Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks

Emad W. Saad, *Student Member, IEEE*, Danil V. Prokhorov, *Member, IEEE*, and Donald C. Wunsch, II, *Senior Member, IEEE*

***Abstract*—Three networks are compared for low false alarm stock trend predictions. Short-term trends, particularly attractive for neural network analysis, can be used profitably in scenarios such as option trading, but only with significant risk. Therefore, we focus on limiting false alarms, which improves the risk/reward ratio by preventing losses. To predict stock trends, we exploit time delay, recurrent, and probabilistic neural networks (TDNN, RNN, and PNN, respectively), utilizing conjugate gradient and multistream extended Kalman filter training for TDNN and RNN. We also discuss different predictability analysis techniques and perform an analysis of predictability based on a history of daily closing price. Our results indicate that all the networks are feasible, the primary preference being one of convenience.**

***Index Terms*—Conjugate gradient, extended Kalman filter, financial engineering, financial forecasting, predictability analysis, probablistic neural network, recurrent neural network, stock market forecasting, time delay neural network, time series analysis, time series prediction, trend prediction.**

I. Introduction

OUR approach to market forecasting capitalizes on two observations: that predictions over a relatively short time are easier to do reliably, and that attempts to profit on short-term moves need this reliability to compensate for risks, taxes, and transaction costs. We arbitrarily select a 2% upwards move in stock price within 22 working days as presenting a reasonable opportunity to profit utilizing a suitable leveraged strategy such as call options or margin trading. The risks inherent in such a strategy require a strong emphasis on reliability of signals. Therefore we focus on limiting false alarms, i.e., inaccurate prediction of 2% or greater increases, adopting a strategy of waiting for the best opportunities to take a position. Incidentally, focusing only on upwards moves is chosen because of the general upward trend of the market over time.

Earlier we demonstrated that short-term predictions are achievable using probabilistic, time-delay, and recurrent networks [1]–[3]. This paper compares the three networks and evaluates them against a conventional method of prediction. Also, a predictability analysis of the stock data is presented and related to the neural-network results. In Sections II–IV, we describe the architecture as well as the training procedure of the time-delay, probabilistic, and recurrent neural networks (TDNN, PNN, and RNN), respectively. Section V describes a linear classifier, for the purpose of comparatively evaluating the neural networks performance. In Section VI, we present several techniques which can be used to analyze the predictability of the different stocks. In Section VII, the different experimental tests as well as their results are described, starting with the predictability tests, followed by the actual forecasting using different neural networks and conventional classifier. Section VIII contains a comparison of the three networks from the point of view of architecture, implementation complexity, training time, and forecasting capability, and Section IX is the conclusion.

Manuscript received July 16, 1996; revised June 3, 1997 and June 30, 1998. This work was supported by the National Science Foundation, Ford Scientific Research Laboratory, Texas Advanced Technology Program, and the Texas Tech Center for Applied Research.

E. W. Saad and D. C. Wunsch, II, are with the Applied Computational Intelligence Laboratory, Department of Electrical Engineering, Texas Tech University, Lubbock, TX 79409-3102 USA.

D. V. Prokhorov is with the Scientific Research Laboratory, Ford Motor Co., Dearborn, MI 48121-2053 USA.

Publisher Item Identifier S 1045-9227(98)07353-6.

II. Time-Delay Neural Networks

The TDNN used in this study are feedforward multilayer perceptrons, where the internal weights are replaced by finite impulse response (FIR) filters (Fig. 1). This builds an internal memory for time series prediction [4]–[8]. Our goal is not price prediction but rather trend prediction, which can be formulated as a problem of pattern classification. An output of "1" corresponds to an upward trend of 2% or more, while an output of "-1" corresponds to a downward trend or upward trend less than 2%. TDNN is appropriate for this because trend prediction requires memory.

By using FIR filters as synaptic connections, each weight is replaced by a weight vector

$$\mathbf{w}_{ji} = [w_{ji}(0), w_{ji}(1), \cdots, w_{ji}(M)]^T \tag{1}$$

where $\mathbf{w}_{ji}$ is the vector carrying signals from neuron i to neuron j.

The output state of neuron i is an M dimensional vector $\mathbf{x}_i(t)$, storing the history of the state of neuron i through M previous time steps

$$\mathbf{x}_i(t) = [x_i(t), x_i(t-1), \cdots, x_i(t-M)]^T. \tag{2}$$

The input signal to every neuron at the application of the input pattern t includes the output signals from the previous layer

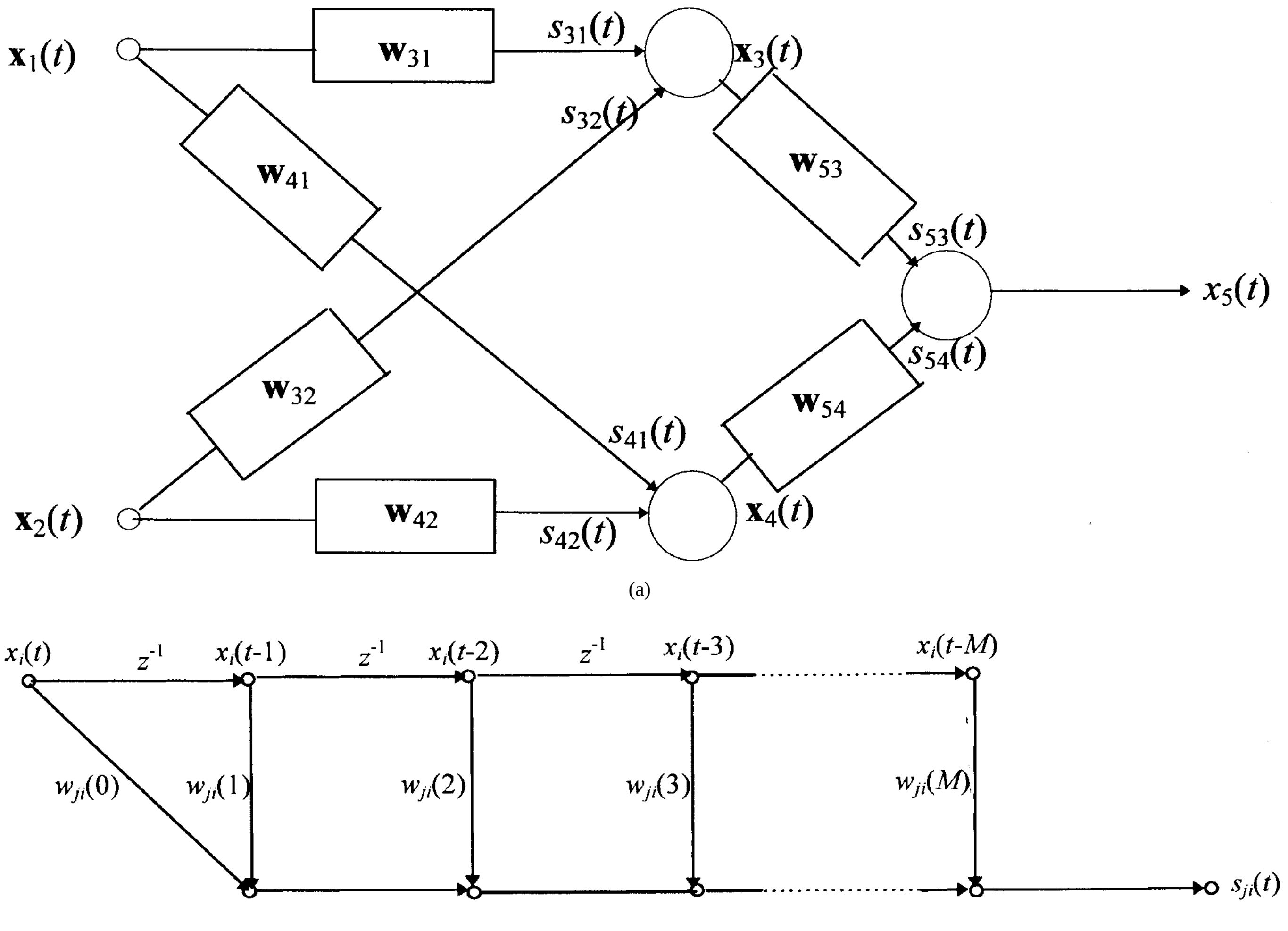


Fig. 1. (a) Three-neuron TDNN with FIR filters ($\mathbf{w}_{ji}$) as synaptic connections. (b) Expanded view of FIR synaptic connections of TDNN. FIR filters build internal memory into the network.

at time steps $t, t-1, t-2, \cdots, t-M$, where M is the delay associated with the corresponding weight vector.

Taking the inner product of $\mathbf{w}_{ji}$ and $\mathbf{x}_i(t)$, we get the output of the weight filter

$$s_{ji}(t) = \mathbf{w}_{ji}^T \mathbf{x}_i(t). \tag{3}$$

Summing the FIR weights outputs, we get the activation potential $v_j(n)$ of the neuron j; i.e.,

$$v_j(t) = \sum_{i=0}^{p} s_{ji}(t) \tag{4}$$

where p is the number of neurons in the previous layer of the network and $s_{ji}(0)$ represents a threshold. The nonlinear activation function of the neuron then produces an output $y_j(t)$; i.e.,

$$y_j(t) = f(v_j(t)). \tag{5}$$

For all neurons, we used the hyperbolic tangent activation function

$$f(x) = (1 - e^{-x})/(1 + e^{-x}) \tag{6}$$

which usually accelerates training [5].

