I.J. Intelligent Systems and Applications, **2018, 6, 49-58**
Published Online June 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2018.06.06

Winograd's Inequality: Effectiveness for Efficient Training of Deep Neural Networks

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

K.V. Ramana
JNTUK College of Engineering, JNTUK University, Kakinada - 533003, Andhra Pradesh, India
E-mail: vamsivihar@gmail.com

Received: 02 November 2017; Accepted: 24 April 2018; Published: 08 June 2018

Abstract—Matrix multiplication is widely used in a variety of applications and is often one of the core components of many scientific computations. This paper will examine three algorithms to compute the product of two matrices: the Naive, Strassen's and Winograd's algorithms. One of the main factors of determining the efficiency of an algorithm is the execution time factor, how much time the algorithm takes to accomplish its work. All the three algorithms will be implemented and the execution time will be calculated and we find that Winograd's algorithm is the best and fast method experimentally for finding matrix multiplication. Deep Neural Networks are used for many applications. Training a Deep Neural Network is a time consuming process, especially when the number of hidden layers and nodes is large. The mechanism of Backpropagation Algorithm and Boltzmann Machine Algorithm for training a Deep Neural Network is revisited and considered how the sum of weighted input is computed. The process of computing the sum of product of weight and input matrices is carried out for several hundreds of thousands of epochs during the training of Deep Neural Network. We propose to modify Backpropagation Algorithm and Boltzmann Machine Algorithm by using fast Winograd's algorithm. Finally, we find that the proposed methods reduce the long training time of Deep Neural Network than existing direct methods.

Index Terms—Deep Neural Networks, Backpropagation Algorithm, Boltzmann Machine Algorithm, Matrix multiplication algorithms: Naive, Strasen's, Winograd's algorithms.

I. Introduction

Matrix multiplication is an important problem in Mathematics and Computer Science. It plays an important role in the areas of Graph Theory, Computer Graphics, Digital Signal Processing and Neural Networks. Fast Matrix Multiplication is still an open problem which has attracted a lot of attention for the past few decades [1]. In this paper we will consider matrix multiplication as the problem, implement various methods to solve this problem and find the best one that takes the least time.

This paper aims to evaluate the theoretical and experimental performance of three key matrix multiplication algorithms (The Naive algorithm, Winograd's algorithm and Strassen's algorithm). Starting with the mathematical definitions of above three algorithms, this paper derives their theatrical run-time complexity and then compares these theoretical values to each algorithm's real-world performance.

Deep Neural Networks(DNN) have been widely used for various tasks, such as classification of various types of data [2-4], clustering of data [5], speech recognition [6], and natural language processing [7] to name just a few. As neural networks research moves from state-of-the-art paradigms to real world applications, the associated training time and computing requirements become an increasingly important factor affecting the comparison of neural networks with alternative, competing techniques. Thus the existence of fast and efficient learning algorithms is crucial for the evolution of this research field in the future.

The computational efficiency depends on the time spent training the network. In the worst cases scenario, the number of epochs can be exponential in n, the number of inputs. In practice, the time required for the networks to converge is highly variable. Several hundreds of thousands of epochs may be required before the weights converge. A number of techniques exist that help speed-up the training time [8]. The same paper proposes a technique of replacing standard matrix multiplication (Naive) for computing the sum of weighted input with Winograd's matrix multiplication to speed-up the training time of a neural network.

This paper is organized as follows. Section II presents related work. Section III presents mathematical definitions, theoretical and experimental comparison of three matrix multiplication algorithms. Section IV presents Winograd's implementation of Backpropagation Algorithm and its performance comparison with direct method. Section V presents Winograd's implementation of Boltzmann Machine Algorithm and its performance

comparison with existing direct method. Finally, section VI contains conclusions and future work.

II. Related Work

For the past few years, a huge research is going on actively to cut the computing cost of complex computer algorithms in various areas by applying innovative techniques like Winograd's method instead of direct and straightforward methods. Kuo-Liang et al. proposed a simple Improved Full Search (IFS) for Vector Quantization (VQ) based on Winograd's Identity [9]. In this letter, authors have mentioned that although VQ is an efficient method for low bit rate image compression, it is time consuming for high-dimensional vectors in the search phase. Employing Winograd's Identity, they presented simple IFS in order to cut the computation time in FS for VQ nearly 50%. They have carried out some experiments on four popular images as the bench mark to evaluate the performance of the proposed method. The four adopted images are Lena, Baboon, F16 and Pepper. The experimental results showed that the proposed method cuts computation time nearly 50%. Winograd's method: a perspective for some pattern recognition problems [10] is presented, In this article, N.B. Venkateswarulu and P.S.V.S.K. Raju used Winograd's method with Euclidean distance, Mahalanobis distance and Maximum Likelyhood (ML) classifiers to reduce their computational time requirements. They performed experiments with six band thematic mapper data. Their result showed that the proposed algorithms for Euclidean and ML classifiers are observed to be two times faster than their literal algorithms. In the case of the Mahalanobis distance classifier, the proposed fast algorithm is showing a speed-up of 7. Utility of this Winograd's method with other Pattern Recognition algorithms is also discussed. Ch. Ramesh et al. implemented fast DCT algorithm using Winograd's method [11]. In this paper, they presented Winograd's matrix multiplication approach for forward Discrete Cosine Transform (DCT) and inverse Discrete Cosine Transform (IDCT) computation to reduce their CPU time. Experiments are conducted with standard images and synthetic images. In their study, authors have used 15 gray level images and 15 color images in tiff format including widely used Lena, Mandrill and Pepper images. From their experiments it is shown that Winograd's based DCT and IDCT algorithms are most preferred algorithms as they consume very less CPU time compared to conventional implementation and MATLAB. D.J. Nagendra Kumar et al. implemented Expectation Maximization (EM) Clustering using Winograd's method to reduce the computing cost [12]. In this paper, authors have proposed eight methods to speed-up quadratic-term computation in EM. All methods are implemented and executed on synthetic datasets with varying number of dimensions, clusters and samples. Out of all approaches, the result showed that Winograd's implemented one is the fastest approach.

