Nonlinear Dynamics and Systems Theory, **7** (1) (2007) 97–112

An Online Learning Algorithm with Adaptive Forgetting Factors for Feedforward Neural Networks in Financial Time Series Forecasting

Lean Yu [ac], Shouyang Wang [ab] and Kin Keung Lai [bc*]

[a] *Institute of Systems Science, Academy of Mathematics and Systems Sciences, Chinese Academy of Sciences, Beijing 100080, China*
[b] *College of Business Administration, Hunan University, Changsha 410082, China*
[c] *Department of Management Sciences, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong*

Received: July 15, 2005; Revised: September 11, 2006

Abstract: In this study, an online learning algorithm for feedforward neural networks (FNN) based on the optimized learning rate and adaptive forgetting factor is proposed for online financial time series prediction. The new learning algorithm is developed for online predictions in terms of the gradient descent technique, and can speed up the FNN learning process substantially relative to the standard FNN algorithm, with simultaneous preservation of stability of the learning process. In order to verify the effectiveness and efficiency of the proposed online learning algorithm, two typical financial time series are chosen as testing targets for illustration purposes.

Keywords: *Online learning algorithm; adaptive forgetting factor; optimal learning rate; feedforward neural network; financial time series forecasting.*

Mathematics Subject Classification (2000): 93C55, 92B20, 91B99.

1 Introduction

The financial market is a complex, evolutionary, and nonlinear dynamical system [1]. Financial time series are inherently noisy, non-stationary, and deterministically chaotic [2]. This means that the distribution of financial time series changes over time. Not only a single data series is non-stationary in the sense of the mean and variance of the series, but the relationship of the data series to other related data series may also be continuously changing. Modeling such dynamical and non-stationary time series is a

[*] Corresponding author: mskklai@cityu.edu.hk

challenging task. Over the past few years, neural networks have been successfully used to model financial time series ranging from options prices [3], corporate bond ratings [4] and stock index trading [5] to currency exchange [6]. Neural networks are universal function approximators that can map any nonlinear function without any *a priori* assumption about the data [7]. Unlike traditional statistical models, neural networks are data-driven, non-parametric weak models, and they let "the data speak for themselves". So neural networks are less susceptible to the model mis-specification problem than most parametric models are, and they are more powerful in describing the dynamics of financial time series than traditional statistical models are [6, 8].

Among these neural network models, the multilayer feedforward neural network (FNN) is widely used for financial time series prediction due to its approximations to nonlinear functions and its self-learning capability [7]. However, the FNN has several limitations. For example, the convergence speed of the FNN algorithm is often slow because the learning rate is fixed [12], thus increasing the network learning time. Therefore, some faster training FNN algorithms, such as adaptive learning algorithms [9-10], real-time learning algorithms [11-12] and other fast learning algorithms [13-15], have been developed in an attempt to reduce these shortcomings. But two main limitations still remain so far.

Firstly, most FNN models do not use the optimized instantaneous learning rates except the work of [11]. In studies in which these are introduced, the learning rate is set to a fixed value. It is, however, critical to determine a proper fixed learning rate for the FNN applications. If the learning rate is large, learning may occur quickly, but it may also become unstable and may even not learn at all [11]. To ensure stable learning, the learning rate must be sufficiently small, but a small learning rate may lead to a long learning time and a slow convergence speed. Also, it is unclear just how small the learning rate should be. In addition, the best fixed learning rate is problem-independent, and it varies with different neural network structure for different applications.

Secondly, in the existing literature, almost all fast algorithms are batch learning algorithms. Although neural network batch learning is highly effective, the computation involved in each learning step is very big, especially when large sample data sets are presented. Furthermore, the neural networks must re-learn from the beginning as new data become available. Therefore, this may overly affect the overall computational efficiency of batch learning. In this sense, batch learning is unsuitable for real-time or online prediction when neural networks are used as a predictor.

For the first problem, an optimized instantaneous learning rate is derived from the gradient descent rule based on optimization techniques. For the second problem, an online learning algorithm should be created to overcome the drawbacks of batch learning algorithm. Actually, there is a difference between online learning algorithm and batch learning algorithm in the neural networks models. In the online learning algorithm, the weight vectors are updated recursively after the presentation of each input vector. While in the batch learning algorithm, the weight vectors of neural networks are updated only at the end of each epoch, which will be further illustrated later. Usually, in the neural networks, a single pass over the input data set is called as an epoch. Furthermore, the weight sequence should be chosen to given a higher weight to more recent data in the time series prediction. So an adaptive forgetting factor is also introduced into the proposed online learning algorithm. In order to verify the effectiveness and efficiency of the proposed online learning algorithm, two typical financial time series, S&P 500 and the exchange rate of euros against US dollars (EUR/USD), are chosen for testing.

The rest of this work is organized as follows. In Section 2, the proposed online learning algorithm with adaptive forgetting factor is first presented in terms of the gradient descent algorithm and optimization techniques. For further illustration, an empirical analysis is then given in Section 3. Finally, some concluding remarks are drawn in Section 4.

2 The Proposed Online Learning Algorithm

In this study, we use a matrix-vector notation of the neural network description in order to be able to express the later by simple formula. Consider a three-layer FNN, which has p nodes in the input layer, q nodes in the hidden layer and k nodes in the output layer. Mathematically, the basic structure of the FNN model is described by

$$
\begin{aligned}
Y(t+1) &= \begin{bmatrix} y_1(t+1) \\ y_2(t+1) \\ \cdots \\ y_k(t+1) \end{bmatrix} = \begin{bmatrix} f_2[\sum_{i=1}^{q} f_1(\sum_{j=1}^{p} w_{ij}(t)x_j(t) + w_{i0}(t))v_{1i}(t) + v_{10}(t)] \\ f_2[\sum_{i=1}^{q} f_1(\sum_{j=1}^{p} w_{ij}(t)x_j(t) + w_{i0}(t))v_{2i}(t) + v_{20}(t)] \\ \cdots \\ f_2[\sum_{i=1}^{q} f_1(\sum_{j=1}^{p} w_{ij}(t)x_j(t) + w_{i0}(t))v_{ki}(t) + v_{k0}(t)] \end{bmatrix} \\
&= \begin{bmatrix} f_2[\sum_{i=0}^{q} f_1(\sum_{j=0}^{p} w_{ij}(t)x_j(t))v_{1i}(t)] \\ f_2[\sum_{i=0}^{q} f_1(\sum_{j=0}^{p} w_{ij}(t)x_j(t))v_{2i}(t)] \\ \cdots \\ f_2[\sum_{i=0}^{q} f_1(\sum_{j=0}^{p} w_{ij}(t)x_j(t))v_{ki}(t)] \end{bmatrix} = \begin{bmatrix} f_2[V_1^T F_1(W(t)X(t))] \\ f_2[V_2^T F_1(W(t)X(t))] \\ \cdots \\ f_2[V_k^T F_1(W(t)X(t))] \end{bmatrix} \\
&= F_2[V^T(t)F_1(W(t)X(t))],
\end{aligned}
\tag{1}
$$