A. Cost Function

Conventional training of TDNN consists of minimizing a cost function, the mean square error of the network. Since we are willing to forego some profit opportunities in order to limit false alarms, we introduce a penalizing factor which punishes the network for false alarms more than for missed profit opportunities

$$E = \sum_{t=1}^{N} j(t) = 1/2 \sum_{t=1}^{N} \alpha(y(t) - d(t))^2 \tag{7}$$

where $d(t)$ is the desired output.

The quadratic error of the network is multiplied by the penalizing factor α

$$\alpha = \begin{cases} 1, & \text{if } y > 0,\, d > 0 \text{ or } y < 0,\, d < 0 \\ M, & \text{if } y < 0,\, d > 0 \\ F, & \text{if } y > 0,\, d < 0. \end{cases} \tag{8}$$

The penalizing factor is equal to one if classification is correct, to the constant M for missed predictions, and to F for false alarms. To limit the false alarm ratio, F is greater than M.

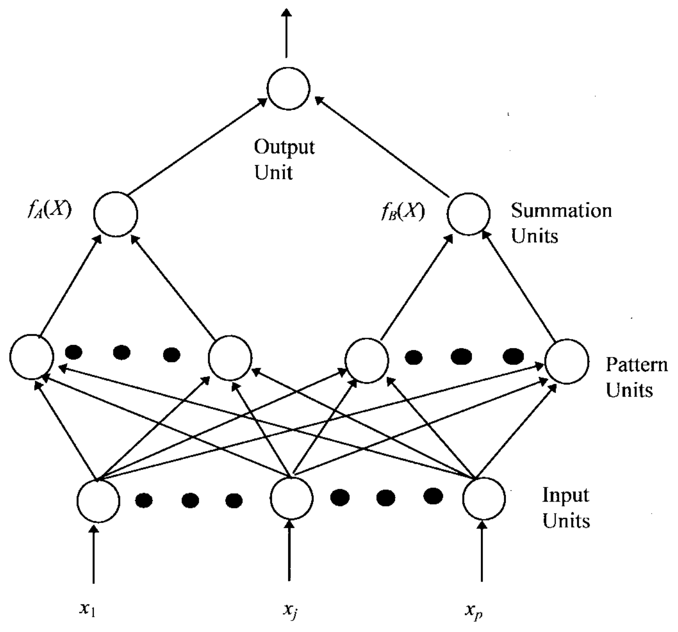


Fig. 2. PNN architecture. This network is the fastest in training, at the cost of requiring one hidden unit (pattern unit) for each pattern in the entire training set.

B. Training TDNN

One way to train TDNN is to transform the network into a static one by unfolding it in time [5]. However, this method has the drawback of creating redundant weights, and therefore needs bookkeeping, in addition to that it does not have symmetry between forward propagation and backpropagation. Another method is temporal backpropagation [4]. While minimizing the total cost, this method incorporates dependence of the future step costs on the current step state. Detailed description of the training method can be found in [5]. We further enhance our system by utilizing conjugate gradient training [5], [9]–[16]. Results of TDNN forecasting using the above techniques appear in Section VII.

III. Probabilistic Neural Network

The probabilistic neural network (PNN) [17] is an algorithm for approximating the Bayesian decision rule [18]. We use a PNN with four layers of dedicated nodes (Fig. 2). Twenty-nine input nodes are fully connected with the next layer of pattern nodes. Input nodes distribute components of the input X. The PNN requires one pattern node for every pattern in the training set. The ith pattern node output function is

$$y_i = \exp(-(X_i - X)^T(X_i - X)/2\sigma^2) \tag{9}$$

where X_i is the ith training pattern, and σ is the smoothing parameter of the Gaussian kernel. Other alternatives to y_i are available [17], including y_i with adaptable σ [19], and full covariance matrices [20].

The third layer is formed by summation nodes which sum the outputs y_i of those pattern units that correspond to two categories of predictions, A and B. In our case, category A is an acceptable profit in the next 22 working days, and category B is the opposite.

The output node adds signals from these two summation nodes, with only the summation node B weighted by the parameter

$$C = -L \tag{10}$$

where $L = l_A/l_B$ is the ratio of losses associated with false alarms to those associated with missed profit opportunities. We have used $L > 1$, emphasizing the importance of avoiding false alarms. At the output, we have a hard-limiting threshold: $+1$ whenever an input pattern X belongs to category (A) and -1 if it is from category (B).

PNN memorizes all training patterns. Each time a new pattern X is presented to the network, it computes (9) for all pattern units, sums up the resulting y_i's in the summation units, and outputs the result. PNN's generalization certainly depends on the value of the smoothing parameter σ, and on how well the training data represents the problem domain.

Our approach to forecasting with PNN is based on the following preprocessing technique [21], [22]. For each pattern, 29 inputs are used: 14 Level-0 features and 15 Level-1 features. The definition for our features are

Level-0:

$$feature(t) = \log \frac{\text{value of } close(t)}{\text{exponential moving average of } close(t)}. \tag{11}$$

Level-1:

$$feature(t) = \frac{close(t) - BA(t-n)}{close(t) + BA(t-n)} \tag{12}$$

where $close(t)$ is the stock's closing price on day t, and

$$BA(t-n) = 1/(m+1) \sum_{k=-(m/2)}^{(m/2)} close(t-n+k). \tag{13}$$

The idea behind (11)–(13) is that the stock price series is assumed to be formed of cycles of different frequencies. If we sample the data at different frequencies, the samples would carry all the information in the series. In order to do this, we use sample blocks of data. The farther in the past is the block, the further it is spaced from the next block, and the larger is the block size. The index n determines how far back in time the center of the block is situated. m is chosen such that it covers the period between the consecutive blocks. The indexes n and m are provided as shown in (13a) at the bottom of the page. After several tests, we selected $n = 97$. Thus, almost six months (about 113 working days) of historical data

$$\begin{array}{ccccccccccccccccccc} n & 1 & 2 & 3 & 4 & 5 & 7 & 9 & 13 & 17 & 25 & 33 & 49 & 65 & 97 & 129 & 193 & 257 & 385 \\ m & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 4 & 4 & 8 & 8 & 16 & 16 & 32 & 32 & 64 & 64 & 128 \end{array} \tag{13a}$$

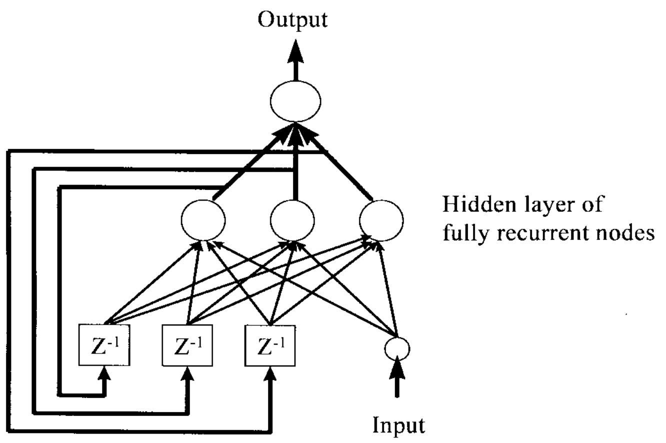


Fig. 3. Recurrent network architecture. Z^{-1} represents a one time step delay unit. This network has a compact memory structure. The EKF described is well-suited for this architecture.

are used to predict the trend of the stock closing price in the next month.

Results of PNN forecasting appear in Section VII.

IV. Recurrent Neural Network and Its Training

The recurrent neural network (RNN) considered in this paper (Fig. 3) is a type of discrete-time recurrent multilayer perceptrons [23]. Temporal representation capabilities of this RNN can be better than those of purely feedforward networks, even with tapped-delay lines. Unlike other networks, RNN is capable of representing and encoding deeply hidden states, in which a network's output depends on an arbitrary number of previous inputs.

Among many methods proposed for training RNN's, extended Kalman filter[1] (EKF) training stands out [24]. EKF training is a parameter identification technique for a nonlinear dynamic system (RNN). This method adapts weights of the network pattern-by-pattern accumulating training information in approximate error covariance matrices and providing individually adjusted updates for the network's weights.

We assume that all weights of the RNN are assembled in a vector W of length M. This can be split into G groups. Denoting W_i as a vector of the ith group of weights, we assume that $W_1 \cup W_2 \cup \cdots \cup W_G = W$ and size(W_1) + size$(W_2) + \cdots +$ size$(W_G) = M$.

It is required by the EKF method that we compute derivatives of the RNN's N outputs, rather than output errors, with respect to the weights [24]. These derivatives are obtained through backpropagation through time or its truncated version [25], [26]. We store them in a set of matrices H_i, where each H_i has dimension: size$(W_i) \times N$. This set is obtained by truncated backpropagation through time, with depth 20, meaning that we do not use more than 20 copies of the network to accumulate appropriate derivatives.

[1]We note that the full name of the EKF method described here is parameter-based node-decoupled EKF.

The following equations form the basis of the EKF training procedure:

$$A(k) = \left(\frac{S(k)^{-1}}{\eta(k)} = \sum_{i=1}^{G} H_i^T(k) P_i(k) H_i(k) \right)^{-1} \tag{14}$$

$$K_i(k) = P_i(k) H_i(k) A(k) \tag{15}$$

$$W_i(k+1) = W_i(k) + K_i(k) e(k) \tag{16}$$

$$P_i(k+1) = P_i(k) - K_i(k) H_i^T(k) P_i(k) + Q_i(k) \tag{17}$$

where $\eta(k)$ is a scalar learning rate, $K_i(k)$ is the Kalman gain matrix for the ith group of weights, $e(k) = d(k) - y(k)$ is the $N \times 1$ error vector, $d(k)$ is a vector of the desired outputs, $y(k)$ is the actual output vector of the RNN; $e^T(k)S(k)e(k)/2$ forms the squared error to be minimized, where $S(k)$ is a user-specified matrix introduced to incorporate weighting of individual components in the error vector $e(k)$ (this matrix is often equal to the unity matrix, but it incorporates the penalty factor α of (7) and (8) in our case), $P_i(k)$ is the size$(W_i) \times$ size(W_i) approximate error covariance matrix which models correlation between each pair of weights within the ith group of weights, and $Q_i(k)$ is a positive diagonal matrix that helps to avoid numerical divergence of the procedure and prevents getting stuck in a local minimum [24].