III. Three Matrix Multiplication Algorithms

As we mentioned before, there are three methods to calculate the multiplication of matrices. All of them give the same result but each one consumes different space in memory and takes different processor time. The methods that we will examine are:

i. Naive algorithm
ii. Strassen's algorithm
iii. Winograd's algorithm

The above three algorithms are defined firstly and then the performances of the algorithms are compared theoretically and by experiments.

A. Naive Algorithm

$$C_{i,j} = \sum_{k=1}^{n} a_{i,k}.b_{k,j} \tag{1}$$

The naive algorithm is solely based on the familiar mathematical definition for the multiplication of two matrices, as shown in (1). To compute each entry in the final $n \times n$ matrix, we need exactly n multiplications and $n-1$ additions. And since each of the n^2 entries in the first matrix (A) is multiplied by exactly n entries from the second matrix (B), the total number of multiplications is $n \times n^2 = n^3$, and the total number of additions is $(n-1)n^2 = n^3 - n^2$.

B. Strassen's Algorithm

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \mathrm{X} \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} \tag{2}$$

$$\begin{aligned} P &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ Q &= (A_{21} + A_{22})B_{11} \\ R &= A_{11}(B_{12} - B_{22}) \\ S &= A_{22}(B_{21} - B_{11}) \\ \mathrm{T} &= (\mathrm{A}_{11} + \mathrm{A}_{12})\mathrm{B}_{22} \\ \mathrm{U} &= (\mathrm{A}_{21} - \mathrm{A}_{11})(\mathrm{B}_{11} + \mathrm{B}_{12}) \\ \mathrm{V} &= (\mathrm{A}_{12} - \mathrm{A}_{22})(\mathrm{B}_{21} + \mathrm{B}_{22}) \end{aligned} \tag{3}$$

$$\begin{aligned} C_{11} &= P + S - T + V \\ C_{12} &= R + T \\ C_{21} &= Q + S \\ C_{22} &= P + R - Q + U \end{aligned} \tag{4}$$

Strassen devised a clever method of dividing the given matrices into four sub-matrices and then recursively multiplying them to obtain the resultant matrix. This method is known as "divide-and-conquer" and adds not only elegance but also improved theoretical performance [13]. First, the two matrices are divided into four

quadrants, or $n/2$ by $n/2$ submatrices each as shown in (2). Now, via a series of intermediate variables, Strassen's method uses only seven multiplications and eighteen additions for each recursive call in order to compute the final matrix product as shown in (3) and (4) instead of the usual eight multiplications and four additions as in case of Naive method.

Since Strassen's algorithm is recursive, we can solve a set of recurrence relations for the number of scalar multiplications and additions. It is relatively straightforward to show that the number of multiplications for an n-by-n matrix is $n^{2.807}$ and the number of additions is $(4.5)n^2 \log_2 n$.

C. Winograd's Algorithm

$$A_i = \sum_{k=1}^{n/2} a_{i,2k-1} . a_{i,2k} \tag{5}$$

$$B_j = \sum_{k=1}^{n/2} b_{2k-1,j} . b_{2k,j} \tag{6}$$

$$C_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - A_i - B_j \tag{7}$$

As demonstrated in [14] Winograd's algorithm is defined as shown in (5), (6), and (7). Instead of multiplying individual numbers as in the naive algorithm, Winograd's algorithm uses pair wise multiplication of couples of entries and then subtracts the accumulated error.

Since A_i and B_j are pre-computed only once for each row and column, they require only n^2 scalar multiplications. The final summation does require $O(n^3)$ multiplications, but only half of those in the naive algorithm. Thus, the total number of scalar multiplications has been reduced to $\frac{1}{2}n^3 + n^2$. However, the number of additions has been increased by $\frac{1}{2}n^3$.

D. Theoretical Study of Matrix Multiplication Algorithms

Table 1 compares the three algorithms in terms of the number of additions, the number of multiplications and complexity as function of the matrix parameter n. Both Naive and Winograd's algorithms appear to run as the cube of n, but Winograd's algorithm is faster by a constant because of the tradeoff between additions and multiplications. The Naive algorithm always had the lowest number of scalar additions with the largest number of scalar multiplications where as Winograd's algorithm had the lowest number of multiplications with the largest number of additions. Strassen's Algorithm performs more quickly as it has fewer multiplications than other two algorithms assuming that the recursive calls of Strassen's algorithm should not take up too much time theoretically. It clearly appears that Strassen's multiplication algorithm is theoretically the best algorithm than Naive and Winograd's algorithms.

Table 1. Theoretical Comparison of Three Algorithms

Algorithm	No. of Multiplications	No. of Additions	Complexity
Naïve	n^3	$n^3 - n^2$	$O(n^3)$
Strassen's	$n^{2.807}$	$(4.5)n^2 \log_2 n$	$O(n^{2.807})$
Winograd's	$\frac{1}{2}n^3 + n^2$	$\frac{3}{2}n^3 - n^2$	$O(n^3)$

E. Experimental Study of Matrix Multiplication Algorithms

The performance comparison of our interest is time consumption. All the algorithms were implemented using C++ and then tested, 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, 64 Bit, 4 GB RAM) with different booting (Windows 7 Ultimate and GNU /Linux 4.4.0 – 57 generic#78-Ubuntu SMP Operating Systems). Software versions are as follows: Microsoft Visual Studio 2008 VC++ and g++ (Ubuntu 5.4.0-6 ubuntu~16.04.4) 5.4.0. The same environment has been used for all other experiments that were carried out in this paper. All the implemented algorithms were rerun for five times and the average execution time was calculated.

