Fast Information Detection in Big Data using Neural Networks and Matrix Decomposition

Hazem M. El-Bakry

Information Systems Dept.,
Faculty of Computers and Information Sciences,
Mansoura University, EGYPT.
Email: elbakry@mans.edu.eg

Abstract: - In previous work, fast neural networks (FNNs) for information extraction was presented. The operation of these networks relies on performing cross-correlation between the input patterns and the weights of processing elements in the frequency domain. In this paper, a new strategy to accelerate this approach is introduced. Such strategy applies the concept of divide and conquer to reduce the number of calculation steps required by FNNs. The big data matrix is decomposed into smaller sub-matrices. Each generated sub-matrix is processed by using a single FNN implemented in the frequency domain. As a result, the speed up ratio is increased with the size of the input big data matrix. This is in contrast to using only FNNs. Simulation results show that the proposed approach for information detection in big data is faster than the conventional neural networks and FNNs. Moreover, experimental results for big data matrices with different sizes show good performance.

Key-Words: - Big data, Neural networks, Cross correlation, Frequency domain, Fast information detection

1 Introduction

Information detection in big data has many important applications in different fields. Examples are the data stored in our computers, storage devices, mobile communications, social networks, Internet documents, Internet search indexing and sensor networks. In these applications, high speed detection is a critical issue. Therefore, in this paper a new algorithm for fast information detection in big data is introduced. Big data is term that describes the exponential growth of vast amount of data. Big data usually includes data sets with large sizes beyond the ability of existing software tools to process input data within a short time. When dealing with big data, organizations face difficulties in being able to manipulate theses big data to get the required information. Analyzing big data is a complex problem in data processing because standard tools and procedures are not designed to search among these massive data [7-12].

The speed of neural networks was enhanced for information extraction from big data [1]. This was done by applying cross-correlation in the frequency domain between the matrix of big data and weights of neural networks. Our theory of FNNs was applied successfully for speeding up object/face detection [2-6]. The fundamental contribution of this paper is to further accelerate FNNs and achieve faster FNNs.

The rest of this paper is organized as follows: section 2 describes the concept of FNNs for information detection in a given matrix. A faster approach for information detection by applying matrix decomposition is presented in section 3. Applying parallel processing techniques to accelerate this new approach is introduced in section 4.

2 Fast Information Detection using Neural Networks

In this section, a fast algorithm for information detection is presented. Such algorithm relies on performing cross correlations between the tested matrix and the sliding window (20x20 elements) in two dimensions. The sliding window contains the weights that located between the input units and the first hidden layer. According to the convolution theorem, the convolution of r with s in the spatial domain is typically equivalent to the final output of the following procedures:

1. Calculate the Fourier transform of r and s and let the results be R and S.

2. Multiply R and S element by element. Let the result of this product D.
3. Finally, compute the inverse Fourier transform of D to achieve the same result of convolution in the spatial domain.

The cross correlations can be realized in the frequency domain by the same way. Thus, information detection process can be accelerated in an order of magnitude [1-6].

In the test phase, a sub matrix I of size mxn (sliding window) is selected from the input big data matrix with size PxT. Such sub-matrix is applied to the first hidden layer of the neural network. Assume that W_i is the vector of weights located between the input sub matrix and the first hidden layer. Such vector has mxn elements and can be described as mxn dimensional matrix. The final result of each neuron in the hidden layer h(i) can be expressed according to the following equation:

$$h_i = a\left(\sum_{j=1}^{m}\sum_{k=1}^{n} X_i(j,k)I(j,k) + b_i\right) \qquad (1)$$

where, b(i) is the bias of each hidden neuron (i) and a is the nonlinear activation function. The output of each hidden processing element for a tested sub-matrix I is represented by Eq.1. It can be evaluated for the big data matrix Z as:

$$h_i(u,v) = a\left(\sum_{j=-m/2}^{m/2}\sum_{k=-n/2}^{n/2} X_i(j,k)\ Z(u+j,v+k) + b_i\right) \qquad (2)$$