where $x_j(t)$, $j = 1, 2, \ldots, p$, are the inputs of the FNN; $y_j(t{+}1)$, $j = 1, 2, \ldots, k$, are the output of the FNN; $w_{ij}(t)$, $i = 1, 2, \ldots, q$, $j = 1, 2, \ldots, p$, are the weights from the input layer to the hidden layer; $w_{i0}(t)$, $i = 1, 2, \ldots, q$, are the biases of the hidden nodes; $v_{ij}(t)$, $i = 1, \ldots, q$, $j = 1, \ldots, k$, are the weights from the hidden layer to the output layer; $v_{i0}(t)$, $i = 1, \ldots, k$, are the bias of the output node; t is a time factor; f_1 is the activation function of the nodes for the hidden layer and f_2 is the activation function of the nodes for the output layer. Generally, the activation function for nonlinear nodes is assumed to be a symmetric hyperbolic tangent function, i.e. $f_1(x) = \tanh(u_0^{-1}x)$, and its first- and second-order derivatives are $f_1'(x) = u_0^{-1}[1 - f_1^2(x)]$, $f_1''(x) = -2u_0^{-1}f_1(x)[1 - f_1^2(x)]$, respectively, where u_0 is the shape factor of the activation function. Specially, some notations in Equation (1) are defined as follows:

$$X = (x_0, x_1, \cdots, x_p)^T \in R^{(p+1)\times 1}, \quad Y = (y_1, y_2, \cdots, y_k)^T \in R^{k\times 1},$$

$$W = \begin{pmatrix} w_{10} & w_{11} & \cdots & w_{1p} \\ w_{20} & w_{21} & \cdots & w_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ w_{q0} & w_{q1} & \cdots & w_{qp} \end{pmatrix} = (W_0, W_1, \cdots, W_p) \in R^{q\times(p+1)},$$

$$V = \begin{pmatrix} v_{10} & v_{20} & \cdots & v_{k0} \\ v_{11} & v_{21} & \cdots & v_{k1} \\ \cdots & \cdots & \cdots & \cdots \\ v_{1q} & v_{2q} & \cdots & v_{kq} \end{pmatrix} = (V_1, V_2, \cdots, V_k) \in R^{(q+1)\times k},$$

$$F_1(W(t)X(t)) \\ = \begin{pmatrix} F_1(\sum_{j=1}^p w_{0j}(t)x_j(t)) & F_1(\sum_{j=1}^p w_{1j}(t)x_j(t)) & \cdots & F_1(\sum_{j=1}^p w_{qj}(t)x_j(t)) \end{pmatrix}^T.$$

$$F_1(W(t)X(t)) \in R^{(q+1)\times 1}.$$

For simplification, let $net_i(t) = \sum_{j=0}^p w_{ij}(t)x_j(t), \quad i = 0, 1, \ldots, q$, then

$$F_1(W(t)X(t)) = \begin{pmatrix} F_1(net_0(t)) & F_1(net_1(t)) & \cdots & F_1(net_q(t)) \end{pmatrix}^T \in R^{(q+1)\times 1}.$$

Usually, through estimating the model parameter vectors (W, V) via FNN learning, we can realize the corresponding tasks such as function approximation, system identification or prediction. In fact, the model parameter vectors (W, V) can be obtained by iteratively minimizing a cost function $E(X\colon W, V)$. In general, $E(X{:}W, V)$ is a sum of the error squares cost function with k output nodes and N training pairs, i.e.,

$$E(X : W, V) = \frac{1}{2}\sum_{j=1}^N \sum_{i=1}^p e_i^2(j) = \frac{1}{2}\sum_{j=1}^N e^T(j)e(j)$$
$$= \frac{1}{2}\sum_{j=1}^N [y_j - \hat{y}_j(X : W, V)]^T [y_j - \hat{y}_j(X : W, V)], \tag{2}$$

where $e(j) = [e_1(j), e_2(j), \cdots, e_k(j)]^T \in R^{k\times 1}$, y_j is the jth actual value and $\hat{y}_j(X : W, V)$ is the jth estimated value, $j = 1, \ldots, N$.

However, the learning algorithm based on Equation (2) is batch learning of neural networks. As earlier mentioned, the computation of the batch learning algorithm is very large if big sample data sets are given. Also, the neural networks must re-learn when new data are available. To overcome the shortcomings, the neural network learning should be iterative or recursive, allowing the network parameters to be updated at each sample interval as new data become available. This idea will be activated to create a new online learning algorithm. In addition, a weighting sequence should be chosen to give a higher weight for more recent data in order to perform online prediction. To arrive at this goal, an adaptive forgetting factor is introduced to this problem. In this study, an exponential forgetting mechanism in the cost function, like a recursive algorithm with the forgetting factor, is used, and then Equation (2) can be rewritten as

$$E(t) = \frac{1}{2}\sum_{j=1}^t \lambda^{t-j} \sum_{i=1}^k e_i^2(j) = \frac{1}{2}\sum_{j=1}^t \lambda^{t-j} e^T(j)e(j)$$
$$= \frac{1}{2}\sum_{j=1}^t \lambda^{t-j} [y(j) - \hat{y}_i(j)]^T [y(j) - \hat{y}_i(j)], \tag{3}$$

where λ is the forgetting factor, $0 < \lambda \leqslant 1$, $e(j) = [e_1(j), e_2(j), \cdots, e_k(j)]^T \in R^{k\times 1}$, $j = 1, \ldots, t$; t is a time factor, representing the number of training pairs here.

By applying the steepest descent method to the error cost function $E(t)$ (i.e., Equation (3)), we can obtain the gradient of $E(t)$ with respect to parameters W and V, respectively.

$$\nabla_W E(t) = \frac{\partial E(t)}{\partial W(t)} = \sum_{j=1}^t \lambda^{t-j} \sum_{i=1}^k e_i(j)\frac{\partial e_i(j)}{\partial W(j)} = -\sum_{j=1}^t \lambda^{t-j} \sum_{i=1}^k e_i(j)\frac{\partial \hat{y}_i(j)}{\partial W(j)}$$

$$= -\sum_{j=1}^{t} \lambda^{t-j} \bar{F}_1'(j)\bar{V}(j)F_2'(j)e(j)x^T(j) = \lambda\nabla_W E(t-1) - \bar{F}_1'(t)\bar{V}(t)F_2'(t)e(t)x^T(t), \tag{4}$$

$$\nabla_V E(t) = \frac{\partial E(t)}{\partial V(t)} = \sum_{j=1}^{t} \lambda^{t-j} \sum_{i=1}^{k} e_i(j)\frac{\partial e_i(j)}{\partial V(j)} = -\sum_{j=1}^{t} \lambda^{t-j} \sum_{i=1}^{k} e_i(j)\frac{\partial \hat{y}_i(j)}{\partial V(j)}$$

$$= -\sum_{j=1}^{t} \lambda^{t-j} F_1(j)e^T(j)F_2'(j) = \lambda\nabla_V E(t-1) - F_1(t)e^T(t)F_2'(t). \tag{5}$$

