNIST handwritten digits database [17] and Olivetti human face database [18] are chosen to evaluate the performance of GPU acceleration. We use the original MNIST handwritten digits database, which contains $O(10k)$ images of handwritten digits in training set and $O(10k)$ images in test set. The original Olivetti human face database contains only 400 frontal-face images of 40 different subjects taken from 10 different angles, which are insufficient to train a DBN. Thus we extend the original Olivetti human face databases through the following steps: at first, each image is rotated from 1° to 360° with increment of 1°; secondly, each of the rotated images is reduced to 25×25 image through cropping and down sampling; finally, the original Olivetti human face database is extended with $O(100k)$ frontal-face images of the first 30 subjects in training set and $O(10k)$ frontal-face images of the rest 10 subjects in test set. In the pre-training process, the configuration we adopted is the same as that used in [5] for both databases. In the fine-tuning process, BP is chosen to train folded-DBNs with a learning rate of 0.001 and a maximum learning epoch of 200 for both databases. Here we choose a small learning rate in order to tune the weights in a fine-grained way. To further accelerate the training process, batch updating method is employed. The properties of two databases and corresponding configurations for the folded-DBN are listed in Table I.

Table I
THE PROPERTIES OF TWO DATABASES: MNIST HAND-WRITTEN DIGITS AND EXTENDED OLIVETTI HUMAN FACE

	MNIST	extended Olivetti
# Training/Test	60000/10000	108000/36000
Image size	28×28	25×25
Image type	Grey scale [0, 1]	Grey scale [0, 1]
folded-DBN	784,1000,500,250,30	625,2000,1000,500,30
learning rate (η)	0.001	0.001
maximum learning epoch	200	200
batch size	1000	100

Table II
THE CONFIGURATION OF CPU PLATFORM AND GPU PLATFORM

	CPU platform	GPU platform
Model	i5 3570K	Tesla K20
memory	16 GB DDRIII	5 GB GDDR5
Software	Matlab 2012a R27	CUDA 5.5
Library	Intel(R) MKL V10.3.5	cuBLAS V2

The configurations of CPU platform and GPU platform used in our simulation are listed in Table II. One major arithmetic operation involved in fine-tuning process of DBN is matrix multiplication, which could be largely accelerated on GPU though built-in paralleling mechanism of cuBLAS library. First, we evaluate the performance of CPU and GPU using a very simple benchmark: matrix multiplication $C = A \times B$, where A and B are two $n \times n$ matrices

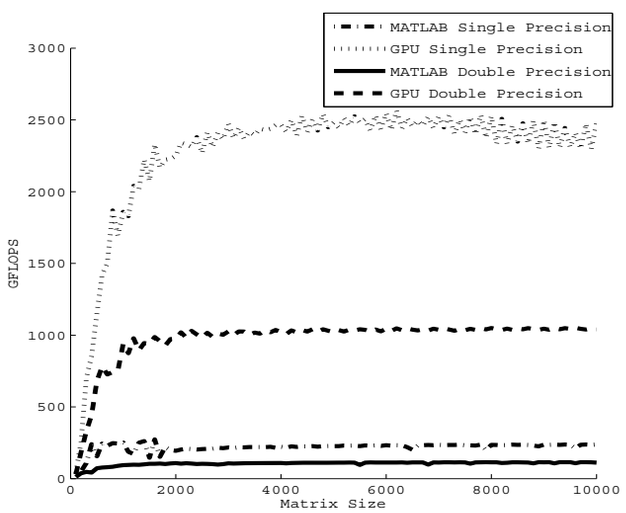

Figure 4. GFLOPS evaluated using $n \times n$ matrix multiplication under single precision and double precision

Table III
MNIST: AVERAGE EXECUTION TIME IN SECONDS ELAPSED ON THREE SECTIONS WITHIN ONE LEARNING EPOCH

	FP	BP	WU	Overall
CPU	7.4648	3.8836	3.3239	14.6724
GPU	0.6126	0.4014	0.2981	1.3120
Speedup	12.19	9.68	11.15	11.18