Grouping of weights can be done in a variety of ways. We employ grouping by node, i.e., weights belonging to the same neuron are grouped together. Thus, we ignore correlation between weights belonging to different neurons. This results in a significant reduction of computational complexity and storage requirements since the size of each error covariance matrix P_i can then be made much smaller than M^2, the size in the case when $G = 1$ [24].

The matrices $P_i(0)$ are initialized as diagonal matrices with large diagonal elements (values between 100 and 10 000). User-specified values of $\eta(k)$ are usually increased from 0.01 to 10 whereas diagonal components of $Q_i(k)$ are decreased from 0.01 to 10^{-6} as training progresses.

When there is a large data base including a variety of operational conditions (different market trends), batch updating of weights is superior to pattern-by-pattern updating since the network learns to simultaneously minimize error on a batch of patterns taken from different regions of the data base. To combine efficiency of EKF training with a batch-like update, without violating consistency between the weights and the approximate error covariance matrices P_i, the multistream training approach was first proposed and tested in [27].

We assume S data streams. It is then useful to consider S copies of the same RNN (weights are identical for all the copies). Each copy is assigned a separate stream. We apply each copy of the RNN to a training vector drawn from its stream. We obtain S stream error vectors and S sets of matrices H_i. We concatenate all H_i to form one H_i of dimension: size$(W_i) \times (N \times S)$, where $N \times S$ denotes the total number of columns. Concatenating all the stream error vectors results in a single error vector e of dimension: $(N \times S) \times 1$, where $N \times S$ is the total number of rows. The number of columns of each of the Kalman gain matrices K_i is increased S times.

The global scaling matrix A also grows S times for each of its dimensions.

Among all these increases of dimensionality, only the matrix A's "swelling" is critical because it must be inverted. This may be a potential bottleneck of the training procedure, particularly when both N and S are large. While applying the multistream EKF procedure to a variety of real-world tasks, we have, however, experienced no problems [3], [28], [29]. We have found singular-value decomposition best for inverting $A(k)$ up to 100 $\times$ 100 elements [9].

Allocation of data streams can be done in various ways. It is impractical to use too short streams since the total number of streams S may otherwise be too large, prohibiting inversion of A. On the other hand, having too few streams may not cause a significant improvement over the one-stream (basic) EKF training. We usually see an order of magnitude faster and superior generalization with the multistream EKF. For large data sets, this advantage increases. One-stream training fails for sets of several thousand vectors [28], [29]. We have used multistream EKF training of RNN for predicting trends of several stocks. A typical stock history amounts to less than 2500 days. Any of 40 training streams starts from an entry point chosen randomly in the whole set. One weight update is then based on 40 training pairs. After the first 40 pairs have been processed, we proceed to the next 40 pairs. We repeat updating until acceptable performance is attained.

Our final network's architecture is the one-hidden-layer RNN with one input of a normalized daily closing price, eight fully recurrent hidden neurons and one output neuron. Recurrent and output nodes have the common bipolar sigmoid nonlinearity defined by (6).

Training usually lasts for around one hundred passes, where one pass corresponds to a complete processing of the whole training set. As in Section II-A, we also incorporated penalizing factors in the error measure for RNN training. We have experimentally determined that the optimal range for the penalizing factor is between 3 and 5. Missed opportunities receive no extra penalty.

V. Conventional Classifier

It is useful to check the performance of a linear classifier applied to our problem. This establishes the lower limit of performance when comparing results of various neural networks (see Section VII).

The linear classifier we used is the Fisher linear classifier [18], [30] which has the form

$$h(X) = V^T X + V_o \tag{18}$$

where X is the vector to be classified. It consists of a delay line of length 50: $x(t)$, $x(t-1)$, $x(t-2)$, $\cdots$, $x(t-49)$ which carries the stock price on the day of purchase as well as on the previous 49 days. V is a linear mapping function, and V_o is a threshold. If $h(X)$ is positive, the pattern is classified as a profit opportunity.

We define the within-class scatter matrix and the between-class scatter matrix, respectively, as

$$S_w = \sum_i P_i E(X - M_i)(X - M_i)^T | \omega_i = \sum_i P_i \sum_i \tag{19}$$

and

$$S_b = \sum_i P_i (M_i - M_o)(M_i - M_o)^T \tag{20}$$

where ω_i means the sample belongs to class i, M_i is the expected vector, $\sum_i$ is the covariance matrix, P_i is the *a priori* probability of class i, and M_o is the expected vector of the mixture distribution which is expressed by

$$M_o = E\{X\} = \sum_i P_i M_i. \tag{21}$$

Our criterion is to find the mapping function V which maximizes the between-class scatter and minimizes the within-class scatter. There are several ways to formulate this criterion. One is

$$J = \text{tr}(S_w^{-1} S_b). \tag{22}$$

The map is then found to be

$$V = \frac{S_w^{-1}(M_2 - M_1)}{\|S_w^{-1}(M_2 - M_1)\|}. \tag{23}$$

The optimum threshold is calculated as follows [18]:

$$V_o = -V^T (P_1 M_1 + P_2 M_2). \tag{24}$$

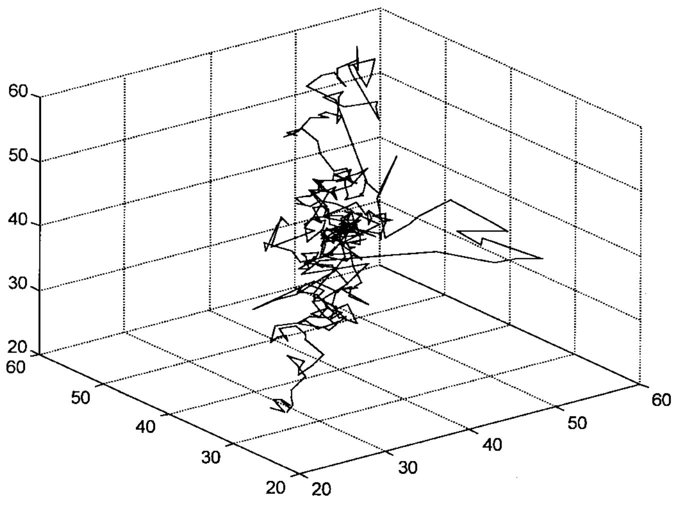
Using $S_w^{-1} S_b$ as criteria is valid only when the class means are distinct. Calculation of the two-class mean vectors showed them to be relatively distinct. We compare performance of the linear classifier (18)–(24) with that of neural networks of the preceding sections in Section VII.

VI. Predictability Analysis

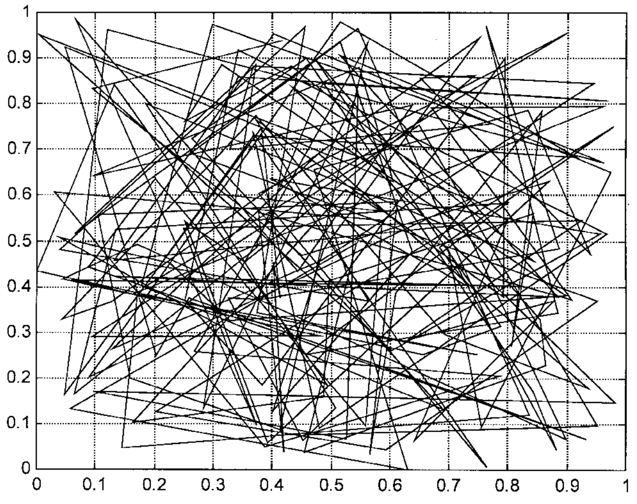
The stock market is governed by a mixture of both deterministic and random factors. It is partially random, partially chaotic [31]. Though chaotic series are hard to distinguish from random ones, they are driven by deterministic equations. While random series are unpredictable, chaotic ones are in many cases predictable by neural networks which can learn to model the underlying function from samples. Predictability analysis is intended to determine the degree to which a stock is chaotic. This can be achieved by several tools, either graphical or quantitative. As we will see, the graphical methods described here are easier to use but carry little information. Therefore, they are typically used as a preliminary test. Quantitative methods require more computations, but they are usually more powerful. However, the state of the art at this writing is still limited.

A. Phase Space Diagrams

A phase space diagram (phase diagram) is the easiest test of chaotic behavior. It is a scatter plot where the independent variable is the value of a time series (i.e., closing price) $x(t)$ at time t, and the dependent variable is $x(t + \tau)$.



(a)



(b)

Fig. 4. (a) Three-dimensional phase diagram of AAPL stock with $\tau = 9$ shows some regularity. (b) Two-dimensional phase diagram of a random series. The path covers all the data range.

The phase diagram of a deterministic system is identified by its regularity. The trajectory is contained in a limited area of the range of the series called an attractor. Furthermore, if the series is chaotic, the attractor has a complicated shape and is called a strange attractor. This is in contrast to a random series where the trajectory covers all the range of the diagram (Fig. 4).

