Multi-step Time Series Forecasting with an Ensemble of Varied Length Mixture Models

DOI:
10.1142/S0129065717500538

Document Version
Accepted author manuscript

Submitted to International Journal of Neural Systems (2016)

Multi-step Time Series Forecasting with an Ensemble of Varied Length Mixture Models

YICUN OUYANG and HUJUN YIN*
School of Electrical and Electronic Engineering, The University of Manchester, Manchester, M13 9PL, UK
E-mail: hujun.yin@manchester.ac.uk

Many real-world problems require modeling and forecasting of time series, such as weather temperature, electricity demand, stock prices and foreign exchange rates. Often, the tasks involve predicting over a long-term period, e.g. several weeks or months. Most existing time series models are inheritably for one-step prediction, that is, predicting one time point ahead. Multi-step or long-term prediction is difficult and challenging due to the lack of information and uncertainty or error accumulation. The main existing approaches, iterative and independent, either use one-step model recursively or treat the multi-step task as an independent model. They generally perform poorly in practical applications. In this paper, as an extension of the self-organizing mixture autoregressive model, the varied length mixture (VLM) models, is proposed to model and forecast time series over multi-steps. The key idea is to preserve the dependencies between the time points within the prediction horizon. Training data are segmented to various lengths corresponding to various forecasting horizons, and the VLM models are trained in a self-organizing fashion on these segments to capture these dependencies in its component autoregressive models of various predicting horizons. The VLM models form a probabilistic mixture of these varied length models. A combination of short and long VLM models and an ensemble of them are proposed to further enhance the prediction performance. The effectiveness of the proposed methods and their marked improvements over the existing methods are demonstrated through a number of experiments on synthetic data, real-world foreign exchange rates and weather temperatures.

Keywords: Time series forecasting; long-term prediction; probabilistic mixture model; regressive models; self-organizing networks.

1. Introduction

Time series are sequences of data points associated with time and are often analyzed for their statistical meaningfulness and predictive traits. Time series models and methods are commonly used for predicting or forecasting the future value on the basis of observed data trends, but their practical applications extend to many fields of studies, including communications, signal processing, economics, finance and social media. In recent years, especially in the era of Big Data, growing interests have been witnessed in time series analysis for extracting meaningful temporal information or patterns in data.[1] Tasks can be divided into clustering, classification and forecasting. Among them, time series forecasting, constructing a model to predict future values based on previous observations, is the most demanding and challenging one. Many models and methods have been proposed, including autoregressive models, heteroskedastic models, neural networks and learning methods.[2]

Once a time series model is established, through either regression or learning of the current point over previous points, the model can then be used for predicting one step ahead. In practical applications, multi-step prediction is often more desirable. How-

*Corresponding author.

ever, predictions over multi-steps are more difficult and challenging to obtain due to lack of measurements within the prediction horizon.[3,4] The existing approaches to multi-step forecasting can be mainly divided into two categories: iterative and independent.[3] For an iterative method, prediction result of the one-step forecast is used as the input for forecasting two-step ahead with the same model, and so on, up to the prediction horizon. Inevitably the errors of every prediction are accumulated along the time steps. Due to this disadvantage, iterative methods are not often preferred in multi-step prediction. For the independent approach, an individual model is built for each forecasting horizon. Although it does not have the problem of accumulating errors, computational complexity of this approach is much greater due to the need to build separate models for each horizon and that dependencies among the time points along the prediction horizon are not extracted and kept.

In the past, time series forecasting has been primarily performed with regressive models such as autoregressive (AR) and autoregressive moving average (ARMA).[5] Both the AR and ARMA models are built on the assumption of linearity and stationarity, which limit their applications. Although, many extended models exist such as the autoregressive integrated moving average (ARIMA) and the generalized autoregressive conditional heteroskedasticity (GARCH) (see[2]), their applications in multi-step prediction are rare due to the aforementioned error accumulation problem. Recently more and more attention has been shifted to neural networks[6,7] and learning methods as general modeling and forecasting tools, such as support vector machine (SVM)[4,8,9] and feed-forward and recurrent neural networks,[10] due to their versatile, nonlinear, adaptive and data-driven nature and capability of modeling complex relations. As a nonlinear model, SVM extracts a group of support vectors to represent characteristics of the data. It constructs a hyperplane by which the largest distances are achieved between the nearest data points of different classes. SVM has been widely used in classification and has also been applied to time series regression, which is regarded as a curve fitting problem. Good performances have been achieved.[8] To further improve the generalization performance and computation speed, a twin SVM (TSVM) has been proposed by employing two nonparallel planes (two related SVM-type problems).[11] SVM and its extensions generally have high computational complexity and difficulties in choosing right kernel functions.