As with the Strassen's recursive algorithm, we assume that square matrices, with a size that is a power of 2, are used for all matrices. With careful implementation, the experiments can be carried out to more general matrices that may not have power of 2 dimensions. Matrix elements are produced by a random function and they are converted into required data type.

The time costs for all three algorithms are demonstrated by the Tables 2, 3, 4, and 5 for integer, long integer, float and double data types respectively. From the tables, It has been observed that the Strassen's algorithm performs more slowly than expected in the theoretical study mainly due to the large number of recursive calls, which theoretically should not take up too much time, large number of stack operations, and large addressing headers in order to contain all four sub-matrix pointers. Winograd's algorithm is faster than the Naïve algorithm both theoretically and experimentally, because computers add faster than they multiply, while the total number of operations remains almost unchanged. It is clearly appeared in all the tables below that the Winograd's algorithm takes less time than other two algorithms for all data types.

Table 2. Experimental Comparison of Three Algorithms for Integer Data under Windows and Linux Environments

Matrix Dimension	Elapsed time (secs)									
	Windows Environment					Linux Environment				
	Naive	Strassen's	Winograd's	Speed-up of winograd's		Naive	Strassen's	Winograd's	Speed-up of winograd	
				Over naive	Over Strassen's				Over naive	Over Strassen's
2	0.000001	0.000001	0.000008	0.08	0.12	0.000001	0.000001	0.000003	0.29	0.47
4	0.000001	0.000162	0.000008	0.12	21.44	0.000001	0.000011	0.000002	0.73	6.76
8	0.000005	0.001051	0.000012	0.41	84.93	0.000006	0.000055	0.000005	1.17	10.36
16	0.000039	0.007692	0.00003	1.32	260.02	0.000048	0.000358	0.000031	1.53	11.53
32	0.000299	0.050205	0.000077	3.89	652.26	0.000354	0.002511	0.000235	1.51	10.67
64	0.001037	0.231829	0.000669	1.55	346.43	0.002847	0.016569	0.001578	1.8	10.5
128	0.008584	1.594599	0.00491	1.75	324.74	0.020583	0.085338	0.008745	2.35	9.76
256	0.073749	10.050447	0.045367	1.63	221.54	0.107674	0.362238	0.062703	1.72	5.78
512	0.775164	78.68554	0.436088	1.78	180.43	0.92492	2.52392	0.562478	1.64	4.49
1024	7.42673	439.95222	6.439806	1.15	68.32	7.510656	17.83601	4.48925	1.67	3.97

Table 3. Experimental Comparison of Three Algorithms for Long Integer Data under Windows and Linux Environments

Matrix Dimension	Elapsed time (secs)									
	Windows Environment					Linux Environment				
	Naive	Strassen's	Winograd's	Speed-up of winograd's		Naive	Strassen's	Winograd's	Speed-up of winograd	
				Over naive	Over Strassen's				Over naive	Over Strassen's
2	0.000001	0.000001	0.000008	0.12	0.15	0.000001	0.000001	0.000003	0.31	0.35
4	0.000001	0.000171	0.000007	0.13	23.54	0.000002	0.000016	0.000002	1.08	10.18
8	0.000006	0.00112	0.000012	0.48	92.73	0.00001	0.000078	0.000007	1.44	11.49
16	0.00004	0.008116	0.00003	1.36	274.38	0.000047	0.000406	0.000032	1.47	12.75
32	0.000301	0.056046	0.000074	4.08	760.99	0.00036	0.002578	0.000225	1.6	11.46
64	0.001018	0.237391	0.000586	1.74	404.98	0.002911	0.018132	0.001604	1.81	11.3
128	0.008475	1.663021	0.00504	1.68	329.97	0.021652	0.087398	0.009198	2.35	9.5
256	0.072891	10.550669	0.045064	1.62	234.13	0.133874	0.388812	0.072268	1.85	5.38
512	0.76461	82.012985	0.429161	1.78	191.1	1.056532	2.679844	0.602787	1.75	4.45
1024	7.376128	463.63663	6.376367	1.16	72.71	9.2931	18.33164	6.081719	1.53	3.01

Table 4. Experimental Comparison of Three Algorithms for Float Data under Windows and Linux Environments

Matrix Dimension	Elapsed time (secs)									
	Windows Environment					Linux Environment				
	Naive	Strassen's	Winograd's	Speed-up of winograd's		Naive	Strassen's	Winograd's	Speed-up of winograd	
				Over naive	Over Strassen's				Over naive	Over Strassen's
2	0.000001	0.000001	0.000008	0.08	0.16	0.000001	0.000001	0.000002	0.68	0.46
4	0.000001	0.000162	0.000007	0.17	23.39	0.000002	0.000014	0.000002	0.68	6.16
8	0.000006	0.001051	0.000012	0.46	84.9	0.000011	0.000063	0.000009	1.15	6.71
16	0.00004	0.007622	0.000031	1.27	245.17	0.000049	0.000378	0.000036	1.37	10.62
32	0.000312	0.046014	0.000077	4.06	597.82	0.000361	0.002545	0.000231	1.56	11.01
64	0.001192	0.228117	0.000617	1.93	369.56	0.002839	0.017424	0.001618	1.76	10.77
128	0.009424	1.596176	0.005357	1.76	297.97	0.020683	0.086754	0.008642	2.39	10.04
256	0.078255	10.074509	0.048628	1.61	207.18	0.118087	0.379269	0.066494	1.78	5.7
512	0.817374	79.029947	0.459184	1.78	172.11	0.949597	2.594051	0.556212	1.71	4.66
1024	7.763171	439.9591	6.310935	1.23	69.71	7.83877	18.12968	4.510095	1.74	4.02

