Fast Word Detection in a Speech Using New High Speed Time Delay Neural Networks

Hazem M. El-Bakry

Faculty of Computer Science & Information Systems,
Mansoura University, EGYPT
E-mail: helbakry20@yahoo.com

Nikos Mastorakis

Technical University of Sofia,
BULGARIA

Abstract—This paper presents a new approach to speed up the operation of time delay neural networks for fast detecting a word in a speech. The entire data are collected together in a long vector and then tested as a one input pattern. The proposed fast time delay neural networks (FTDNNs) use cross correlation in the frequency domain between the tested data and the input weights of neural networks. It is proved mathematically and practically that the number of computation steps required for the presented time delay neural networks is less than that needed by conventional time delay neural networks (CTDNNs). Simulation results using MATLAB confirm the theoretical computations.

Keywords—Fast Time Delay Neural Networks, Cross Correlation, Frequency Domain, Word Detection in a speech.

1. INTRODUCTION

Recently, time delay neural networks have shown very good results in different areas such as automatic control, speech recognition, blind equalization of time-varying channel and other communication applications. The main objective of this research is to reduce the response time of time delay neural networks. The purpose is to perform the testing process in the frequency domain instead of the time domain. Our approach was successfully applied for sub-image detection using fast neural networks (FNNs) as proposed in [1-3]. Furthermore, it was used for fast face detection [7,9,18], and fast iris detection [8]. Another idea to further increase the speed of FNNs through image decomposition was suggested in [7]. In addition, it was applied for fast attack detection in computer networks [15]. By using this theory, fast painting with different colors was presented in [19]. Moreover, high speed decision tree classifier for identifying protein coding regions was presented in [20]. An interesting Internet application for fast search on web pages was presented in [11].

FNNs for detecting a certain code in one dimensional serial stream of sequential data were described in [4,5]. Compared with conventional neural networks, FNNs based on cross correlation between the tested data and the input weights of neural networks in the frequency domain showed a significant reduction in the number of computation steps required for certain data detection [1,2,3,4,5,7,8,9,11,12]. Here, we make use of our theory on FNNs implemented in the frequency domain to increase the speed of time delay neural networks.

The idea of moving the testing process from the time domain to the frequency domain is applied to time delay neural networks. Theoretical and practical results show that the proposed FTDNNs are faster than CTDNNs. In section 2, our theory on FNNs for detecting certain data in one dimensional matrix is described. Experimental results for FTDNNs are presented in section 3.

2. THEORY OF FNNS BASED ON CROSS CORRELATION IN THE FREQUENCY DOMAIN

Finding a certain word in a speech is a searching problem. Each position in the input matrix is tested for the presence or absence of the required word. At each position in the input matrix, each sub-matrix is multiplied by a window of weights, which has the same size as the sub-matrix. The outputs of neurons in the hidden layer are multiplied by the weights of the output layer. When the final output is high, this means that the sub-matrix under test contains the required word and vice versa. Thus, we may conclude that this searching problem is a cross correlation between the matrix under test and the weights of the hidden neurons.

The convolution theorem in mathematical analysis says that a convolution of f with h is identical to the result of the following steps: let F and H be the results of the Fourier Transformation of f and h in the frequency domain. Multiply F and H* (conjugate of H) in the frequency domain point by point and then transform this product into the spatial domain via the inverse Fourier Transform. As a result, these cross correlations can be represented by a product in the frequency domain. Thus, by using cross correlation in the frequency domain, speed up in an order of magnitude can be achieved during the detection process

[1,2,3,4,5,7,8,9],[14-20]. In the detection phase, a sub matrix I of size 1xn (sliding window) is extracted from the tested matrix, which has a size 1xN, and fed to the neural network. Let W_i be the matrix of weights between the input sub-matrix and the hidden layer. This vector has a size of 1xn and can be represented as 1xn matrix. The output of hidden neurons h(i) can be calculated as follows:

$$h_i = g\left(\sum_{k=1}^{n} W_i(k) I(k) + b_i \right) \quad (1)$$

where g is the activation function and b(i) is the bias of each hidden neuron (i). Equation 1 represents the output of each hidden neuron for a particular sub-matrix I. It can be obtained to the whole input matrix Z as follows:

$$h_i(u) = g\left(\sum_{k=-n/2}^{n/2} W_i(k)\, Z(u+k) + b_i \right) \quad (2)$$

Eq.2 represents a cross correlation operation. Given any two functions f and d, their cross correlation can be obtained by:

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

Therefore, Eq. 2 may be written as follows [1]:

$$h_i = g\left(W_i \otimes Z + b_i \right) \quad (4)$$

where h_i is the output of the hidden neuron (i) and $h_i(u)$ is the activity of the hidden unit (i) when the sliding window is located at position (u) and $(u) \in [N-n+1]$.