The existing neural networks that have been widely used for forecasting time series[12] include multilayer perceptron (MLP),[13–16] recurrent neural network (RNN),[17–22] and self-organizing map (SOM).[23]

MLP is one of the widely used feed-forward neural networks, consisting of multiple layers of nodes with nonlinear activation functions. It is commonly trained with the back-propagation algorithm,[14] in which the error signals are back-propagated through the network, for training the network weights. MLP can be used for approximating smooth and measurable functions,[14] as an alternative to traditional techniques.[16] When used for time series modeling, a segment of consecutive (delayed) time points is used as the input. MLP typically has local minima and overfitting problems.

RNN, which has directed-cycle connections (feedback) among units, is different from the feed-forward networks.[20] To deal with the dynamic temporal behaviors in time series, internal states are incorporated into the network. RNN is competitive when applied to the temporal tasks such as handwriting recognition and time series prediction[21,22] and can achieve good performance in portfolio optimization, speech recognition and price analysis. However, RNN is not easy to train for large numbers of neurons due to convergence and stability problems.

SOM[24,25] is an unsupervised learning that maps an input space onto a grid of neurons, in which topological relationships are preserved. When time points are grouped into segments of consecutive points as input vectors, SOM can be used to model time series. There were also a number of earlier extensions of SOM for time series, such as NG,[26] self-organizing autoregressive (SOAR),[27] temporal Kohonen map (TKM),[28] recurrent SOM (RSOM)[29] and recursive SOM (RecSOM).[30] TKM attempts to integrate temporal information in SOM by allowing some previous activation into the activation of the current input,[28] and RSOM, as a modification of TKM, moves the leaky integrators into the difference vector.[29] RecSOM incorporates context information into the reference vectors.[30] To further improve on SOM's time series modeling capability, the self-organizing mixture autoregressive (SOMAR)[31,32] and its neural gas variant, the neural gas mixture autoregressive (NG-

MAR)[33] have been proposed. They are based on mixture of autoregressive models and employ a new way to measure similarity between input batches and reference vectors. All these models, however, are for single step prediction.

This paper extends SOMAR for multi-step time series forecasting. In this method, termed as varied length mixture (VLM) models, probabilistic dependencies between time points within the prediction horizon are extracted and kept, rather than excluded as in the independent methods. The VLM models are trained with input segments of various lengths. The reference vectors (or weights) of the VLM models are of the maximum length for the required prediction horizon. For each individual input segment, the reference vectors only update those parts of the reference vectors that correspond to the length of the input segment.

Further more the VLM models replace the winner-takes-all mechanism used in SOM with a more general mixture model such as the self-organizing mixture network (SOMN).[34] That is, all the component models contribute to the final output of the mixture model, where contributions are determined by the mixing weights, which are also adaptive to the training data. The model acts as an ensemble learning. For multi-step forecasting, the VLM models serve as a mixture of various length models, which are co-trained and complementary to each other. A graphic description of the VLM models is given in Fig. 1. Furthermore, a dynamic combination of various models (e.g. short-term and long-term), together with an integrating or ensemble mechanism, can be used on the VLM models for enhanced performance. The proposed methods have been tested on a variety of benchmark datasets as well as real-world time series and the results show that the proposed neural network outperforms existing competitors.

In section 2, after a brief description of SOM and SOMAR models for time series modeling, the VLM models are introduced for multi-step prediction. Section 3 extends the VLM models to an ensemble learning. In section 4, performances of the proposed methods are evaluated and compared with the existing methods on benchmark data, foreign exchange rates and weather temperatures. Section 5 discusses the advantages of the proposed algorithms, followed by conclusions in section 6.