Eq.2 is equivalent to cross correlation operation performed in time domain. For any two matrices f and d, their cross correlation can be determined by [13]:

$$f(x,y)\otimes d(x,y) = \left(\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} f(x+m,y+n)d(m,n)\right) \qquad (3)$$

Therefore, Eq.2 may be expressed as follows:

$$h_i = a\left(Z\otimes X_i + b_i\right) \qquad (4)$$

where, h_i is the activity of the hidden processing element (i) when the weight matrix is applied at location (p,t) and (p,t) $\in$[U-m+1,V-n+1].

Eq. 4 can be realized in the frequency domain as follows:

$$Z\otimes X_i = F^{-1}\left(F(Z)\bullet F^*\left(X_i\right)\right) \qquad (5)$$

So, by performing cross correlation operation between the input big data matrix and the weights of the first hidden layer, the speed of information detection in big data will be increased compared to traditional neural networks. By the same way, the final result can be represented according to the following equation:

$$O(p,t) = a\left(\sum_{i=1}^{\mu} w_o(i)\, h_i(p,t) + b_o\right) \qquad (6)$$

where, μ is the number of neurons in the first hidden layer. b_o is the bias of output neuron. O(p,t) is the output of the neural network when the sliding window of weights is located at the position (p,t) in the tested big data matrix Z, and w_o represents the vector of weights situated between the last hidden layer and the output stage.

The computational complexity of cross correlation implemented in the frequency space can be investigated as explained below [1-6]:

1- For a given matrix of NxN elements, a number of complex calculation steps equal to $N^2\log_2 N^2$ is required to compute 2D-FFT. Computing the 2D-FFT of the weight matrix for each neuron in the first hidden layer requires the same number of complex calculation steps.

2- Furthermore, the inverse 2D-FFT must be determined for each processing element in the first hidden layer. Therefore, μ backward and (μ+1) forward transforms have to be computed. As a result, for a NxN tested matrix, the total number of the 2D-FFT to compute is $(2\mu+1)N^2\log_2 N^2$.

3- In addition, the processed big data matrix and the weights must be dot multiplied in the frequency domain. Therefore, a number of complex calculation steps equal to μN^2 should be considered. So, $6\mu N^2$ real calculation steps are required to perform complex dot product in the frequency space.

4- The resulted number of calculation steps required by fast neural networks is complex. In order to compare with traditional neural networks such number must be converted into a real version. It is known that $(N^2/2)\log_2 N^2$ complex multiplications and $N^2\log_2 N^2$ complex additions are required to perform 2D-FFT [14]. Every complex multiplication requires 6 real calculation steps and every complex addition needs 2 real calculation steps. So, the total number of calculation steps required to perform the 2D-FFT for a big data two dimensional matrix is:

$$\rho=6((N^2/2)\log_2 N^2) + 2(N^2\log_2 N^2) \qquad (7)$$

Eq.7 can be reformulated as:

$$\rho=5(N^2\log_2 N^2) \qquad (8)$$

5- Moreover, to perform cross correlation in the frequency domain, the weight and input big data matrices must have the same size. Because the weight matrix is the smaller, a number of zeros = (N^2-n^2) must be added to that matrix. This operation requires a total real number of calculation steps = $\mu(N^2-n^2)$ for all processing elements. Furthermore, after the 2D-FFT of the weight matrix is determined, the conjugate of this matrix must be evaluated. So, a real number of calculation steps $=\mu N^2$ should be added in order to compute the conjugate of the weight matrix for all processing elements on the hidden layer. In addition, a number of real calculation steps equal to N is required to create butterflies complex numbers $(e^{-jk(2\Pi n/N)})$, where $0<K<L$. These (N/2) complex numbers are multiplied by the elements of the input big data matrix or by previous complex numbers during the calculation of 2D-FFT. Moreover, two real floating point calculation steps are required to create a complex number. As a result, the total number of calculation steps required for fast neural networks becomes:

$$\sigma=((2\mu+1)(5N^2\log_2 N^2) +6\mu N^2+\mu(N^2-n^2)+ \mu N^2 +N) \quad (9)$$

which can be simplified to:

$$\sigma=((2\mu+1)(5N^2\log_2 N^2) +\mu (8N^2-n^2) +N) \qquad (10)$$

6- Under the same conditions, by using a moving window of size nxn for the same big data matrix of NxN elements, $(\mu(2n^2-1)(N-n+1)^2)$ calculation steps are required by conventional traditional neural networks for detecting specific information in the input big data matrix. The theoretical speed up factor η can be computed as follows:

$$\eta=\frac{\mu(2n^2-1)(N^2-n^2+1)}{(2\mu+1)(5N^2\log_2 N^2)+\mu(8N^2-n^2)+N}$$
(11)

The theoretical speed up ratio with various sizes of the tested big data matrix and different sizes for the weight matrix is listed in Table 1. The practical speed up ratio for processing big data matrices of various sizes with weight matrices of different sizes is described in Table 2.

3 Faster Neural Networks for Information Detection in Big Data

Here, a novel faster neural algorithm for information detection in big data is presented. Table 3 shows the number of calculation steps required for FNNs with various matrix sizes. By investigating these values, it is clear that as the size of the big data matrix is increased, the number of calculation steps required by FNNs is much increased. For example, the number of calculation steps consumed by a matrix of size (50x50) is much less than that needed for a matrix of size (100x100). Also, the number of calculation steps needed for a matrix of size (200x200) is much less than that required for a matrix of size (400x400). As a result, for example, if a matrix of size (100x100) is divided into 4 smaller sub-matrices of size (50x50), then the speed up ratio for information detection in big data will be increased. The number of calculation steps required by faster neural networks (fast neural networks + matrix decomposition) to process a big data matrix after decomposition can be computed as follows:

1. Suppose that the size of the big data matrix has (NxN) elements.
2. Such matrix is divided into α (LxL elements) sub-matrices. So, the speed up ratio can be determined according to the following equation:

$$\alpha=(N/L)^2 \qquad (12)$$

3. Let the number of calculation steps required for processing one (LxL) sub-matrix is ε. So, the total number of calculation steps (T) needed to manipulate these smaller sub-matrices generated after matrix decomposition will be:

$$T = \alpha\,\varepsilon \tag{13}$$

The final speed up ratio (η_d) can be evaluated as follows:

$$\eta_d = \frac{\mu(2n^2-1)(N-n+1)^2}{(\mu(\alpha+1)+\alpha)(5N_s^2\log_2 N_s^2)+\alpha\mu(8N_s^2-n^2)+N_s+\Delta} \tag{14}$$

where,

Ns: is the size of each small sub-matrix, and Ns=L.

Δ: is a small number of calculation steps required to determine the final values at the boundaries between sub-matrices. Such number depends on the size of the sub-matrices.