Table IV
OLIVETTI FACE: AVERAGE EXECUTION TIME IN SECONDS ELAPSED ON THREE SECTIONS WITHIN ONE LEARNING EPOCH

	FP	BP	WU	Overall
CPU	31.5393	21.8812	27.6103	81.0308
GPU	4.5189	3.2842	3.1524	10.9556
Speedup	6.98	6.66	8.76	7.40

and Giga Floating Point Operations Per Second (GFLOPS) is chosen as the criterion for comparison. Fig.4 shows the GFLOPS when matrices with different sizes are multiplied on the CPU using Matlab and on GPU using cuBLAS. Generally, 10x speedup is readily achievable for both single-precision and double-precision floating-point arithmetic. Consider that double precision is the default data type for variables in Matlab and its higher computing precision, we evaluate the speedup of GPU over CPU in DBN applications using the higher computing precision data type of Matlab, double precision floating point numbers. As is mentioned before, we could only optimize matrix operations, therefore we expect that a 10x speedup from our implementation of folded-DBN. First, two DBNs are pre-trained using RBM with two different databases: MNIST handwritten digits databases and Olivetti human face databases, respectively. Second, the initial weights stored in MAT files are loaded and the two folded-DBNs are further tuned using the algorithm described in Section II. Note that, the batch size is set to 1000 and 100 for two databases, which means the training set of MNIST handwritten digits database is divided into 60 data chunks with 1000 samples in each chunk and the training set of Olivetti human face database is divided into 1080 data chunks with 100 samples in each chunk. A complete epoch for fine-tuning has o three sections: forward propagation (FP), backward propagation (BP), and weights update (WU). To eliminate the artifact caused by outliers, we use median of time, rather than mean of time, to represent the "average time" needed by each on the three sections within each epoch, which are listed in Table III and Table IV. Also the speedup of GPU over CPU is listed in the bottom row. The simulation results on real-world database are consistent with our expectation and thus verify the efficiency of our GPU platform in training DBN with free parameters of $O(10^5)$ magnitude. Through 200 learning epochs, the MSE drops from 13.3515 to 5.0891 and from 9.3568 to 3.2343, for MNIST handwritten digits database and Olivetti human face database respectively, which verifies the validity of our GPU implementation.

V. CONCLUSIONS

In this paper, we have briefly introduced RBM, the building block of DBN. We also provide the detailed training algorithm of folded-DBN and elaborate its advantage over conventional DBN. We implement the folded-DBN on GPU platform with cuBLAS by comparing with the CPU implementation using Matlab. The GPU implementation achieves a speedup of 7 to 11 times over CPU program, upon which we conclude that GPU-based computing acceleration is a cost-effective, highly efficient method for training deep learning networks. Also we only compare the computing time of folded-DBN implemented on CPU and GPU platforms in our work. Thus we would like to perform a complexity analysis over folded-DBN in our future work.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under grant ECCS 1053717, the Army Research Office under grant W911NF-12-1-0378, and the NSF-DFG Collaborative Research on "Autonomous Learning", a supplement grant to CNS 1117314.

REFERENCES

[1] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.
[2] H. He, S. Chen, K. Li, and X. Xu, "Incremental learning from stream data," *IEEE Trans. Neural Networks and Learning Systems*, vol. 22, no. 12, pp. 1901–1914, 2012.
[3] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.
[4] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.
[5] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, pp. 504 – 507, 2006.
[6] W. B. Langdon, "Performing with cuda," in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, 2011, pp. 423–430.
[7] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Accelerating fully homomorphic encryption using GPU," in *2012 IEEE Conference on High Performance Extreme Computing (HPEC)*, 2012, pp. 1–5.
[8] *CUDA C PROGRAMMING GUIDE*, 5th ed., NVIDIA, July 2013.
[9] NVIDIA, "Cuda basic linear algebra subroutines (cublas) library." [Online]. Available: https://developer.nvidia.com/cublas
[10] *CUDA Toolkit 5.0 Performance Report*, NVIDIA, January 2013.
[11] J. Wang, H. He, and D. V. Prokhorov, "A Folded Neural Network Autoencoder for Dimensionality Reduction," *Proceedings of the International Neural Network Society Winter Conference(INNS-WC 2012)*, vol. 13, pp. 120–127, 2012.
[12] G. E. Hinton and T. J. Sejnowski, "In Parallel Distributed Processing: Explorations in the Microstructure of Cognition." *Learning and Relearning in Boltzmann Machines*, vol. 1, pp. 282–317, 1986.
[13] P. Smolensky, "Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1:Foundation, pp. 194–281, 1986.
[14] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," 2010. [Online]. Available: http://www.cs.toronto.edu/ hinton/absps/guideTR.pdf
[15] A. Fischer and C. Igel, "An introduction to restricted boltzmann machines," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Springer, 2012, pp. 14–36.
[16] LISA Lab, University of Montreal, "Deep Learning Tutorial: Restricted Boltzmann Machines (RBM)." [Online]. Available: http://deeplearning.net/tutorial/rbm.html
[17] Y. LeCun and C. Cortes, "THE MNIST DATABASE of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/index.html
[18] Olivetti database at ATT, "Olivetti Faces." [Online]. Available: http://www.cs.nyu.edu/∼roweis/data.html