Prediction of Market Price by using Fast Time Delay Neural Networks

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

Nikos Mastorakakis
Technical University of Sofia, BULGARIA

Abstract: Prediction of market price is very important for strategic planning. In this paper, a new approach for fast market price prediction is presented. Such algorithm uses fast time delay neural networks (FTDNNs). The operation of these networks relies on performing cross correlation in the frequency domain between the input data and the input weights of neural networks. It is proved mathematically and practically that the number of computation steps required for the presented FTDNNs is less than that needed by conventional time delay neural networks (CTDNNs). Simulation results using MATLAB confirm the theoretical computations.

Keywords: Fast Neural Network; Cross Correlation, Frequency Domain, Market Price Prediction.

1. Introduction

Fast market prediction is very important for strategic planning in business companies. With the advent of the digital computer, market price prediction has since moved into the technological realm [33-64]. The most prominent technique involves the use of artificial neural networks (ANNs). 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 [5]. Furthermore, it was used for fast face detection [7,9], and fast iris detection [8]. Another idea to further increase the speed of FNNs through image decomposition was suggested in [7].

FNNs for detecting a certain code in one dimensional serial stream of sequential data were described in [10,11]. 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 [5-15]. Here, we make use of the theory of FNNs implemented in the frequency domain to increase the speed of time delay neural networks for market price prediction. 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. Theory of neural networks for pattern detection are described in section 2. Section 3 presents FTDNNs for detecting certain packet in serial data. Experimental results for fast market price prediction by using FTDNNs are given in section 4.

2. Theory of ANNs for Prediction

Artificial neural network (ANN) is a mathematical model, which can be set one or more layered and occurred from many artificial neural cells. The wide usage of the ANN may be due to the three basic properties:

1. The ability of the ANN as a parallel processing of the problems, for which if any of the neurons violate the constraints would not affect the overall output of the problem.

2. The ability of the ANN to extrapolate from historical data to generate forecasts.

3. The successful application of the ANN to solve non-linear problems. The history and theory of the ANN have been described in a large number of published literatures and will not be covered in this paper except for a very brief overview of how neural networks operate.

The ANN computation can be divided into two phases: learning phase and testing phase. The learning phase forms an iterative updating of the synoptic weights based upon the error back propagation algorithm. Back propagation algorithm is generalized of least mean square learning rule, which is an approximation of steepest descent technique. To find the best approximation, multi-layer feed forward neural network architecture with back propagation learning rule is used. A schematic diagram of typical multi-layer feed-forward conventional time delay neural network architecture is shown in Fig. 1. The network has five inputs and one output neuron in its linear output layer. The number of neurons in the hidden layer is varied to give the network enough power to solve the problem. Each neuron
computes a weighted sum of the individual inputs \((I_1, I_2, \ldots, I_j)\) it receives and adding it with a bias \((b)\) to form the net input \((s)\). The bias is included in the neurons to allow the activation function to be offset from zero.

\[
\text{sum} = w_{1,1}I_1 + w_{1,2}I_2 + \ldots + w_{1,j}I_j + b
\]  
(1)

Where, \(w_{ji}\) is the connection weight between neuron \(j\) and neuron \(i\). The net input \((\text{sum})\) is then passed to the subsequent layer through a non-linear sigmoid function to form its own output \((y_j)\):

\[
y_j = \frac{1}{1 + e^{-\text{sum}}}
\]  
(2)

Afterward, the output \(y_j\) was compared with the target output \(\hat{y}_j\) using an error function of the form:

\[
\delta_j = (t_j - y_j)y_j(1-y_j)
\]  
(3)

For the neuron in the hidden layer, the error term is given by the following equation:

\[
\delta_j = y_j(1-y_j) \sum_k \delta_k w_{kj}
\]  
(4)

where \(\delta_k\) is the error term of the output layer, and \(w_{kj}\) is the weight between the hidden layer and output layer. The error was then propagated backward from the output layer to the input layer to update the weight of each connection at iteration \((t+1)\) as follows:

\[
w_{ji}(t+1) = w_{ji}(t) + \eta \delta_j y_j + \alpha (w_{ji}(t) - w_{ji}(t-1))
\]  
(5)

Choosing a small learning rate \(\eta\) leads to slow rate of convergence, and too large \(\eta\) leads to oscillation. The term \(\alpha\) is called momentum factor and determines the effect of past weight changes on the current direction of movement. Both of these constant terms are specified at the start of the training cycle and determine the speed and stability of the network. The process was repeated for each input pattern until the error was reduced to a threshold value.

### 3. Market Price Prediction by using FTDNNs

Computing the resulted output, for a certain pattern of information; in the incoming serial data, is a prediction problem. First neural networks are trained to predict the estimated variable and this is done in time domain. In pattern detection phase, each position in the incoming matrix is processed to predict the estimated variable 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 detection process [5-25]. Assume that the size of the attack code is \(1xn\). In attack detection 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 may be an attack 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 \(1xn\) and can be represented as \(1xn\) matrix. The output of hidden neurons \(h_i\) can be calculated as follows [5]:

\[
h_i = g \left( \sum_{k=1}^{n/2} W_i(k) I(k) + b_i \right)
\]  
(6)

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 [5]:

\[
h_i(u) = g \left( \sum_{k=-n/2}^{n/2} W_i(k) Z(u+k) + b_i \right)
\]  
(7)

Eq.6 represents a cross correlation operation. Given any two functions \(f\) and \(d\), their cross correlation can be obtained by [4]:

\[
d(x) \otimes f(x) = \left( \sum_{n=-\infty}^{\infty} f(x+n)d(n) \right)
\]  
(8)

Therefore, Eq. 7 may be written as follows [5]:

\[
h_i = g \left( W_i \otimes Z + b_i \right)
\]  
(9)

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) ∈ [N-n+1].

Now, the above cross correlation can be expressed in terms of one dimensional Fast Fourier Transform as follows [5]:

\[ W_i \otimes Z = F^{-1} \left[ F(Z) \cdot F^*(W_i) \right] \]  \hspace{1cm} (10)

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:

\[ O(u) = g \left( \sum_{i=1}^{q} W_o(i) h_i(u) + b_o \right) \]  \hspace{1cm} (11)

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.

4. Complexity Analysis of FTDNNs for Market Price Prediction

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 Nlog_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)Nlog_2 N.

3- The number of computation steps required by FTDNNs 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 Nlog_2 N complex additions [3]. 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:

\[ \rho = 6 \left( \frac{N}{2} \right) \log_2 N + 2N \log_2 N \]  \hspace{1cm} (12)

which may be simplified to:

\[ \rho = 5N \log_2 N \]  \hspace{1cm} (13)

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 FTDNNs.

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 \( e^{jk \pi / (2q \log_2 N)} \), 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 FTDNNs becomes:

\[ \sigma = (2q+1)(5N \log_2 N) + 6qN + q(N-n) + qN + N \]  \hspace{1cm} (14)

which can be reformulated as:

\[ \sigma = (2q+1)(5N \log_2 N) + q(8N-n) + N \]  \hspace{1cm} (15)

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 attack detection or processing (n) input data. The theoretical speed up factor \( \eta \) can be evaluated as follows:

\[ \eta = \frac{q(2n-1)(N-n+1)}{(2q+1)(5N \log_2 N) + q(8N-n) + N} \]  \hspace{1cm} (16)

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

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 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.

The theoretical speed up ratio for searching short successive (n) code in a long input vector (L) using time delay 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 FTDNN. This is because the number of variables is reduced compared with CTDNN.

5. Conclusion
A fast neural algorithm for prediction of market price has been presented. Such strategy has been realized by using our design for 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 weights of neural networks. Simulation results have confirmed this proof by using MATLAB.

References


Fig. 1. CTDNNs.

- 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.

Fig. 2. FTDNNs.

- Cross correlation in the frequency domain between the total (N) input data and the weights of the hidden layer.
Table 1: The theoretical speed up ratio for market price prediction (n=400).

<table>
<thead>
<tr>
<th>Length of serial data</th>
<th>Number of computation steps required for CTDNNs</th>
<th>Number of computation steps required for FTDNNs</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>4.3565</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 2: The theoretical speed up ratio for market price prediction (n=625).

<table>
<thead>
<tr>
<th>Length of serial data</th>
<th>Number of computation steps required for CTDNNs</th>
<th>Number of computation steps required for FTDNNs</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>1.8373e+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 for market price prediction (n=900).

<table>
<thead>
<tr>
<th>Length of serial data</th>
<th>Number of computation steps required for CTDNNs</th>
<th>Number of computation steps required for FTDNNs</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>0.8587e+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>8.9693</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 for market price prediction.

<table>
<thead>
<tr>
<th>Length of serial 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>