A New Fast Neural Network Model

Hazem M. El-Bakry
Faculty of Computer Science & Information Systems, Mansoura University, EGYPT
helbakry20@yahoo.com

Nikos Mastorakis
Technical University of Sofia, BULGARIA

Abstract- In this paper, a new model for testing patterns with neural networks is presented. The idea is to accelerate the operation of testing patterns by using neural networks. This is done by applying cross correlation between the input patterns and the input weights of neural networks in the frequency domain rather than time domain. Furthermore, such model is very useful for understanding the internal relation between the tested patterns. In addition, the input patterns are collected in one vector and manipulated as a one pattern. Then, the important data (code) can be hidden and encrypted inside the whole input data and this is very useful for security applications. Moreover, it can be applied successfully for pattern/data analysis application. Simulation results confirm the theoretical considerations.

Keywords- Neural Networks, Cross Correlation, Frequency Domain.

1. Introduction

It has been shown that neural networks are efficient in many different applications. The fast response of these neural networks is very important especially in real-time applications. In this paper, a new model for accelerating the operation of neural networks in the test phase is presented. The idea is to collect all patterns in one vector and treat them at the same time as a one pattern. Then cross correlations are applied between such vector and the input weights of neural networks. These cross correlations are applied in the frequency domain rather than time domain. Performing cross correlation in the frequency domain is faster than time one while the resulted final output is the same in both cases.

The idea of applying cross correlation between the input data pattern and the input weights was applied successfully in many different applications [3-83]. Compared to traditional feedforward neural networks (TFNNs) shown in Fig. 1, 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 processing certain data [3-83]. Here, we make use of the theory of cross correlation implemented in the frequency domain to increase the speed of neural networks.

By using the presented model, important data can be hidden inside the whole input data. Then it can be processed without appearing to the neural network as a certain pattern. This is very useful for reasons of security. Furthermore, these important codes can be encrypted in the input data.

This paper is organized as follows. Realization of in the frequency domain is presented in section II. Simulation Results are introduced in section III. Finally conclusions are given.

2. Information Processing by Using Neural Networks Implemented in the Frequency Domain

Information processing by using neural networks is divided into two parts. First neural networks are trained to recognize the input patterns. In the test phase, each position in the incoming matrix is processed and tested for the required data (code) by using neural networks. At each position in the input one dimensional 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. Thus, we may conclude that the whole problem is a cross correlation between the incoming serial data and the weights of neurons in the hidden layer. 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 test phase [3-83]. Assume that the size of the input pattern is 1xn. In the test phase, a sub matrix I of size 1xn (sliding window) is extracted from the tested matrix, which has a size of 1xN. Such sub matrix, which contains the input pattern, 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 1xn and can be represented as 1xn matrix. The output of hidden neurons h(i) can be calculated as follows [3-83]:

\[ h_i = g \left( \sum_{k=1}^{n} W_i(k)l(k) + b_i \right) \]  

(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 [2]:

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

Equation 1 represents the output of each hidden neuron \((i)\). Equation 1 represents the output of each hidden neuron \((i)\). Equation 1 represents the output of each hidden neuron \((i)\). Equation 1 represents the output of each hidden neuron \((i)\).

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

Eq.1 represents a cross correlation operation. Given any two functions \( f \) and \( d \), their cross correlation can be obtained by [1]:

\[
d(x) \otimes f(x) = \left( \sum_{n=-\infty}^{\infty} f(x+n) d(n) \right)
\]

Therefore, Eq. 2 may be written as follows [3-83]:

\[
h_i = g \left( W_i \otimes Z + b_i \right)
\]

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 [3-83]:

\[
W_i \otimes Z = F^{-1} \left( F(Z) \cdot F^*(W_i) \right)
\]

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

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

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 [2]. 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 fast neural model (FNM) shown in Fig. 2 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 [2]. 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:

\[
\rho = 6((N/2) \log_2 N) + 2(N \log_2 N)
\]

which may be simplified to:

\[
\rho = 5N \log_2 N
\]

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 FNM.

