J Comput Virol (2010) 6:115–122
DOI 10.1007/s11416-009-0120-x

CORRESPONDENCE

# Fast virus detection by using high speed time delay neural networks

**Hazem M. El-Bakry**

Received: 17 January 2007 / Revised: 10 July 2007 / Accepted: 26 March 2009 / Published online: 15 April 2009


**Abstract** This paper presents an intelligent approach to detect unknown malicious codes by using new high speed time delay neural networks. 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.

## 1 Introduction

Fast virus detection is very important for computer and network security. Since the appearance of the first computer virus in 1986, many new viruses have been created every year. The number of these viruses is growing rapidly and this threatens to outpace the manual efforts of anti-virus experts in designing solutions for detecting these viruses and removing them from the computer system [1]. There are a wide variety of protection mechanisms to over come virus attack like firewalls and antivirus tools. As the number and intensity of malware attacks is on the rise, computer security companies, researchers and users do their best to find new solutions thwart and defend against such assaults [25–27].

New technology exists for detecting known viruses. Programs such as Norton and MacAfee's Antivirus are ubiquitous. These programs search for the executable code of known patterns. One drawback of this method is that a copy of a malicious program must be known before extracting the pattern necessary for its detection [2].

Some researchers tried to overcome this intrusion by using intelligent algorithms to detect virus codes. In an early attempt, the authors in [3] conducted an analysis of several programs evidently by hand and identified tell-tale signs, which they subsequently used as a filter to protect new programs. IBM researchers have applied neural networks for virus detection and incorporated a similar approach for detecting boot-sector viruses into IBM's Anti-Virus software [4]. Others used data mining techniques such as naïve bayes classifiers to detect virus codes [5]. However, the work in literature has shown that the ability of neural networks to generalize is far better than that of the bayes classifier [6–10]. This is because of the powerful learning capability of neural networks rather than bayes classifier.

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 paper is to improve the speed of time delay neural networks for fast virus detection. The purpose is to perform the testing process in the frequency domain instead of the time domain. This approach was successfully applied for sub-image detection using fast neural networks (FNNs) as proposed in [11–13]. Furthermore, it was used for fast face detection [14,15], and fast iris detection [16]. Another idea to further increase the speed of FNNs through image decomposition was suggested in [14].

FNNs for detecting a certain code in one dimensional serial stream of sequential data were described in [17,18]. 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

H. M. El-Bakry (✉)
Faculty of Computer Science and Information Systems,
Mansoura University, Mansoura, Egypt
e-mail: helbakry20@yahoo.com

a significant reduction in the number of computation steps required for certain data detection [11–20]. Here, we make use of our theory on FNNs implemented in the frequency domain to increase the speed of time delay neural networks for fast virus detection.

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 Sect. 2, our theory on FNNs for detecting certain data in one dimensional matrix is described. Experimental results for FTDNNs are presented in Sect. 3.

## 2 Theory of FNNs based on cross correlation in the frequency domain

Finding a certain virus in the input one dimensional matrix is a searching problem. Each position in the input matrix is tested for the presence or absence of the required virus. 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 virus 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* 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 [11–18,21]. Assume that the size of the virus code in $1 \times n$. In virus detection phase, a sub matrix I of size $1 \times n$ (sliding window) is extracted from the tested matrix, which has a size $1 \times N$. Such sub matrix, which may be a virus code, is 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 $1 \times n$ and can be represented as $1 \times n$ matrix. The output of hidden neurons h(i) can be calculated as follows:

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

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:

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

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

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

Therefore, Eq. 2 may be written as follows [11]:

$$h_i = g\left(W_i \otimes Z + b_i\right) \tag{4}$$

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 [11]:

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

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:

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

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 $1 \times N$ elements, the 1D-FFT requires a number equal to $N \log_2 N$ of complex computation steps [22]. 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 [22]. 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 $1 \times N$ matrix is:

$$\rho = 6\left((N/2)\log_2 N\right) + 2\left(N \log_2 N\right) \tag{7}$$

which may be simplified to:

$$\rho = 5N \log_2 N \tag{8}$$

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:

$$\sigma = (2q+1)\left(5N \log_2 N\right) + 6qN + q(N-n) + qN + N \tag{9}$$

which can be reformulated as:

$$\sigma = (2q+1)\left(5N \log_2 N\right) + q(8N-n) + N \tag{10}$$

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

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

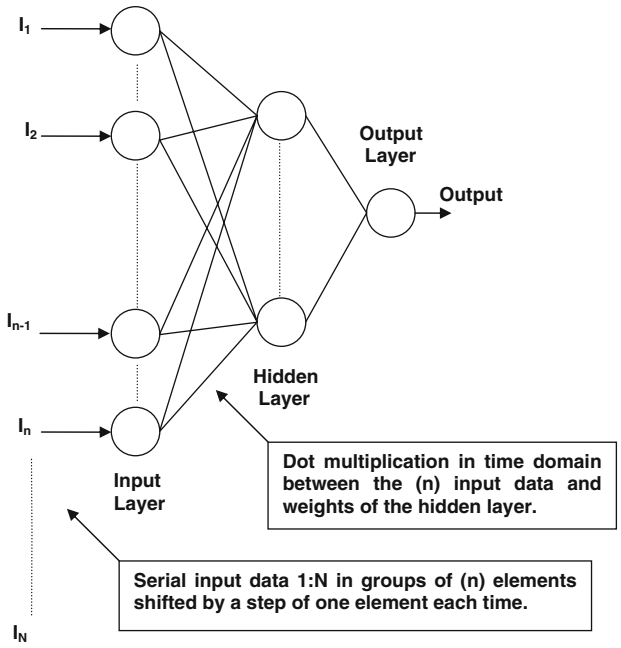


**Fig. 1** Classical time delay neural networks

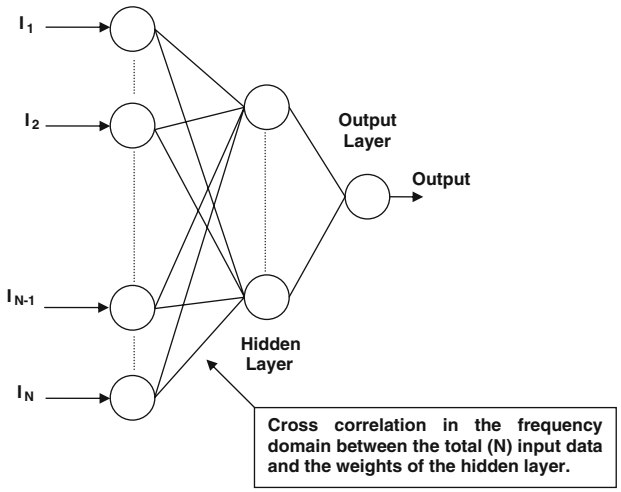


**Fig. 2** Fast time delay neural networks

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

## 3 Experimental results of time delay neural networks for fast virus detection

First neural networks are trained to classify virus from non virus examples and this is done in time domain. In the virus detection phase, each sub-matrix $(1 \times n)$ in the incoming data (probe matrix $1 \times N$) is tested for the presence or absence of the virus. At each position in the incoming input matrix, each

sub-matrix is multiplied by a window of weights which has the same size as the sub-matrix. This multiplication is done in the time domain. 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 a virus and vice versa. Thus, we may conclude that this searching problem is cross correlation in the time domain between the incoming data and the input weights of 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, the proposed new approach is to collect all the incoming data together in a long vector (for example 100 × n). Then the input data is tested by time delay neural networks as a single pattern with length L (L = 100 × n). Such a test is performed in the frequency domain as described in Sect. 2. The virus inserted in the incoming data may have real or complex values in a form of one or two dimensional array. 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 [23,24]. 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 (real /complex) will be presented. Also, the speed up ratio in the case of a one and two dimensional incoming 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:

## 3.1 In case of real inputs

### *3.1.1 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. The multiplication and addition operations are repeated (N−n+1) for all possible sub matrices in the incoming input matrix. In addition, all of these procedures are repeated at each neuron in the hidden layer. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q(2n-1)\,(N-n+1) \tag{12}$$

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

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

The theoretical speed up ratio for searching short successive (n) data in a long input vector (L) using complex-valued time delay neural networks is shown in Figs. 3, 4, and 5. 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 Fig. 6.