To find information of a specific size (for ex. 20x20) in a given big data matrix of any size by using faster neural networks after matrix decomposition into smaller sub-matrices, the optimal size of these sub-matrices must be computed. By investigating the values clarified in Table 3, it is obvious that, the best size for the sub-matrix which consumes the smallest number of calculation steps is 25x25 elements. Furthermore, Fig. 1 clarifies that the fastest speed up ratio is obtained when using a sub-matrix of size (25x25). In addition, the same figure explains that the speed up ratio is increased when the size of the sub-matrix *(L)* is decreased. A comparison between the speed up ratio for FNNs and faster neural networks with various sizes of the processed big data matrices is listed in Table 4. In contrast to using only FNNs, by using faster neural networks, it is obvious that the values of speed up ratio are increased with the size of big data matrix. As shown in Fig. 2, the number of calculation steps needed by FNNs increases rapidly with the size of the input big data matrix. So, the speed up ratio is decreased with the size of the input big matrix. While in case of using faster neural networks, the number of calculation steps needed by faster neural networks is increased smoothly. Therefore, the speed up ratio is increased. Increasing the speed up ratio with the size of the input big data matrix is very important for many applications. In addition, for massively big data matrices, while the speed up ratio of FNNs for information detection is decreased, the speed up ratio still increases in case of using faster neural networks as shown in Table 5. Moreover, as shown in Fig. 3, the speed up ratio in case of faster neural networks is increased with the size of the weight matrix which has the same size (n) as the required information. For example, it is obvious that the speed up ratio for detecting information of size 30x30 is larger than that of size 20x20. Simulation results for the speed up ratio in case of using FNNs and faster neural networks are clarified in Table 6. Practical results confirm the theoretical considerations. This proves that the speed up ratio after matrix decomposition is faster than using only FNNs. In addition, the practical speed up ratio is increased with the size of the input big data matrix.

It should be taken into account that the dimensions of the generated sub matrix after matrix decomposition *(L)* must not be less than the dimensions of the required information to be detected which has the same size as the weight matrix as follows:

$$L \geq n \tag{15}$$

In order not to loss any information in the input big data matrix, Eq. 15 clarifies the constrain which governs the relation between the length of the generated sub matrix after matrix decomposition and the size of weight matrix.

For example, when searching for information of size (20x20), the big data matrix must be divided into smaller sub matrices of size not less than 20x20.

4 Faster Information Detection in Big Data using a Neural Parallel Processing Technique

In this section, a neural parallel processing technique is presented to enhance the performance of information detection in big data. As described in the previous section, for each sub-matrix, a single FNN is used to extract the required information from the matrix of big data. In order to reduce the manipulation time as well as enhance the speed up ratio of information detection, parallel processing approaches are used. Each sub-matrix is manipulated by using a single FNN simulated on a separated node in a grid/clustered system or using a single processor. The number of calculation steps (ω) manipulated by each node/processor (sub-matrices processed by one node/processor) =

$$\omega = \frac{\text{The total number of sub}-\text{images}}{\text{Number of Processors / nodes}} \tag{16}$$

$$\omega = \frac{\alpha}{\Pr} \qquad (17)$$

where, Pr is the number of nodes or processors.

The total number of calculation steps (γ) required to test a big data matrix by using this approach can be computed as follows:

$$\gamma=\omega\varepsilon \qquad (18)$$

As shown in Table 7, using a symmetric multiprocessing system with 16 parallel processing elements or 16 nodes in either a massively parallel processing system or a clustered system, the speed up ratio (with respect to conventional neural networks) for information detection is increased. By dividing each sub-matrix into smaller groups, a further reduction in the calculation steps can be done. For each group, the neural processing (multiplication of input values by weights and summation) is performed for each group by using a single processing element. This operation is done for all of these groups as well as other groups in all of the sub-matrices at the same time. The optimal case can be achieved when each group contains only one element. In this case, one operation is required to multiply one element by its weight and also a small number of calculation steps is needed to compute the total summation for each sub-matrix. If the sub-matrix has n^2 elements, then the required number of processing elements will be n^2. Therefore, the number of calculation steps will be $\alpha q(1+\beta)$, where β is a small number depending on the value of n. For example, when n=20, then $\beta=6$ and if n=25, then $\beta=7$. The speed up ratio can be computing as:

$$\eta=O((2n^2-1)(N-n+1)^2/\alpha(1+\beta)) \qquad (19)$$

Furthermore, if the number of processing elements = αn^2, then the number of calculation steps will be $q(1+\beta)$, and the speed up ratio can be calculated:

$$\eta=O((2n^2-1)(N-n+1)^2/ (1+\beta)) \qquad (20)$$

Moreover, if the number of processing elements = $q\alpha n^2$, then the number of calculation steps will be $(1+\beta)$, and the speed up ratio can be computed as:

$$\eta=O(q(2n^2-1)(N-n+1)^2/ (1+\beta)) \qquad (21)$$

In this case, as the length of each group is very small. As a result, there is no need to apply cross correlation between the input big data matrix and the weights of the neural network in the frequency domain.