Phase diagrams can be plotted only in two or three dimensions, which is the main shortcoming of this technique. Each dimension is called the embedding dimension. The process of representing a system by one variable and its lagged versions is called embedding [32]. A system, with an n dimensional attractor, can be embedded in an m dimensional space provided that

$$m \geq 2n + 1. \tag{25}$$

This is according to Takens embedding theorem [33]. The attractor dimension can be estimated by different ways, including the method explained in the following section. While Takens theorem gives no indication of how to choose τ, Abarbanel *et al.* [34] proposes the following method to calculate a suitable value for τ. We first compute the correlation factor

$$C = \frac{1/N \sum_{i=1}^{N} [x(i+\tau) - \bar{x}][x(i) - \bar{x}]}{1/N \sum_{i=1}^{N} [x(i) - \bar{x}]^2} \tag{26}$$

as a function of τ. The denominator is simply the variance and it serves for normalization. We then find the value of τ which gives the first zero correlation, as the lag for the phase space. An alternative method is to choose τ which gives the first minimum of the mutual information $I[x(i)|x(i+\tau)]$, [35]. This is supported by the argument that the different coordinates should be an uncorrelated delay (see also Section VI-D1). Unfortunately, according to (25), the method of plotting the phase diagram in two or three dimensions is valid only for attractors of dimension less than unity and hence unusable for most economic systems due to their high dimensionality.

B. Correlation Dimension

This is one of the most popular measures of chaos. It has been introduced by Grassberger and Procaccia [36], and it is a measure of the fractal dimension of a strange attractor. The name fractal comes from the fact that the dimension is not an integer [37]. For example, the attractor of a stable system is a point which has zero dimension. The attractor of an oscillating system is a circle which has two dimensions. These attractors have integer dimensions. A chaotic attractor is characterized by a noninteger dimension. The correlation dimension d can be viewed as measure of the deviation of a time series from randomness and is calculated as follows.

1) We choose a suitable time lag τ. From the scalar time series, we form T points of m embedding dimensions

$$X(t) = [x(t), x(t+\tau), x(t+2\tau), \cdots, \\ \cdot x(t+(m-1)\tau)]. \tag{27}$$

2) For every point, we count the number N_i of points within a hypersphere of radius r centered at that point. Assuming random distribution, the probability that a point lies within r of the chosen point will be

$$p_i = N_i/T - 1. \tag{28}$$

Over T points, we get the average probability

$$C(r, m) = 1/[T(T-1)] \\ \sum_{i=1}^{T}\sum_{j=1}^{T} \cdot H(r - \|X(i) - X(j)\|) i \neq j \tag{29}$$

where

$$H(y) = \begin{cases} 1, & \text{if } y > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{30}$$

3) We vary r, calculate $C(r, m)$, and plot log $C(r)$ versus log r for a given value of m.

4) The slope of the linear part of this curve is the correlation dimension d, based on the relation

$$C(r, m) = r^d \quad \text{for} \quad m > d \quad \text{and} \quad r \to 0. \tag{31}$$

5) We calculate then $C(m)$ and plot it against the embedding dimension over a sufficiently large range of m. If $C(m)$ increases infinitely with m, this indicates that the process is random. If it stabilizes after some value of m, this indicates that we have a chaotic series.

1) Effect of Finite Number of Data Points: If we use a series of finite length T, it establishes an upper limit on the calculated correlation dimension [38]. Consider the slope of $C(r)$ over q orders of magnitude, extending from r_1 to r_2, which correspond to $C_1(r)$ and $C_2(r)$, respectively. We then find

$$d = [\log C_2(r) - \log C_1(r)]/q. \tag{32}$$

The lower limit of $C_2(r)$ is $2/T(T-1)$, and the upper limit is one. For large T, the calculated fractal dimension will have an upper limit of 2 $\log T/q$. If we calculate the slope over just one order of magnitude, the upper limit becomes

$$d \leq 2 \log T. \tag{33}$$

For financial data series of length of the order of 1000 (corresponding to about four-year history), the upper limit becomes six. Hence this method fails for a large class of financial series, because of their high dimensionality. In addition, nonstationarity of the stock price will hinder its use even if data covering a long history is available.

C. Lyapunov Exponent

Chaos is characterized by sensitivity to initial conditions. The Lyapunov Exponent measures divergence of two orbits starting with slightly different initial conditions [37]. If one orbit starts at x_o and the other at $x_o + \Delta x_o$, after n steps, the divergence between orbits becomes

$$\Delta x_n = |f^n(x_o + \Delta x_o) - f^n(x_o)| \tag{34}$$

where $x_{n+1} = f(x_n)$. For chaotic orbits, Δx_n increases exponentially for large n

$$\Delta x_n = \Delta x_o e^{\lambda n} \tag{35}$$

where λ is the Lyapunov Exponent

$$\lambda = \lim_{n \to \infty} [(1/n) \ln(\Delta x_n / \Delta x_o)]. \tag{36}$$

A positive exponent indicates chaotic behavior. From our financial data the Lyapunov Exponent is calculated as follows.

1) We test the series by plotting $\log \Delta x_n$ against n. If the relation is close to linear, then the divergence is exponential and we can proceed to calculate the Lyapunov Exponent.

2) Starting from two points x_i and x_j close in value but not in time, we have

$$\Delta x_o = |x_i - x_j| \tag{37}$$

and

$$\Delta x_n = |x_{i+n} - x_{j+n}|. \tag{38}$$

3) We repeat the last step for an arbitrary N number of times and calculate the Lyapunov exponent at each time using (36) and take the average value.

The Lyapunov exponent method can be used for high dimensionality series since it does not involve calculating the system dimension. The lower limit on the series length is the number of points N on which the exponent is averaged. Usually several hundred points are enough.

D. Relation of the Predictability Analysis to Choices of Neural Network Techniques

1) Network Architecture: The neural-network architecture and the phase space characteristics of a time series are strongly related. Calculating the fractal dimension of a series by the method of Grassberger gives us an estimate of the number of degrees of freedom in the system. If we then calculate a suitable delay using (26), we can reconstruct the phase space of the system. Since highly correlated inputs to the network are redundant [39], using the reconstructed phase space variables $x(t), x(t-\tau), x(t-2\tau), \cdots, x(t-(m-1)\tau)$ as inputs to the time delay network, might be an improvement.

If the conventional methods fail to calculate the system dimension, we can minimize output error of a neural network as a function of the number of hidden neurons [40], [41]. This number can estimate the system dimension.

2) Multistep Prediction: Chaotic systems are characterized by the exponential divergence between two closely starting paths. This makes the multistep prediction of a chaotic system an impossible task [41]. Even if the network is very well trained to make one step predictions, a very small error at the first step prediction will increase exponentially with time and may be unacceptable after the n-step prediction according to the relation

$$E_n = E_o e^{\lambda n} \tag{39}$$

where E_o and E_n are the first and nth step prediction errors, respectively.

We have previously compared neural and conventional (ARMA) techniques for price prediction [2], as opposed to trend classification considered in this paper. While performance of both TDNN and ARMA is similar for single step predictions, TDNN outperforms ARMA in accuracy of multistep predictions.

Comparing neural network prediction to a threshold autoregressive (TAR) model predictions of the sunspot series, Weigend [41] found that both were comparable in single step predictions, but the neural network outperformed the TAR model in multistep predictions. He also suggests in [40] that instead of using the Lyapunov exponent to calculate the n-step prediction error, we can estimate Lyapunov exponent in (39)

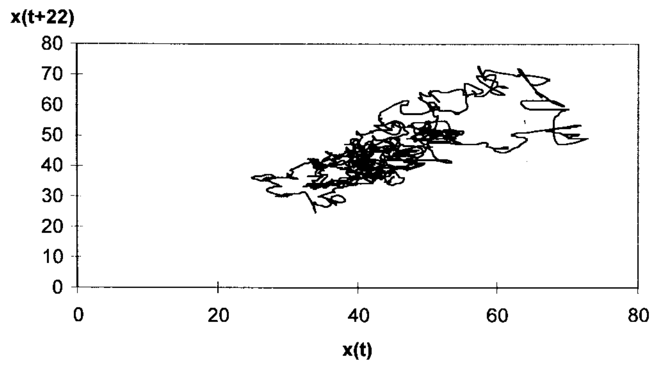


Fig. 5. Phase diagram for Apple stock using a delay of 22 days, shows some regularity in contrast with the random series phase diagram in Fig. 4.

through the calculation of the multistep prediction error. This is particularly convenient when conventional methods fail due to insufficiency of data points or due to other practical problems.

Among neural networks used in this study, only RNN utilizes a concept of multistep predictions when its hidden nodes feed themselves with their outputs from the previous time step. Though distinct Lyapunov exponents of some stocks may suggest using different truncation depths of backpropagation through time for RNN training (see Section IV), we found depth 20 to be a reasonable choice for all the stocks.

VII. Results

Predictability and forecasting tests have been run on a variety of stocks and on different data sets, where some are more recent than the others. Generally the later tests confirm the earlier ones. The tests run in or before April 1996 include only Apple, IBM, Motorola, and Microsoft stocks. The most recent tests run in 1997 include the following stocks which cover a larger variety of categories: Apple (AAPL), IBM (IBM), Motorola (MOT), Microsoft (MSFT) represent the technology group which generally has high volatility. American Express (AXP), Wells Fargo (WFC) represent the banks. Walt Disney Co. (DIS), McDonald (MCD) represent the consumer stocks. Public Svc New Mexico (PNM), Energas (V.EEG) are cyclical stocks.

A. Predictability Analysis

1): Phase Diagrams: Phase diagrams of Apple, IBM, Motorola, and Microsoft stock prices have been plotted in two dimensions, with a delay of 22 days, following the procedure described in Section VI-A. They showed some regularity, which indicate the presence of determinism in the stocks. Fig. 5 shows the phase diagram of Apple stock. The other stocks have similar diagrams.