So, the updated formulae of weights are given by, respectively

$$\Delta W(t) = -\eta\nabla_W E(t) = -\eta\left(\lambda\nabla_W E(t-1) - \bar{F}_1'(t)\bar{V}(t)F_2'(t)e(t)x^T(t)\right), \tag{6}$$

$$\Delta V(t) = -\eta\nabla_V E(t) = -\eta\left(\lambda\nabla_V E(t-1) - F_1(t)e^T(t)F_2'(t)\right), \tag{7}$$

where η is the learning rate; λ is the forgetting factor; Δ is the incremental operator; ∇ is the gradient operator; ΔW and ΔV are the weight adjustment increments;

$$\bar{F}_{1(j)}' = diag[f_{1(1)}' \ f_{1(2)}' \ \cdots \ f_{1(q)}'] \in R^{q\times q};$$

$$F_2' = diag[f_{2(1)}' \ f_{2(2)}' \ \cdots \ f_{2(k)}'] \in R^{k\times k};$$

$$f_{1(i)}' = f_1'(net_i) = \frac{\partial f_1(net_i)}{\partial net_i}, i = 1, 2, \cdots, q;$$

$$f_{2(i)}' = f_2'[v_i^T F_1(WX)] = \frac{\partial f_2[v_i^T F_1(WX)]}{\partial[v_i^T F_1(WX)]}, i = 1, 2, \cdots k;$$

$$\bar{V} = \begin{bmatrix} v_{11} & v_{21} & \cdots & v_{k1} \\ v_{12} & v_{22} & \cdots & v_{k2} \\ \cdots & \cdots & \cdots & \cdots \\ v_{1q} & v_{2q} & \cdots & v_{kq} \end{bmatrix} = [\bar{v}_1 \ \bar{v}_2 \ \cdots \ \bar{v}_k] \in R^{q\times p};$$

$$\bar{v}_i = [v_{i1} \ \cdots \ v_{iq}]^T \in R^{q\times 1}, i = 1, 2, \cdots, q.$$

To derive the optimal learning rate, consider the following error increment equation:

$$\Delta e(t+1) = e(t+1) - e(t) = y(t+1) - \hat{y}(t+1) - y(t) + \hat{y}(t). \tag{8}$$

Let $\Delta y(t+1) = y(t+1) - y(t)$ be the change of the actual series and let $\Delta\hat{y}(t+1) = \hat{y}(t+1) - \hat{y}(t)$ be the change of the neural network output. Here we assume that the absolute value of the change of the actual series is much smaller than the absolute value of the change of the neural network output, i.e., $|\Delta y(t+1)| << |\Delta\hat{y}(t+1)|$. This implies that the value $y(t)$ can approximate $y(t{+}1)$ locally during the training process, that is to say, the change of the actual series can be ignored comparing with the change of neural network output during the learning process. This assumption is realistic for many processes of actual series due to energy constraints in practical systems, while no constraints are imposed to the output of the neural networks [11]. Also, if this condition is not satisfied, then the neural network prediction system will not be able to adapt sufficiently fast to change in the actual series and the prediction of the actual series will be impossible. With the above assumption, the increment in Equation (8) can be approximated as

$$\Delta e(t+1) = e(t+1) - e(t) = \Delta y(t+1) - \Delta\hat{y}(t+1) \approx -\Delta\hat{y}(t+1). \tag{9}$$

Usually, in the recursive algorithm, the change of output of neural networks with adaptive forgetting factors can be represented as

$$\Delta\hat{y}(t+1) = -\eta\lambda\zeta(t-1) + \eta\xi(t)e(t), \tag{10}$$

where $\zeta(t-1) = F_2'[\nabla_V^T E(t-1)F_1 + \bar{V}^T\bar{F}_1'\nabla_W E(t-1)X]$, $\xi(t) = F_2'[(F_1^T F_1)I_{k^2} + \bar{V}^T F_1' F_1' \bar{V} X^T X]F_2'$ with

$$F_2' = \begin{bmatrix} F'_{2(1)} & 0 & \cdots & 0 \\ 0 & F'_{2(2)} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & F'_{2(N)} \end{bmatrix},$$

$$F_1^T F_1 = \begin{bmatrix} F^T_{1(1)}F_{1(1)} & F^T_{1(1)}F_{1(2)} & \cdots & F^T_{1(1)}F_{1(N)} \\ F^T_{1(2)}F_{1(1)} & F^T_{1(2)}F_{1(2)} & \cdots & F^T_{1(2)}F_{1(N)} \\ \cdots & \cdots & \cdots & \cdots \\ F^T_{1(N)}F_{1(1)} & F^T_{1(N)}F_{1(2)} & \cdots & F^T_{1(N)}F_{1(N)} \end{bmatrix},$$

$$X^T X = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & \cdots & x_1^T x_N \\ x_2^T x_1 & x_2^T x_2 & \cdots & x_2^T x_N \\ \cdots & \cdots & \cdots & \cdots \\ x_N^T x_1 & x_N^T x_2 & \cdots & x_N^T x_N \end{bmatrix},$$

$$F_1' F_1' = \begin{bmatrix} \bar{F}'_{1(1)}\bar{F}'_{1(1)} & \bar{F}'_{1(1)}\bar{F}'_{1(2)} & \cdots & \bar{F}'_{1(1)}\bar{F}'_{1(N)} \\ \bar{F}'_{1(2)}\bar{F}'_{1(1)} & \bar{F}'_{1(2)}\bar{F}'_{1(2)} & \cdots & \bar{F}'_{1(2)}\bar{F}'_{1(N)} \\ \cdots & \cdots & \cdots & \cdots \\ \bar{F}'_{1(N)}\bar{F}'_{1(1)} & \bar{F}'_{1(N)}\bar{F}'_{1(2)} & \cdots & \bar{F}'_{1(N)}\bar{F}'_{1(N)} \end{bmatrix}.$$

In order to prove Equation (10), a lemma must be introduced firstly.

Lemma 2.1 *The total time derivative of the FNN single output $V^T F_1(WX)$is given by*

$$\frac{d[V^T F_1(WX)]}{dt} = F_1(WX)\frac{dV}{dt} + \bar{V}^T\bar{F}'_1(WX)\frac{dW}{dt}X$$

$$= \frac{dV^T}{dt}F_1(WX) + \bar{V}^T\bar{F}'_1(WX)\frac{dW}{dt}X,$$

where $V^T F_1(WX)$ is the single output of FNN; $\frac{dW}{dt}$ and $\frac{dV}{dt}$ are the derivatives with *respect to timet; W, V are the weight vectors; X is the input vector; and*

$$F_1(WX) = [f_1(net_0), f_1(net_1), \cdots, f_1(net_q)]^T; X = [x_0, x_1, \cdots, x_p]^T;$$

$$\bar{F'}_1(WX) = \begin{bmatrix} f'(net_1) & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & f'(net_q) \end{bmatrix}; \quad \frac{dW}{dt} = \begin{bmatrix} \frac{dw_{00}}{dt} & \frac{dw_{01}}{dt} & \cdots & \frac{dw_{0p}}{dt} \\ \frac{dw_{10}}{dt} & \frac{dw_{11}}{dt} & \cdots & \frac{dw_{1p}}{dt} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{dw_{q0}}{dt} & \frac{dw_{q1}}{dt} & \cdots & \frac{dw_{qp}}{dt} \end{bmatrix};$$

$$\bar{V} = [v_1, v_2, \cdots, v_q]^T; \quad \frac{dV}{dt} = \begin{bmatrix} \frac{dv_0}{dt} & \frac{dv_1}{dt} & \cdots & \frac{dv_q}{dt} \end{bmatrix}^T.$$