Now, the above cross correlation can be expressed in terms of one dimensional Fast Fourier Transform as follows [1]:

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

Hence, by evaluating this cross correlation, a speed up ratio can be obtained comparable to conventional neural networks. Also, the final output of the neural network can be evaluated as follows:

$$O(u) = g\left(\sum_{i=1}^{q} W_o(i)\, h_i(u) + b_o \right) \quad (6)$$

where q is the number of neurons in the hidden layer. O(u) is the output of the neural network when the sliding window located at the position (u) in the input matrix Z. W_o is the weight matrix between hidden and output layer.

The complexity of cross correlation in the frequency domain can be analyzed as follows:

1- For a tested matrix of 1xN elements, the 1D-FFT requires a number equal to $N\log_2 N$ of complex computation steps [13]. Also, the same number of complex computation steps is required for computing the 1D-FFT of the weight matrix at each neuron in the hidden layer.

2- At each neuron in the hidden layer, the inverse 1D-FFT is computed. Therefore, q backward and (1+q) forward transforms have to be computed. Therefore, for a given matrix under test, the total number of operations required to compute the 1D-FFT is $(2q+1)N\log_2 N$.

3- The number of computation steps required by FNNs is complex and must be converted into a real version. It is known that, the one dimensional Fast Fourier Transform requires $(N/2)\log_2 N$ complex multiplications and $N\log_2 N$ complex additions [13]. Every complex multiplication is realized by six real floating point operations and every complex addition is implemented by two real floating point operations. Therefore, the total number of computation steps required to obtain the 1D-FFT of a 1xN matrix is:

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

which may be simplified to:

$$\rho = 5N\log_2 N \quad (8)$$

4- Both the input and the weight matrices should be dot multiplied in the frequency domain. Thus, a number of complex computation steps equal to qN should be considered. This means 6qN real operations will be added to the number of computation steps required by FNNs.

5- In order to perform cross correlation in the frequency domain, the weight matrix must be extended to have the same size as the input matrix. So, a number of zeros = (N-n) must be added to the weight matrix. This requires a total real number of computation steps = q(N-n) for all neurons. Moreover, after computing the FFT for the weight matrix, the conjugate of this matrix must be obtained. As a result, a real number of computation steps = qN should be added in order to obtain the conjugate of the weight matrix for all neurons. Also, a number of real computation 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 matrix or by previous complex numbers during the computation of FFT. To create a complex number requires two real floating point operations. Thus, the total number of computation steps required for FNNs becomes:

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

which can be reformulated as:

$$\sigma = (2q+1)(5N\log_2 N) + q(8N-n) + N \quad (10)$$

6- Using sliding window of size 1xn for the same matrix of 1xN pixels, q(2n-1)(N-n+1) computation steps are required when using CTDNNs for certain code detection or processing (n) input data. The theoretical speed up factor η can be evaluated as follows:

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

CTDNNs and FTDNNs are shown in Figures 1 and 2 respectively.