2) Correlation Dimension: Calculated correlation dimension for the stocks under test was found to increase as the embedding dimension is increased (up to an embedding dimension of six for Apple stock). We interpret the failure of this method as a consequence of the high dimensionality of the financial system as explained in Section VI-B-1.

3) Lyapunov Exponent: This is the most successful predictability analysis technique. Averaging over 50 points, we found Lyapunov exponents of 0.44, 0.55, 0.31, and 0.3 for Apple, IBM, Motorola, and Microsoft stocks, respectively, for the data ending in April 1996. They are all positive, suggesting that all the stocks under test are chaotic, hence predictable. The Lyapunov exponents have been recalculated after updating the data in April 1997 and adding more stocks. These newer results are summarized in Table I. Most of the stocks have a Lyapunov exponents in the same range, except the cyclical stocks which have relatively lower Lyapunov exponents. A zero Lyapunov exponent means either the series is random or the series is periodic. The low Lyapunov exponent in this case can be attributed to the periodicity present in the cyclic stocks.

4) Summary of Results: Though the Apple phase diagram showed some regularity, it doesn't carry much information for the reasons explained in Section VI-A. The correlation dimension failed as was expected due to the reasons explained in Section VI-B. The Lyapunov exponent was the most successful test, and it is in agreement with the practical neural networks results, since all stocks gave positive exponents which indicates their deviation from randomness, and hence the possibility of prediction. The periodicity in the PNM and VEEG stocks was confirmed by the small Lyapunov exponent.

B. Neural-Network Forecasting

In the following four tests, profit opportunities are defined as days which are followed by a 2% up move of the stock closing price, within the following 22 working days. Test number 5 also include predictions of 5 and 10% up moves.

1) Results of PNN: The probabilistic network was tested for $L = 1$, 2, 4, and 8, and for a smoothing parameter ranging from $\sigma = 0.025$ to $\sigma = 0.2$ with a step of 0.025. These tests were repeated using 500, 1000, 1500, and 1800 training points ending on April 31, 1995. The test set had 200 points starting from the end of the training set and ending on February 15, 1996. The best results of these tests are listed in Table II.

2) Results of TDNN: We have trained the time-delay network for iterative price prediction from which the profit opportunities were extracted. The stopping criteria, as explained in Section II-C, was stabilization of the error. The network had 50 input delays and no delays in the hidden layer. The test data extended from March 1, 1995 to August 23, 1995 (126 days). The best number of training days varied. Table III shows the results.

3) Results of RNN: We trained RNN on data dating back to January 2, 1987. The training was stopped as soon as the network was trained long enough. Long enough was typically below 100 passes, with a false alarm rate of less than 10%, and missed about 80% of opportunities. We used a penalty factor [see equations of Section II-A and matrix S in (14)] between 1 and 3, gradually increasing it. We usually started training with values of the learning rate η as low as 0.01, increasing it by an order of magnitude once in ten or 20 passes. A similar strategy was applied to vary the diagonal components of the matrices Q_i except that we kept decreasing them to 10^{-5} or 10^{-6} (see Section IV). All diagonal components were fixed to be the same and to be varied in synchrony with adjustments of the learning rate. Our results are given in Table IV.

TABLE I
LYAPUNOV EXPONENT'S FOR THE STOCK DATA, CALCULATED USING THE METHOD IN SECTION VI-C

Stock	AAPL	IBM	MOT	MSFT	AXP	WFC	DIS	MCD	PNM	VEEG
λ	.4236	.4563	.3068	.1851	.2454	.5424	.3061	.1678	.0276	.0987

TABLE II
BEST RESULTS OF PNN TESTED ON DATA ENDING ON FEBRUARY 15, 1996

Stock	σ	L	Training Days	No. of profit opportunities	Network Predictions	Missed Opportunities	False Alarms	% F.A.
Apple	0.025	1	500	131	8	124	1	12.5
IBM	0.175	4	1800	150	38	113	1	2.63
IBM	0.2	8	1800	150	16	134	0	0
Motorola	0.025	1*	1800	147	8	139	0	0
Microsoft	0.025	1*	500	166	22	146	2	9.09
Microsoft	0.025	1*	1800	166	8	158	0	0

* The same result was obtained with all other values of L.

TABLE III
RESULTS OF TDNN TESTED ON DATA ENDING ON AUGUST 23, 1995

Stock	No. of profit opportunities	Network Predictions	Missed Opportunities	False Alarms	% F.A.
Apple	94	30	64	0	0
IBM	117	37	83	3	8.1
Motorola	108	29	83	4	13.8

TABLE IV
RESULTS OF RNN TESTED ON DATA ENDING ON AUGUST 23, 1995

Stock	No. of profit opportunities	Network Predictions	Missed Opportunities	False Alarms	% F.A.
Apple	94	36	59	1	2.8
IBM	117	78	40	1	1.3
Motorola	108	39	71	2	5.1

4) Comparative Test: Increasing the number of participating stocks, we added Microsoft for all three networks, and we updated our data to the end of April 1996. The three networks were tested on a 100 days set ending on March 20, 1996 (the last month data was needed to calculate the profit opportunity of the last day in test). The results indicated that retraining the networks was necessary. We retrained the three networks by augmenting our old training sets with new data up to October 18, 1995. The results of the three networks are summarized in Table V and shown in Fig. 6.

For the PNN, we tried combinations of σ and L in some range around those values which gave the best results in the former test. The new results were as good as the old ones.

The TDNN was trained to do direct trend prediction, which is easier than the price prediction. We used the technique described in Section VI-D1 to determine the delays between the network inputs. In training the networks, different combinations of the network parameters have been examined. These variable parameters were the number of layers, number of neurons in each layer, number of delays in each layer, delay τ between inputs, penalty factor, length of training set and range of initial random weights. In all cases a one-hidden-layer network performed better than a two-hidden-layer one. We used the correlation coefficient described in Section VI-A to estimate the delay between inputs. We used seven inputs with delay $\tau = 70$ between one another for all stocks except Microsoft for which we used ten inputs with delay $\tau = 50$. The hidden layer had from three to seven neurons, and the input and hidden layer delays were of equal values between seven and ten. Thus, the input layer had both a coarse and a fine resolution delay lines. For Motorola, the false alarm rate was always lower when training on the maximum available history data, while for other stocks shorter training sets (300 to 500 days) were necessary to achieve best results. A small penalty factor resulted in a high false alarm rate while a large one resulted in zero profit prediction. For all stocks, the optimum penalty factor was $F = 3$ [See (8)].

For the RNN, the training strategy was similar to the one described above. We however found that an increase in the penalty factor was necessary in order to avoid making an unacceptable number of false predictions (above 10–15%). We typically started with the penalty factor of two gradually increasing it to five, and trained for less than 150 passes.

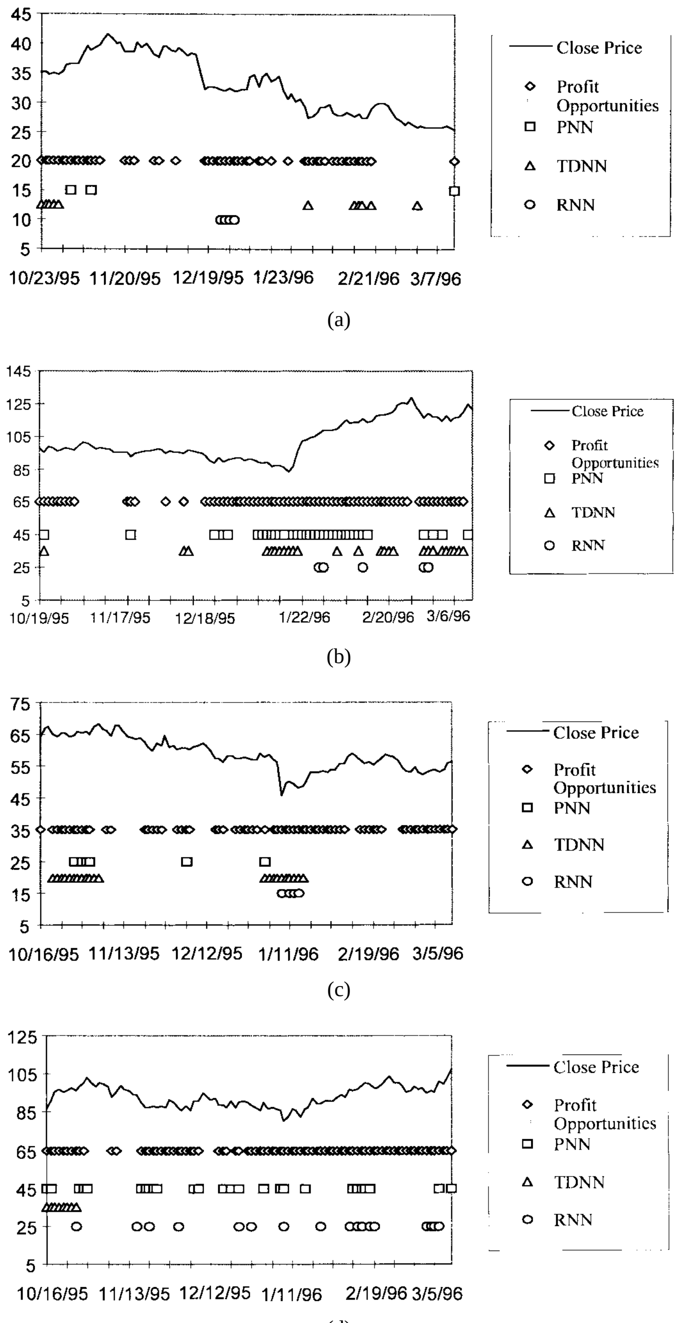


Fig. 6. Actual and predicted profit opportunities for (a) Apple, (b) IBM, (c) Motorola, and (d) Microsoft. The predictions are generally spread over the period, with a low false alarm rate.