Proof Derivation of $\frac{d[V^T F_1(WX)]}{dt}$ is as follows:

$$\frac{d[V^T F_1(WX)]}{dt} = \frac{d\left[\sum_{i=0}^{q} v_i f_1\left(\sum_{j=0}^{p} w_{ij}x_j\right)\right]}{dt}$$

$$= \sum_{i=0}^{q} \frac{\partial\left[\sum_{i=0}^{q} v_i f_1\left(\sum_{j=0}^{p} w_{ij}x_j\right)\right]}{\partial v_i}\frac{dv_i}{dt} + \sum_{i=0}^{q}\sum_{j=0}^{p} \frac{\partial\left[\sum_{i=0}^{q} v_i f_1\left(\sum_{j=0}^{p} w_{ij}x_j\right)\right]}{\partial w_{ij}}\frac{dw_{ij}}{dt}$$

$$= \sum_{i=0}^{q} f_1\left(\sum_{j=0}^{p} w_{ij}x_j\right)\frac{dv_i}{dt} + \sum_{i=0}^{q}\sum_{j=0}^{p} v_i f_1'\left(\sum_{j=0}^{p} w_{ij}x_j\right)x_j\frac{dw_{ij}}{dt}$$

$$= f_1(net_0)\frac{dv_0}{dt} + f_1(net_1)\frac{dv_1}{dt} + \cdots + f_1(net_q)\frac{dv_q}{dt}$$

$$+ v_0 f_1'(net_0)[x_0\frac{dw_{00}}{dt} + x_1\frac{dw_{01}}{dt} + \cdots + x_p\frac{dw_{0p}}{dt}]$$

$$+ v_1 f_1'(net_1)[x_0\frac{dw_{10}}{dt} + x_1\frac{dw_{11}}{dt} + \cdots + x_p\frac{dw_{1p}}{dt}] + \cdots$$

$$+ v_q f_1'(net_q)[x_0\frac{dw_{q0}}{dt} + x_1\frac{dw_{q1}}{dt} + \cdots + x_p\frac{dw_{qp}}{dt}]$$

$$= [f_1(net_0)\ f_1(net_1)\ \cdots\ f_1(net_q)]\begin{bmatrix} \frac{dv_0}{dt} & \frac{dv_1}{dt} & \cdots & \frac{dv_q}{dt} \end{bmatrix}^T$$

$$+[v_1\ v_2\ \cdots\ v_q]\begin{bmatrix} f'(net_1) & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & f'(net_q) \end{bmatrix}\begin{pmatrix} \text{due to } f(net_0) \equiv 1, \\ f'(net_0) = 0 \end{pmatrix}$$

$$\times \begin{bmatrix} \frac{dw_{00}}{dt} & \frac{dw_{01}}{dt} & \cdots & \frac{dw_{0p}}{dt} \\ \frac{dw_{10}}{dt} & \frac{dw_{11}}{dt} & \cdots & \frac{dw_{1p}}{dt} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{dw_{q0}}{dt} & \frac{dw_{q1}}{dt} & \cdots & \frac{dw_{qp}}{dt} \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_p \end{bmatrix}$$

$$= F_1^T(WX)\frac{dV}{dt} + \bar{V}^T\bar{F'}_1(WX)\frac{dW}{dt}X = \frac{dV^T}{dt}F_1(WX) + \bar{V}^T\bar{F'}_1(WX)\frac{dW}{dt}X.$$

□

In the following, we start to prove Equation (10). The above Lemma together with Equations (6) and (7) gives

$$\Delta\hat{y}(t{+}1) \approx \left(\tfrac{d\hat{y}(t+1)}{dt}\right)\Delta t = \left(\begin{array}{c} \frac{d\hat{y}_1(t+1)}{dt} \\ \frac{d\hat{y}_2(t+1)}{dt} \\ \cdots \\ \frac{d\hat{y}_k(t+1)}{dt} \end{array}\right)\Delta t = \left(\begin{array}{l} f'_{2(1)}\cdot\left(F_1^T\frac{dv_1}{dt}+v_1^T\bar{F}'_1\frac{dW}{dt}X\right) \\ f'_{2(2)}\cdot\left(F_1^T\frac{dv_2}{dt}+v_2^T\bar{F}'_1\frac{dW}{dt}X\right) \\ \cdots \\ f'_{2(k)}\cdot\left(F_1^T\frac{dv_k}{dt}+v_k^T\bar{F}'_1\frac{dW}{dt}X\right) \end{array}\right)\Delta t$$

$$\approx \left(\begin{array}{l} f'_{2(1)}\cdot\left(F_1^T\frac{\Delta v_1}{\Delta t}+v_1^T\bar{F}'_1\frac{\Delta W}{\Delta t}X\right) \\ f'_{2(2)}\cdot\left(F_1^T\frac{\Delta v_2}{\Delta t}+v_2^T\bar{F}'_1\frac{\Delta W}{\Delta t}X\right) \\ \cdots \\ f'_{2(k)}\cdot\left(F_1^T\frac{\Delta v_k}{\Delta t}+v_k^T\bar{F}'_1\frac{\Delta W}{\Delta t}X\right) \end{array}\right)\Delta t = \left[\begin{array}{l} f'_{2(1)}\cdot(F_1^T\Delta v_1+v_1^T\bar{F}'_1\Delta WX) \\ f'_{2(2)}\cdot(F_1^T\Delta v_2+v_2^T\bar{F}'_1\Delta WX) \\ \cdots \\ f'_{2(k)}\cdot(F_1^T\Delta v_k+v_k^T\bar{F}'_1\Delta WX) \end{array}\right]$$

$$= \left[\begin{array}{l} f'_{2(1)}\cdot(F_1^T[-\eta\nabla_{V_1}E(t)]+\bar{v}_1^T\bar{F}'_1[-\eta\nabla_W E(t)]X) \\ f'_{2(2)}\cdot(F_1^T[-\eta\nabla_{V_2}E(t)]+\bar{v}_2^T\bar{F}'_1[-\eta\nabla_W E(t)]X) \\ \cdots \\ f'_{2(k)}\cdot(F_1^T[-\eta\nabla_{V_k}E(t)]+\bar{v}_k^T\bar{F}'_1[-\eta\nabla_W E(t)]X) \end{array}\right]$$