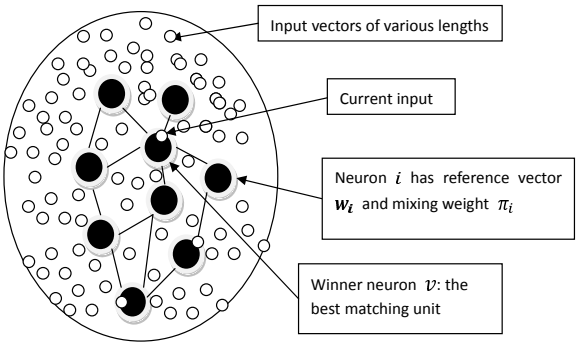

Figure 1. Graphical description of VLM models. Reference vector of each neuron contains the coefficients of an AR model and a mixing weight.

2. Varied Length Mixture Model for Multi-step Prediction

This section starts with SOM and SOMAR models for time series modeling and then introduces the proposed VLM models for multi-step forecasting. The training procedure, including constructing input segments and updating reference vectors, and predicting procedure are described in detail.

2.1. *Prior work based on SOM, SOMAR and NGMAR*

2.1.1. *SOM*

SOM is a topology-preserving vector quantization method. When SOM is used directly for modeling time series, the sequence has to be segmented into vectors (e.g. segments of l consecutive points) and the reference or weight vector of neuron i is defined as,

$$\mathbf{w}_i = [w_{i,0}, w_{i,1}, \cdots, w_{i,l-1}], i = 0, 1, \cdots, M-1 \quad (1)$$

where l denotes the length of the input vectors (and length of the weight vectors) and M is the total number of neurons.

For a time series of total length L, $\{x_\tau\}_{\tau=0}^{L-1}$, a sliding window of length l can be used to divide the series into input segments as $\mathbf{x}(\tau) = [x_{\tau-l+1}, x_{\tau-l+2}, \ldots, x_\tau]_{\tau=l-1}^{L-1}$. There are in total $L-l+1$ input segments in the input segment set. Then the updating rule for the weight vectors is described as,

$$\Delta\mathbf{w}_i = \gamma(n) g_{i,v}(n)(\mathbf{x}(n) - \mathbf{w}_i), i = 0, 1, \cdots, M-1 \quad (2)$$

where n is the iteration number; $\mathbf{x}(n)$ is the input segment at iteration n, randomly selected from the input segment set; $\gamma(n)$ is the learning rate; $g_{i,v}(n)$ is the neighborhood function typically a Gaussian and v is the best matching unit (BMU) based on the minimum distance to the input $\|\mathbf{x}(n) - \mathbf{w}_i\|$.

This direct way of using SOM for time series is also called vector SOM (VSOM). Once the SOM is trained, the last element of the winner's weight vector can be used as the estimated value in one-step forecasting and the winner is chosen based on the all the elements but the last one. VSOM (as well as its other variants, RSOM and RecSOM) can capture some temporal information in the weight vectors. However, these weight vectors do not represent regressive temporal models. To turn SOM into time series models, each neuron has to cast a regressive model.

2.1.2. *SOMAR*

SOMAR was introduced to model non-linear time series by a mixture of linear AR processes in a self-organizing fashion.[31,32] Although its weight vectors are defined in the same way as in the VSOM, the meaning of these weight vectors is different. They represent the coefficients of learned component AR models. Given the current input point x_τ and the previous segment $\mathbf{x}(\tau - 1)$, each node yields a modeling error as,

$$e_i(\tau) = x_\tau - \mathbf{w}_i^T \mathbf{x}(\tau - 1), i = 0, 1, \cdots, M - 1 \quad (3)$$

After a small batch of input vectors, the autocorrelation coefficients of the error sequences, instead of the errors themselves, are used to measure the similarity between the input and candidate regressive models represented by the reference vectors,

$$R_i(\xi) = \frac{1}{s\sigma^2} \sum_{j=0}^{s-\xi-1} (e_i(\tau - j) - \mu)(e_i(\tau + \xi - j) - \mu), \quad i = 0, 1, \cdots, M - 1 \quad (4)$$

where ξ is the lag, s is the size of the batch, μ and σ^2 are the mean and variance of the error sequences $e_i(\tau)$, i.e. $\mu = \frac{\sum_{\xi=1}^{s} e_i(\xi)}{s}$ and $\sigma^2 = \frac{\sum_{\xi=1}^{s} (e_i(\xi) - \mu)^2}{s}$.