The new results in Table V show that the RNN made a small number of predictions comparing with its performance illustrated in the Table IV. It is a manifestation of a higher penalty imposed for false alarms during training.

5) Predicting Higher Trends: In order to test the ability of the neural networks to predict higher percentages of increase, more tests have been performed on data last updated in April 1997. In addition, a wider variety of stocks have been tested. The same approach have been used as in the previous comparative test (see Section VII-B4). The simulation results are summarized in Table VI.

In Table VI, the tests giving zero predictions would give some predictions if the penalizing factor was decreased, but with unacceptably high false alarm rate (50% or more). Generally, the prediction of 5% upward trend is possible with acceptable false alarm rate. Predicting 10% upward usually leads to either a significantly higher false alarm rate, or to zero predictions.

Comparing the different stock categories, we think that technology stocks are generally harder to predict due to their higher volatility.

C. Conventional Classifier

The linear classifier was tested on the same 100-day test set used for the neural networks (see Table IV). We found that the shorter training set of 500 points, rather than all the available history data since January 1987, gave the best results summarized in Table VII.

We then performed a linear classifier test with data updated until April 1997 to predict 2, 5, and 10% trend. The best results are shown in Table VIII.

Comparing Tables VI and VIII, we can see that the linear classifier produced higher false alarm rates than the neural net methods, as expected.

VIII. Comparison

A. Network Architecture

The PNN is a four-layer network. The number of neurons in the first hidden layer is equal to the number of training patterns. This means that using 1000 training patterns with 29 inputs each requires 29 000 weights between the input and first hidden layer. While training TDNN or RNN with so many weights would be difficult, training PNN is essentially an allocation of memory. However, testing PNN on every new pattern requires carrying out operations prescribed by (9) with each stored pattern, which may take significantly more time than testing TDNN or RNN.

The TDNN is a moderate-size network, generally of three or four layers. In most of our tests we used a three-layer network with five to ten input nodes, three to seven hidden nodes, and one output node. We used seven to ten delays in the input and hidden layers. The largest network required 841 weights.

The RNN has a compact structure. In all tests we used a three-layer network of one input, one output, and eight fully recurrent hidden neurons. The resulting network had 89 weights.

B. Implementation Complexity and Training Time

Among the three networks, the PNN is the simplest to implement. All connections are in the forward direction. No derivatives are calculated. Therefore it is the fastest network in training. Other architectures exist which similarly have the advantage of fast training [42]–[46]. The weights between the input and pattern units are directly determined by training patterns. The weights of the output neuron are set according to (10). Training time is determined by duration of the leave-one-out test [17] to estimate the best σ, and it is typically much less than training time for either TDNN or RNN.

TABLE V
COMPARISON OF TDNN, PNN, AND RNN BEST RESULTS FOR DATA ENDING MARCH 23, 1996. THE FALSE ALARM RATE IS RELATIVELY LOW FOR ALL OF THEM

Stock	Profit Opportunities	PNN Predictions	PNN %F.A	TDNN Predictions	TDNN %F.A	RNN Predictions	RNN %F.A
Apple	54	3	0	11	9.09	4	0
IBM	72	34	2.94	26	3.8	5	0
Motorola	69	6	0	22	13.6	4	0
Microsoft	83	26	7.69	8	0	17	5.8

TABLE VI
PREDICTING 2, 5, AND 10% COMPARISON FOR DATA ENDING IN APRIL 1997. A LOW FALSE ALARM RATE COULD STILL BE ACHIEVED

Stock	%	Profit Opportunities	PNN Predictions	PNN %F.A	TDNN Predictions	TDNN %F.A	RNN Predictions	RNN %F.A
AAPL	2	62	51	7.84	10	0	16	0
AAPL	5	41	44	22.72	8	12.5	9	11.11
AAPL	10	17	7	14.29	75	25	4	0
IBM	2	72	25	4	9	0	22	0
IBM	5	47	17	11.76	6	16.67	17	11.76
IBM	10	27	9	33	11	27.27	0	0
MOT	2	81	48	18.75	27	0	33	3.03
MOT	5	67	31	16.13	14	0	19	0
MOT	10	50	28	28.57	15	20	1	0
MSFT	2	87	49	4.08	61	0	46	2.17
MSFT	5	75	13	7.69	9	0	9	11.11
MSFT	10	56	6	16.66	0	0	0	0
AXP	2	92	20	0	17	0	49	0
AXP	5	81	47	0	12	0	6	0
AXP	10	57	18	0	11	45	0	0
WFC	2	85	45	4.44	19	0	29	0
WFC	5	60	13	7.69	37	13.5	0	0
WFC	10	14	10	40	0	0	0	0
DIS	2	74	19	0	7	0	48	0
DIS	5	58	11	0	8	0	9	0
DIS	10	28	9	0	4	25	0	0
MCD	2	73	4	0	6	0	53	5.66
MCD	5	32	5	20	8	12.5	25	4
MCD	10	7	0	0	18	66	2	0
PNM	2	50	63	36.5	22	18	35	17.14
PNM	5	24	58	68.96	0	0	6	16.67
PNM	10	0	0	0	0	0	0	0
V.EEG	2	70	14	0	8	12.5	37	8.11
V.EEG	5	60	10	0	8	12.5	18	5.56
V.EEG	10	32	8	0	49	42	13	0

TABLE VII
RESULTS OF THE LINEAR CLASSIFIER TEST ON DATA ENDING ON AUGUST 23, 1995

Stock	No. of profit opportunities	Classifier Predictions	Missed Opportunities	False Alarms	% F.A.
Apple	54	67	17	30	44.78
IBM	72	45	37	10	22.22
Motorola	69	33	43	7	21.21
Microsoft	83	53	42	12	22.64

TABLE VIII
RESULTS OF THE LINEAR CLASSIFIER TEST ON DATA ENDING IN APRIL 1997. MOST OF THE RESULTS ARE WORSE THAN THE NEURAL NETWORKS ONES

Stock	% increase	Training points	No. of profit opportunities	Classifier Predictions	Missed Opportunities	False Alarms	% F.A.
AAPL	2	500	62	82	6	26	31.71
AAPL	5	500	41	96	0	55	57.3
AAPL	10	500	17	89	0	72	80.9
IBM	2	1999	72	24	54	5	20.83
IBM	5	1999	47	11	44	8	72.73
IBM	10	1999	21	1	27	1	100
MOT	2	500	81	87	10	16	18.39
MOT	5	1000	67	29	46	8	27.58
MOT	10	500	50	95	1	46	48.42
MSFT	2	2000	87	17	70	0	0
MSFT	5	1000	75	49	35	9	18.37
MSFT	10	500	56	7	51	2	28.57
AXP	2	500	92	10	82	0	0
AXP	5	1000	81	5	76	0	0
AXP	10	500	57	13	47	3	23:08
WFC	2	500	85	22	66	3	13.64
WFC	5	500	60	22	42	4	18.18
WFC	10	1000	14	8	12	6	75
DIS	2	500	74	2	72	0	0
DIS	5	1000	58	5	53	0	0
DIS	10	1000	28	1	28	1	100
MCD	2	1000	73	32	48	7	21.88
MCD	5	2003	32	27	14	9	33.33
MCD	10	500	7	35	3	31	88.57
PNM	2	500	50	20	42	12	60
PNM	5	500	24	21	19	16	76.19
PNM	10	500	0	41	0	41	100
V.EEG	2	500	70	76	9	15	19.74
V.EEG	5	500	60	76	5	21	27.63
V.EEG	10	500	32	80	1	49	61.25

TDNN training is an iterative process where each cycle consists of one or more forward propagations through the network, and one backpropagation to obtain derivatives of the cost function with respect to the network weights. The weights form a three-dimensional matrix of size equal to the square of the total number of neurons multiplied by the total delay in all layers plus one. The implementation of both forward and backward passes requires careful bookkeeping of indexing.

In order to accelerate the training, it is necessary to use an efficient training method like the conjugate gradient presented above. This requires extra subroutines to calculate the direction of descent and to perform the line minimization. Training is an iterative process. The number of iterations varies from tens to several thousands. Every iteration includes one backpropagation and one or more forward propagations for the line minimization required by the conjugate gradient algorithm. When training in batch mode, we need to repeat the above iterations for all the training patterns in every cycle. The training speed is proportional to the number of training patterns and depends on the network's size. Training time of our experimentations varied between a few minutes and several hours on a Pentium Pro 200 PC.

Implementation complexity of the multistream EKF training of the RNN consists of three main components: truncated backpropagation through time, EKF computations, and the matrix A's inversion [see (14)]. Compared with complexity of the TDNN training, only the last two components matter since backpropagation computes derivatives in both training methods.

Computational complexity of the EKF algorithm scales linearly with the number of data streams used. Also, the matrix A's inversion imposes an additional computational burden proportional to the cube of the number of streams. Nonetheless, we did not find increasing computational complexity a problem while using up to 50 streams. Furthermore, the multistream EKF algorithm spends the same computer processing time as does the conjugate gradient technique for a similar neural network and identical training set on the same workstation.

While TDNN training with the conjugate gradient method invokes the line minimization routine, RNN training utilizes the EKF and matrix inversion routines. Adding to it a nontrivial bookkeeping for recurrent connections, we thus conclude that the overall implementation complexity of the

RNN training is the highest among the training methods discussed in this paper.