3. Experimental Results for Time Delay Neural Networks

Time delay neural networks accept serial input data with fixed size (n). Therefore, the number of input neurons equals to (n). Instead of treating (n) inputs, our new approach is to collect all the input data together in a long vector (for example 100xn). Then the input data is tested by time delay neural networks as a single pattern with length L (L=100xn). Such a test is performed in the frequency domain as described in section II. Complex-valued neural networks have many applications in fields dealing with complex numbers such as telecommunications, speech recognition and image processing with the Fourier Transform [6,10]. Complex-valued neural networks mean that the inputs, weights, thresholds and the activation function have complex values. In this section, formulas for the speed up ratio with different types of inputs will be presented. The special case of only real input values (i.e. imaginary part=0) will be considered. Also, the speed up ratio in the case of a one and two dimensional input matrix will be concluded. The operation of FNNs depends on computing the Fast Fourier Transform for both the input and weight matrices and obtaining the resulting two matrices. After performing dot multiplication for the resulting two matrices in the frequency domain, the Inverse Fast Fourier Transform is calculated for the final matrix. Here, there is an excellent advantage with FNNs that should be mentioned. The Fast Fourier Transform is already dealing with complex numbers, so there is no change in the number of computation steps required for FNNs. Therefore, the speed up ratio in the case of complex-valued time delay neural networks can be evaluated as follows:

1) In case of real inputs

A) For a one dimensional input matrix

Multiplication of (n) complex-valued weights by (n) real inputs requires (2n) real operations. This produces (n) real numbers and (n) imaginary numbers. The addition of these numbers requires (2n-2) real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q(2n-1)(N-n+1) \quad (12)$$

The speed up ratio in this case can be computed as follows:

$$\eta = \frac{2q(2n-1)(N-n+1)}{(2q+1)(5N\log_2 N)+q(8N-n)+N} \quad (13)$$

The theoretical speed up ratio for searching short successive (n) code in a long input vector (L) using complex-valued time delay neural networks is shown in Tables 1, 2, and 3. Also, the practical speed up ratio for manipulating matrices of different sizes (L) and different sized weight matrices (n) using a 2.7 GHz processor and MATLAB is shown in Table 4.

B) For a two dimensional input matrix

Multiplication of (n^2) complex-valued weights by (n^2) real inputs requires ($2n^2$) real operations. This produces (n^2) real numbers and (n^2) imaginary numbers. The addition of these numbers requires ($2n^2$-2) real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q(2n^2-1)(N-n+1)^2 \quad (14)$$

The speed up ratio in this case can be computed as follows:

$$\eta = \frac{2q(2n^2-1)(N-n+1)^2}{(2q+1)(5N^2\log_2 N^2)+q(8N^2-n^2)+N} \quad (15)$$

The theoretical speed up ratio for detecting (nxn) real valued submatrix in a large real valued matrix (NxN) using complex-valued time delay neural networks is shown in Tables 5, 6, 7. Also, the practical speed up ratio for manipulating matrices of different sizes (NxN) and different sized code matrices (n) using a 2.7 GHz processor and MATLAB is shown in Table 8.

2) In case of complex inputs

A) For a one dimensional input matrix

Multiplication of (n) complex-valued weights by (n) complex inputs requires (6n) real operations. This produces (n) real numbers and (n) imaginary numbers. The addition of these numbers requires (2n-2) real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q(4n-1)(N-n+1) \quad (16)$$

The speed up ratio in this case can be computed as follows:

$$\eta = \frac{2q(4n-1)(N-n+1)}{(2q+1)(5N\log_2 N)+q(8N-n)+N} \quad (17)$$

The theoretical speed up ratio for searching short complex successive (n) code in a long complex-valued input vector (L) using complex-valued time delay neural networks is shown in Tables 9, 10, and 11. Also, the practical speed up ratio for manipulating matrices of different sizes (L) and different sized weight matrices (n) using a 2.7 GHz processor and MATLAB is shown in Table 12.

B) For a two dimensional input matrix

Multiplication of (n^2) complex-valued weights by (n^2) real inputs requires ($6n^2$) real operations. This produces (n^2) real numbers and (n^2) imaginary numbers. The addition of these

numbers requires ($2n^2$-2) real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta=2q(4n^2-1)(N-n+1)^2 \quad (18)$$

The speed up ratio in this case can be computed as follows:

$$\eta=\frac{2q(4n^2-1)(N-n+1)^2}{(2q+1)(5N^2\log_2 N^2)+q(8N^2-n^2)+N} \quad (19)$$

The theoretical speed up ratio for detecting (nxn) complex-valued submatrix in a large complex-valued matrix (NxN) using complex-valued neural networks is shown in Tables 13, 14, and 15. Also, the practical speed up ratio for manipulating matrices of different sizes (NxN) and different sized code matrices (n) using a 2.7 GHz processor and MATLAB is shown in Table 16.