5 Conclusion

Faster neural networks have been presented for accelerating information detection in big data. Such approach has decomposed the input big data matrix, which contains big data, into many small in size sub-matrices. This has been achieved by performing cross-correlations in the frequency domain between these sub-matrices and the weights of neural processing elements. Furthermore, by using faster neural networks (FNNs + matrix decomposition), the speed up ratio has been increased with the size of the big data. This is in contrast to using only FNNs. Experimental results for different big data matrices have proven that the proposed faster neural networks detects the required information in real-time efficiently. Moreover, it can locate the required information despite noise and missing data. Experimental results have confirmed the theoretical considerations.

References:

[1] Hazem M. El-Bakry, Nikos E. Mastorakis, Michael E. Fafalios, “Fast Information Retrieval from Big Data by using Cross Correlation in the Frequency Domain,” International Journal of Neural Networks and Advanced Applications, Vol. 1, 2014, pp. 68-72.

[2] Hazem M. El-Bakry, "An Efficient Algorithm for Pattern Detection using Combined Classifiers and Data Fusion," Information Fusion Journal, vol. 11, 2010, pp. 133-148.

[3] Hazem M. El-Bakry, "Fast Virus Detection by using High Speed Time Delay Neural Networks," Journal of Computer Virology, vol. 6, no. 2, 2010, pp. 115-122.

[4] H. M. El-Bakry, and Q. Zhao, "Fast Normalized Neural Processors For Pattern Detection Based on Cross -Correlation Implemented in the Frequency Domain," Journal of Research and Practice in Information Technology, Vol. 38, No.2, May 2006, pp. 151-170.

[5] H. M. El-Bakry, and Q. Zhao, "Speeding-up Normalized Neural Networks For Face/Object Detection," Machine Graphics & Vision Journal (MG&V), vol. 14, No.1, 2005, pp. 29-59.

[6] H. M. El-Bakry, and Q. Zhao, "Fast Time Delay Neural Networks," the International Journal of Neural Systems, vol. 15, No.6, December 2005, pp.445-455.

[7] S. del Rio, V. L pez, J.M. Benitez, F. Herrera, On the use of MapReduce for imbalanced big data using Random Forest, ElSEVIER, (2014).
[8] X.-W. Chen, Big Data Deep Learning: Challenges and Perspectives, IEEE, (2014).
[9] Yingyi Bu, Vinayak Borkar, Michael J. Carey, Joshua Rosen, Neoklis Polyzotis, Tyson Condie, Markus Weimer, Raghu Ramakrishnan, Scaling Datalog for Machine Learning on Big Data, (2012).
[10] A. Cassioli , A. Chiavaioli , C. Manes , M. Sciandrone, An incremental least squares algorithm for large scale linear classification, ElSEVIER, (2012).
[11] C.L. Philip Chen, Chun-Yang Zhang, Data-intensive applications, challenges, techniques and technologies: A survey on Big Data, ElSEVIER, (2013).
[12] Alicia Fernández a, ÁlvaroGómez a, FedericoLecumberry a,n, ÁlvaroPardo b, Ignacio Ramírez a, Pattern Recognitionin Latin Americainthe “Big Data” Era, ElSEVIER,(2014).
[13] R. Klette, and Zamperon, "Handbook of image processing operators," John Wiley & Sons, Ltd, 1996.
[14] James W. Cooley and John W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput*. 19, 297–301 (1965).