C. Summary and Extension

The three networks showed comparable results. It was possible to control the false alarm rate and even reduce it to zero, in the case of PNN through the loss factor, and in the case of TDNN and RNN through the penalty factor. Although this generally reduces the number of the network predictions to about 10% of the total number of profit opportunities, this can be acceptable while seeking conservative predictions.

One obvious application of our strategy is in options trading. Considering a typical example like IBM stock, we made the TDNN predict profit opportunities for a period of 100 days ending April 21, 1997, and we picked up a typical day when the network predicted a 2% increase like April 7, 1997, when the stock price was $129.125. Following a two-month holding period strategy, a January 1998, $110 option (in the money) could be bought at the network "buy" signal for $26.75 and sold on May 21, 1997 for $69. In the case of trading on 1000 shares and assuming $50 commission each way, the profit would be $42.15/share, which is equivalent to 157.57%. Or a January 1998 $130 option (at the money) could be bought for $15.375 and sold on May 21, 1997 for $51. In this case the profit would be $35.525/share, which is equivalent to 231.06%. If we consider an out of the money option like January 1998 $140, it could be bought for $11 and sold on May 21, 1997 for $43, making a profit of $31.9/share, equivalent to 290%. On any selling day, not being the day of peak price, these profits would be typical ones. Of course, any false alarms could incur losses of up 100%. Therefore, the aggressive trading strategy outlined above really does need our emphasis on eliminating them.

IX. Conclusion

Predicting short term stock trends based on history of daily closing prices is possible using any of the three different networks discussed here. Preliminary predictability analysis and careful neuroengineering are important premises for successful performance. The correlation coefficient calculated using (26) is useful in estimating the delay between the TDNN inputs. A small Lyapunov exponent indicates either a random or cyclic behavior of the stock. In the later case, it was noticed that a relatively short training set should be used.

TDNN is moderate with respect to implementation complexity and memory requirement.

PNN has advantages of extreme implementation simplicity and low false alarm rate even for stocks with low predictability. Its minor disadvantages are the storage requirement of every training pattern and the increased testing time. PNN is more suitable for stocks which do not need training on long history, like Apple stock. This can be determined using a validation set.

RNN has the capability to dynamically incorporate past experience due to internal recurrence, and it is the most powerful network among the three in this respect. Like TDNN, RNN does not need large memory storage, but its minor disadvantage is the implementation complexity—a one-time task.

Acknowledgment

The authors gratefully acknowledge the help of the undergraduate students: E. Stevens, R. Montgomery, J. Casall, J. Escabi, C. Hoffman, A. Davis, and J. Haggard in data collection and running simulations. They would also like to thank the reviewers for their valuable comments.

References

[1] H. Tan, D. Prokhorov, and D. Wunsch, "Probabilistic and time-delay neural-network techniques for conservative short-term stock trend prediction," in *Proc. World Congr. Neural Networks,* Washington, D.C., July 1995.

[2] W. Kreesuradej, D. Wunsch, and M. Lane, "Time-delay neural network for small time series data sets," in *Proc. World Congr. Neural Networks,* San Diego, CA, June 1994.

[3] E. Saad, D. Prokhorov, and D. Wunsch, "Advanced neural-network training methods for low false alarm stock trend prediction," in *Proc. IEEE Int. Conf. Neural Networks,* Washington, D.C., June 1996, pp. 2021–2026.

[4] E. Wan, "Time series prediction by using a connectionist network with internal delay lines," in *Time Series Prediction: Forecasting the Future and Understanding the Past,* A. S. Weigend and N. A. Gershenfeld, Eds. Reading, MA: Addison-Wesley, 1993, pp. 195–217.

[5] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Englewood Cliffs, NJ: Prentice-Hall, 1998.

[6] K. Lang and G. Hinton, "A time-delay neural-network architecture for speech recognition," Carnegie-Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-88-152, 1988.

[7] A. Waibel, "Modular construction of time-delay neural networks for speech recognition," *Neural Comput.,* vol. 1, no. 1, pp. 39–46, 1989.

[8] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Accoust., Speech, Signal Processing,* vol. 37, no. 3, pp. 328–339, 1989.

[9] W. Press *et al.*, *Numerical Recipes in C, The Art of Scientific Computing,* 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1994.

[10] J. Hornell, "Industrial experiences with neural networks," in *Proc. 6th Oklahoma Symp. Artificial Intell.,* Tulsa, OK, Nov. 11–12, 1992, pp. 127–133.

[11] P. Vauder, "Minimization methods for training feedforward neural networks," *Neural Networks,* vol. 7, no. 1, 1994.

[12] A. Gorban, "Training neural networks," *Paragraph,* 1990, in Russian.

[13] R. Fletcher and C. Reeves, "Function minimization by conjugate gradients," *Comput. J.,* pp. 149–154, 1964.

[14] E. Polak and G. Ribiére, "Note sur la Convergence de Methods de Directions Conjugués," *Revue Francáise Informat. Recherche Operationnelle,* vol. 16, pp. 35–43, 1969.

[15] M. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *J. Res. Nat. Bur. Standards,* sec. B, vol. 48, pp. 409–436, 1952.

[16] D. Shanno, "Conjugate gradient methods with inexact searches," *Math. Operations Res.,* vol. 3, no. 3, pp. 244–256, 1978.

[17] D. Specht, "Probabilistic neural networks," *Neural Networks,* vol. 3, pp. 109–118, 1990.

[18] K. Fukunaga, *Introduction to Statistical Pattern Recognition,* 2nd ed. San Diego, CA: Academic, 1990.

[19] D. Specht, "Probabilistic neural networks and general regression neural networks," *Fuzzy Logic and Neural Network Handbook,* C. Chen, Ed. New York: McGraw-Hill, 1996.

[20] M. Musavi, K. Kalantri, W. Ahmed, and K. Chan, "A minimum error neural network," *Neural Networks,* vol. 6, pp. 397–407, 1993.

[21] H. Tan, "Neural-network model for stock forecasting," M.S.E.E. thesis, Texas Tech Univ., 1995.

[22] M. Jurik, "The care and feeding of a neural network," *Futures,* vol. 21, pp. 40–42, Oct. 1992, Cedar Falls, IA.

[23] D. Hush and B. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Mag.,* Jan. 1993.

[24] G. Puskorius, L. Feldkamp, and L. Davis, "Dynamic neural-network methods applied to on-vehicle idle speed control," *Proc. IEEE,* vol. 84, no. 10, pp. 1407–1420, 1996.

[25] P. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE,* vol. 78, no. 10, pp. 1550–1560, 1990.

[26] R. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, Architecture, and Applications,* M. Chauvin and D. E. Rumelhart, Eds. LEA, 1995, ch. 13.