For a one dimensional matrix, from Tables 1,2,3,4,9,10,11, and 12, we can conclude that the response time for vectors with short lengths are faster than those which have longer lengths. For example, the speed up ratio for the vector of length 10000 is faster that of length 1000000. The number of computation steps required for a vector of length 10000 is much less than that required for a vector of length 40000. So, if the vector of length 40000 is divided into 4 shorter vectors of length 10000, the number of computation steps will be less than that required for the vector of length 40000. Therefore, for each application, it is useful at the first to calculate the optimum length of the input vector. The same conclusion can be drawn in case of processing the two dimensional input matrix as shown in Tables 5,6,7,8,13,14,15, and 16. From these tables, it is clear that the maximum speed up ratio is achieved at image size (N=200) when n=20, then at image size (N=300) when n=25, and at image size (N=400) when n=30. This confirms our previous results presented in [7] on fast subimage detection based on neural networks and image decomposition. Using this technique, it was proved that the speed up ratio of neural networks becomes faster when the input image is divided into many subimages and each subimage is processed in the frequency domain separately using a single fast neural processor. Another point of interest should be noted. In CTDNNs, if the whole input data (N) is available, then there is a waiting time for each group of (n) input data so that conventional neural networks can release their output for the previous group of (n) data. In contrast, FTDNNs can process the total N data directly with zero waiting time. For example, if the total (N) input data is appeared at the input neurons, then:

1- CTDNNs can process only data of size (n) as the number of input neurons = (n).

2- The first group of (n) data is processed by CTDNNs.

3- The second group of (n) data must wait for a waiting time = τ, where τ is the response time consumed by CTDNNs for treating each group of (n) input data.

4- The third group of (n) data must wait for a waiting time = 2τ corresponding to the total waiting time required by CTDNNs for treating the previous two groups.

5- The fourth (n) data must wait for a waiting time = 3τ.

6- The last group of (n) data must wait for a waiting time = $(N-n)\tau$.

As a result, the wasted waiting time in the case of CTDNNs is $(N-n)\tau$. In the case of FTDNNs, there is no waiting time as the whole input data (Z) of length (N) will be processed directly and the time consumed is the only time required by FNNs themselves to produce their output.

4. Conclusion

A new approach to increase the speed of time delay neural networks for detecting a word in a speech has been presented. This has been done by designing a novel model of time delay neural networks called FTDNNs. Theoretical computations have shown that FTDNNs require fewer computation steps than conventional ones. This has been achieved by applying cross correlation in the frequency domain between the input data and the input weights of time delay neural networks. Simulation results have confirmed this proof by using MATLAB. This model can be successfully applied to any application that uses time delay neural networks.

References

[1] H. M. El-Bakry, "New Faster Normalized Neural Networks for Sub-Matrix Detection using Cross Correlation in the Frequency Domain and Matrix Decomposition," Applied Soft Computing journal, vol. 8, issue 2, March 2008, pp. 1131-1149.

[2] H. M. El-Bakry, and N. Mastorakis "New Fast Normalized Neural Networks for Pattern Detection," Image and Vision Computing Journal, vol. 25, issue 11, 2007, pp. 1767-1784.

[3] H. M. El-Bakry and M. Hamada, "A New Implementation for High Speed Neural Networks in Frequency Space," Lecture Notes in Computer Science, Springer, KES 2008, Part I, LNAI 5177, pp. 33-40.

[4] H. M. El-Bakry, and Q. Zhao, "A Fast Neural Algorithm for Serial Code Detection in a Stream of Sequential Data," International Journal of Information Technology, vol.2, no.1, pp. 71-90, 2005.

[5] H. M. El-Bakry, and H. Stoyan**,** "FNNs for Code Detection in Sequential Data Using Neural Networks for Communication Applications, " Proc. of the First International Conference on Cybernetics and Information Technologies, Systems and Applications: CITSA 2004, 21-25 July, 2004. Orlando, Florida, USA, Vol. IV, pp. 150-153.

[6] A. Hirose, "Complex-Valued Neural Networks Theories and Applications", Series on innovative Intellegence, vol.5. Nov. 2003.