Consequently, the BMU is selected with the minimum sum of autocorrelation coefficients (SAC) as,

$$v = arg\min_i (\sum_{\xi=-s}^{s} \|R_i(\xi)\|), i = 0, 1, ..., M - 1 \quad (5)$$

The reference vectors are then updated by the LMS algorithm, further coupled with SOM's neighborhood mechanism,

$$\Delta \mathbf{w}_i = \gamma(n) g_{i,v}(n) e_i(n) \mathbf{x}(n - 1), i = 0, 1, ..., M - 1 \quad (6)$$

The SOMAR model has been shown successful in converging to underlying AR processes. It yields better performance than previous SOM-based models on various benchmark data and foreign exchange (FX) rates.[32] However, its fixed grid structure of neurons can be further relaxed for more optimal performance in time series where topological lattice is not essential.

2.1.3. *NGMAR*

To free the arrangement of neurons in SOMAR, a neural gas version, NGMAR, was proposed by replacing SOM with NG.[33] Local mixture autoregressive models and the SAC similarity measure are kept in the NGMAR model, while the reference vectors are updated by,

$$\Delta \mathbf{w}_{i_k} = \gamma(n) \exp(-k/\lambda(n)) e_{i_k}(n) \mathbf{x}(n - 1) \quad (7)$$

$$\gamma(n) = \gamma_0 (\gamma_N / \gamma_0)^{n/N} \quad (8)$$

$$\lambda(n) = \lambda_0 (\lambda_N / \lambda_0)^{n/N} \quad (9)$$

where k is the ranking of neuron i_k, the learning rate $\gamma(n)$ and the neighborhood range $\lambda(n)$ decay to zero with time, and N is the total number of iterations. Due to that the dynamic neighborhood of NG is based on similarity rankings, improved performance of NGMAR in time series modeling has been demonstrated over other neural networks.[33]

2.1.4. *GSOMAR*

SOMAR is in fact a simplified version of the mixture autoregressive (MAR) model,[35] proposed for dealing with non-stationary time series. The conditional cumulative distribution of the MAR model can be expressed as

$$F(x_t|\mathscr{F}_{t-1}) = \sum_{i=1}^{K} P_i \Psi(x_t - \phi_{i0} - \phi_{i1} x_{t-1} - ... - \phi_{il_i} x_{t-l_i}) \quad (10)$$

where $\mathscr{F}_{t-1}$ represents the information at time $t-1$, K is the number of AR processes, l_i is the order of AR process i, Ψ refers to the cumulative distribution functions of normal distribution, ϕ_{ij} are the parameters and P_i is the mixing weight represented by winning frequency of neuron i.

In SOMAR and NGMAR, usually only the winning AR model is selected to represent the time series at any time in prediction. This means that the mixing factor of only the winner is unit and others' are zero.

To fully employ the mixture model, the generalized SOMAR (GSOMAR)[36] was proposed by learning the values of the mixing weights π_i based on the winning frequencies. Then the updating rule for the mixing weights is described as follows

$$\Delta\pi_i = \gamma(n)(\hat{P}_i - \pi_i) \tag{11}$$

where $\hat{P}_i$ is estimated by winning percentage of node i. In forecasting, the mixing weights are used, acting as the posterior probability of a component class given an input sample as in the self-organizing mixture network (SOMN).[34]

2.2. *Input segments of various lengths*

In SOMAR, NGMAR and GSOMAR, the input segments are of the same length and the learned AR models are of the same order and these methods are inherently for one-step prediction. For multi-step prediction, these methods can be applied repeatedly as in the iterative approach. For instance, given the current input, $\mathbf{x}(t) = [x_{t-l+1}, \cdots, x_{t-1}, x_t]$, one-step forecast is obtained directly from the model as $\hat{x}_{t+1} = f(\mathbf{x}(t))$; and then two-step forecast uses one-step forecast result and the current input as the input, $\hat{x}_{t+2} = f(\mathbf{x}(t), \hat{x}_{t+1})$, ..., and so on. In the iterative method, errors in previous steps are accumulated to the next steps, resulting in poor accuracy for long-term prediction.[37]