### *3.1.2 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. The multiplication and addition operations are repeated $(N-n+1)^2$ for all possible sub matrices in the incoming input matrix. In addition, all of these procedures are repeated at each neuron in the hidden layer. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

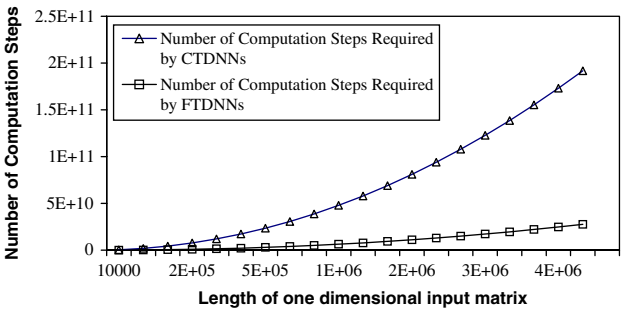


**Fig. 3** A comparison between the number of computation steps required by FTDNNs and CTDNNs in case of real-valued one dimensional input matrix and complex-valued weight matrix (n=400)

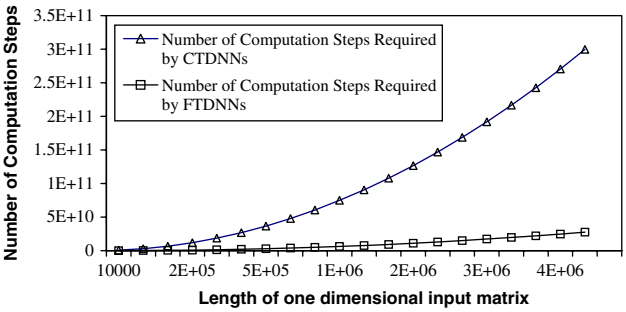


**Fig. 4** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of real-valued one dimensional input matrix and complex-valued weight matrix (n=625)

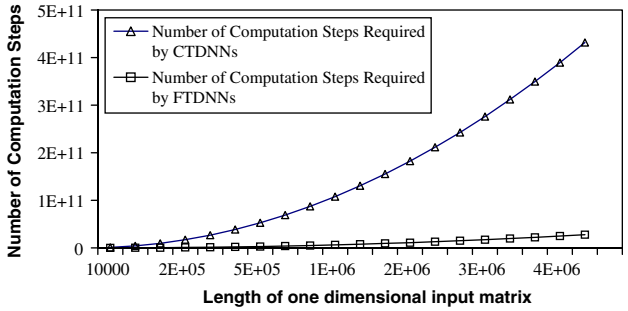


**Fig. 5** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of real-valued one dimensional input matrix and complex-valued weight matrix (n=900)

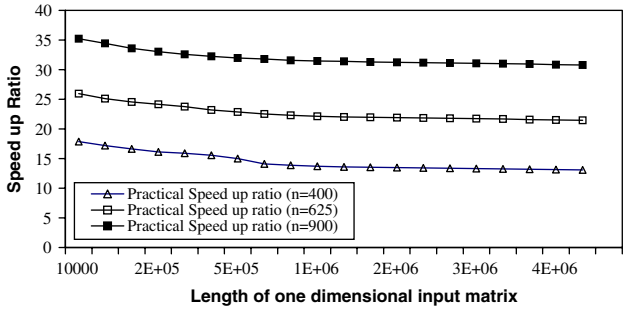


**Fig. 6** Practical speed up ratio for time delay neural networks in case of one dimensional real-valued input matrix and complex-valued weights

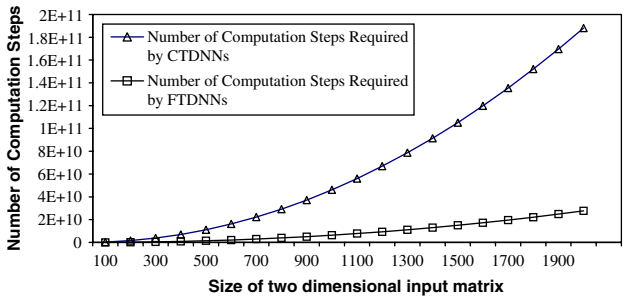


**Fig. 7** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of real-valued two dimensional input matrix and complex-valued weight matrix (n=20)

$$\theta = 2q\left(2n^2 - 1\right)(N - n + 1)^2 \tag{14}$$

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

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

The theoretical speed up ratio for detecting (n × n) real valued submatrix in a large real valued matrix (N × N) using complex-valued time delay neural networks is shown in Figs. 7, 8, 9. Also, the practical speed up ratio for manipulating matrices of different sizes (N × N) and different sized weight matrices (n) using a 2.7 GHz processor and MATLAB is shown in Fig. 10.