[7] H. M. El-Bakry, "Face detection using fast neural networks and image decomposition," Neurocomputing Journal, vol. 48, 2002, pp. 1039-1046.

[8] H. M. El-Bakry, "Human Iris Detection Using Fast Cooperative Modular Neural Nets and Image Decomposition,"

Machine Graphics & Vision Journal (MG&V), vol. 11, no. 4, 2002, pp. 498-512.
[9] H. M. El-Bakry, "Automatic Human Face Recognition Using Modular Neural Networks," Machine Graphics & Vision Journal (MG&V), vol. 10, no. 1, 2001, pp. 47-73.
[10] S. Jankowski, A. Lozowski, M. Zurada, " Complex-valued Multistate Neural Associative Memory," IEEE Trans. on Neural Networks, vol.7, 1996, pp.1491-1496.
[11] H. M. El-Bakry, "New High Speed Normalized Neural Networks for Fast Pattern Discovery on Web Pages," the International Journal of Computer Science and Network Security, vol.6, No. 2A, Feb. 2006, pp.142-152.
[12] 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.
[13] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput. 19, 297–301 (1965).
[14] R. Klette, and Zamperon, "Handbook of image processing operators, " John Wiley & Sonsltd, 1996.
[15] H. M. El-Bakry, and N. Mastorakis, "An Effective Routing Algorithm for Real-Time Applications," International Journal of Communication, vol. 1, issue 4, 2007, pp. 156-168.
[16] 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.
[17] H. M. El-Bakry, and Q. Zhao, "Fast Pattern Detection Using Normalized Neural Networks and Cross Correlation in the Frequency Domain," EURASIP Journal on Applied Signal Processing, Special Issue on Advances in Intelligent Vision Systems: Methods and Applications—Part I, Vol. 2005, No. 13, 1 August 2005, pp. 2054-2060.
[18] H. M. El-Bakry, "New Fast Principal Component Analysis for Face Detection," Journal of Advanced Computational Intelligence and Intelligent Informatics, vol.11, no.2, 2007, pp. 195-201.
[19] Hazem M. El-Bakry, "Fast Painting with Different Colors Using Cross Correlation in the Frequency Domain," International Journal of Computer Science, vol.1, no.2, 2006, pp. 145-156.
[20] H. M. El-Bakry, and M. Hamada, " New Fast Decision Tree Classifier for Identifying Protein Coding Regions," Accepted for Publication in ICSIA 2008 Conf., China, Dec. 3-5, 2008.

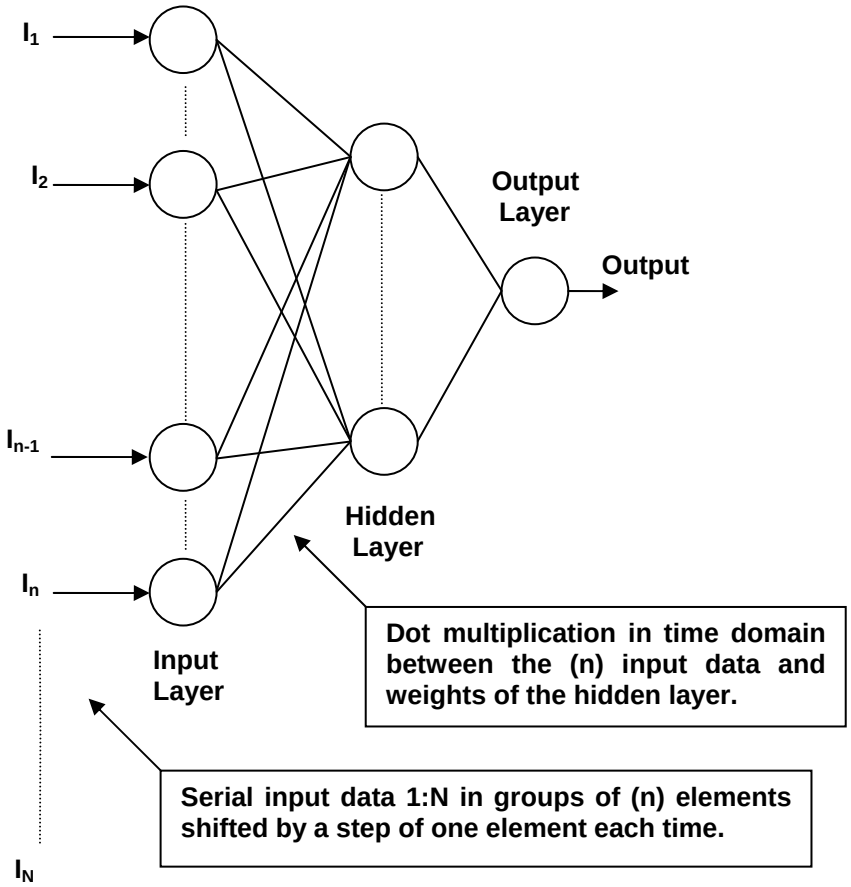

Fig.1. Classical time delay neural networks.

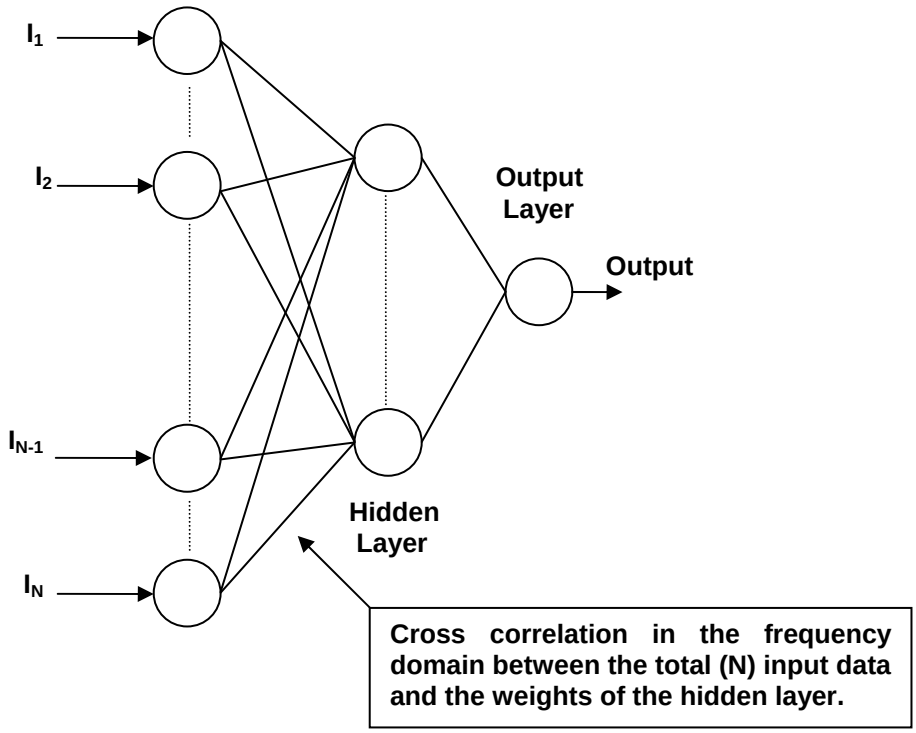

Fig.2. Fast time delay neural networks.

Table 1: The theoretical speed up ratio for time delay neural networks (1D-real values input matrix, n=400).

Length of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
10000	4.6027e+008	4.2926e+007	10.7226
40000	1.8985e+009	1.9614e+008	9.6793
90000	4.2955e+009	4.7344e+008	9.0729
160000	7.6513e+009	8.8219e+008	8.6731
250000	1.1966e+010	1.4275e+009	8.3823
360000	1.7239e+010	2.1134e+009	8.1571
490000	2.3471e+010	2.9430e+009	7.9752
640000	3.0662e+010	3.9192e+009	7.8237

Table 2: The theoretical speed up ratio for time delay neural networks (1D-real values input matrix, n=625).

Length of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
10000	7.0263e+008	4.2919e+007	16.3713
40000	2.9508e+009	1.9613e+008	15.0452
90000	6.6978e+009	4.7343e+008	14.1474
160000	1.1944e+010	8.8218e+008	13.5388
250000	1.8688e+010	1.4275e+009	13.0915
360000	2.6932e+010	2.1134e+009	12.7433
490000	3.6674e+010	2.9430e+009	12.4612
640000	4.7915e+010	3.9192e+009	12.2257

Table 3: The theoretical speed up ratio for time delay neural networks (1D-real values input matrix, n=900).

Length of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
10000	9.823 e+008	4.2911e+007	22.8933
40000	4.2206e+009	1.9612e+008	21.5200
90000	9.6176e+009	4.7343e+008	20.3149
160000	1.7173e+010	8.8217e+008	19.4671
250000	2.6888e+010	1.4275e+009	18.8356
360000	3.8761e+010	2.1134e+009	18.3409
490000	5.2794e+010	2.9430e+009	17.9385
640000	6.8985e+010	3.9192e+009	17.6018

Table 4: Practical speed up ratio for time delay neural networks (1D-real values input matrix).

Length of input matrix	Speed up ratio (n=400)	Speed up ratio (n=625)	Speed up ratio (n=900)
10000	17.88	25.94	35.21
40000	17.19	25.11	34.43
90000	16.65	24.56	33.59
160000	16.14	24.14	33.05
250000	15.89	23.76	32.60
360000	15.58	23.23	32.27
490000	15.28	22.87	31.99
640000	14.08	22.54	31.78

Table 5: The theoretical speed up ratio for time delay neural networks (2D-real values input matrix, n=20).

Size of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	3.1453e+008	4.2916e+007	7.3291
200x200	1.5706e+009	1.9610e+008	8.0091
300x300	3.7854e+009	4.7335e+008	7.9970
400x400	6.9590e+009	8.8203e+008	7.8898
500x500	1.1091e+010	1.4273e+009	7.7711
600x600	1.6183e+010	2.1130e+009	7.6585
700x700	2.2233e+010	2.9426e+009	7.5556
800x800	2.9242e+010	3.9186e+009	7.4623

Table 6: The theoretical speed up ratio for time delay neural networks (2D-real values input matrix, n=25).

Size of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	4.3285e+008	4.2909e+007	10.0877
200x200	2.3213e+009	1.9609e+008	11.8380
300x300	5.7086e+009	4.7334e+008	12.0602
400x400	1.0595e+010	8.8202e+008	12.0119
500x500	1.6980e+010	1.4273e+009	11.8966
600x600	2.4863e+010	2.1130e+009	11.7667
700x700	3.4246e+010	2.9425e+009	11.6381
800x800	4.5127e+010	3.9185e+009	11.5163

Table 7: The theoretical speed up ratio for time delay neural networks (2D-real values input matrix, n=30).

Size of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	5.4413e+008	4.2901e+007	12.6834
200x200	3.1563e+009	1.9608e+008	16.0966
300x300	7.9272e+009	4.7334e+008	16.7476
400x400	1.4857e+010	8.8201e+008	16.8444
500x500	2.3946e+010	1.4273e+009	16.7773
600x600	3.5193e+010	2.1130e+009	16.6552
700x700	4.8599e+010	2.9425e+009	16.5160
800x800	6.4164e+010	3.9185e+009	16.3745

Table 8: Practical speed up ratio for time delay neural networks (2D-real values input matrix).

Size of input matrix	Speed up ratio (n=20)	Speed up ratio (n=25)	Speed up ratio (n=30)
100x100	17.19	22.32	31.74
200x200	17.61	22.89	32.55
300x300	16.54	23.66	33.71
400x400	15.98	22.95	34.53
500x500	15.62	22.49	33.32
600x600	15.16	22.07	32.58
700x700	14.87	21.83	32.16
800x800	14.64	21.61	31.77

Table 9: The theoretical speed up ratio for time delay neural networks (1D-complex values input matrix, n=400).

Length of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	9.2111e+008	4.2926e+007	21.4586
200x200	3.7993e+009	1.9614e+008	19.3706
300x300	8.5963e+009	4.7344e+008	18.1571
400x400	1.5312e+010	8.8219e+008	17.3570
500x500	2.3947e+010	1.4275e+009	16.7750
600x600	3.4500e+010	2.1134e+009	16.3245
700x700	4.6972e+010	2.9430e+009	15.9604
800x800	3.9192e+009	6.1363e+010	15.6571

Table 10: The theoretical speed up ratio for time delay neural networks (1D-complex values input matrix, n=625).

Length of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	1.4058e+009	4.2919e+007	32.7558
200x200	5.9040e+009	1.9613e+008	30.1025
300x300	1.3401e+010	4.7343e+008	28.3061
400x400	2.3897e+010	8.8218e+008	27.0883
500x500	3.7391e+010	1.4275e+009	26.1934
600x600	5.3885e+010	2.1134e+009	25.4969
700x700	7.3377e+010	2.9430e+009	24.9324
800x800	9.5868e+010	3.9192e+009	24.4612

Table 11: The theoretical speed up ratio for time delay neural networks (1D-complex values input matrix, n=900).

Length of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	1.9653e+009	4.2911e+007	45.7993
200x200	8.4435e+009	1.9612e+008	43.0519
300x300	1.9240e+010	4.7343e+008	40.6410
400x400	3.4356e+010	8.8217e+008	38.9450
500x500	5.3791e+010	1.4275e+009	37.6817
600x600	7.7544e+010	2.1134e+009	36.6920
700x700	1.0562e+011	2.9430e+009	35.8870
800x800	1.3801e+011	3.9192e+009	35.2134

Table 12: Practical speed up ratio for time delay neural networks (1D-complex values input matrix).

Length of input matrix	Speed up ratio (n=400)	Speed up ratio (n=625)	Speed up ratio (n=900)
10000	37.90	53.58	70.71
40000	36.82	52.89	69.43
90000	36.34	52.47	68.69
160000	35.94	51.88	68.05
250000	35.69	51.36	67.56
360000	35.28	51.02	67.15
490000	34.97	50.78	66.86
640000	34.67	50.56	66.58

Table 13: The theoretical speed up ratio for time delay neural networks (2D-complex values input matrix, n=20).

Size of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	6.2946e+008	4.2916e+007	14.6674
200x200	3.1431e+009	1.9610e+008	16.0281
300x300	7.5755e+009	4.7335e+008	16.0040
400x400	1.3927e+010	8.8203e+008	15.7894
500x500	2.2197e+010	1.4273e+009	15.5519
600x600	3.2386e+010	2.1130e+009	15.3266
700x700	4.4493e+010	2.9426e+009	15.1206
800x800	5.8520e+010	3.9186e+009	14.9340

Table 14: The theoretical speed up ratio for time delay neural networks (2D-complex values input matrix, n=25).

Size of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	8.6605e+008	4.2909e+007	20.1836
200x200	4.6445e+009	1.9609e+008	23.6856
300x300	1.1422e+010	4.7334e+008	24.1301
400x400	2.1198e+010	8.8202e+008	24.0333
500x500	3.3973e+010	1.4273e+009	23.8028
600x600	4.9746e+010	2.1130e+009	23.5427
700x700	6.8519e+010	2.9425e+009	23.2856
800x800	9.0290e+010	3.9185e+009	23.0418

Table 15: The theoretical speed up ratio for time delay neural networks (2D-complex values input matrix, n=30).

Size of input matrix	Number of computation steps required for classical complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	1.0886e+009	4.2901e+007	25.3738
200x200	6.3143e+009	1.9608e+008	32.2021
300x300	1.5859e+010	4.7334e+008	33.5045
400x400	2.9722e+010	8.8201e+008	33.6981
500x500	4.7904e+010	1.4273e+009	33.5640
600x600	7.0405e+010	2.1130e+009	33.3197
700x700	9.7225e+010	2.9425e+009	33.0412
800x800	1.2836e+011	3.9185e+009	32.7581

Table 16: Practical speed up ratio for time delay neural networks (2D-complex values input matrix).

Size of input matrix	Speed up ratio (n=20)	Speed up ratio (n=25)	Speed up ratio (n=30)
100x100	38.33	46.99	62.88
200x200	39.17	47.79	63.77
300x300	38.44	48.86	64.83
400x400	37.92	47.23	65.99
500x500	37.32	46.89	64.89
600x600	36.96	46.48	64.01
700x700	36.67	46.08	63.31
800x800	36.38	45.78	62.64