$$= -\eta\left[\begin{array}{l} f'_{2(1)}\cdot(F_1^T\nabla_{V_1}E(t)+\bar{v}_1^T\bar{F}'_1\nabla_W E(t)X) \\ f'_{2(2)}\cdot(F_1^T\nabla_{V_2}E(t)+\bar{v}_2^T\bar{F}'_1\nabla_W E(t)X) \\ \cdots \\ f'_{2(k)}\cdot(F_1^T\nabla_{V_k}E(t)+\bar{v}_k^T\bar{F}'_1\nabla_W E(t)X) \end{array}\right]$$

$$= -\eta\left[\begin{array}{llll} f'_{2(1)} & 0 & \cdots & 0 \\ 0 & f'_{2(2)} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & f'_{2(k)} \end{array}\right]$$

$$\times\left[\begin{array}{l} F_1^T[\lambda\nabla E_{V_1}(t-1)-e_1f'_{2(1)}F_1]+\bar{v}_1^T\bar{F}'_1\nabla_W E(t)X \\ F_1^T[\lambda\nabla E_{V_2}(t-1)-e_2f'_{2(2)}F_1]+\bar{v}_2^T\bar{F}'_1\nabla_W E(t)X \\ \cdots \\ F_1^T[\lambda\nabla E_{V_k}(t-1)-e_kf'_{2(k)}F_1]+\bar{v}_k^T\bar{F}'_1\nabla_W E(t)X \end{array}\right]$$

$$= -\eta F'_2\left[\lambda\nabla_V^T E(t-1)F_1 - F'_2eF'_1F_1 + \bar{V}^T\bar{F}'(\lambda\nabla E_W(t-1)-F'_1VF'_2eX^T)X\right]$$

$$= -\eta\lambda F'_2\left[\nabla_V^T E(t-1)F_1+\bar{V}^T\bar{F}'\nabla E_W(t-1)X\right]+\eta F'_2(F'_2eF'_1F_1$$

$$-\bar{V}^T\bar{F}'F'_1VF'_2eX^TX)$$

$$= -\eta\lambda F'_2\left[\nabla_V^T E(t-1)F_1+\bar{V}^T\bar{F}'\nabla E_W(t-1)X\right]+\eta F'_2(F'_1F_1I_{k^2}$$

$$-\bar{V}^T\bar{F}'F'_1VX^TX)F'_2e$$

$$= -\eta\lambda\zeta(t-1)+\eta\xi(t)e(t).$$

□

Substituting (10) into (9), we obtain

$$e(t+1) \approx e(t)+\eta\lambda\zeta(t-1)-\eta\xi(t)e(t). \tag{11}$$

The objective here is to derive an optimal learning rate η. That is, at iteration t, an optimal value of the learning rate, $\eta^*(t)$, which minimizes $E(t{+}1)$, is obtained. Define the cost function:

$$E(t+1) = 0.5e^T(t+1)e(t+1). \tag{12}$$

Using Equation (11), Equation (12) may be written as

$$E(t+1) = 0.5\left[e(t) + \eta\lambda\zeta(t-1) - \eta\xi(t)e(t)\right]^T \left[e(t) + \eta\lambda\zeta(t-1) - \eta\xi(t)e(t)\right]. \tag{13}$$

In Equation (13), the first and second order conditions are as

$$\begin{aligned}\left.\frac{dE(t+1)}{d\eta}\right|_{\eta=\eta^*(t)} &= -0.5\left[\xi(t)e(t) - \lambda\zeta(t-1)\right]^T \left[e(t) - \eta^*(t)\xi(t)e(t) + \eta^*(t)\lambda\zeta(t-1)\right] \\ &\quad - 0.5[e(t) - \eta^*(t)\xi(t)e(t) + \eta^*(t)\lambda\zeta(t-1)]^T[\xi(t)e(t) - \lambda\zeta(t-1)] = 0,\end{aligned}$$

$$\left.\frac{d^2E(t+1)}{d\eta^2}\right|_{\eta=\eta^*(t)} = [\xi(t)e(t) - \lambda\zeta(t-1)]^T[\xi(t)e(t) - \lambda\zeta(t-1)] > 0.$$

Since $\xi(t)$ and $\zeta(t-1)$ is positively defined, the second condition is met, the optimum learning rate can be obtained from the first order condition, as illustrated in Equation (14). Interestedly, the optimized learning rate that we obtained is distinctly different from the result produced by the work [11]

$$\eta^*(t) = \frac{[\xi(t)e(t) - \lambda\zeta(t-1)]^T e(t)}{[\xi(t)e(t) - \lambda\zeta(t-1)]^T[\xi(t)e(t) - \lambda\zeta(t-1)]}. \tag{14}$$

Finally, the increments of the neural network parameters, found using the optimal learning rate, are obtained by replacing the η^* given by Equation (14) to Equations (6) and (7), which yield

$$\begin{aligned}\Delta W(t) &= -\left(\frac{[\xi(t)e(t) - \lambda\zeta(t-1)]^T e(t)}{[\xi(t)e(t) - \lambda\zeta(t-1)]^T[\xi(t)e(t) - \lambda\zeta(t-1)]}\right) \\ &\quad \times \left(\lambda\nabla_W E(t-1) - \bar{F}_1'(t)\bar{V}(t)F_2'(t)e(t)x^T(t)\right), \qquad (15)\\ \Delta V(t) &= -\left(\frac{[\xi(t)e(t) - \lambda\zeta(t-1)]^T e(t)}{[\xi(t)e(t) - \lambda\zeta(t-1)]^T[\xi(t)e(t) - \lambda\zeta(t-1)]}\right) \\ &\quad \times \left(\lambda\nabla_V E(t-1) - F_1(t)e^T(t)F_2'(t)\right). \qquad (16)\end{aligned}$$

It is worth noting that the forgetting factor λ is adaptive. During the neural network learning process, if the prediction error $e(t)$ grows, this may mean that the neural network parameters have changed. This implies that the network model is incorrect and needs adjustment. So we should reduce the forgetting factor and allow the neural network model to adapt. An adaptive forgetting factor which allows this is

$$\lambda(t) = s(t-1)/s(t), \tag{17}$$

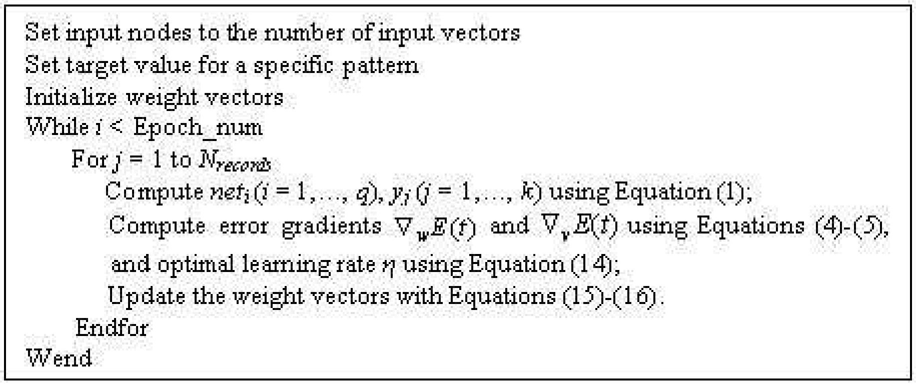
where $s(t)$ is a weighted average of the past values of e^Te and is calculated by

$$s(t) = [(\tau-1)/\tau]s(t-1) + (e^Te/\tau), \tag{18}$$

τ is the time constant of the forgetting factor determining how fast $\lambda(t)$ changes.

Using the updated weight formula with optimal learning rates and adaptive forgetting factors, a new online recursive learning algorithm is generated. For convenience, the proposed online learning algorithm is summarized as follows, as shown in Figure 2.1.

To verify the effectiveness of the proposed online learning algorithm, two typical financial time series: S&P 500, a famous stock index, and one foreign exchange rate, euros against US dollars (EUR/USD), are used as testing targets. The simulation experiments are presented in the following section.



```
Set input nodes to the number of input vectors
Set target value for a specific pattern
Initialize weight vectors
While $i$ < Epoch_num
    For $j$ = 1 to $N_{records}$
        Compute $net_i$ ($i$ = 1,..., $q$), $y_j$ ($j$ = 1,..., $k$) using Equation (1);
        Compute error gradients $\nabla_w E(t)$ and $\nabla_v E(t)$ using Equations (4)-(5),
        and optimal learning rate $\eta$ using Equation (14);
        Update the weight vectors with Equations (15)-(16).
    Endfor
Wend
```

Figure 2.1: Outline of the proposed online learning algorithm.

3 Experimental Analysis

In this section, there are two main motivations: (1) to evaluate the performance of the proposed online learning algorithm, and (2) to compare the efficiency of the proposed online learning algorithm with other similar algorithms. To perform the two motivations, two real-world data experiments are carried out. In this section, we first describe the research data and experiment design and then report the experimental results.

3.1 Research data and experiment design

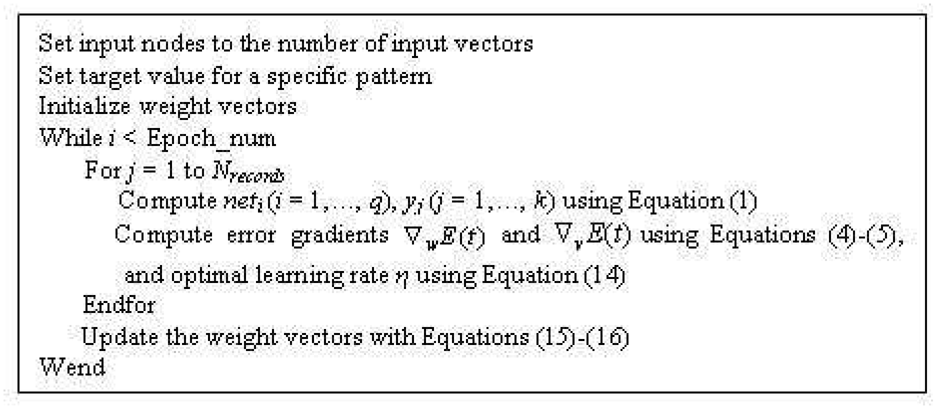
In the experiments, one stock index, S&P 500, and one foreign exchange rate, euros against US dollars (EUR/USD), are used for testing purpose. The historical data are daily and are obtained from Wharton Research Data Service (WRDS), provided by Wharton School of the University of Pennsylvania. The entire data set covers the period from January 1, 2000 to December 31, 2004 with a total of 1256 observations. The data sets are divided into two periods: the first period covers January 1, 2000 to December 31, 2003 with 1004 observations, while the second period is from January 1, 2004 to December 31, 2004 with 252 observations. The first period, which is assigned to in-sample estimation, is used for network learning, i.e., training set. The second period, which is reserved for out-of-sample evaluation, is used for validation, i.e., testing set. For space limitation, the original data are not listed in this paper, and detailed data can be obtained from the WRDS.

For comparison, four related algorithms, standard FNN algorithm [7, 16], batch learning algorithm, Levenberg-Marquart (LM) based learning algorithm [15-16], and extended Kalman filter (EKF) based learning algorithm [12, 17], are employed in this study. For standard FNN learning algorithm, the learning rate is fixed at 0.3, more details about standard FNN learning algorithm can be referred to [7]. In the batch learning algorithm, the weights are updated only at the end of each epoch. Similar to the online learning algorithm, the batch learning algorithm can also be summarized as follows, as illustrated in Figure 3.1.

The Levenberg-Marquart (LM) based algorithm [15-16] is a kind of quick convergence algorithm which has the little computation time for per iteration. Basically, the link weights of the neural network are updated based on the Jacobian matrix, J, collecting the partial derivatives of the neural network error e with respect to the weights. In other words, the update increment ΔW collecting the corrections of the weights in matrix W is computed by

$$\Delta W = -[J^T J + \mu I]^{-1} J^T e, \tag{19}$$

$$J = \begin{bmatrix} \frac{\partial e}{\partial w_{11}} & \frac{\partial e}{\partial w_{12}} & \cdots & \frac{\partial e}{\partial w_{1n}} \\ \frac{\partial e}{\partial w_{21}} & \frac{\partial e}{\partial w_{22}} & \cdots & \frac{\partial e}{\partial w_{2n}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e}{\partial w_{m1}} & \frac{\partial e}{\partial w_{m2}} & \cdots & \frac{\partial e}{\partial w_{mn}} \end{bmatrix}. \tag{20}$$



```
Set input nodes to the number of input vectors
Set target value for a specific pattern
Initialize weight vectors
While $i$ < Epoch_num
    For $j$ = 1 to $N_{records}$
        Compute $net_i$ ($i$ = 1,..., $q$), $y_j$ ($j$ = 1,..., $k$) using Equation (1)
        Compute error gradients $\nabla_w E(t)$ and $\nabla_v E(t)$ using Equations (4)-(5),
        and optimal learning rate $\eta$ using Equation (14)
    Endfor
    Update the weight vectors with Equations (15)-(16)
Wend
```

Figure 3.1: Outline of the batch learning algorithm.

It is worth noting that the LM-based algorithm is rather flexible. If μ is sufficiently large, the above weight update algorithm is similar to the gradient descent algorithm. If μ is equal to zero, the above algorithm will be a Gaussian-Newton algorithm. In this sense, the LM-based algorithm has the characteristics of both the gradient descent algorithm and the Gaussian-Newton algorithm.

The extended Kalman filter (EKF) based algorithm [12, 17] is a novel weight adjustment algorithm for FNN. In this algorithm, the Kalman filter is used to update the weight vector of FNN. The generic principle of EKF-based algorithm is that the EKF can modify the weight parameters to maximize the posterior probability of current instance with respect to its predicted probability distribution of weight parameters. Recent work proposed by Ruck [17] has revealed that the FNN algorithm is actually a degenerated form of the EKF. Due to its excellent convergence properties, a lot of successful applications have been reported. Basically, the EKF-based weight adjustment formulae are illustrated as follows.

$$W(t) = W(t-1) + K(t)[y(t) - \hat{y}(t)], \tag{21}$$

$$K(t) = P(t-1)H^T(t)[H(t)P(t-1)H^T(t) + R(t)]^{-1}, \tag{22}$$

$$P(t) = P(t-1) - K(t)H(t)P(t-1), \tag{23}$$

where $W(t)$ is the connect weight of FNN, $K(t)$ is called the Kalman gain, $y(t)$ is the actual value, $\hat{y}(t)$ is the predicted value produced by neural networks, $P(t)$ is the error covariance matrix, defined by $P(t) = E\{[y(t) - \hat{y}(t)]^T[y(t) - \hat{y}(t)]\}$ and $H(t)$ is the gradient, defined by $H(t) = \frac{\partial \hat{y}(t)}{\partial W}$. Usually, the system actual output$y(t) = \hat{y}(t) + \varepsilon(t)$, $\varepsilon(t)$ is assumed to be white noise vector with covariance $R(t)$ regarded as a modeling error. For more details, please refer to [12, 17].

