Speed Up Method for Neural Network Back Propagation Learning by Using GPGPU

Yuta Tsuchida, Michifumi Yoshioka, and Hidekazu Yanagimoto

Graduate School of Engineering, Osaka Prefecture University,
Gakuen-cho Naka-ku Sakai city Osaka, Japan
tsuchida@sig.cs.osakafu-u.ac.jp

Abstract. Neural network is the basic machine learning technique with many inputs and outputs, so it applied many recognition systems. And recently, many applications with neural network are required to process a big data in real time. In this paper, we discuss how to make the processing time of the neural network learning faster by using GPU. GPGPU is a technique by which GPUs are used for a general computation approach. GPU is a dedicated circuit to draw the graphics, so it has a characteristic in which many simple arithmetic circuits are implemented. This characteristic is applied to not only graphic processing but also general purpose like this proposed method. In order to employ it effectively, the calculations which the neurons in the layer and many patterns are processed is parallelized. And we propose the parallelize method for calculation of back propagation. As the result, the proposed method is 40 times faster than the non-parallelized.

Keywords: Neural network, GPGPU, CUDA

1 Introduction

Neural network is the basic machine learning method with many inputs and outputs, so it applied many recognition systems. For example, the face recognition[1], speech recognition[2] or odor identification[3]. There are two main types of the neural network, supervised or un-supervised. We treat feed-forward neural network, one of the supervised basic network model. This model has many dimensional vector as input and output which is distributed nonlinearly. And recently, many applications with neural network are required to process a big data in real time.

On the other hand, the GPU, Graphic Processing Units, becomes popular and low cost due to development of high quality 3D CGs, movies processing and computer games. GPU is a dedicated circuit to draw graphics, so it has a few hundred simple arithmetic units. These units are able to be applied to not only graphic processing but also general parallel computations. The usage of GPU to parallel computations is called General-Purpose Computing on GPU. In order to assist developments with GPGPU, Compute Unified Device Architecture (CUDA) is distributed by NVIDIA, which optimizes parallel computations with

GPU. Shuai, at el [4] verified the characteristics of GPGPU by using CUDA by comparing with CPU.

In this study, we consider a GPGPU application to machine learning, especially in neural networks. The neural network is one of the supervised machine learning algorithms with some input signals and outputs. The outputs are compared with the supervisor signals, and the weight coefficients in the network are updated to decrease the difference between the outputs and supervisors. Recently, the applications of the neural network are required to treat the huge data in real time, so the neural network learning has to achieve high speed.

The goal of this paper is that the neural network learning method is modified to speed up the processing time with GPGPU. As the previous works, Kyoung-S, at el[5] proposed the neural network implementation on GPU. And Hongboom, at el[6] proposed how to implement with CUDA. However, in these conventional researches, they didn't propose the speed up of the learning, because these assumed the learned networks. Under these circumstances, we propose a neural network training method which is optimized for GPGPU. At first, we describe the conventional method. And then we rewrite the method in order to parallelization. Finally, we show the effectiveness of our proposed algorithms by some simulations.

2 Proposed methods

2.1 Neural network

In this section, the architecture of the three layered neural network is summarised[7]. Fig.1 shows the three layered neural network. It has / neurons in an input layer, m neurons in a hidden layer and n neurons in an output layer, respectively. Following equations indicate the oth *(o* = 1, 2, • • • *P)* pattern. The output $y_{\cdot}{}^{\circ}{}_{7}$ from the neuron in hidden layer is defined as

$$= f\left(\sum_{t=o} v_{ii} e_j\right) \ (j = 1, 2, \bullet \bullet \bullet m), \qquad (^{1})$$

where $x^{\circ}(i = 1, 2, \bullet \bullet \bullet /)$ is input signal. $x6 = —1$ is a threshold for neurons in hidden layer and identical for all patterns. v_{ii} is a weight coefficient from the input layer to the hidden layer, and $f(x) = 1 / (1 + \exp(—x))$ is sigmoid function. The outputs from the neuron in output layer zk is:

$$zr, = f\left(\sum_{j=o}^{(m} w_i k y y \right) \qquad (k = 1, 2 \bullet \bullet \bullet n). \qquad (2)$$

yo = —1 is a threshold and identical for all patterns, and $w_i k$ is a weight from the neuron in hidden layer to output layer. Since the conventional method is assumed an online learning approach, the weights 74_3, and $w_i k$ are updated for every pattern. The backpropagation method is applied to the update of the weights [7]. We propose the method which is trained simultaneously for different patterns modified from the conventional method.