Table 1. The Theoretical Speed up Ratio for Big Data Matrices with Different Sizes

Matrix size	Speed up ratio (n=20)	Speed up ratio (n=25)	Speed up ratio (n=30)
100x100	3.67	5.04	6.34
200x200	4.01	5.92	8.05
300x300	4.00	6.03	8.37
400x400	3.95	6.01	8.42
500x500	3.89	5.95	8.39
600x600	3.83	5.88	8.33
700x700	3.78	5.82	8.26
800x800	3.73	5.76	8.19
900x900	3.69	5.70	8.12
1000x1000	3.65	5.65	8.05
1100x1100	3.62	5.60	7.99
1200x1200	3.58	5.55	7.93
1300x1300	3.55	5.51	7.93
1400x1400	3.53	5.47	7.82
1500x1500	3.50	5.43	7.77
1600x1600	3.48	5.43	7.72
1700x1700	3.45	5.37	7.68
1800x1800	3.43	5.34	7.64
1900x1900	3.41	5.31	7.60
2000x2000	3.40	5.28	7.56

Table 2. Practical Speed up Ratio for Big Data Matrices with Different Sizes

Matrix size	Speed up ratio (n=20)	Speed up ratio (n=25)	Speed up ratio (n=30)
100x100	7.88	10.75	14.69
200x200	6.21	9.19	13.17
300x300	5.54	8.43	12.21
400x400	4.78	7.45	11.41
500x500	4.68	7.13	10.79
600x600	4.46	6.97	10.28
700x700	4.34	6.83	9.81
800x800	4.27	6.68	9.60
900x900	4.31	6.79	9.72
1000x1000	4.19	6.59	9.46
1100x1100	4.24	6.66	9.62
1200x1200	4.20	6.62	9.57
1300x1300	4.17	6.57	9.53
1400x1400	4.13	6.53	9.49
1500x1500	4.10	6.49	9.45
1600x1600	4.07	6.45	9.41
1700x1700	4.03	6.41	9.37
1800x1800	4.00	6.38	9.32
1900x1900	3.97	6.35	9.28
2000x2000	3.94	6.31	9.25

Table 3. The Number of Calculation Steps Required by FNNs for Big Data Matrices of Sizes (25x25 - 1000x1000 elements), μ=30, n=20

Matrix size	No. of calculation steps in case of using FNNs
25x25	1.9085e+006
50x50	9.1949e+006
100x100	4.2916e+007
150x150	1.0460e+008
200x200	1.9610e+008
250x250	3.1868e+008
300x300	4.7335e+008
350x350	6.6091e+008
400x400	8.8203e+008
450x450	1.1373e+009
500x500	1.4273e+009
550x550	1.7524e+009
600x600	2.1130e+009
650x650	2.5096e+009
700x700	2.9426e+009
750x750	3.4121e+009
800x800	3.9186e+009
850x850	4.4622e+009
900x900	5.0434e+009
950x950	5.6623e+009
1000x1000	6.3191e+009

Table 4. The Speed up Ratio in case of using FNNs and FNNs after Big Data Matrix Decomposition into Sub-Matrices (25x25 elements) for Big Data Matrices of Different Sizes (from N=50 to N=1000, n=25, μ=30)

Matrix size	Speed up ratio in case of using FNNs	Speed up ratio in case of using FNNs after matrix decomposition
50x50	2.7568	5.0713
100x100	5.0439	12.4622
150x150	5.6873	15.6601
200x200	5.9190	17.3611
250x250	6.0055	18.4073
300x300	6.0301	19.1136
350x350	6.0254	19.6218
400x400	6.0059	20.0047
450x450	5.9790	20.3034
500x500	5.9483	20.5430
550x550	5.9160	20.7394
600x600	5.8833	20.9032
650x650	5.8509	21.0419
700x700	5.8191	21.1610
750x750	5.7881	21.2642
800x800	5.7581	21.3546
850x850	5.7292	21.4344
900x900	5.7013	21.5054
950x950	5.6744	21.5689
1000x1000	5.6484	21.6260

Table 5. The Speed up Ratio in case of using FNNs and FNNs after Big Data Matrix Decomposition into Sub-Matrices (25x25) for very large Matrices (from N=100000 to N=2000000, n=25, μ=30)

Matrix size	Speed up ratio in case of using FNN	Speed up ratio in case of using FNN after matrix decomposition
100000x100000	3.6109	22.7038
200000x200000	3.4112	22.7092
300000x300000	3.3041	22.7110
400000x400000	3.2320	22.7119
500000x500000	3.1783	22.7125
600000x600000	3.1357	22.7128
700000x700000	3.1005	22.7131
800000x800000	3.0707	22.7133
900000x900000	3.0448	22.7134
1000000x1000000	3.0221	22.7136
1100000x1100000	3.0018	22.7137
1200000x1200000	2.9835	22.7138
1300000x1300000	2.9668	22.7138
1400000x1400000	2.9516	22.7139
1500000x1500000	2.9376	22.7139
1600000x1600000	2.9245	22.7140
1700000x1700000	2.9124	22.7140
1800000x1800000	2.9011	22.7141
1900000x1900000	2.8904	22.7141
2000000x2000000	2.8804	22.7141

Table 6. The Practical Speed up Ratio in case of using FNNs and FNNs after Big Data Matrix Decomposition into Sub-Matrices (25x25 elements) for Big Data Matrices of Different Sizes (from N=100 to N=2000, n=25, μ=30)

Matrix size	Speed up ratio in case of using FNNs	Speed up ratio in case of using FNNs after matrix decomposition
100x100	10.75	34.55
200x200	9.19	35.65
300x300	8.43	36.73
400x400	7.45	37.70
500x500	7.13	38.66
600x600	6.97	39.61
700x700	6.83	40.56
800x800	6.68	41.47
900x900	6.79	42.39
1000x1000	6.59	43.28
1100x1100	6.66	44.14
1200x1200	6.62	44.95
1300x1300	6.57	45.71
1400x1400	6.53	46.44
1500x1500	6.49	47.13
1600x1600	6.45	47.70
1700x1700	6.41	48.19
1800x1800	6.38	48.68
1900x1900	6.35	49.09
2000x2000	6.31	49.45

Table 7. The Speed up Ratio in case of using FNN after Big Data Matrix Decomposition into Sub-matrices (25x25 elements) for Big Data matrices of Different Sizes (from N=50 to N=1000, n=25, µ=30) using 16 Parallel Processors or 16 Nodes

Matrix size	Speed up ratio
50x50	81.1403
100x100	199.3946
150x150	250.5611
200x200	277.7780
250x250	294.5171
300x300	305.8174
350x350	313.9482
400x400	320.0748
450x450	324.8552
500x500	328.6882
550x550	331.8296
600x600	334.4509
650x650	336.6712
700x700	338.5758
750x750	340.2276
800x800	341.6738
850x850	342.9504
900x900	344.0856
950x950	345.1017
1000x1000	346.0164