Table 5. Experimental Comparison of Three Algorithms for Double Data under Windows and Linux Environments

Matrix Dimension	Elapsed time (secs)									
	Windows Environment					Linux Environment				
	Naive	Strassen's	Winograd's	Speed-up of winograd's		Naive	Strassen's	Winograd's	Speed-up of winograd	
				Over naive	Over Strassen's				Over naive	Over Strassen's
2	0.000001	0.000001	0.000008	0.07	0.11	0.000001	0.000001	0.000002	0.4	0.54
4	0.000001	0.000157	0.000007	0.13	21.71	0.000001	0.00001	0.000002	0.54	4.59
8	0.000005	0.001236	0.000007	0.78	178.04	0.000008	0.000063	0.000006	1.38	11.16
16	0.000039	0.009409	0.000031	1.24	302.65	0.000046	0.000379	0.000031	1.47	12.12
32	0.000326	0.053596	0.000081	4.03	662.55	0.00035	0.002609	0.000228	1.53	11.43
64	0.001103	0.177866	0.000645	1.71	275.88	0.002883	0.017403	0.001611	1.79	10.81
128	0.009063	1.723981	0.005358	1.69	321.74	0.023482	0.08734	0.01021	2.3	8.55
256	0.101053	10.34754	0.058246	1.73	177.65	0.131894	0.384518	0.070332	1.88	5.47
512	0.86402	72.14182	0.494861	1.75	145.78	1.001293	2.597018	0.577752	1.73	4.5
1024	8.339114	603.20766	6.755024	1.23	89.3	8.821907	18.19347	5.370864	1.64	3.39

IV. Backpropagation Algorithm (BPA)

A neural network is a set of connected input/output units in which each connection has a weight associated with it. There are many different kinds of neural networks and neural network algorithms. The most popular neural network algorithm is Backpropagation learning algorithm [15].

The Backpropagation algorithm performs learning on a multilayer feed-forward neural network. It iteratively learns a set of weights for prediction of the class label of tuples. A multilayer feed-forward Deep Neural Network (DNN) consists of an input layer, more number of hidden layers and an output layer as shown in Fig. 1 [16, 17].

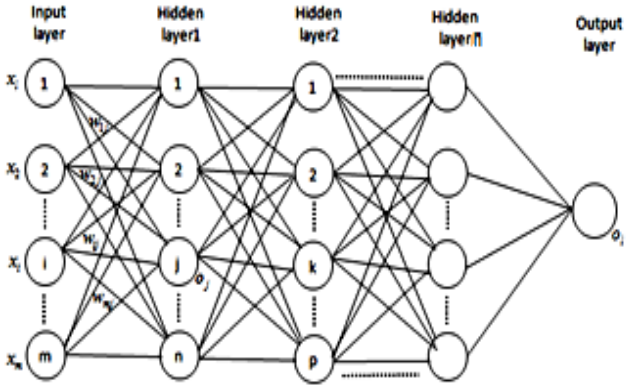
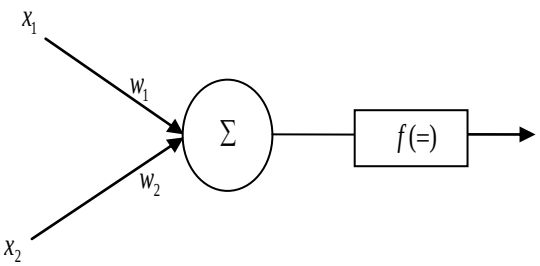


Fig.1. A multilayer feed-forward Deep Neural Network

A. Winograd's Modified Backpropagation Algorithm (WMBPA):

In order to reduce the long training time, the standard Backpropagation Algorithm (BPA) gets modified by using Winograd's matrix multiplication algorithm as explained below.

Consider a Neural Network with two-inputs and single output



Output of the neuron $= f$ (activation value)

Activation value $= w_1x_1 + w_2x_2$

Now, the above equation can be written as

$$w_1x_1 + w_2x_2 = w_1x_1 + w_2x_2 - w_1w_2 - x_1x_2 + w_1w_2 + x_1x_2$$
$$= w_1(w_2 + x_1) + x_2(w_2 + x_1) - w_1w_2 - x_1x_2$$
$$= (w_1 + x_1)(w_2 + x_1) - w_1w_2 - x_1x_2$$

i.e.

$$w_1x_1 + w_2x_2 = (w_1 + x_1)(w_2 + x_1) - w_1w_2 - x_1x_2 \quad (8)$$

Winograd's Inequality is an expansion of (8) and it can be generalized to neural networks with n-inputs as shown in (9), where n is even number. If n is odd, we can make it even by padding zero elements to both vectors to apply the method.

$$w_1x_1 + w_2x_2 + \ldots\ldots + w_nx_n = \sum_{i-1}^{n/2}(w_{2i-1} + x_{2i})(w_{2i} + x_{2i-1}) \sum_{i=1}^{n/2} w_{2i}w_{2i-1} - \sum_{i=1}^{n/2} x_{2i}x_{2i-1} \quad (9)$$

Consider a multilayer networks m - n - p, where

m designates number of nodes in the input layer.

n designates number of nodes in the hidden layer.

p designates number of nodes in the output layer.

Terminology:

$X_i = [x_{i1}\ x_{i2}\ \ldots.x_{im}\ ]$ = Training sample.

$D = \{X_1, X_2, \ldots.\ X_t\ \}$ = Training Data Set.

W_{ji} = Weight connecting neuron j in the current layer from neuron i in the previous layer.

Oj = Output at the neuron j.

$P_j = \sum_{i-1}^{s/2} w_{j2i-1}\, w_{j2i}$ Partial weight sum at the neuron j

$S = m/n$ depending upon hidden or output layers.