[27] L. Feldkamp and G. Puskorius, “Training controllers for robustness: Multistream DEKF,” in *Proc. World Congr. Comput. Intell.,* Orlando, FL, June/July 1994, pp. 2377–2382.
[28] K. Marko, J. James, T. Feldkamp, G. Puskorius, L. Feldkamp, and D. Prokhorov, “Training recurrent networks for classification: Realization of automotive engine diagnostics,” in *Proc. World Congr. Neural Networks (WCNN),* San Diego, CA, Sept. 1996, pp. 845–850.
[29] A. Petrosian, R. Homan, D. Prokhorov, and D. Wunsch, “Classification of epileptic EEG using recurrent neural networks with wavelet filtering,” in *Proc. SPIE Conf. Signal and Image Processing,* Denver, CO, Aug. 1996, vol. 2825, pp. 834–843.
[30] A. Fisher, *The Mathematical Theory of Probabilities.* New York: McMillan, 1923, vol. 1.
[31] C. Klimasauskas, “Basics of building market timing systems: Making money with neural networks,” *Tutorial at IEEE World Congr. Comput. Intell.,* Orlando, FL, 1994.
[32] R. Hilborn, *Chaos and Nonlinear Dynamics: An Introduction for Scientists and Engineers.* New York: Oxford Univ. Press, 1994.
[33] F. Takens, “Detecting strange attractors in turbulence,” in *Lecture Notes in Mathematics.* Berlin, Germany: Springer-Verlag, 1981, vol. 898, p. 366.
[34] H. Abarbanel *et al.*, “The analysis of observed chaotic data in physical systems,” *Rev. Modern Phys.,* vol. 65, no. 4, pp. 1331–1392, Oct. 1993.
[35] A. Fraser and H. Swinney, “Independent coordinates for strange attractors from mutual information,” *Phys. Rev.,* vol. A33, p. 1134, 1986.
[36] P. Grassberger and Procaccia, “Characterization of strange attractors,” *Phys. Rev. Lett.,* vol. 50, pp. 346–349, 1983.
[37] H. Korsch and H. Jodl, *Chaos: A Program Collection for the PC.* Berlin, Germany: Springer-Verlag, 1994.
[38] D. Ruelle, “Deterministic chaos: The science and the fiction,” in *Proc. R. Soc. Lond.,* vol. A427, pp. 241–248, 1990.
[39] K. Koehler, “Forecasting economic time-series with neural networks,” in *Informatica e diritto/ Seminar Proc.,* 1993, pp. 53–73.
[40] N. Gershenfeld and A. Weigend, “The future of time series: Learning and understanding,” in *Time Series Prediction: Forecasting the Future and Understanding the Past,* A. S. Weigend and N. A. Gershenfeld, Eds. Reading, MA: Addison-Wesley, 1993, pp. 1–70.
[41] A. Weigend, B. Huberman, and D. Rumelhart, “Predicting the future: A connectionist approach,” in *Int. J. Neural Syst.,* vol. 1, pp. 193–209, 1990.
[42] O. Bashkirov, E. Braverman, and I. Muchnik, “Potential function algorithms for pattern recognition learning machines,” *Automat. Remote Contr.,* vol. 25, pp. 692–695, 1964.
[43] T. Poggio and F. Girosi, “A theory of networks for approximation and learning,” Artificial Intell. Lab., Massachusetts Inst. Technol., Cambridge, MA, A.I. Memo 1140, July 1989.
[44] J. Moody and C. Darken, “Fast learning in networks of locally tuned processing units,” *Neural Comput.,* vol. 1, pp. 281–294, 1989.
[45] E. Kussul, L. Kasatkina, D. Rachkovskij, and D. Wunsch, “Application of random threshold neural networks for diagnostics of micro machine tool condition,” in *Proc. IEEE Int. Conf. Neural Networks,* Anchorage, AK, May 1998.
[46] G. Carpenter, S. Grossberg, and J. Reynolds, “ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network,” *Neural Networks,* vol. 4, no. 5, pp. 565–588, 1991.
[47] V. Martin and K. Sawyer, “Statistical techniques for modeling nonlinearities,” *Chaos and Nonlinear Models in Economics,* J. Creedy and V. Martin, Eds. Edward Elgar, 1994.
[48] P. Grassberger, T. Schreiber, and C. Schaffrath, “Nonlinear time sequence analysis,” *Int. J. Biffurcation and Chaos,* vol. 1, no. 3, pp. 521–547, 1991.
[49] K. Bergerson and D. Wunsch, “A commodity trading model based on a neural network-expert system hybrid,” in *Proc. IEEE Int. Conf. Neural Networks,* Seattle, WA, 1991, pp. 1289–1293.
[50] P. Grassberger, T. Schreiber, and S. Carsten, “Nonlinear time sequence analysis,” *Int. J. Biffurcation and Chaos,* vol. 1, pp. 521–547, 1991.
[51] A. Ansari and F. Lin, “Forecasting financial trends in stock markets with neural networks,” in *Proc. 3rd Int. Conf. Syst. Integration,* São Paulo, Brazil, 1996, pp. 188–196.
[52] J. Wang and J. Leu, “Stock trend prediction using ARIMA-based neural networks,” in *Proc. 1996 IEEE Int. Conf. Neural Networks,* 1996, pp. 2160–2165.
[53] D. Wood and B. Dasgupta, “Classifying trend movements in the MSCI USA capital market index—A comparison of regression, ARIMA and neural-network methods,” *Comput. Operations Res.,* vol. 23, no. 6, p. 611, 1996.
[54] J. Tsukuda and S. Baba, “Predicting Japanese corporate bankruptcy in terms of financial data using neural network,” *Comput. Ind. Eng.,* vol. 27, nos. 1–4, p. 445, Sept. 1994.
[55] P. Coats and L. Fant, “Recognizing distress patterns using a neural-network tool,” *Financial Management,* vol. 22, no. 3, pp. 142–145, Fall 1993.
[56] E. Azoff and N. Meade, “Neural-network time series forecasting of financial markets,” *Int. J. Forecasting,* vol. 11, no. 4, p. 601, 1995.
[57] I. Kaastra and M. Boyd, “Designing a neural network for forecasting financial and economic time series,” *Neurocomputing,* vol. 10, no. 3, p. 215, 1996.
[58] J. Utans, J. Moody, S. Rehfuss, and H. Siegelmann, “Input variable selection for neural networks: Application to predicting the U.S. business cycle,” in *Conf. Comput. Intell. Financial Eng.,* New York, Apr. 1995, pp. 118–122.
[59] S. Piché, “Trend visualization,” in *Conf. Comput. Intell. Financial Eng.,* New York, Apr. 1995, pp. 146–150.
[60] R. Yager, “Multicriteria decision making using fuzzy quantifiers,” in *Conf. Comput. Intell. Financial Eng.,* New York, Apr. 1995, pp. 42–46.
[61] A.-P. Refenes, A. Zapranis, and G. Francis, “Stock performance modeling using neural networks: A comparative study with regression models,” *Neural Networks,* vol. 7, no. 2, pp. 375–388, 1994.
[62] E. Schoneburg, “Stock price prediction using neural networks: A project report,” *Neurocomputing,* vol. 2, pp. 17–17, 1990.
[63] F. Wong and P. Wang, “A stock selection strategy using fuzzy neural networks,” in *Neurocomputing,* vol. 2. Elsevier, 1990/1991, pp. 233–242.
[64] H. Ahmadi, “Testability of the arbitrage pricing theory by neural networks,” in *Proc. IEEE Int. Conf. Neural Networks,* 1990.
[65] S. Dutta and S. Shekhar, “Bond rating: A nonconservative application of neural networks,” in *Proc. IEEE Int. Conf. Neural Networks,* 1988, vol. 2, pp. 443–450.
[66] S. Ghosh and C. Scotfield, “An application of a multiple neural-network learning system to emulation of mortgage underwriting judgments,” in *Proc. IEEE Conf. Neural Networks,* 1988.
[67] C. Haefke and C. Helmenstein, “A neural-network model to exploit the econometric properties of Austrian IPO’s,” in *Proc. Comput. Intell. Financial Eng. (CIFEr),* 1995, pp. 128–135.
[68] A. J. Surkan and J. C. Singleton, “Neural networks for bond rating improved by multiple hidden layers,” in *Proc. IEEE Int. Conf. Neural Networks,* 1990.
[69] D. L. Reilly, E. Collins, C. Scofield, and S. Ghosh, “Risk assessment of mortgage applications with a neural-network system: An update as the test portfolio ages,” in *Proc. IEEE Int. Conf. Neural Networks,* vol. II, pp. 479–482.
[70] J. Utans, J. Moody, and S. Rehfus, “Input variable selection for neural networks: Application to predicting the U.S. business cycle,” in *Proc. IEEE/IAFE Conf. Comput. Intell. Financial Eng. (CIFEr),* New York, NY, Apr. 9–11, 1995.
[71] H. White, “Economic prediction using neural networks: The case of IBM daily stock returns,” in *Proc. IEEE Int. Conf. Neural Networks,* 1988.
[72] R. Trippi and E. Turban, *Neural Networks in Finance and Investing.* New York: Irwin, 1996.
[73] *Proceedings of the Neural Networks in the Capital Markets Conference,* Abu-Mostafa, Ed. Singapore: World Scientific, 1996.
[74] A. Refenes, A. Burgess, and Y. Bentz, “Neural networks in financial engineering: A study in methodology,” *IEEE Trans. Neural Networks,* vol. 8, pp. 1222–1267, 1997.

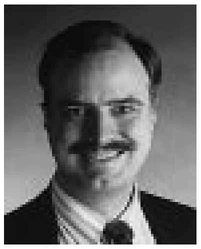
Emad W. Saad (S’95) received the B.S. degree with Honors from Ain Shams University, Cairo, Egypt, in 1992. He later joined the Applied Computational Intelligence Laboratory at Texas Tech University, Lubbock, TX, where he received the M.S. degree in electrical engineering in 1996. Currently he is pursuing the Ph.D. degree in electrical engineering.

He worked for two years as an Electronic Engineer for the ELSAFE. In the summers of 1997 and 1998, he worked at The Boeing Company, Seattle, WA, under a contract with Texas Tech University. He was working on neural networks with query-based learning and computational applications for aerospace control problems. His research interests include financial engineering, time series prediction, explanation capability of neural networks, query-based learning, adaptive critic designs, and neurocontrol.

Danil V. Prokhorov (S'95–M'97) received the M.S. degree from the State Academy of Aerospace Instrument Engineering, St. Petersburg, Russia, in 1992, and the Ph.D. degree in electrical engineering from Texas Tech University, Lubbock, in 1997.

Prior to joining Texas Tech in 1994, he worked at the Institute for Informatics and Automation of Russian Academy of Sciences, St. Petersburg, as a Research Engineer. He also worked in the Vehicle Electronic Systems Department of Ford Research Laboratory, Dearborn, MI, as a Summer Intern in 1995 to 1997. He is currently with the group of Artificial Neural Systems and Fuzzy Logic of Ford Research Laboratory. His research interests include adaptive critics, signal processing, system identification, control, and optimization-based on various neural network approaches.

Dr. Prokhorov is a member of the International Neural Network Society.



Donald C. Wunsch, II (SM'94) completed a Humanities Honors Program at Seattle University in 1981, received the B.S. degree in applied mathematics from the University of New Mexico, Albuquerque, in 1984, the M.S. degree in applied mathematics in 1987 and the Ph.D. degree in electrical engineering and 1991, both from the University of Washington, Seattle.

He was Senior Principal Scientist at Boeing, where he invented the first optical implementation of the ART1 neural network, featured in the 1991 Boeing Annual Report, and other optical neural networks and applied research contributions. He has also worked for International Laser Systems and Rockwell International. He is currently Associate Professor of Electrical Engineering and Director of the Applied Computational Intelligence Laboratory at Texas Tech University, involving eight other faculty, several postdoctoral associates, doctoral candidates, and other graduate and undergraduate students. His research activities include adaptive critic designs, neural optimization, forecasting and control financial engineering, fuzzy risk assessment for high-consequence surety, wind engineering, characterization of the cotton manufacturing process intelligent agents, and Go. He is heavily involved in research collaboration with former Soviet scientists.

Dr. Wunsch is a current NSF Career Award recipient. He is also an Academician in the International Academy of Technological Cybernetics, and the International Informatization Academy; and is recipient of the Halliburton Award for excellence in teaching and research at Texas Tech. He is a member of the International Neural Network Society, Association for Computing Machinery, a life member of the American Association of Artificial Intelligence, and a past member of the IEEE Neural Network Council. He is an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS.