Dynamic Ridge Polynomial Neural Network with a Real Time Recurrent Learning Algorithm: Forecasting the S&P 500

Rozaida GHAZALI, Nazri MOHD NAWI, and Mohd Najib MOHD SALLEH

Faculty of Information Technology and Multimedia, Universiti Tun Hussein Onn Malaysia

Abstract: This paper presents a supervised higher order polynomial neural network which is called Dynamic Ridge Polynomial Neural Network. The network combines the characteristics of higher order and recurrent neural networks. It functionally extends the input space into a higher dimensional space, where linear separability is possible, without suffering from the combinatorial explosion in the number of weights. Furthermore, the presence of the recurrent link expands the network’s ability for attractor dynamics and storing information for later use. In order to predict the future trends of the S&P 500 signals, a Real Time Recurrent Learning algorithm was employed in training the network. Extensive simulations for the prediction of five steps ahead were performed on the signals. Experimental results indicate that the Dynamic Ridge Polynomial Neural Network demonstrated advantages in capturing chaotic movement in the signals with an improvement in the profit return, and rapid convergence over the widely known Multilayer Perceptrons.

Keywords: Dynamic Ridge Polynomial Neural Network, financial time series, higher order neural network, Multilayer Perceptrons.

1. Introduction

Time series forecasting is one of the most important and interesting problem when studying natural occurring phenomena. Its importance stems from the fact that it has wide-ranging applications, including control systems, engineering processes, environmental systems and economics. From the knowledge of some aspect of previous behavior of the system, the aim of the prediction process is to determine or predict its future behavior.

Traditional methods for time series forecasting are statistics-based, including moving average (MA), autoregressive (AR), autoregressive moving average (ARMA) models, linear regression and exponential smoothing. These approaches do not produce fully satisfactory results, due to the nonlinear behavior of most of the natural occurring time series.

Other more advanced techniques such as neural networks, fuzzy logic, and genetic algorithm have been successfully used in time series prediction. The application of neural networks in time series prediction has shown better performance in comparison to statistical methods because of their nonlinear nature and training capability. In addition, it has been shown that neural networks are universal approximators and have the ability to produce complex nonlinear mappings.

However, when the number of inputs to the model and the number of training examples becomes extremely large, the training procedure for ordinary neural network architectures becomes tremendously slow and unduly tedious. To overcome such time-consuming operations, this research work focuses on using Dynamic Ridge Polynomial Neural Network (DRPNN) [1] which has a single layer of learnable weights, therefore reducing the networks' complexity. The Real Time Recurrent Learning algorithm by Williams and Zipser [2] was used to train the DRPNN. The network was used to predict the daily S&P 500 index.

2. Dynamic Ridge Polynomial Neural Network

DRPNN is a type of Higher Order Neural Net-
network (HONN) which contains summing unit and product units that multiply their inputs. These high order terms or product units can increase the information capacity of higher order network in comparison to standard neural networks with summation units only. The utilization of higher order terms allows the neural networks to expand the input space into a higher dimensional space where linear separability is possible.

The structure of the DRPNN is constructed from a number of increasing order of Pi-Sigma Neural Networks (PSNN), as shown in Fig. 1, with the addition of a feedback connection from the output layer back to the input layer [3]. The feedback connection feeds the activation of the output node to the summing nodes in each PSNN unit, thus allowing each building block of PSNN unit to see the resulting output of the previous patterns. In contrast to the ordinary feedforward Ridge Polynomial Neural Network (RPNN) [4], the proposed DRPNN, as shown in Fig. 2 is provided with memories which give the network the ability of retaining information to be used later. All the connection weights from the input layer to the first summing layer are learnable, while the rest are fixed to unity.

Suppose that $M$ is the number of external inputs $U(n)$ to the network, and let $y(n-I)$ to be the output of the DRPNN at previous time step. The overall input to the network are the concatenation of $U(n)$ and $y(n-I)$, and is referred to as $Z(n)$ where:

$$Z_i(n) = \begin{cases} 
U_i(n) & \text{if } 1 \leq i \leq M \\
y(n-I) & i = M + 1 
\end{cases}$$

(1)

The output of the $k$th order DRPNN is determined as follows:

$$y(n) = \sigma \left( \sum_{i=1}^{k} P_i(n) \right)$$

$$P_i(n) = \prod_{j=1}^{i-1} h_j(n)$$

$$h_j(n) = \sum_{i=1}^{m_j} W_{ji} Z_i(n) + W_{bj}$$

where $k$ is the number of PSNN units used, $P_i(n)$ is the output of each PSNN block, $h_j(n)$ is the net sum of the sigma unit in the corresponding PSNN block, $W_{ji}$ is the bias, $\sigma$ is the sigmoid activation function, and $n$ is the current time step.

---

**Fig 1. Pi-Sigma Neural Network of K-th order**
3. Real Time Recurrent Learning Algorithm for DRPNN

DRPNN follows the Real Time Recurrent Learning algorithm (RTRE) [2] for updating the weights of the PSNN unit in the network. This algorithm computes the derivatives of states and outputs with respect to all weights as the network processes the sequence, that is, during the forward step.

A standard error measure used for training the network is the Sum Squared Error:

\[
E(n) = \frac{1}{2} \sum e^2(n)
\]  

(3)

The error between the target and forecast signal is determined as follows:

\[
e(n) = d(n) - y(n)
\]  

(4)

where \(d(n)\) is the target output at time \(n\), \(y(n)\) is the forecast output at time \(n\). At every time \(n\), the weights are updated according to:

\[
\Delta W_u(n) = -\eta \left( \frac{\partial \tilde{y}(n)}{\partial W_u} \right)
\]  

(5)

where \(\eta\) is the learning rate. The value \(\left( \frac{\partial \tilde{y}(n)}{\partial W_u} \right)\) is determined as:

\[
\left( \frac{\partial \tilde{y}(n)}{\partial W_u} \right) = e(n) \frac{\partial \hat{y}(n)}{\partial W_u}
\]  

(6)

\[
\frac{\partial \hat{y}(n)}{\partial W_u} = \frac{\partial y(n)}{\partial P_i(n)} \frac{\partial P_i(n)}{\partial W_u}
\]  

(7)

\[
\frac{\partial y(n)}{\partial P_i(n)} = \frac{\partial y(n)}{\partial P_i(n)} \frac{\partial P_i(n)}{\partial W_u}
\]  

(8)

and

\[
\frac{\partial P_i(n)}{\partial W_u} = \left( \frac{W_i - \hat{y}(n-1)}{W_u} \right) + Z_{ij}(n) \delta_k
\]  

(9)

where \(\delta_k\) is the Kroencker delta. Assume \(D\) as the dynamic system variable (the state of the \(ij^{th}\) neuron), where \(D\) is:

\[
D_{ij}(n) = \frac{\partial y(n)}{\partial W_u}
\]  

(10)

The state of a dynamical system is formally defined as a set of quantities that summarizes all the information about the past behavior of the system that is needed to uniquely describe its future behavior [5]. Substituting Equation (8) and (9) into (7) results in:

\[
D_{ij}(n) = \frac{\partial y(n)}{\partial W_u} = f \left( \sum_{i'=1}^{P} \left( \prod_{j=1}^{h_{ij}(n)} \right) [W_i, D_{ij}(n-1) + Z_{ij}(n) \delta_k] \right)
\]  

(11)

For simplification, the initial values for \(D_{ij}(n-1) = 0\), and \(Z_{ij}(n-1) = 0.5\). Then the weights updating rule is:

\[
\Delta W_u(n) = \eta \left( \frac{\partial \tilde{y}(n)}{\partial W_u} \right) \frac{\partial y(n)}{\partial W_u}
\]  

(12)

where \(W_u\) are adjustable weights and \(\Delta W_u\) are total of weight changes.