$Q_j = \sum_{i-1}^{s/2} x_{j2i-1}\, x_{j2i}$ Partial input sum at the neuron j

$S = m/n$ depending upon hidden or output layers.

X_{ji} = Input from neuron j in the current layer coming from the neuron i in the previous layer.

I_j = Activation value of the neuron j.

E_j = Error at neuron j in the output layer.

$PE_j = \sum_{i=1}^{s/2} E_{2i-1}E_{2i}$ Partial error sum for the neuron j in the hidden layer.

T_i = Target of the neuron j in the out layer.

l = Learning rate

Activation value

$$I_j = \sum_{i=1}^{s/2}(W_{j\,2i-1} + x_{j\,2i})(w_{j2i} + x_{j2i-1}) - P_j - Q_j)$$

Where $S = n/m$ depending on hidden or output layer.

Algorithm 1: Winograd's implementation of BPA

1. Initialize all weights and bias 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. Compute $P_j = \sum_{i=1}^{s/2}(w_{j\,2i-1} + w_{j\,2i})$
8. Compute $Q_j = \sum_{i=1}^{s/2}(x_{j\,2i-1} + x_{j\,2i})$
9. Activation Value $I_j = \sum_{i=1}^{s/2}(w_{j\,2i-1} + x_{j\,2i})$ $(w_{j\,2i} + x_{j\,2i-1}) - P_j - Q_j$
10. $O_j = \frac{1}{1 = e^{-I_j}}$ }
11. For each unit j in the output layer
12. $Ej = O_j(1-O_j)(T_j - O_j)$
13. For each unit j in the hidden layer, from the last to first hidden layer
14. $Ej = O_j(1-O_j)\sum_{i=1}^{S/2}(W_{j\,2i} + E_{2i})(W_{j\,2i-1} + E_{2i-1})$ $-P_j - PE_j$
15. For each weight W_{ji} in network {
16. $\Delta W_{ji} = (1)E_jO_j$
17. $W_{ji} = W_{ji} + \Delta W_{ji}$ }
18. For each bias θ_j in the network. {
19. $\Delta\theta_j = (1)E_j$
20. $\theta_j = \theta_j + \Delta\theta_j$ }

}}

B. Performance Comparison of BPA and WMBPA:

Updating the weights and biases after the presentation of each tuple referred to as case updating. Alternatively, the weight and bias increments could be accumulated in variables, so that the weights and biases are updated after all of the tuples in the training set have been presented. The latter strategy is called epoch updating, where one iteration through the training set is an epoch [18]. In our implementation, Backpropagation employed epoch updating.

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. We have considered 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 epochs. The logistic or sigmoid function is used as activation function.

Table 6 Demonstrates that proposed Winograd's implementation (WMBPA) shows significant improvement in decreasing neural network training time compared to Naive implementation (BPA). From the Fig. 2 it is completely clear that the training time further reduced as we are increasing training pattern size. Fig. 3 shows speed-up of WMBPA over BPA under windows & Linux environments.

Table 6. Training Time Comparison of BPA and WMBPA under Windows and Linux Environments

Pattern size	Elapsed time (Secs)					
	Windows Environment			Linux Environment		
	BPA	WMBPA	SPEED UP	BPA	WMBPA	SPEED UP
10	0.93	0.65	1.43	1.06	0.74	1.42
20	6.52	4.42	1.48	7.74	5.25	1.47
30	22.82	13.59	1.68	25.59	16.60	1.54
40	54.24	30.71	1.77	62.43	38.37	1.63
50	104.6	58.85	1.78	118.15	74.22	1.59
60	185.9	101.1	1.84	205.55	128.34	1.60
70	293.7	161.4	1.82	327.11	202.62	1.61
80	443.7	238.9	1.86	493.19	309.80	1.59
90	620.4	340.3	1.82	696.63	425.90	1.64
100	855.1	463.8	1.84	963.22	585.39	1.65

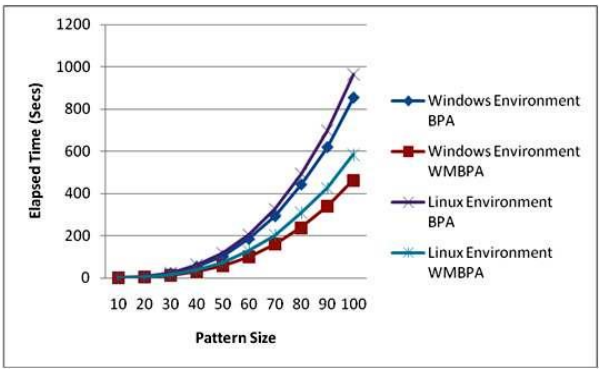


Fig.2. Performance of WMBPA vs BPA

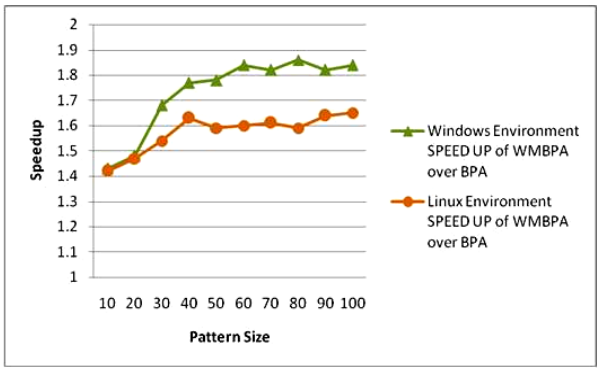


Fig.3. Speed-up of WMBPA over BPA

V. Boltzmann Machine Algorithm (BMA)

A general Boltzmann Machine is a network of symmetrically coupled stochastic binary units. It Contains a set of visible units and a set of hidden units with visible-to-visible, visible-to-hidden, and hidden-to-hidden connections. As shown in Fig. 4 a Restricted Boltzmann Machine (RBM) is Boltzmann Machine with no Hidden-to-hidden and no visible-to-visible connnections. A stack of RBMs are then composed to create a Deep Boltzmann Machine (DBM) [19].