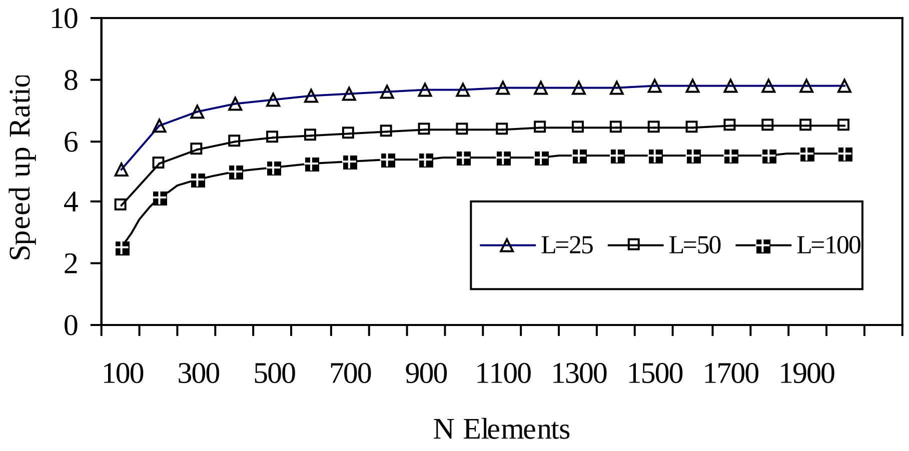

Fig. 1: The speed up ratio for big data matrices decomposed into different in size sub-matrices (L).

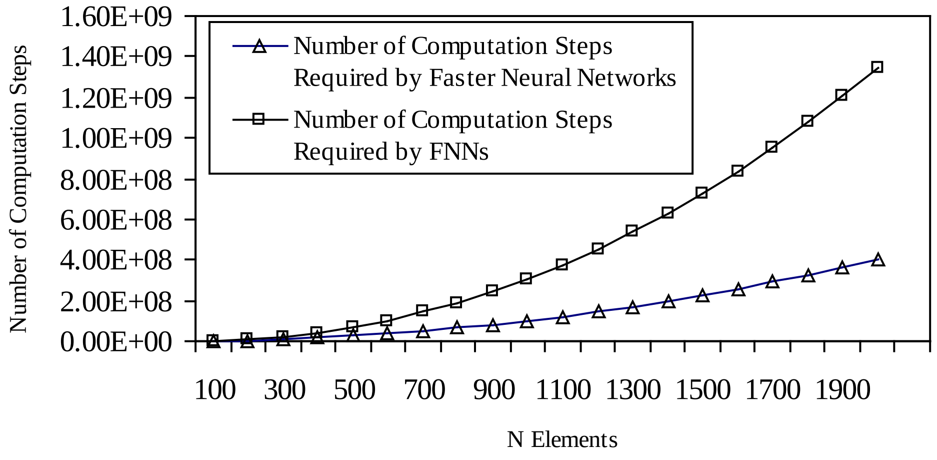

Fig. 2: A comparison between the number of calculation steps required by FNNs before and after big data matrix decomposition.

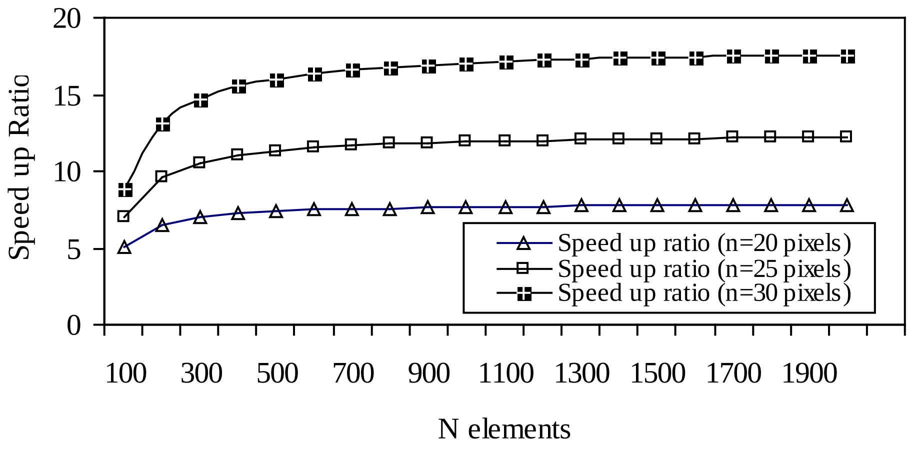

Fig. 3: The speed up ratio in case of big data matrix decomposition and different window size (n), (L=25x25).