In all the neural network predictors, five input nodes are determined by auto-regression testing. The appropriate number of hidden nodes is set to 12 in terms of trial and error. The training epochs are set to 3000 due to trial and error and the problem complexity.

To examine the forecasting performance, the root mean square error ($RMSE$) and directional change statistics (D_{stat}) [16] of financial time series are employed as the performance measurement of the testing set. In addition, training time and training mean square error ($TMSE$) are used as the efficiency measurement of different algorithms.

3.2 Experiment Results

When the data are prepared, we begin to perform experiments according to the previous experiment design. First of all, the prediction results with five algorithms are reported. Figures 3.2 and 3.3 give graphical representations of the forecasting results for two typical financial time series using different FNN learning algorithms. Table 3.1 shows a detailed prediction performance of the different algorithms in terms of both the level measurement ($RMSE$) and direction measurement (D_{stat}). From the figures and table, we can generally find that the prediction results of the proposed online learning algorithm are very promising for two typical financial time series under study either where the measurement of forecasting performance is the goodness-of-fit such as $RMSE$ or where the forecasting performance criterion is the D_{stat}.

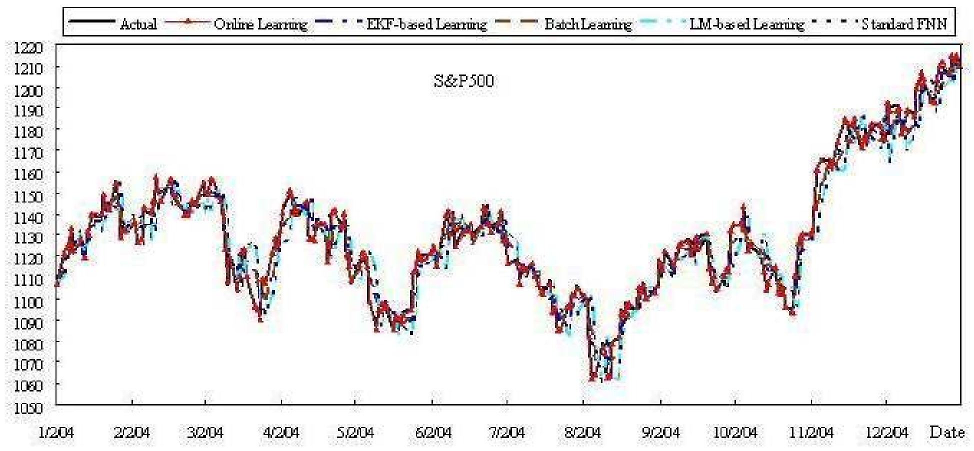


Figure 3.2: The forecasting results with different learning algorithm for S&P 500.

In detail, Figure 3.2 reveals that the comparison for the S&P 500 of the proposed online learning algorithm versus the other four learning algorithm. Similarly, it can be seen from Figure 3.3 that the forecasting performance for ERU/USD has significantly improved using the proposed online learning algorithm. The graphical results indicate that the proposed online learning algorithm performs than the other algorithms presented here.

Subsequently, the concrete prediction performance comparison of various algorithms for two different financial time series via $RMSE$ and D_{stat} are given in Table 3.1.

For the S&P 500, the proposed online learning algorithm outperforms the other four learning algorithms in terms of both $RMSE$ and D_{stat}. Focusing on the $RMSE$ indicator, the proposed online learning algorithm performs the best, followed by batch learning, EKF-based learning, LM-based learning and Standard FNN learning algorithm. Comparing with standard FNN learning algorithm, the RMSE of the proposed online learning algorithm is much smaller. From the viewpoint of D_{stat}, the performance of the proposed online learning algorithm is the best of the all. Relative to the standard FNN learning algorithm, the performance improvement arrives at 26.82% (80.31%-53.41%) While the performance of the proposed online learning algorithm is slightly improved relative to batch learning algorithm, EKF-based learning algorithm and LM-based algorithm.

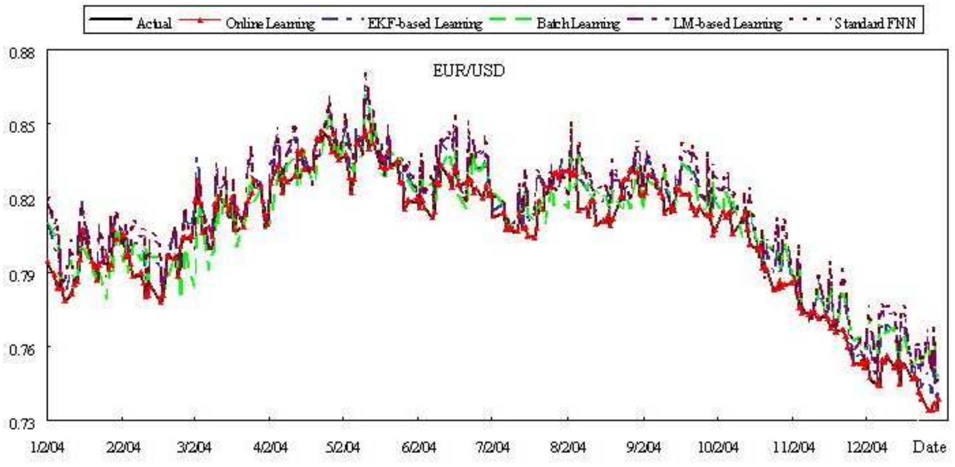


Figure 3.3: The forecasting results with different learning algorithm for EUR/USD.

Algorithms	S&P 500		EUR/USD	
	$RMSE$	$D_{stat}(\%)$	$RMSE$	$D_{stat}(\%)$
Online learning	1.2859	80.31	0.0799	79.87
Batch learning	2.1467	71.42	0.0943	69.75
EKF-based learning	2.1538	70.35	0.1051	72.29
LM-based learning	4.3531	71.69	0.1544	69.34
Standard FNN	7.8553	53.49	0.3362	55.64

Table 3.1: Performance comparison of four neural network learning algorithms.

Algorithms	S&P 500		EUR/USD	
	Time (seconds)	*TMSE*	*Time (seconds)*	*TMSE*
Online learning	197	3.41	192	1.17×10^{-3}
Batch learning	186	7.96	177	5.04×10^{-3}
EKF-based learning	173	8.42	154	4.85×10^{-3}
LM-based learning	535	8.77	573	3.56×10^{-3}
Standard FNN	249	12.55	269	6.09×10^{-3}

Table 3.2: The comparisons of the computational efficiency and training performance.