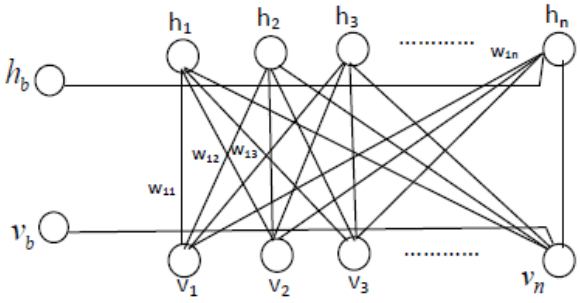


Fig.4. Restricted Boltzmann Machine (RBM)

A. *Winograd's Modified Boltzmann Machine Algorithm (WMBMA)*

In order to reduce the long training time, the direct Boltzmann Machine Algorithm(BMA) gets modified by using Winograd's matrix multiplication algorithm as per (9). Consider the following simple RBN with visible and hidden units as shown in Fig. 4.

Terminology:

Here $v_1, v_2, \ldots\ldots v_n$ are visible units.

$h_1, h_2, \ldots\ldots, h_n$ are hidden units.

v_b bias unit connecting all visible units

h_b bias unit connecting all hidden units

W_{ij} the connecting weight from visible node v_i to hidden node h_j

$V = \{v_1, v_2, \ldots\ldots v_n\}$ visible values.

$H = \{h_1, h_2, \ldots\ldots h_n\}$ hidden values.

Weight matrix W is defined as

$$W = \begin{array}{c} \\ v_1 \\ v_2 \\ . \\ . \\ v_n \end{array} \begin{pmatrix} h_1 & h_2 & \ldots\ldots & h_n \\ w_{11} & w_{12} & \ldots\ldots & w_{1n} \\ w_{21} & w_{22} & \ldots\ldots & w_{2n} \\ . & . & \ldots\ldots & . \\ . & . & \ldots\ldots & . \\ w_{n1} & w_{n2} & \ldots\ldots & w_{nn} \end{pmatrix}$$

$W^j{}_{hidden} = [w_{1j}\, w_{2j} \ldots\ldots\ldots w_{nj}]$ is the weight vector for the hidden node h_j as shown in Fig. 5.

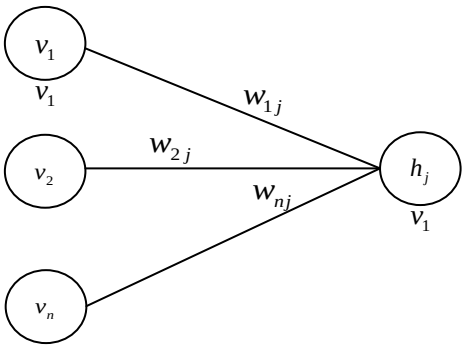


Fig.5. Representation of Vector W^j_{hidden}

$W^j visible = [w_{j1}\ w_{j2}.....w_{jn}]$ is the weight vector for the visible node v_j as shown in Fig. 6.

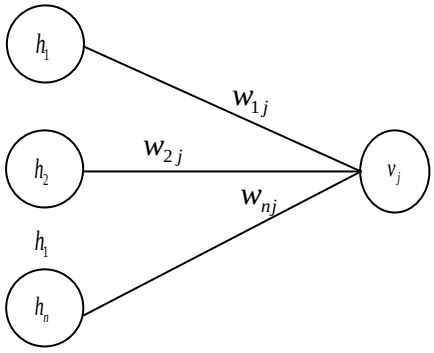


Fig.6. Representation of Vector $W^J_{vissible}$

$win(xy) =$ Winograd's multiplication between x and y^T.

$\sigma(x) = \frac{1}{1+e^{-x}}$ is the activation function for nodes.

Now we can see a Deep RBN as stack of simple RBN as illustrated in Fig. 7 and Fig. 8.

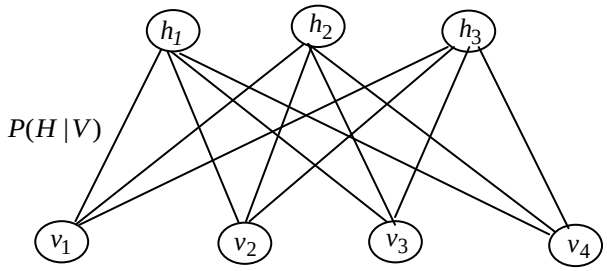


Fig.7. RBM

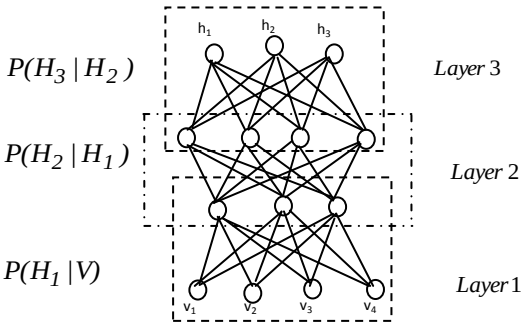


Fig.8. Stack of RBMs

For an RBM network all the visible units are conditionally independent of given hidden vector i.e.

$$P(V|H) = \prod_{i=1}^{n} P(v_i|H)$$

and also all the hidden units are conditionally independent of given visible vector i.e.

$$P(H|V) = \prod_{i=1}^{n} P(h_i|V)$$

Conditional distribution over hidden and visible units with Winograd's multiplication is given by

$$p(h_j = 1|V) = \sigma(W_{hb_j} + win(W^j_{hidden} * V))$$
$$= \frac{1}{1+e^{-(w_{hbj} + win(W^j_{hidden} * V))}}$$
$$p(v_j = 1|H) = \sigma(W_{vb_j} + win(W^j_{visible} * H))$$
$$= \frac{1}{1+e^{-(w_{vbj} + win(W^j_{visible} * H))}}$$

Algorithm 2 : Winograd's 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
$$P(h_j = 1/V) = \frac{1}{1+e^{-(w_{hbj} + win(w^j_{hidden} * V))}}$$
3. Negative phase: Reconstruct visible units using negative statistics (E_j) is given by
$$P(v_j = 1/H) = \frac{1}{1+e^{-(w_{vbj} + win(w^j_{visible} * H))}}$$
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.

B. Performance Comparison of BMA and WMBMA:

Deep Neural Networks (DNNs) are often much harder to train than shallow neural networks. If we could train deep nets they would be much more powerful than shallow nets [20]. A stack of six RBM layers are arranged for the composition of a DBM. The observations are obtained by training the network for 30,000 epochs in both forward and backward direction for each layer.

Table 7 demonstrates that proposed Winograd's implementation (WMBMA) shows significant improvement in decreasing neural network training time compared to Naive implementation (BMA). From Fig. 9 it is completely clear that the training time further reduced as we are increasing training pattern size. Fig. 10 shows speed-up of WMBMA over BMA under Windows & Linux environments.

Table 7. Training Time Comparison of BMA & WMBMA under Windows and Linux Environments

Pattern Size	Elapsed time (Secs)					
	Windows Environment			Linux Environment		
	BMA	WMBMA	SPEED UP	BMA	WMBMA	SPEED UP
10	1.65	1.17	1.41	1.75	1.27	1.37
20	11.70	7.65	1.53	13.17	8.97	1.47
30	39.16	23.21	1.69	44.24	28.49	1.55
40	93.55	52.94	1.77	107.10	65.76	1.63
50	178.98	101.24	1.77	202.61	126.80	1.60
60	316.45	175.55	1.80	352.00	218.14	1.61
70	509.24	280.55	1.82	560.24	346.57	1.62
80	757.59	408.04	1.85	850.29	528.02	1.61
90	1056.53	586.53	1.80	1198.00	733.86	1.63
100	1432.29	794.37	1.80	1656.76	1003.04	1.65

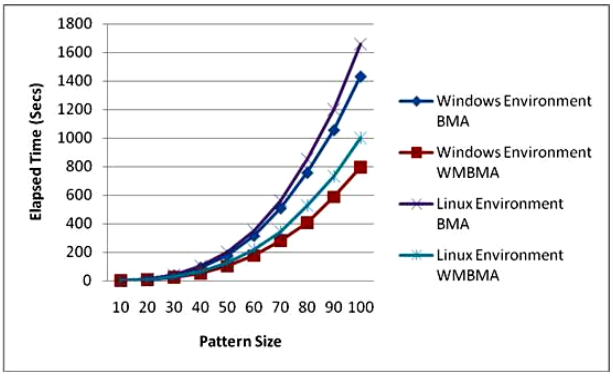


Fig.9. Performance of WMBMA vs BMA

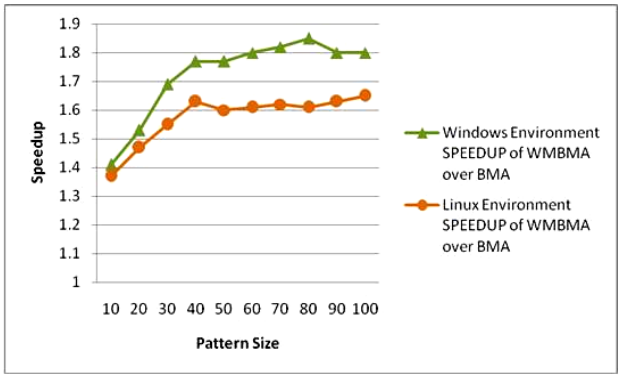


Fig.10. Speed-up of WMBMA over BMA

VI. Conclusions and Future Work

In this paper, we examined three matrix multiplication algorithms: the Naive, Strassen's and Winograd's. After our experiments, we found that Winograd's method is faster than other two algorithms for finding the matrix multiplication. Then it has been proposed to use Winograd's algorithm to modify Backpropagation Algorithm (BPA) and Boltzmann Machine Algorithm (BMA). It was experimentally verified that the proposed methods (WMBPA and WMBMA) perform the training of multilayer Deep Neural Networks much faster than the existing algorithms.

For the future work, we will parallelize the Deep Neural Network algorithms. These algorithms will further be implemented using parallel programming models and tested on multi core systems.

References

[1] Khaled Thabet, "Matrix Multiplication Algorithms", IJCSNS, vol.12, No.2, February 2012, pp: 74-79.

[2] Rita Georgina Guimaraes, Renata L. Rosa, Denise De Gaetano, Demostenes Z. Rodriguez, and Graca Bressan, "Age Groups Classification in Social Network Using Deep Learning", IEEE Access, Vol. 5, pp. 10805-10816, May 2017.DOI: 10.1109/ACCESS.2017.2706674

[3] Kamini Goyal, Dapinder Kaur,"A Novel Vehicle Classification Model for Urban Traffic Surveillance Using the Deep Neural Network Model", International Journal of Education and Management Engineering(IJEME), Vol.6, No.1, pp.18-31, 2016.DOI: 10.5815/ijeme.2016.01.03

[4] Xinhuai Zou, Ming Cheng, Cheng Wang, Yan Xia, and Jonathan Li, "Tree Classification in Complex Forest Point Clouds Based on Deep Learning", IEEE Geoscience and Remote Sensing Letters, Vol. 14, No. 12, December 2017, pp. 2360-2364.

[5] C. Bhanuprakash, Y. S. Nijagunarya, and M. A. Jayaram, "Clustering of Faculty by Evaluating their Appraisal Performance by Using Feed Forward Neural Network Approach", International Journal of Intelligent Systems and Applications (IJISA), Vol.9, No.3, pp.34-40, 2017.DOI: 10.5815/ijisa.2017.03.05

[6] Haytham M. Fayek, Margaret Lech, and Lawrence Cavedon, "Evaluating deep learning architectures for Speech Emotion Recognition", Neural Networks, Special Issue 2017, Article in Press.

[7] Ruhi Srikaya, Geoffrey E. Hinton, and Anoop deoras, "Application of Deep Belief Networks for Natural Language Understanding", IEEE/ACM Transactions on Audio, Speech, and Language Processing, Vol. 22, No. 4, April 2014.

[8] John Paul T. Yusiong, "Optimizing Artificial Neural Networks using Cat Swarm Optimization Algorithm," I. J. Intelligent Systems and Applications, 2013, 01, 69-80, DOI: 10.5815/ijisa.2013.01.07

[9] Kuo-Liang Chung, Wen-Ming Yan, and Jung-Gen Wu, "A Simple Improved Full Search for Vector Quantization Based on Winograd's Identity", IEEE Signal Processing Letters, vol. 7, No. 12, December 2000, pp. 342-344.

[10] N.B. Venkteswarulu and P.S. V.S.K. Raju "Winograd's method: Aperspective for some pattern recognintion problems", Pattern Recognitions Letters, Vol-15, No.2, PP. 105-109, 1994.

[11] Ch. Ramesh, Dr. N. B. Venkateswarlu, and Dr. J. V. R. Murthy, "Fast DCT Algorithm using Winograd's Method," IJECET, vol. 3, Issue 1, January-June 2012, pp. 98-110.

[12] D. J. Nagendra Kumar, J.V.R. Murthy and N.B. Venkateswarlu, "Computation Reduction of Expectation Maximization Clustering using Winograd's Method", 4th International Conference on Electronics Computer Technology (ICECT 2012), pp. 255 to 259.

[13] Steven Huss-Lederman, Elaine M. Jacobson, Jeremy R. Johnson, Anna Tsao, and Thomas Turnbull, "Implementation of Strassen's Algorithm for Matrix Multiplication", IEEE Conference on Super Computing, January 1996. DOI: 10.1109/SUPERC.1996.183534.

[14] Udi Manber, Introduction to Algorithms: A Creative

Approach, pp.301-304, Pearson Education, New Jersey, 1989.
[15] Shaminder Singh, and Jasmeen Gill, "Temporal Weather Prediction using Backpropagation based Genetic Algorithm Technique", International Journal of Intelligent Systems and Applications, 2014, 12, 55-61. DOI:10.5815/ijisa.2014.12.08.
[16] Jurgen Schmidhuber, "Deep learning in neural networks: An overview", Neural Networks, 61, 85-117, http://dx.doi.org/10.1016/j.neunet.2014.09.003.
[17] Musab Coskun, Ozal Yildirim, and Aysegul Ucar, and Yakup Demir, "An Overview of Popular Deep Learning Methods", EJT,Vol. 7, Number 2, pp. 165-176, December 2017, DOI: 10.23884/EJT.2017.7.2.11
[18] Pierre Baldi, Peter Sadowski, and Zhiqin Lu, "Learning in the Machine: The Symmetries of the Deep Learning Channel," Neural Networks 95(2017), pp. 110-114, http://dx.doi.org/10.1016/j.neunet.2017.08.008
[19] C. L. Philip Chen, Chun-Yang Zhang, Long Chen, and Min Gan, "Fuzzy Restricted Boltzmann Machine for the Enhancement of Deep Learning", IEEE Transactions on Fuzzy Systems, vol.23, No.6, Dec.2015, pp.2163-2173.
[20] Mohammad Rafiqul Alam, Mohammed Bennamoun, Roberto Togneri, and Ferdous Sohel, "A joint Deep Boltzmann Machine (jDBM) Model for Person Identification Using Mobile Phone Data", IEEE Transactions on Multimedia, vol. 19, No. 2, February 2017, pp. 317-326.

Authors' Profiles

D.T.V. Dharamajee Rao is currently working as Professor in the Department of Computer Science and Engineering at Aditya Institute of Technology and Management, Tekkali, Srikakulam, Andhra Pradesh, India. He received B.Tech. degree in Computer Science and Engineering in 1993 and M.Tech. degree in Computer Science and Technology in 2001 from Andhra University, Visakhapatnam, Andhra Pradesh, India. He is pursuing Ph.D. in the Department of Computer Science and Engineering, JNT University, Kakinada, Andhra Pradesh, India. He got published more than 12 papers in International and National, Conferences and Journals. His current Research interests include Data Mining, Neural Networks, Parallel Programming and Linear Algebra Techniques.

K.V. Ramana received B.Tech. degree in Electronics and Communication Engineering from JNT University, Hyderabad. Telangana, India in 1986, M.Tech. degree in Computer Science and Engineering from University of Hyderabad, Hyderabad, Telangana, India in 1990, and Ph.D. in Computer Science and Engineering from Rayalaseema University, Kurnool, Andhra Pradesh, India in 2011. He is working as Professor in the Department of Computer Science and Engineering, JNTUK College of Engineering, JNTUK University, Kakinada, Andhra Pradesh, India. He got published more than 20 papers in International and National, Conferences and Journals. His research interests include Data Warehousing and Mining, Neural Networks, Image Processing, and Pattern Recognition.

How to cite this paper: 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(IJISA), Vol.10, No.6, pp.49-58, 2018. DOI: 10.5815/ijisa.2018.06.06