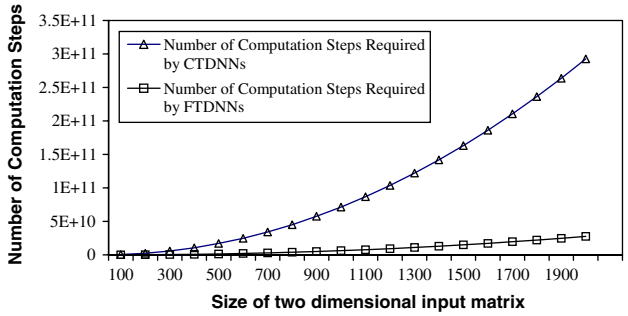


**Fig. 8** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of real-valued two dimensional input matrix and complex-valued weight matrix (n=25)

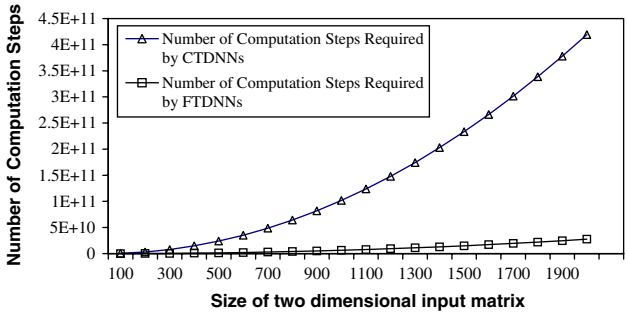


**Fig. 9** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of real-valued two dimensional input matrix and complex-valued weight matrix (n=30)

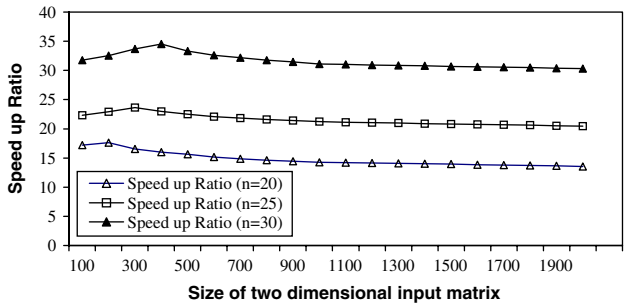


**Fig. 10** Practical speed up ratio for time delay time neural networks in case of two dimensional real-valued input matrix and complex-valued weights

## 3.2 In case of complex inputs

### *3.2.1 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,

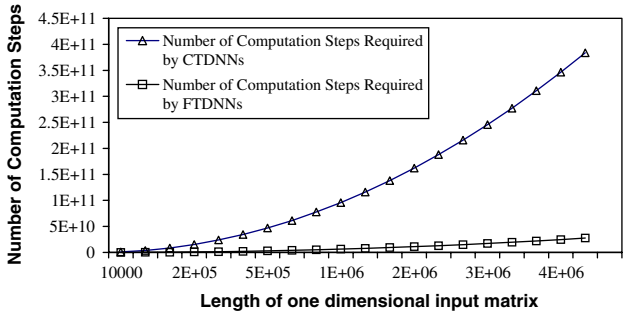


**Fig. 11** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of complex-valued one dimensional input matrix and complex-valued weight matrix (n=400)

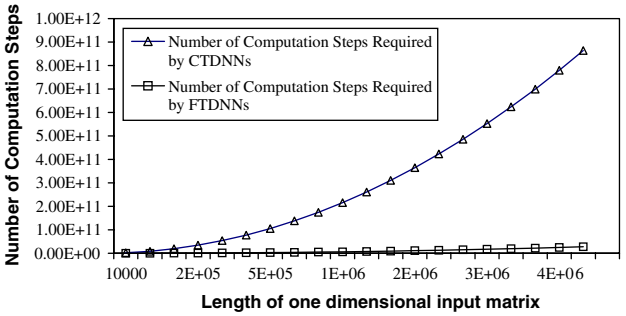


**Fig. 13** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of complex-valued one dimensional input matrix and complex-valued weight matrix (n=900)