DRPNN follows the following steps for updating its weights.
1. Start with low order DRPNN
2. Carry out the training and update the weights asynchronously after each training pattern.
3. When the observed change in error falls below the predefined threshold \( r \), i.e.,
\[
\left| \frac{e(n) - e(n-1)}{e(n-1)} \right| < r ,
\]
a higher order PSNN unit is added.
4. The threshold, \( r \), for the error gradient together with the learning rate, \( n \), are reduced by a suitable factor \( \text{dec} \ r \) and \( \text{dec} \ n \), respectively.
5. The updated network carries out the learning cycle (repeat steps 1 to 4) until the maximum number of epoch is reached.
Notice that every time a higher order PSNN unit is added, the weights of the previously trained PSNN units are kept frozen, whilst the weights of the latest added PSNN are trained.

### 4. Experimental Designs

The performance of the new proposed DRPNN is benchmarked against the performance of ordinary RPNN and MLP. The prediction performance of our networks was evaluated using the Annualized Return (AR) and Normalized Mean Square of the Error (NMSE) matrices as shown in Table 1.

The MLP was trained using the incremental backpropagation learning algorithm [5]. Early stopping was utilized and the signal was divided into three data sets which are the training, validation and the out-of-sample data which represent 50%, 25%, and 25% of the entire data set, respectively. The MLP was trained with hidden units varies from 3 to 8.

For the RPNN and DRPNN, the data were partitioned into two categories: the training and the out-of-sample data, with a distribution of 75% and 25%, respectively. The RPNN were trained with a constructive learning algorithm [4]. The performance of RPNN and DRPNN was evaluated with the number of higher order terms increased from 1 to 5. DRPNN follows the training steps discussed previously.

All network models used in this work were trained at a maximum epoch of 3000. For all neural networks, an average performance of 20 trials was used. The learning rate was selected between 0.05 and 0.5 and the momentum term was experimentally selected between 0.4 and 0.9. A random weights initialization was employed in the range of -0.5 to 0.5.

### Tab. 1. Performance Metrics

<table>
<thead>
<tr>
<th>Annualized Return (%AR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( AR = \frac{\text{Profit}}{\text{All profit}} \times 100 )</td>
</tr>
<tr>
<td>( \text{Profit} = \frac{252}{n} \times CR ), ( CR = \sum_{i=1}^{n} R_i )</td>
</tr>
</tbody>
</table>
| \( R_i = \begin{cases} 
+ |y_i| & \text{if } (y_i)(\hat{y}_i) > 0, \\
- |y_i| & \text{otherwise} 
\end{cases} \) |
| \( All \text{ profit } = \frac{252}{n} \times \sum_{i=1}^{n} \text{abs}(R_i) \) |

<table>
<thead>
<tr>
<th>Normalized Mean Squared Error (NMSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{NMSE} = \frac{1}{\sigma^2 n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 )</td>
</tr>
<tr>
<td>( \sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \overline{\hat{y}})^2 )</td>
</tr>
<tr>
<td>( \overline{\hat{y}} = \frac{1}{n} \sum_{i=1}^{n} \hat{y}_i )</td>
</tr>
</tbody>
</table>

\( n \) is the total number of data patterns.
\( y \) and \( \hat{y} \) represent the actual & predicted output value, respectively.

### 4.1. Forecasting the S&P 500 signals

The Standard & Poor 500 stock index futures (S&P 500) is a market free float-weighted index published since 1957 of the prices of 500 common stocks actively traded in the United States. The stocks included in the S&P 500 are those of large publicly held companies that trade on either of the two largest American stock market companies; the NYSE Euronext and the NASDAQ OMX. It contains the stocks of 500 leading companies in leading industries of the U.S. economy. The index is the most notable of the many indices owned and maintained by Standard & Poor’s, a division of McGraw-Hill.
and lower bounds of the transfer function in order to avoid computational problems and to meet algorithm requirements.

The S&P 500 signals were obtained from a historical database provided by DataStream [6], dated from 01/01/1988 until 11/07/1995, giving a total of 1962 data points. To smooth out the noise and to reduce the trend, the original raw data was pre-processed into a stationary series (as shown in Fig. 3) by transforming them into measurements of relative different in percentage of price (RDP) [7]. The signals were transformed into 5-day RDP, in which the output variable signifies a relative different price of 5 days ahead. The input variables were determined from 4 lagged RDP values based on 5-day periods (RDP-5, RDP-10, RDP-15, and RDP-20) and one transformed signal (EMA15) which is obtained by subtracting a 15-day exponential moving average from the original signal. It is worth noting that the precise values of daily prices (the non-stationary signals) are often not as meaningful to trading as its relative magnitude and the high-frequency component in financial data are often more difficult to be modelled [8]. Furthermore, the distribution of the transformed data will become more symmetrical and will follow more closely to normal distribution (refer to Fig. 4). Subsequent to transformation, all the input and output variables were scaled between the upper

Fig. 3. Signal before and after pre-processing

(a) S&P 500; non-stationary signals

(b) S&P 500; Stationary signals

Fig. 4. Histograms of the signal before and after pre-processing

(a) S&P 500; before pre-processed

(b) S&P 500; after pre-processed

5. Simulation Results

In this simulation, we focus on how the networks generate profits, as this is the main interest in financial time series forecasting. Therefore, during generalization, the network model that endows the highest profit on unseen data is considered the best model.

Table 2 shows the average performance over 20 simulations for various neural network architectures, in which the DRPNN model is benchmarked against the ordinary RPNN and the MLP. In each network, the best average results were chosen from all different network topologies and different learning parameters setting. As it can be noticed from Table 2, all neural network architectures produced good simulation results for the prediction of S&P 500 signal. However, DRPNN offers better profit return when compared to other models. There is a difference of 0.12 percentage of AR between the lowest average result produced by the MLP and the best average result produced by the DRPNN. Meanwhile for the
NMSE, DRPNN has successfully produced the lowest result when compared to other neural networks. The results indicate that reasonable forecasting performance has been achieved through the developed network model.

**Tab. 2. Average Results on Stationary Signals for the Prediction of 5-Steps ahead**

<table>
<thead>
<tr>
<th>Performance Measures</th>
<th>Neural Networks</th>
<th>CMESP</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR(%)</td>
<td>MLP</td>
<td>85.645</td>
</tr>
<tr>
<td></td>
<td>RPNN</td>
<td>85.644</td>
</tr>
<tr>
<td></td>
<td>DRPNN</td>
<td>85.769</td>
</tr>
<tr>
<td>NMSE</td>
<td>MLP</td>
<td>0.295</td>
</tr>
<tr>
<td></td>
<td>RPNN</td>
<td>0.296</td>
</tr>
<tr>
<td></td>
<td>DRPNN</td>
<td>0.291</td>
</tr>
</tbody>
</table>

**Table 3. Average Epoch and CPU Time usage During Training of the Networks**

<table>
<thead>
<tr>
<th>Neural Networks</th>
<th>Num.of Epochs</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>907</td>
<td>520.98</td>
</tr>
<tr>
<td>RPNN</td>
<td>193</td>
<td>42.97</td>
</tr>
<tr>
<td>DRPNN</td>
<td>255</td>
<td>60.06</td>
</tr>
</tbody>
</table>

The average number of epochs reached during the training of the networks for the prediction of five steps ahead are shown in Table 3. In the same table, the amount of CPU time used to learn the S&P 500 signals is presented in order to compare the speed of the networks to execute and complete the training. The CPU time was based on a machine with Windows XP 2000, Intel processor (Pentium 4), CPU of 3.00 GHz, and 1 GB of RAM. Results in Table 3 show that the ordinary RPNN appeared to use the least number of epochs to converge during the training and took least CPU time to learn the signals in comparison to other networks. Despite the fact that the number of epoch and CPU time for DRPNN are slightly higher than that of the ordinary RPNN, the results do not reflect the significant predictive value offered by DRPNN. This is because we are more concern with the out-of-sample prediction value of the network using the percentage of AR, as explained previously. Meanwhile, MLP revealed to utilize large number of epochs and CPU time to finish up the training. In this case, the performance gained by the proposed DRPNN is still much better when compared to the MLP, and it is still acceptable considering good prediction results achieved by the network.

Fig. 5 shows the best prediction on out of sample S&P 500 signal using the proposed DRPNN. As it can be noticed, the plot for the original and predicted signals is very close to each other and at some points they are nearly overlapping. This indicates that DRPNN are capable of learning the behavior of chaotic and highly non-linear financial time series data and they can capture the underlying movements in financial markets.

Simulation results clearly demonstrate that the proposed DRPNN is potentially profitable and beneficial as money-making predictor. The network manifests highly nonlinear dynamical behavior induced by the recurrent feedback, therefore leads to a better input-output mapping and a better forecast. With the recurrent connection, the network outputs depend not only on the initial values of external inputs, but also on the entire history of the system inputs.

**6. Conclusion**

This paper presents the predictive capability of new developed recurrent neural network architecture; Dynamic Ridge Polynomial Neural Network, on the S&P 500 signals. Simulation results confirmed that DRPNN model has the ability to predict financial time series with a better performance than MLP. Apart from generating higher profit return, DRPNN also attained lower forecast error which indicates that the networks can track the signal better than other benchmarked models. Furthermore, DRPNN used smaller number of epochs and less CPU time during the training in comparison to the MLP. This
is obviously due to the presence of only a single layer of adaptive weights. The enhanced performance in the prediction of S&P 500 signal using DRPNN is due to the networks robustness caused by the reduced number of free parameters compared to the MLP.

In view of the fact that the behavior of the financial signal itself related to some past inputs on which the present inputs depends, it therefore requires explicit treatment of dynamics. The merit of DRPNN, as compared to the RPPN is its increased inherited nonlinearity which results from the use of recurrent neural networks architecture, giving it an advantage when dealing with nonlinear and complex time series forecasting.

References

Rozaida GHAZALI
She received her B.Sc. (Hons) degree in Computer Science from Universiti Sains Malaysia, and M.Sc. degree in Computer Science from Universiti Teknologi Malaysia. She obtained her Ph.D. degree in Higher Order Neural Networks at Liverpool John Moores University, UK. She is currently a teaching staff at Faculty of Information technology and Multimedia, Universiti Tun Hussein Onn Malaysia (UTHM). Her research area includes neural networks, fuzzy logic, financial time series prediction and physical time series forecasting.

Nazri MOHD NAWI
He received his B.S. degree in Computer Science from University of Science Malaysia (USM), Penang, Malaysia. His M.Sc. degree in computer science was received from University of Technology Malaysia (UTM), Skudai, Johor, Malaysia. He re eived his Ph.D. degree in Mechanical Engineering department, Swansea University, Wales Swansea. He is currently a lecturer in Software Engineering Department at Universiti Tun Hussein Onn Malaysia (UTHM). His research interests are in optimization, data mining techniques and neural networks.

Mohd Najib MOHD SALLEH
He is a lecturer at Faculty of Information Technology and Multimedia at Universiti Tun Hussein Onn Malaysia since 2001. He had Bachelors degree in Computer Science from Universiti Pertanian Malaysia, Selangor in 1988. He received a Master degree in Computer Science in Information System from Universiti Teknologi Malaysia in 2000. He has a PhD in Data Mining from Universite De La Rochelle, France in 2008. His doctoral thesis was on decision tree modeling with incomplete information in classification task problem. His research interests include rough set theory, artificial intelligence in data mining and knowledge discovery.