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 \((e^{j2\pi L/n})\), 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 FNM becomes:

\[
\sigma = (2q+1)(5N \log_2 N)+6qN+q(N-n)+qN+N
\]

which can be reformulated as:

\[
\sigma = (2q+1)(5N \log_2 N)+q(8N-n)+N
\]

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 TFNNs to process \((n)\) input data. The theoretical speed up factor \( \eta \) can be evaluated as follows:

\[
\eta = \frac{q(2n-1)(N-n+1)}{(2q+1)(5N \log_2 N)+q(8N-n)+N}
\]
3. Simulation Results

TFNNs 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 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 before.

Eq. 11 is also true for recurrent neural networks. The theoretical speed up ratio for processing short successive (n) data in a long input vector (L) using recurrent neural networks is listed 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. An interesting point is that the memory capacity is reduced when using FNMs. This is because the number of variables is reduced compared to TFNNs. Another point of interest should be noted. In TFNNs, 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, FNMs 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- TFNNs can process only data of size (n) as the number of input neurons \(= n\).
2- The first group of (n) data is processed by TFNNs.
3- The second group of (n) data must wait for a waiting time \(= \tau\), where \(\tau\) is the response time consumed by TFNNs for treating each group of (n) input data.
4- The third group of (n) data must wait for a waiting time \(= 2\tau\) corresponding to the total waiting time required by TFNNs for treating the previous two groups.
5- The fourth (n) data must wait for a waiting time \(= 3\tau\).
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 TFNNs is \((N-n)\tau\). In the case of FNMs, 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 FNMs itself to produce their output.

<table>
<thead>
<tr>
<th>Length of input data</th>
<th>Number of computation steps required for TFNNs</th>
<th>Number of computation steps required for FNMs</th>
<th>Speed up ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>2.3014e+008</td>
<td>4.2926e+007</td>
<td>5.3613</td>
</tr>
<tr>
<td>40000</td>
<td>0.9493e+009</td>
<td>1.9614e+008</td>
<td>4.8397</td>
</tr>
<tr>
<td>90000</td>
<td>2.1478e+009</td>
<td>4.7344e+008</td>
<td>4.5365</td>
</tr>
<tr>
<td>160000</td>
<td>3.8257e+009</td>
<td>8.8219e+008</td>
<td>4.3366</td>
</tr>
<tr>
<td>250000</td>
<td>5.9830e+009</td>
<td>1.4275e+009</td>
<td>4.1912</td>
</tr>
<tr>
<td>360000</td>
<td>8.6195e+009</td>
<td>2.1134e+009</td>
<td>4.0786</td>
</tr>
<tr>
<td>490000</td>
<td>1.1735e+010</td>
<td>2.9430e+009</td>
<td>3.9876</td>
</tr>
<tr>
<td>640000</td>
<td>1.5331e+010</td>
<td>3.9192e+009</td>
<td>3.9119</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length of input data</th>
<th>Number of computation steps required for TFNNs</th>
<th>Number of computation steps required for FNMs</th>
<th>Speed up ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>3.5132e+008</td>
<td>4.2919e+007</td>
<td>8.1857</td>
</tr>
<tr>
<td>40000</td>
<td>1.4754e+009</td>
<td>1.9613e+008</td>
<td>7.5226</td>
</tr>
<tr>
<td>90000</td>
<td>3.3489e+009</td>
<td>4.7343e+008</td>
<td>7.0737</td>
</tr>
<tr>
<td>160000</td>
<td>0.5972e+010</td>
<td>8.8218e+008</td>
<td>6.7694</td>
</tr>
<tr>
<td>250000</td>
<td>0.9344e+010</td>
<td>1.4275e+009</td>
<td>6.5458</td>
</tr>
<tr>
<td>360000</td>
<td>1.3466e+010</td>
<td>2.1134e+009</td>
<td>6.3717</td>
</tr>
<tr>
<td>490000</td>
<td>1.8337e+010</td>
<td>2.9430e+009</td>
<td>6.2306</td>
</tr>
<tr>
<td>640000</td>
<td>2.3958e+010</td>
<td>3.9192e+009</td>
<td>6.1129</td>
</tr>
</tbody>
</table>
Table 3. The theoretical speed up ratio (n = 900).

<table>
<thead>
<tr>
<th>Length of input data</th>
<th>Number of computation steps required for TRNNs</th>
<th>Number of computation steps required for FNM</th>
<th>Speed up ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>4.9115e+008</td>
<td>4.2911e+007</td>
<td>11.4467</td>
</tr>
<tr>
<td>40000</td>
<td>2.1103e+009</td>
<td>1.9612e+008</td>
<td>10.7600</td>
</tr>
<tr>
<td>90000</td>
<td>4.8088e+009</td>
<td>4.7343e+008</td>
<td>10.1575</td>
</tr>
<tr>
<td>160000</td>
<td>8.8587e+010</td>
<td>8.8217e+008</td>
<td>9.7336</td>
</tr>
<tr>
<td>250000</td>
<td>1.3444e+010</td>
<td>1.4275e+009</td>
<td>9.4178</td>
</tr>
<tr>
<td>360000</td>
<td>1.9381e+010</td>
<td>2.1134e+009</td>
<td>9.1705</td>
</tr>
<tr>
<td>490000</td>
<td>2.6397e+010</td>
<td>2.9430e+009</td>
<td>9.0693</td>
</tr>
<tr>
<td>640000</td>
<td>3.4493e+010</td>
<td>3.9192e+009</td>
<td>8.8009</td>
</tr>
</tbody>
</table>

Table 4. Practical speed up ratio.

<table>
<thead>
<tr>
<th>Length of input data</th>
<th>Speed up ratio (n=400)</th>
<th>Speed up ratio (n=625)</th>
<th>Speed up ratio (n=900)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>8.94</td>
<td>12.97</td>
<td>17.61</td>
</tr>
<tr>
<td>40000</td>
<td>8.60</td>
<td>12.56</td>
<td>17.22</td>
</tr>
<tr>
<td>90000</td>
<td>8.33</td>
<td>12.28</td>
<td>16.80</td>
</tr>
<tr>
<td>160000</td>
<td>8.07</td>
<td>12.07</td>
<td>16.53</td>
</tr>
<tr>
<td>250000</td>
<td>7.95</td>
<td>17.92</td>
<td>16.30</td>
</tr>
<tr>
<td>360000</td>
<td>7.79</td>
<td>11.62</td>
<td>16.14</td>
</tr>
<tr>
<td>490000</td>
<td>7.64</td>
<td>11.44</td>
<td>16.00</td>
</tr>
<tr>
<td>640000</td>
<td>7.04</td>
<td>11.27</td>
<td>15.89</td>
</tr>
</tbody>
</table>

4. CONCLUSION

A new model for testing patterns with neural networks has been presented. The operation of neural networks during the test phase has been accelerated. This has been done by applying cross correlation between the whole input patterns and the input weights of neural networks in the frequency domain rather than time domain. Simulation results have confirmed the theoretical considerations. The main advantage of this model is that it is very useful for understanding the internal relation between the tested patterns. In addition, important codes can be hidden inside the input pattern and this is very important for data encryption. Moreover, it can be applied for many other applications including pattern/data analysis.

REFERENCES


APPLICATIONS OF COMPUTER ENGINEERING (ACE '11), Spain, March 24-26, 2011, pp. 237-244.


Fig. 1. TFNNs.

Cross correlation in time domain between the \(n\) input data and weights of the hidden layer.

Serial input data 1:N in groups of \(n\) elements shifted by a step of one element each time.
Cross correlation in the frequency domain between the total (N) input data and the weights of the hidden layer.

Fig.2. FNM.