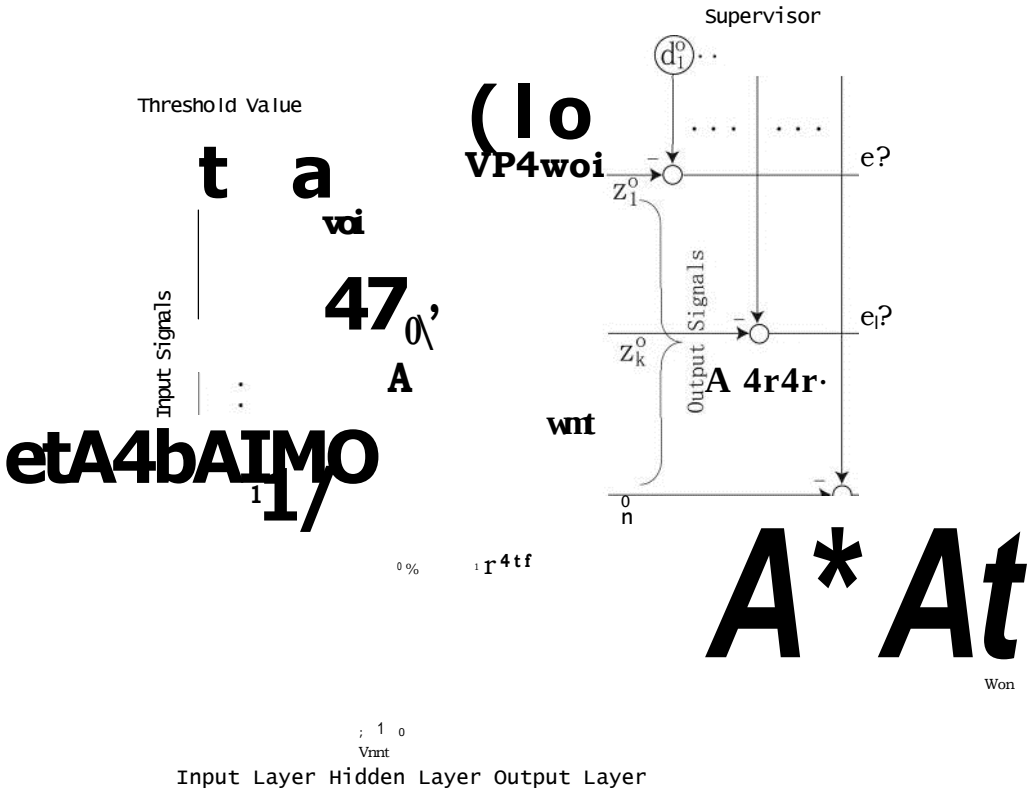

Fig. 1. Three layered neural network.

2.2 The parallelized learning algorithm

The parallelized learning algorithm is modified as follows. In the feed forward phase, the calculation Eq.(1) and (2) are the same weights for all patterns. In the back propagation phase, the update quantities of the weights .wo_k and .vtf_3 are added for the result of the each pattern respectively. Namely, when the Eq.(1) and (2) are calculated, weights v_{i3}, w_{3k} are fixed, and the output for each layer,

are stored for each patterns. In this method, the evaluation function is the summation of the errors for every patterns:

$$E = \frac{1}{2} \sum_{o=1}^{P} \sum_{k=0}^{n} E(c/z - zn2 \tag{3}$$

And the weights in the output layer are updated as follows:

$$ek = c/Z - zk. \tag{4}$$

$$= e\,4(1 \tag{5}$$

$$AiWjk = \sum_{0=1}^{7} /4^{6}i, \quad (6)$$

where 8 is stored for each pattern *o*. Then, the weights in the hidden layer are updated as follows:

$$= (1 - \sum_{k=1} W^{i}_{r} \quad (7)$$

$$= \sum \quad (8)$$

In the training step, each value for every equation is calculated simultaneously, and these equations are processed sequentially. For example, Eq.(5) is calculated in the total *P* x n times in the same time, and Eq.(5) and (6) are calculated sequentially. By changing this way, we avoid that a value which is rewritten by the oth change is not overwritten by the one of o'th.

2.3 The implementation method for GPGPU

By using the method described in the previous section, we show method of the implementation for GPGPU. Fig.2 shows the proposed method. The circles show values, and triangles show the calculation of the feed forward phase. The inputs of the neural network *x,* the output of the hidden layer y and outputs of the network z are depicted *(1* x *P),* (m x *P),* (n x *P)* matrixes respectively. In the back-propagation phase, the supervisors *d* are shown as (n x *P)* matrix too. The input signals for each pattern are input to the network which has identical weights. In the feed forward phase, the calculations for neurons in same layer are performed in parallel. However, it is important that we need to wait until all calculations in the same layer are completed because calculations in the next layer need the results in the previous layer. In order to guarantee this synchronization, we introduce a synchronization mechanism by using "Syncthread" function in CUDA. In the backpropagation phase, the equations which are proposed in previous section can be calculated simultaneously because of the each value rewritten by the equation is not accessed from other equation. The procedure is as follows. In the output layer, {n x P} Eq.(4) and (5) are processed. And {n x (m + 1)} Eq.(6) are processed. In the hidden layer, {m x P} Eq.(7) is processed. And then, {m x *(1* + 1)} Eq.(8) is processed. Finally, the process is paused until the each equation has completed, so we need "Syncthread" function.

3 Simulation

In order to confirm the effectiveness of our proposed methods, we have a simulation. We use the method which is not parallelized for inside of the neural network and patterns as the conventional method, and implemented in as same hardware as proposed method. We compared the calculation times of the proposed methods with the conventional method. The GPU used for the simulation is GeForce GTX480. Table 1 shows the specification of the parameter settings for this simulation. Fig.3 shows the result of the simulation. As the result, the processing speed is about 40 times faster than that of the conventional.

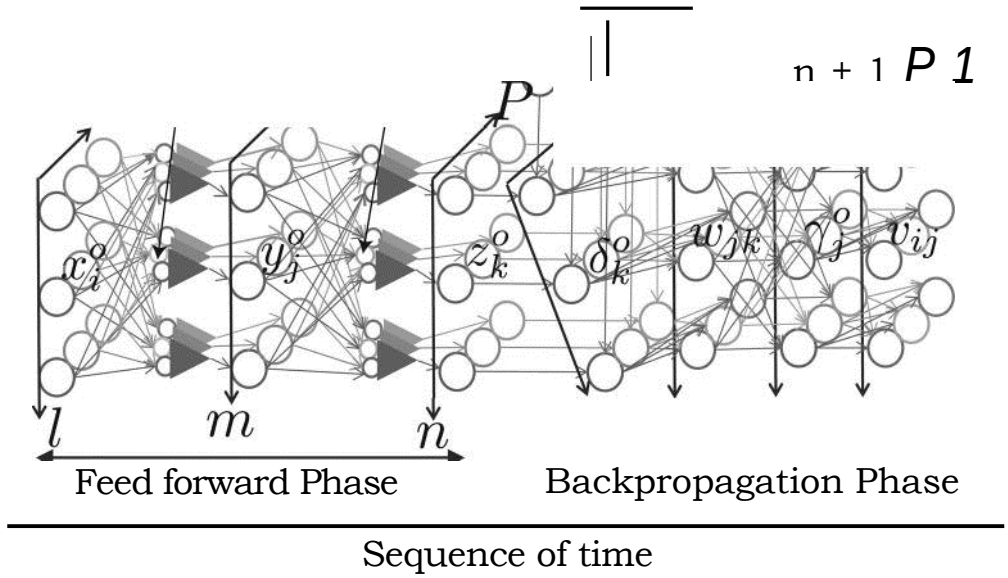

Fig. 2. Parallelization for each process. The circles show the values, and the triangles show the calculation of the neurons.

Table 1. Parameter settings used in the simulations

Symbols	Descriptions	Values
N	The number of the neural networks	20
1	The number of inputs	3
m	The number of the neurons in the hidden layer	6
n	The number of output	5
P	The number of the patterns	100
s	The number of the steps of training	10000

4 Conclusion

In this paper, we proposed the neural network training algorithm optimized for GPGPU. This method is parallelized for the neurons and patterns by changing the learning method. The simulation results show the effectiveness of our proposed method. Our proposed method achieves about 40 times faster than the conventional method. In the future works, this method is required to verify by using the larger scale network.

References

1. Chelsia A. D., Jamal A. D., Ali C, and Sigeru 0. ,"COMBINING NEURAL NET-WORKS FOR SKIN DETECTION", Signal and Image Processing : An International Journal(SIPIJ) Vol.1, No.2, December 2010, DOI : 10.5121/sipij.2010.1201

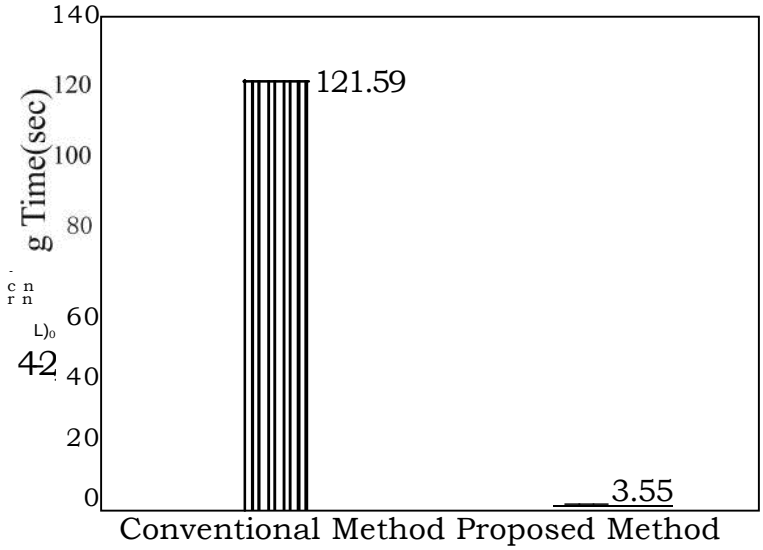

Fig. 3. The simulation method. The vertical axis shows the processing time.

2. Joe Tebelskis , "Speech Recognition using Neural Networks" School of Computer Science, Carnegie Mellon University , CMU-CS-95-142 (1995)
3. Jamal, M.; Khan, M.R.; Imam, S.A.; Jamal, A.; , "Artificial neural network based e-nose and their analytical applications in various field," Control Automation Robotics and Vision (ICARCV), 2010 11th International Conference on , vol., no., pp.691-698, 7-10 Dec. 2010
4. Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, and Kevin Skadron. 2008. "A performance study of general-purpose applications on graphics processors using CUDA". J. Parallel Distrib. Comput. 68, 10 (October 2008), 1370-1380.
5. K.-S.Kyoung-Su Oh, Keechul Jung, "GPU implementation of neural networks" Pattern Recognition, Volume 37, Issue 6, June 2004, Pages 1311-1314, ISSN 00313203, DOI: 10.1016/j.patcog.2004.01.013.
6. Honghoon Jang, Anjin Park, Keechul Jung "Neural Network Implementation Using CUDA and OpenMP" Computing: Techniques and Applications, 2008. DICTA '08.Digital Image, vol., no., pp.155-161, 1-3 Dec. 2008 doi: 10.1109/DICTA.2008.82.
7. Edgar Sanchez-Sinencio, Clifford Lau, editors. Artificial neural networks: Paradigms, applications, and hardware implementations, IEEE press, Piscataway, N.J, 1992