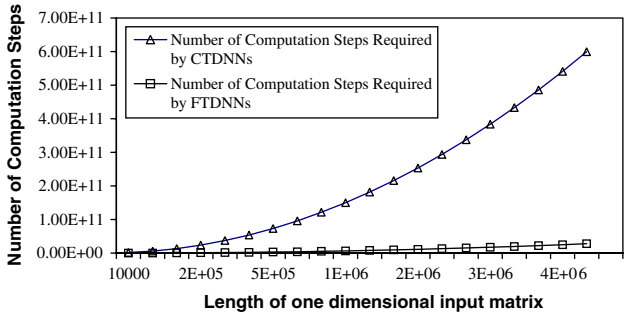


**Fig. 12** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of complex-valued one dimensional input matrix and complex-valued weight matrix (n=625)

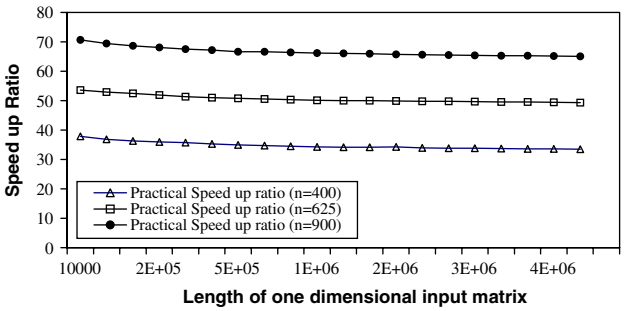


**Fig. 14** Practical speed up ratio for time delay neural networks in case of one dimensional complex-valued input matrix and complex-valued weights

the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q\,(4n-1)\,(N-n+1) \tag{16}$$

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

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

The theoretical speed up ratio for searching short complex successive (n) data in a long complex-valued input vector (L) using complex-valued time delay neural networks is shown in Figs. 11, 12, and 13. Also, the practical speed up ratio for manipulating matrices of different sizes (L) and different sized weight matrices (n) using a 700 MHz processor and MATLAB is shown in Fig. 14.

### *3.2.2 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:

$$\theta = 2q\left(4n^2-1\right)(N-n+1)^2 \tag{18}$$

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

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

The theoretical speed up ratio for detecting (n×n) complex-valued submatrix in a large complex-valued matrix (N × N) using complex-valued neural networks is shown in Figs. 15, 16, and 17. Also, the practical speed up ratio for manipulating matrices of different sizes (N × N) and different sized weight matrices (n) using a 2.7 GHz processor and MATLAB is shown in Fig. 18.

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

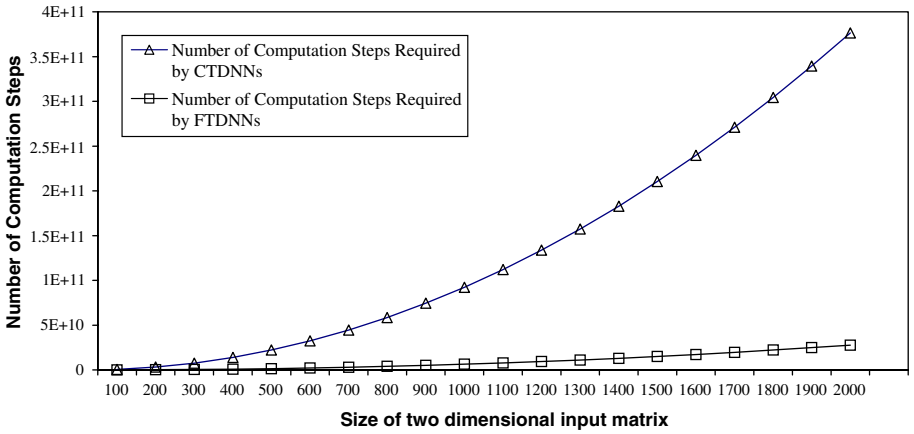


**Fig. 15** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of complex-valued two dimensional input matrix and complex-valued weight matrix (n=20)

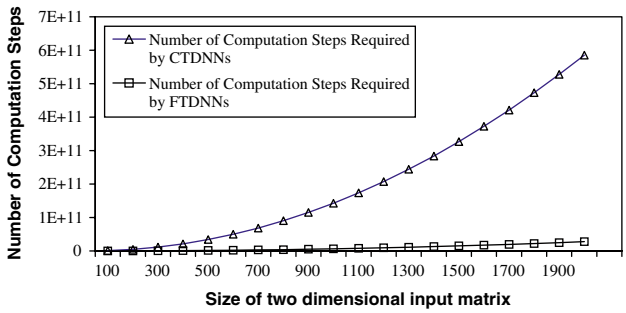


**Fig. 16** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of complex-valued two dimensional input matrix and complex-valued weight matrix (n=25)

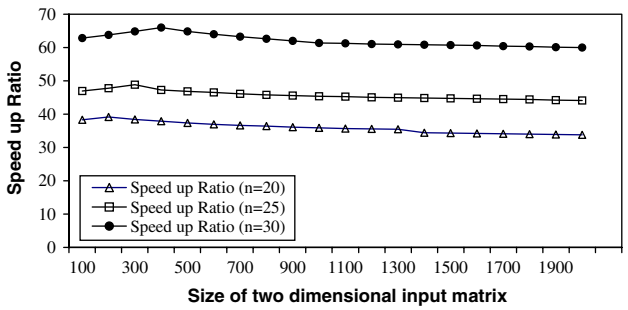


**Fig. 18** Practical speed up ratio for time delay neural networks in case of two dimensional complex-valued input matrix in and complex-valued weights

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.

Another interesting point is that the memory capacity is reduced when using FTDNN. This because the number of variables compared to CTDNN is reduced. The neural algorithm presented here can be inserted very easily in any Anti-Virus gateway software.

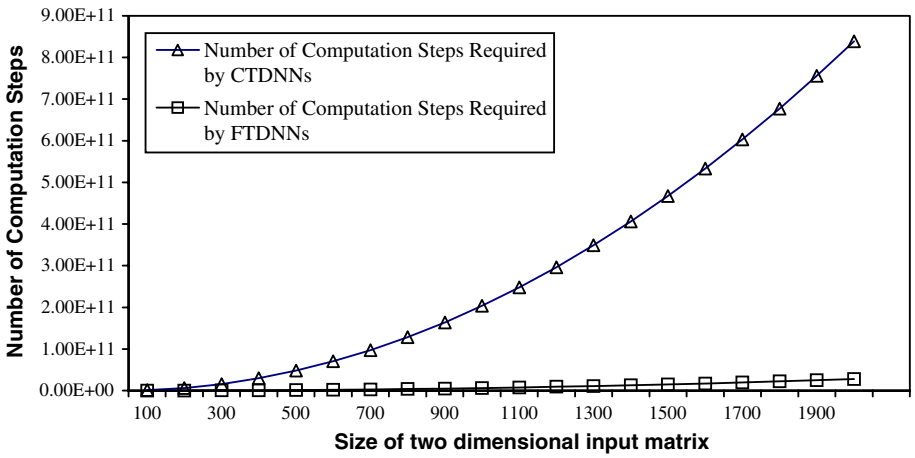


**Fig. 17** A comparison between the number of computation steps required by FTDNNs and CTDNNs in the case of complex-valued two dimensional input matrix and complex-valued weight matrix (n=30)

## 4 Conclusion

New FTDNNs for fast virus detection have been presented. 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. Furthermore, the memory complexity has been reduced when using the fast neural algorithm. In addition, this algorithm can be combined in any Anti-Virus gateway software. Moreover, successfully it can be applied to any application that uses time delay neural networks.

## References


1. Kephert, L., Arnold, W.: Automatic extraction of computer virus signatures. In: Proc. of the 4th Virus Bulletin International Conference, Abingdon, pp. 178–184 (1994)
2. Zhang, B., Yin, J., Hao, J.: Malicious Codes Detection based on Neural Network Ensembles. IJCNN2007, August 12–14, 2007, Orlando, Florida, USA (to appear)
3. Lo, R., Levitt, K., Olsson, R.: MCF: a malicious code filter. Comput. Secur. **14**(6), 541–566 (1995)
4. Tesauro, G., Kephart, J., Sorkin, G.: Neural networks for computer virus recognition. IEEE Expert. **11**(4), 5–6 (1996)
5. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 470–478. ACM Press, New York (2004)
6. Slezak, D., Wróblewski, J., Szczuka, M.: Constructing extensions of Bayesian classifiers with use of normalizing neural networks. LNCS, Found. Intell. Syst. **2871**, 408–416 (2003)
7. Guterman, H., Nehmadi, Y., Chistyakov, A., Soustiel, J.F., Feinsod, M.: A comparison of neural network and Bayes recognition approaches in the evaluation of the brainstem trigeminal evoked potentials in multiple sclerosis. Int. J. Bio-Med. Comput. **43**(3), 203–213 (1996)
8. Kjell, B.: Authorship determination using letter pair frequency features with neural network. J. Lit. Linguistic Comput. **9**, 119–124 (1996)
9. Shahin, M.A., Tollner, E.W., Mcclendon, R.W.: Artificial intelligence classifiers for sorting apples based on watercore. J. Agric. Eng. Res. **79**(3), 265–274 (2001)
10. http://web.njit.edu/~shi/Steganalysis/method.htm
11. El-Bakry, H.M.: A new neural design for faster pattern detection using cross correlation and matrix decomposition. Neural World J. (Accepted)
12. El-Bakry, H.M.: New faster normalized neural networks for submatrix detection using cross correlation in the frequency domain and matrix decomposition. Appl. Soft Comput. J. **8**(2), 1131–1149 (2008)
13. El-Bakry, H.M.: New fast principal component analysis for face detection. J. Adv. Comput. Intell. Intell. Inform. **11**(2), 195–201 (2007)
14. El-Bakry, H.M.: Face detection using fast neural networks and image decomposition. Neurocomput. J. **48**, 1039–1046 (2002)
15. El-Bakry, H.M.: Automatic human face recognition using modular neural networks. Mach. Graph. Vis. J. (MG&V) **10**(1), 47–73 (2001)
16. El-Bakry, H.M.: Human Iris detection using fast cooperative modular neural nets and image decomposition. Mach. Graph. Vis. J. (MG&V) **11**(4), 498–512 (2002)
17. El-Bakry, H.M., Zhao, Q.: A fast neural algorithm for serial code detection in a stream of sequential data. Int. J. Inform. Technol. **2**(1), 71–90 (2005)
18. El-Bakry, H.M., Stoyan, H.: FNNs for code detection in sequential data using neural networks for communication applications. In: 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 (2004)
19. El-Bakry, H.M., Zhao, Q.: Fast pattern detection using neural networks realized in frequency domain. In: Proc. of the International Conference on Pattern Recognition and Computer Vision, The Second World Enformatika Congress WEC'05, Istanbul, Turkey, 25–27 Feb, pp. 89–92 (2005)
20. El-Bakry, H.M., Zhao, Q.: Sub-image detection using fast neural processors and image decomposition. In: Proc. of the International Conference on Pattern Recognition and Computer Vision, The Second World Enformatika Congress WEC'05, Istanbul, Turkey, 25–27 Feb, pp. 85–88 (2005)
21. Klette, R., Zamperon, P.: Handbook of Image Processing Operators. Wiley, New York (1996)
22. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**, 297–301 (1965)
23. Hirose, A.: Complex-Valued Neural Networks Theories and Applications. Series on innovative Intellegence, vol. 5. World Scientific, Singapore (2003)
24. Jankowski, S., Lozowski, A., Zurada, M.: Complex-valued multistate neural associative memory. IEEE Trans. Neural Netw. **7**, 1491–1496 (1996)
25. Arnold, W., Tesauro, G.: Automatically Generated Win32 Heuristics Virus Detection, pp. 123–132. Virus Bulltien Conference (1995)
26. Coates, G., Leigh, D.: Virus Detection: the Brainy Way, pp. 211–224. Virus Bulltien Conference (1995)
27. Zwienenberg, R.: Heuristics Scanners: Artificial Intelligence?, pp. 203–210. Virus Bulltien Conference (1995)