For the exchange rate of EUR/USD, the performance of the proposed online algorithm is the best, similar to the results of the S&P 500. Likewise, the proposed online algorithm has gained much improvement relative to the standard FNN learning algorithm. Interestedly, the *RMSE* of the batch learning algorithm is slightly better than that of the EKF-based learning algorithm, but the directional performance (i.e., D_{stat}) of the batch learning is somewhat worse than that of the EKF-based learning algorithm. The possible reasons are needed to be further addressed later.

In summary, we can conclude that (1) the proposed online learning algorithm with adaptive forgetting factors performs consistently better than other comparable learning algorithm for both the stock index and foreign exchange rate; (2) the evaluation value of the two criteria of the proposed online learning is much better than that of the standard FNN learning algorithm, indicating that the proposed online learning algorithm can effectively reflect error changes and significantly improve network learning performance. One possible reason for this is that the optimal learning rate and adaptive forgetting factors are used in the online learning algorithm.

In addition, the computation speed of the proposed online algorithm is very fast during the experiments when using a personal computer (PC) and the training performance is also well in predicting time series, indicating that the proposed learning algorithm is an efficient online algorithm. For comparison purpose, Table 3.2 reports the comparison of the computation time and training performance between the proposed online learning algorithm and the other four learning algorithm presented here.

From Table 3.2, we can find the following conclusions. First of all, for both S&P 500 and EUR/USD series, the computational time of the EKF-based learning algorithm is the smallest and the LM-based learning algorithm is the largest. The results reported here are basically consistent with the work of Iiguni et al. [12]. The main reason is that the number of iterations of LM-based learning algorithm is much larger than that of EKF-based learning algorithm, although the computation time per iteration of the EKF-based learning algorithm is larger than that of the LM-based learning algorithm [12]. Secondly, the computation time of the batch learning algorithm is smaller than that of the online learning algorithm due to the batch processing of the data. However, relative to the standard FNN learning and LM-based learning algorithms, the computation time of the online learning algorithm is much smaller. The main reason may be that the proposed online learning algorithm adopts optimal learning rate, resulting in the increase of convergence speed. Thirdly, although the computation time of the proposed online learning algorithm is not the best, the training performance (refer to *TMSE* presented by Table 3.2) and the generalized performance (refer to *RMSE* and D_{stat} reported by

Table 3.1) is the best among the entire learning algorithms presented in this study. The possible reason is that the adaptive forgetting factor helps improve the performance of this proposed algorithm. Finally, since the difference of the computation time between the proposed online learning algorithm and EKF-based and batch learning algorithm is marginal, and the difference of the performance between the proposed online learning algorithm and EKF-based and batch learning is significant, in this sense, the computational efficiency of the proposed online learning algorithm is satisfactory when forecasting financial time series. In general, the experimental results reveal that the proposed online learning algorithm provide a feasible solutions to financial time series online prediction.

4 Conclusions

In this study, an online learning algorithm with optimized learning rates and adaptive forgetting factors is first proposed. This exploratory research examines the potential of using the proposed online learning algorithm to predict two main financial time series – S&P 500 and the exchange rate for euros against US dollars. Our empirical results suggest that the online learning algorithm may provide much better forecasts than the other four learning algorithms. Furthermore, the learning efficiency is also satisfactory relative to the learning performance. This implies that the proposed online learning algorithm with adaptive forgetting factors is very suitable for online prediction of financial time series.

Acknowledgements

The authors would like to thanks the guest editors and two anonymous referees for their valuable comments and suggestions. Their comments helped to improve the quality of the paper immensely. The work described in this paper was partially support by National Natural Science Foundation of China (NSFC No. 70221001, 70601029); Key Laboratory of Management, Decision and Information Systems of Chinese Academy of Sciences (CAS) and Strategic Research Grant of City University of Hong Kong (SRG No. 7001677, 7001806).

References

[1] Hall, J.W. Adaptive selection of US stocks with neural nets. In: *Trading On the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets* (Deboeck, G.J., ed.). Wiley, New York, 1994, 45–65.

[2] Yaser, S.A.M., Atiya, A.F. Introduction to financial forecasting. *Applied. Intelligence* **6** (1996) 205–213.

[3] Hutchinson, J., Lo, A., Poggio, T. A non-parametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance* **49** (1994) 851–889.

[4] Moody, J., Utans, J. Architecture selection strategies for neural networks: Application to corporate bond rating prediction. In: *Neural Networks in the Capital Markets* (Refenes, A.P., ed.). Wiley, New York, 1994, 277–300.

[5] Kimoto, T., Asakawa, K., Yoda, M., Takeoka, M. Stock market prediction system with modular neural networks. In: *Neural Networks Finance Investing: Using Artificial Intelligence to Improve Real-World Performance* (Trippi, R.R., Turban, E., eds.). Irwin Publishers, Chicago, 1996, 497–510.

[6] Zhang, G.Q., Michael, Y.H. Neural network forecasting of the British Pound/US Dollar exchange rate. *Omega* **26** (1998) 495–506.

[7] Haykin, S. *Neural Networks: A Comprehensive Foundation.* Prentice-Hall Inc., Englewood Cliffs, New-Jersey, 1999.

[8] Kaastra, I., Boyd, M.S. Forecasting futures trading volume using neural networks. *Journal of Futures Markets* **15** (1995) 953–970.

[9] Tollenaere, T. SuperSAB: Fast adaptive back propagation with good scaling properties. *Neural Networks* **3** (1990) 561–573.

[10] Park, D.C., El-Sharkawi, M.A., Marks II, R.J. An adaptive training neural network. *IEEE Transactions on Neural Networks* **2** (1991) 334–345.

[11] Sha, D., Bajic, V.B. An online hybrid learning algorithm for multilayer perceptron in identification problems. *Computers and Electrical Engineering* **28** (2002) 587–598.

[12] Iiguni, Y., Sakai, H., Tokumaru, H. A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter. *IEEE Transactions on Signal Processing* **40** (1992) 959–966.

[13] Jacobs, R.A. Increase rates of convergence through learning rate adaptation. *Neural Networks* **1** (1988) 295–307.

[14] Brent, R.P. Fast training algorithms for multilayer neural nets. *IEEE Transactions on Neural Networks* **2** (1991) 346–35.

[15] Hagan, M.T., Menhaj, M. Training feedforward networks with Marquart algorithm. *IEEE Transactions on Neural Networks* **5** (1994) 989–993.

[16] Yu, L., Wang, S.Y., Lai, K.K. A novel nonlinear ensemble forecasting model incorporating GLAR and ANN for foreign exchange rates. *Computers & Operations Research* **32** (2005) 2523–2541.

[17] Ruck, D.W., Rogers, S.K., Kabrisky, M., Maybeck, P.S., Oxley, M.E. Comparative analysis of backpropagation and the extended Kalman filter for training multilayer perceptrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14** (1992) 686–691.