Indonesian Journal of Electrical Engineering and Computer Science
Vol. 11, No. 3, September 2018, pp. 954~961
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v11.i3.pp954-961

A Novel Approach for Efficient Training of Deep Neural Networks

D.T.V. Dharmajee Rao[1], K.V. Ramana[2]
[1]Aditya Institute of Technology and Management, Tekkali-532201, Srikakulam, Andhra Pradesh, India
[2]JNTUK College of Engineering, JNTUK University, Kakinada - 533003, Andhra Pradesh, India

Article Info

Article history:

Received May 12, 2018
Revised Jun 11, 2018
Accepted Jun 14, 2018

Keywords:

Backpropagation and Boltzmann Machine algorithms,
Deep Neural Network,
Multi-core processor system,
OpenMP,
Parallel Blocked Matrix multiplication.

ABSTRACT

Deep Neural Network training algorithms consumes long training time, especially when the number of hidden layers and nodes is large. Matrix multiplication is the key operation carried out at every node of each layer for several hundreds of thousands of times during the training of Deep Neural Network. Blocking is a well-proven optimization technique to improve the performance of matrix multiplication. Blocked Matrix multiplication algorithms can easily be parallelized to accelerate the performance further. This paper proposes a novel approach of implementing Parallel Blocked Matrix multiplication algorithms to reduce the long training time. The proposed approach was implemented using a parallel programming model OpenMP with collapse() clause for the multiplication of input and weight matrices of Backpropagation and Boltzmann Machine Algorithms for training Deep Neural Network and tested on multi-core processor system. Experimental results showed that the proposed approach achieved approximately two times speedup than classic algorithms.

Corresponding Author:

D.T.V. Dharmajee Rao,
Aditya Institute of Technology and Management,
Tekkali-532201, Srikakulam, Andhra Pradesh, India.
Email: dtvdrao@gmail.com

1. INTRODUCTION

In recent years, Deep Neural Networks (DNN) has been used in wide range of research areas related to data classification, clustering and pattern recognition [1]. State-of-the-art performances have been achieved with DNN techniques on various applications, such as traffic sign classification [2], feature extraction [3] and face recognition [4] to name just a few. DNN is a multilayer feed-forward neural network that consists of an input layer, an output layer and more number of hidden layers with different number of neurons in each layer. These multiple hidden layers and more number of neurons in each layer add more computational complexity to the network which results in slow rate of learning capacity [5]. Moreover with the growth in size of datasets, it has become hard to train DNNs that require higher computing power [6].

To reduce the training time (convergence time), the idea is not based on changing the existing algorithms such as Backpropagation Algorithm (BPA) and Boltzmann Machine Algorithm (BMA) or to develop new algorithms for training DNN. It is based on using some optimization techniques like Blocking and parallelization of algorithms on multi-core systems. Blocking is a popular and well-proven optimization technique to improve the performance of matrix multiplication [7]. An overview of common optimization techniques for matrix multiplication on multi-core processor for optimized performance is discussed in [8].

This paper aims to select the optimal block size suitable for computer system environment and configuration where all experiments are carried out. The same paper proposes to modify BPA and BMA by using Parallel Blocked Matrix multiplication. These proposed techniques and algorithms speed up the training of DNN on multi-core CPU architectures.

Remainder of this paper is organized as follows. Section 2 presents proposed novel approach of Parallel Blocked Matrix multiplication. Section 3 presents implementation of Backpropagation Algorithm and Boltzmann Machine Algorithm using proposed method. Performance of proposed technique is shown through results in section 4, and finally the conclusions and future work are summarized in section 5.

2. PROPOSED METHOD

The total execution of matrix multiplication consists of two parts: The memory access latency of two input matrices elements to be multiplied and the processing time of product of two matrices. Blocking is a technique that really reduces the memory access latency. Blocking brings a sort of divide-and-conquer technique and this allows one to parallelize the product of two matrices more effectively on multi-core processer system.

2.1. Blocking

Matrix multiplication is a compute intensive algorithm. Blocking is an optimization technique which boosts the performance of matrix multiplication, especially for large matrices [9]. The idea behind the blocking is to organize the matrix data in the form of chunks called blocks. With the use of cache memory it is obvious that the memory access time to all parts of the matrix would not be the same. If a required block is present in the cache it is possible to retrieve the data without much delay. This is called cache hit. Contrarily, if the required block is not in the cache it has to be loaded from the main memory. This increases the latency, and is called cache miss.

1. Cache hit ratio = Percentage of memory accesses hit in the cache.
2. Cache miss ratio = Percentage of memory accesses miss in the cache.
3. Average memory access time = Cache hit ratio × Time to access Cache + Cache miss ratio × Time to access memory.

Instead of operating entire rows and columns of matrix, blocked algorithm operates on sub-matrices or blocks, so that data loaded into the cache memory are reused. This algorithm even increases the number of operations, but significantly reduces the average memory access time by minimizing the number of cache misses [10]. The two input and one output matrices are divided into blocks. The size of each block is bs × bs. The block C_{11} of output matrix is computed as a product of A_{11} block of matrix A and B_{11} block of matrix B as shown in (1). In a similar manner, all other blocks of output matrix C are computed. The Blocked Matrix multiplication is depicted in Figure 1. The Pseudo code for Blocked Matrix multiplication to reduce cache misses versus sequential matrix multiplication for matrices of size n × n is shown in Figure 2.

$$C_{11}=A_{11}\times B_{11}+A_{12}\times B_{21}+ \text{----} +A_{1n}\times B_{n1} \quad (1)$$

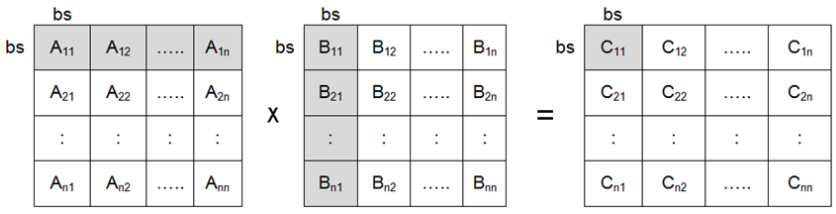

Figure 1. Blocked Matrix multiplication

Sequential	Blocked

for(i = 0 ; i < n ; i++)
for(j = 0 ; j < n ; j++)
{
sum = 0 ;
for(k = 0 ; k < n ; k++)
{
sum += a[i*n+k]*b[k*n+j] ;
}
c[i*n+j] = sum;
}
	for (i = 0; i < n; i=i+bs)
	for (j = 0; j < n; j=j+bs)
	{
	for (ii = i; ii < i+bs && ii < n; ii++)
	for (jj = j; jj < j+bs && jj < n; jj++)
	c[ii*p+jj] = 0;
	for (k = 0; k < n ; k=k+bs)
	for (ii = i; ii < i+bs && ii < n; ii++)
	for (jj = j; jj < j+bs && jj < n; jj++)
	for (kk = k; kk < k+bs && kk < n; kk++)
	c[ii*p+jj] += a[ii*n+kk] * b[kk*p+jj];
	}

Figure 2. Pseudo code for matrix multiplication (Sequential vs Blocked)

2.2. Parallelization Using OpenMP

Greater speedup is achieved if parallelization is used for matrix multiplication [11]. The Blocked Matrix multiplication algorithm can be parallelized using OpenMP because it does not have data dependency. OpenMP (Open Multiprocessing) is an open-source library for shared memory parallelization on multi-core processor architecture. Since most processors are now at least dual core, it is usually of advantage to parallelize the matrix product computations. This ensures that the large number of iterations is distributed among multiple cores, parallel and concurrent threads to increase performance. There are two more loops namely the ii loop, and jj loop that split the matrix multiplication into a simple bs × bs matrix which is multiplied the traditional way. Instead of just the outer one by #pragma omp parallel for, more than one outer most "for" loops can be parallelized and distributed among multiple cores by using OpenMP collapse() clause to increases the amount of work. The Figure 3 describes pseudo code for Parallel Blocked Matrix multiplication using OpenMP with collapse() clause versus Sequential Matrix multiplication.

Blocked	Parallel Blocked with collapse() clause

for (i = 0; i < n; i=i+bs)
for (j = 0; j < n; j=j+bs)
{
for (ii = i; ii < i+bs && ii < n; ii++)
for (jj = j; jj < j+bs && jj < n; jj++)
c[ii*p+jj] = 0;
for (k = 0; k < n ; k=k+bs)
for (ii = i; ii < i+bs && ii < n; ii++)
for (jj = j; jj < j+bs && jj < n; jj++)
for (kk = k; kk < k+bs && kk < n; kk++)
c[ii*p+jj] += a[ii*n+kk] * b[kk*p+jj];
}
	#pragma omp parallel for collapse(2)
	for (i = 0; i < n; i=i+bs)
	for (j = 0; j < n; j=j+bs)
	{
	for (ii = i; ii < i+bs && ii < n; ii++)
	for (jj = j; jj < j+bs && jj < n; jj++)
	c[ii*p+jj] = 0;
	for (k = 0; k < n ; k=k+bs)
	for (ii = i; ii < i+bs && ii < n; ii++)
	for (jj = j; jj < j+bs && jj < n; jj++)
	for (kk = k; kk < k+bs && kk < n; kk++)
	c[ii*p+jj] += a[ii*n+kk] * b[kk*p+jj];
	}

Figure 3. Pseudo code for matrix multiplication (Blocked vs Parallel Blocked with collapse())

3. RESEARCH METHOD

Multilayer feed-forward Deep Neural Networks are found to be characterized with the major drawback such as slow rate of learning capacity and simple problems may take hundreds of thousands of iterations to converge. Many researches focused on improving the speed of convergence. The traditional way of Backpropagation Algorithm (BPA) and Boltzmann Machine Algorithm (BMA) for training DNN is revisited and observed that the sum of product of weight and input matrices is computed for several hundreds of thousands of times during the training of DNN [12]. Here the input matrix gets multiplied by weight matrix by using sequential matrix multiplication algorithm. By implementing a novel approach of replacing this sequential matrix multiplication algorithm by Parallel Blocked Matrix multiplication algorithm, one can speed up the convergence time of BPA and BMA training process.

3.1. Parallel Blocked Backpropagation Algorithm (PBBPA)

The development of fast and efficient learning algorithms of DNNs has been a subject of interest in the recent years. As a result, several new algorithms have been proposed with various optimization techniques for the faster convergence of BPA [13]. A multilayer feed-forward Deep Neural Network (DNN) consists of an input layer, six hidden layers and an output layer used for experimentation is shown in Figure 4.

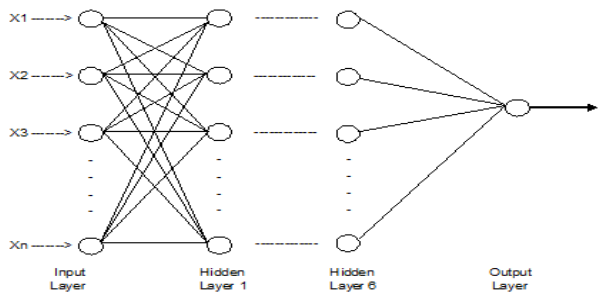

Figure 4. A multilayer feed-forward Deep Neural Network

In order to reduce long convergence time, the Backpropagation Algorithm with classic matrix multiplication gets modified by using Parallel Blocked Matrix Multiplication as listed in Algorithm 1. The resultant algorithm is now onwards called Parallel Blocked Backpropagation Algorithm (PBBPA). The PBBPA can be used instead of BPA to speed up the convergence time of DNN training process. The PBBPA has same steps of BPA but Parallel Blocked Matrix Multiplication is used instead of traditional matrix multiplication to compute the product of input matrix and weight matrix.

Algorithm 1: Parallel Blocked implementation of BPA

1. Initialize all weights and biases in network
2. While terminating condition is not satisfied {
3. For each training sample Xi in D {
4. For each Input layer unit j {
5. $O_j = I_j$;
6. For each hidden or output layer unit j {
7. $I_j = \sum_i ParallelBlocked(w_{ij} O_i) + \theta_j$;
8. $O_j = \frac{1}{1 + e^{-I_j}}$; }
9. For each unit j in the output layer
10. $Ej = O_j(1 - O_j)(T_j - O_j)$;
11. For each unit j in the hidden layers, from the last to first hidden layer
12. $Ej = O_j(1 - O_j)\sum_k ParallelBlocked(E_k W_{jk})$;
13. For each weight W_{ji} in network {
14. $\Delta W_{ji} = (l) E_j O_j$;
15. $W_{ji} = W_{ji} + \Delta W_{ji}$; }
16. For each bias θ_j in the network. {
17. $\Delta\theta_j = (l) E_j$;
18. $\theta_j = \theta_j + \Delta\theta_j$; }
19. }}

3.2. Parallel Blocked Boltzmann Machine Algorithm (PBBMA)

A Restricted Boltzmann Machine (RBM) is structurally a shallow neural network with just two layers that are the visible layer (input layer) and hidden layer as shown in Figure 5. In RBM networks, neurons in each layer have no connections between them and are connected to all other neurons in other layer and connections between neurons are bidirectional and symmetric. RBMs have received a lot of attention in the recent years after being proposed as building blocks of multilayer learning architectures called Deep Boltzmann Machines (DBM) as shown in Figure 6.

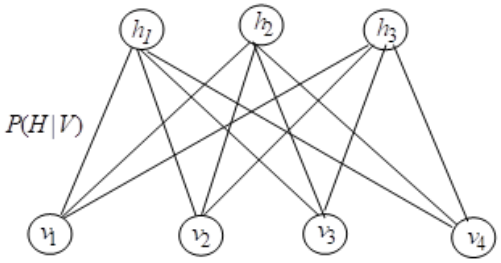

Figure 5. Restricted Boltzmann Machine (RBM)

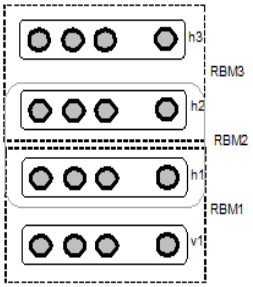

Figure 6. DBM: A stack of RBMs

Training a DBM is a computationally expensive task that involves several RBMs and requires a considerable amount of time [14]. As usual, Boltzmann Machine Algorithm with standard sequential matrix multiplication gets replaced by Parallel Blocked Matrix Multiplication as listed in Algorithm 2. Now modified BMA is called Parallel Blocked Boltzmann Machine Algorithm (PBBMA). The PBBMA can be used instead of BMA to speed up the training of DNN. The PBBMA has same steps of BMA but the classic matrix multiplication is replaced by Parallel Blocked Matrix Multiplication to compute the product of input matrix and weight matrix.

Algorithm 2: Parallel Blocked implementation of BMA

1. Take the training dataset, set the states of the visible units to the training data.
2. Positive phase : Reconstruct hidden units using positive statistics (E_j) is given by

$$P(h_j = 1/V) = \frac{1}{1+e^{-(w_{hbj}+ParallelBlocked(w^j_{hidden}*V))}}$$

3. Negative phase: Reconstruct visible units using negative statistics (E_j) is given by

$$P(v_j = 1/H) = \frac{1}{1+e^{-(w_{vbj}+ParallelBlocked(w^j_{visible}*H))}}$$

4. Update Phase $w_{ij} = w^{old}_{ij} + \eta(positive(E_j)) - negative(E_j)$

Repeat with all training vectors until required threshold gets satisfied.

4. RESULTS AND DISCUSSION

The performance comparison of our interest is convergence time consumption of DNN training process. All the algorithms were implemented using C++ and then tested on multi-core processor system, the information about execution time was collected. In order to compare the performance, testing was performed on a personal desktop(HP Compaq 8200 Elite SFF Intel Core I7-2600 CPU@3.40 GHz, 4 Cores, 8 Threads, 64 Bit, 8 MB Cache, 4 GB RAM) running Ubuntu operating system (GNU/ Linux 4.4.0-57 generic#78-Ubuntu SMP) with software version g++ (Ubuntu 5.4.0-6 ubuntu~16.04.4) 5.4.0. The same environment has been used for all experiments that were carried out for this paper. All the implemented algorithms were rerun for five times and the average execution time was calculated.

4.1. Optimal Block Size

Another thing that one should take care when using blocking is to fit the blocks which are being multiplied into the cache line comfortably so that the cache misses are minimized. So it is also important to select the optimal block size [15]. The block size (bs) is chosen so that the bs × bs submatrix or block of A and B matrices can fit in the cache. In this experiment, the execution time is calculated for the matrix multiplication of matrices of size 1024 × 1024 by varying the block size from 4 to 512 in step of power of 2. It is practically found that optimal block size (bs) is 16 for the above mentioned environment as shown in Figure 7.

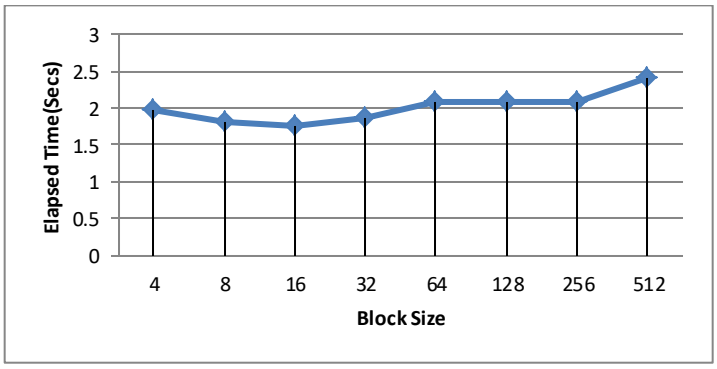

Figure 7. Execution time of Parallel Blocked Matrix multiplication

4.2. Performance Comparison of PBBPA and BPA

The classification of odd and even numbers was chosen for the performance analysis of Backpropagation training algorithm as it is simple problem to implement. The training pattern (binary form of given number) size is a power of 2 and equal to the number of units in the input layer. Number of units is equal in input and all hidden layers except output layer. One output unit is used to represent two classes where the value 1 represents odd class, and the value zero represents even class. The training patterns are generated using random function. The weights in the networks are initialized to small random numbers ranging from -1.0 to 1.0. The biases are similarly initialized to small random numbers. Considered the Deep Neural Network with six hidden layers in addition to input and output layers, the observations are obtained by training the network for 30,000 iterations. The logistic or sigmoid function is used as activation function.

Table 1 demonstrates that proposed Parallel Blocked implementation (PBBPA) shows significant improvement in decreasing Deep Neural Network training time compared to direct implementation (BPA). From the Figure 8, it is completely clear that the training time further reduced as the size of training pattern is increasing.

Table 1. Training time comparison of BPA and PBBPA

Pattern Size	Elapsed time(Secs)		
	BPA	PBBPA	Speedup
16	4.61	15.35	0.30
32	36.04	37.47	0.96
48	122.07	99.20	1.23
64	338.29	152.81	2.21
80	573.99	342.74	1.67
96	979.43	548.99	1.78
112	1523.43	853.87	1.78
128	2454.65	1166.90	2.10
144	3290.57	1737.87	1.89
160	2742.20	1350.74	2.03

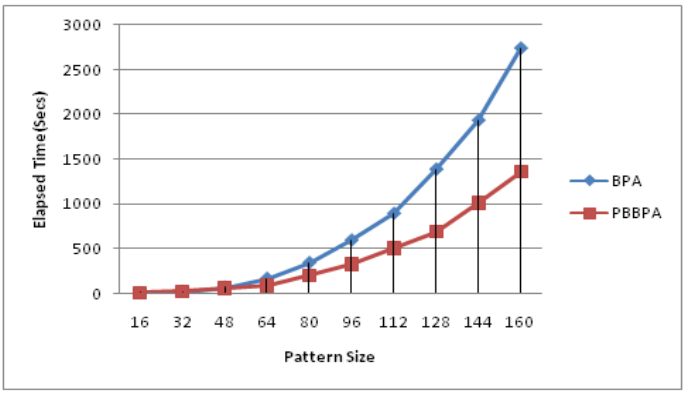

Figure 8. Performance of PBBPA vs BPA

4.3. Performance Comparison of PBBMA and BMA

A stack of six RBM layers are arranged for the composition of a DBM to generate the original binary pattern for a given incorrect pattern as input. The observations are obtained by training the network for 30,000 iterations in both forward and backward directions for each layer. Table 2 demonstrates that proposed Parallel Blocked implementation (PBBMA) shows significant improvement in decreasing Deep Neural Network training time compared to direct implementation (BMA). From the Figure 9 it is completely clear that the training time further reduced as the size of training pattern is increasing.

Table 2. Training time comparison of BMA and PBBMA

Pattern Size	Elapsed time(Secs)		
	BMA	PBBMA	Speedup
16	4.61	15.35	0.30
32	36.04	37.47	0.96
48	122.07	99.20	1.23
64	338.29	152.81	2.21
80	573.99	342.74	1.67
96	979.43	548.99	1.78
112	1523.43	853.87	1.78
128	2454.65	1166.90	2.10
144	3290.57	1737.87	1.89
160	4640.19	2353.60	1.97

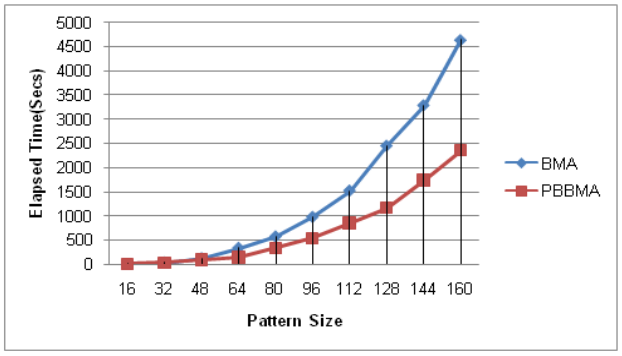

Figure 9. Performance of PBBMA vs BMA

5. CONCLUSIONS AND FUTURE WORK

In this paper, optimal block size of matrix has been chosen experimentally for Parallel Blocked Matrix multiplication to improve the performance of matrix multiplication which is a key operation and used for several times during the training of Deep Neural Network algorithms. Then it has been proposed to use Parallel Blocked Matrix multiplication with optimal block size to modify Backpropagation Algorithm (BPA) and Boltzmann Machine Algorithm (BMA). It was experimentally verified that the proposed methods (PBBPA and PBBMA) perform the training of Deep Neural Networks approximately two times faster than the existing classic algorithms.

For the future work, these Deep Neural Network algorithms will be implemented using parallel programming model CUDA and tested on many-core GPU systems.

REFERENCES

[1] Teddy Surya Gunawan, Ahmad Fakhrur Razi Mohd Noor, Mira Kartiwi, "Development of English Handwritten Recognition Using Deep Neural Network", *Indonesian Journal of Electrical Engineering and Computer Science*, May 2018, 10(2), 562-568.
[2] Dan Ciresan, Ueli Meier, Jonathan Masci, Jurgen Schmidhuber, "Multi-column deep neural network for traffic sign classification", Neural Networks, 2012 special issue, 333-338.
[3] Andre Stuhlsatz, Jens Lippel, and Thomas Zielke, "Feature Extraction with Deep Neural Networks by a Generalized Discriminate Analysis", IEEE Transactions on Neural Networks and Learning Systems, April 2012, 23(4), 596-608.
[4] Dr. Priya Gupta, Nidhi Saxena, Meetika Sharma, Jagriti Tripathi, "Deep Neural Network for Human Face Recognition", International Journal of Engineering and Manufacturing, 2018, 1, 63-71.
[5] Sarat Chandra Nayak, "Development and Performance Evaluation of Adaptive Hybrid High Order Neural Networks for Exchange Rate Prediction", International Journal of Intelligent Systems and Applications, 2017, 8, 71-85.
[6] Asma EIAdel, Mourad Zaied, Chokri Ben Amar, "Fast DCNN based on FWT, intelligent dropout and layer skipping for image retrieval", Neural Networks, 2017, 10-18.
[7] Monica S, Lam, Edward E., Rothberg and Michael E. Wolf, "The Cache Performance and Optimizations of Blocked Algorithms", Proceedings of the fourth international conference on architectural support for programming languages and operating systems – ASPLOS-IV, 1991.
[8] Nenad Anchev, Marjan Gusev, Sasko Ristov, and Blagoj Atanasovski,"Some Optimization Techniques of the Matrix Multiplication Algorithm", Proceedings of the 35th International Conference on Information Technology Interfaces, ITI – 2013.

[9] Asif Muhammad, Muhammad Arshad Islam, "Performance Evaluation of Matrix Multiplication in Virtual Machine", International Conference on Communication, Computing and Digital Systems, 2017.
[10] Ananth M., Vishwas S., Dr. Anala M R, "Cache Friendly Strategies to Optimize Matrix Multiplication", IEEE 7th International Advance Computing Conference, 2017.
[11] Sasko Ristov, Marjan Gusev, "Super linear Speedup for Matrix Multiplication", Proceedings of the 35th International Conference on Information Technology Interfaces, ITI – 2012.
[12] D.T.V. Dharmajee Rao, K.V. Ramana, "Winograd's Inequality: Effectiveness for Efficient Training of Deep Neural Networks", International Journal of Intelligent Systems and Applications, accepted on 25-04- 2018, "In Press".
[13] Omaima N., Ahmad AL-Allaf, "Fast Back Propagation Neural Network Algorithm for Reducing Convergence Time of BPNN Image Compression", Proceedings of the 5th International Conference on IT & Multimedia, 2011.
[14] Noel Lopes, Bernardete Ribeiro and Joao Goncalves,"Resticted Boltzmann Machines and Deep Belief Networks on Multi-Core Processors", IEEE World Congress on Computational Intelligence, WCCI 2012.
[15] Sasko Ristov, Marjan Gusev, Goran Velkoski, "Optimal Block Size for Matrix Multiplication Using Blocking", Proceedings of the 37th International Conference on Information and communication Technology, Electronics and microelectronics, MIPRO, May 2014, 295-300.