To build independent models for multi-step prediction, time series points are segmented vectors according to prediction horizons. For example, for one, two, or h-step prediction, the current, 1st, 2nd, $\cdots$, or h-th future values are concatenated with the previous values to form the input segments as follows.

$$\mathbf{x}(\tau, 1) = [x_{\tau-l+1}, \cdots, x_{\tau-1}, x_\tau] \tag{12}$$
$$\mathbf{x}(\tau, 2) = [x_{\tau-l+1}, \cdots, x_{\tau-1}, x_{\tau+1}] \tag{13}$$
$$\cdots \quad \cdots$$
$$\mathbf{x}(\tau, h) = [x_{\tau-l+1}, \cdots, x_{\tau-1}, x_{\tau+h-1}] \tag{14}$$

Compared to the input vectors used in the iterative method, these input vectors replace the present value with a future value corresponding to the forecasting horizon. For each horizon, an independent model is trained. The problem of error accumulation is avoided. However, the dependencies among consecutive future points (between the current and the horizon) are not taken into account.

To further improve the performance in multi-step forecasting with the SOMAR/NGMAR model, the proposed VLM method effectively uses component AR models of varied orders instead of the same order. Although the weight vectors are of the same, the maximum length, their effective elements update according to the input vectors of various length. All the consecutive (future points) are kept in the segmented training vectors, as follows:

$$\mathbf{x}^{(1)}(\tau) = [x_{\tau-l+1}, \cdots, x_{\tau-1}, x_\tau] \tag{15}$$
$$\mathbf{x}^{(2)}(\tau) = [x_{\tau-l+1}, \cdots, x_{\tau-1}, x_\tau, x_{\tau+1}] \tag{16}$$
$$\cdots \quad \cdots$$
$$\mathbf{x}^{(h)}(\tau) = [x_{\tau-l+1}, \cdots, x_{\tau-1}, x_\tau, x_{\tau+1}, \cdots, x_{\tau+h-1}] \tag{17}$$
$$\cdots \quad \cdots$$
$$\mathbf{x}^{(H)}(\tau) = [x_{\tau-l+1}, \cdots, x_{\tau-1}, x_\tau, x_{\tau+1}, \cdots, x_{\tau+H-1}] \tag{18}$$

where H is the maximum horizon to be predicted. When $H = 1$, the method becomes the one step method. That is, for a given prediction horizon, H, all other prediction horizons (up to H) can also learned in the mixture model. The mixture is now a set of heterogenous AR models.

2.3. *Stochastic gradient descent learning*

The input vectors are built with various lengths of $l, l+1, \cdots$, and $l+H-1$, where l is the minimum length of these inputs or the minimum order of local AR models. The reference vectors of the nodes are

all of the maximum length, $(l + H - 1)$,

$$\mathbf{w}_i = [w_{i,0}, \cdots, w_{i,l-2}, w_{i,l-1}, \cdots, w_{i,l+H-2}] \quad (19)$$

Different elements of the reference vectors are updated by the corresponding elements of the input vectors. For input vector $\mathbf{x}^{(1)}(\tau)$, the components $[w_{i,0}, \cdots, w_{i,l-1}]$ are updated; For $\mathbf{x}^{(2)}(\tau)$, $[w_{i,0}, \cdots, w_{i,l}]$ are updated; so on; and likewise, for input vector $\mathbf{x}^{(H)}(\tau)$, all components $[w_{i,0}, \cdots, w_{i,l+H-2}]$ are updated.

An input segment is randomly drawn from all possible input segments constructed by eqs. (15)-(18). Depending on the input vector, the updating rule can be rewritten as a series of rules,

$$[w_{i,0}, \cdots, w_{i,l-1}] = [w_{i,0}, \cdots, w_{i,l-1}] + \Xi(n)\mathbf{x}^{(1)} \quad (20)$$

$$[w_{i,0}, \cdots, w_{i,l}] = [w_{i,0}, \cdots, w_{i,l}] + \Xi(n)\mathbf{x}^{(2)} \quad (21)$$

$$\cdots \qquad \cdots$$

$$[w_{i,0}, \cdots, w_{i,l+H-2}] = [w_{i,0}, \cdots, w_{i,l+H-2}] + \Xi(n)\mathbf{x}^{(H)} \quad (22)$$

where $\Xi(n) = \gamma(n)\exp(-k/\lambda(n))e_{i_k}(n)$.

Mirroring to the cost function of the SOM (e.g.[24]), if the probability distribution of data vectors $\mathbf{x}$ is described by $p(\mathbf{x})$, then the mean modeling error is determined by

$$E(\mathbf{w}_1, \cdots, \mathbf{w}_M) = \sum_i \int_{D_i} \sum_k g_\lambda(k_i)(x_{t+h} - \mathbf{w}_i^T\mathbf{x})^2 p(\mathbf{x})d\mathbf{x} \quad (23)$$

Instead of minimizing the quantization errors, $\|\mathbf{x} - \mathbf{w}_i\|^2$, as by the SOM with the further consideration of neighborhood learning, here the regression errors, $(x_{t+h} - \mathbf{w}_i^T\mathbf{x})^2$, are minimized, Then the adaption of reference vectors $\mathbf{w}_i$ in general can be expressed by

$$\Delta\mathbf{w}_{i_k} = \gamma(n)g_\lambda(k_i(\mathbf{x}, \mathbf{w}))e_{i_k}(n)\mathbf{x}^{(h)}(n) \quad (24)$$

where $\gamma(n)$ is the step size decaying to zero for increasing i_k and e_i is the error on the input vector $\mathbf{x}$ by the reference vector $\mathbf{w}_i$, i.e. $e_i(\mathbf{x}, \mathbf{w}) = x_{n+1} - \mathbf{w}_i^T\mathbf{x}$. The dynamics of the reference vector $\mathbf{w}_{i_k}$ obeys a stochastic gradient descent on the cost function.

The proposed VLM models are readily for predicting up to the given horizon, H, with the corresponding reference weights learned. To further improve the prediction performance, the proposed VLM models adopt probabilistic mixture principle, which forms an ensemble or combination of models of various orders (e.g. daily and weekly models), as described next.

Algorithm 1 Algorithms for eVLM model

1: Initialise the weight matrix $\mathbf{w}_i$, and specify training iterations N;
2: **for** $\tau = 1$ to N **do**
3: Build individual input vectors $\mathbf{x}^{(h)}(\tau), h = 1, \cdots, H$ for each forecasting horizon h;
4: **end for**
5: **for** $\tau = 1$ to N **do**
6: Calculate SAC value of each batch of input vectors and all reference vectors;
7: Build RANK based on SAC values;
8: Update the corresponding reference vector $\mathbf{w}_i$ and mixing weights π_i of neuron i as in Eq. (11);
9: **end for**
10: **for** $h = 1$ to H **do**
11: Obtain individual prediction of each neuron $\hat{x}_{t+h}(i)$ by using BMU v and corresponding components of $\mathbf{w}_v$;
12: Obtain weighted average by Eq. (26) as prediction
13: **end for**
14: Combine short-term and long-term prediction outputs adaptively as the prediction output in Eq. (28).
15: Calculate the final prediction by eVLM as in Eq. (35).

3. Ensemble of VLM Models

As combining forecasts or ensemble predicting can lead to performance improvement over individual models by reducing prediction variance or errors, an adaptive combination of VLM models of various orders, short-term (e.g. daily) and long-term (e.g. weekly), together with an ensemble, is employed for multi-step forecasting of time series, resulting the ensemble VLM (eVLM) method. Ensemble learning can reduce the variance of the learning. The main reason why ensemble can perform better than individual models is that the ensemble employs multiple models, each of which can focus on some particular part of the input space. This is the very case in financial time series, especially FX rates. The financial time series has different patterns, which are embedded in different time series segments. Therefore, an ensemble can be useful and effective for describing different time series patterns and reducing the prediction error. Furthermore, the probabilistic mixture principle can be adopted as an efficient way of the ensemble.

3.1. *Probabilistic mixture principle*

To achieve effectiveness and robustness, the winner-takes-all principle in the previous described models is replaced by a probabilistic mixture where all components contribute to the mixture,[34] with the mixing weights updating rule described by eq. (11).

After training, the updated reference vectors of the network can be used for predicting over all the horizons $1, 2, \cdots, H$. Given a current input vector $\mathbf{x}(t)$, various predicted future values are given as the

output mixture of K individual component models,

$$\hat{x}_{t+1} = \sum_{i=1}^{K} \pi_i \sum_{j=0}^{l-1} w_{i,j} x_{t-(l-1)+j} \tag{25}$$

$$\cdots \qquad \cdots$$

$$\hat{x}_{t+h} = \sum_{i=1}^{K} \pi_i (\sum_{j=0}^{l-1} w_{i,j} x_{t-(l-1)+j} + \sum_{j=l}^{l+h-2} w_{i,j} \hat{x}_{t-(l-1)+j}) \tag{26}$$

$$\cdots \qquad \cdots$$

$$\hat{x}_{t+H} = \sum_{i=1}^{K} \pi_i (\sum_{j=0}^{l-1} w_{i,j} x_{t-(l-1)+j} + \sum_{j=l}^{l+H-2} w_{i,j} \hat{x}_{t-(l-1)+j}) \tag{27}$$

3.2. *Adaptively combining VLM models*

Most existing time series predictions are concerned with predicting the next step, i.e. $H = 1$. The so-called long term prediction is concerned of prediction when $H >> 1$; while the so-called short-term prediction is about predicting near next step, i.e. H is a small number (usually 1, 2 or 3). In FX forecasting, when daily data is used, short-time usually means a day, or a couple of days or few days ahead; while long terms could be weekly or fortnight, or even months ahead.

In general, performance of combined forecasting models (e.g. combining short-term and long-term approaches) is better than that of single models.[13] Such combination can benefit from performance advantages of individual models while complementing their shortcomings. Furthermore, to overcome the limitations of a fixed combination, adaptive, horizon-dependent weighting is employed in the combination.

For example, consider that one can build two forecasting models, a short-term, say daily model $f_d(h)$ and a long-term, say weekly model $f_w(h)$, $h = 1, 2, \cdots, H$. The parameters of the daily model are estimated with the training data segments of short forecasting horizons. Similarly, the weekly model is trained with the training vectors revealing relationship between values with long distance as described in the previous section. The adaptive combination of the both can be defined as

$$y(h) = \alpha(h) f_d(h) + (1 - \alpha(h)) f_w(h) \tag{28}$$

where $\alpha(h)$ is real function of horizon h, and $0 \leq \alpha(h) \leq 1$.

There are two factors to consider in the combination:

1) For a short forecasting horizon, the combination should play more emphasis on the daily forecasting as the short-term model works better.

2) For a long forecast horizon, the performance of the daily model deteriorates. Therefore, the weighting of the daily model should decrease while the weekly model will dominate the combination.

Taking these two factors into consideration, $\alpha(h)$ can be defined as

$$\alpha(h) = \exp(-\theta h) \tag{29}$$

where θ is a small positive value controlling the influence of the short term. For example, $\theta = 0.05$ has been applied to all the experiments.

All the input vectors are used as input segments to train the daily model $f_d(h)$.

$$\mathbf{x}^{(1)}(\tau), \mathbf{x}^{(2)}(\tau), \cdots, \mathbf{x}^{(H)}(\tau) \tag{30}$$

Similarly, the weekly input vectors are used as inputs to train the weekly model $f_w(h)$.

$$\mathbf{x}_w^{(1)}(\tau) = [x_{\tau-(l-1)*5}, \cdots, x_{\tau-5}, x_\tau] \tag{31}$$

$$\mathbf{x}_w^{(2)}(\tau) = [x_{\tau-(l-1)*5}, \cdots, x_{\tau-5}, x_\tau, x_{\tau+5}] \tag{32}$$

$$\cdots \qquad \cdots$$

$$\mathbf{x}_w^{(h)}(\tau) = [x_{\tau-(l-1)*5}, \cdots, x_{\tau-5}, x_\tau, \cdots, x_{\tau+(h-1)*5}] \tag{33}$$

$$\cdots \qquad \cdots$$

$$\mathbf{x}_w^{(H)}(\tau) = [x_{\tau-(l-1)*5}, \cdots, x_{\tau-5}, x_\tau, \cdots, x_{\tau+(H-1)*5}] \tag{34}$$

With the trained daily and weekly models, $f_d(h)$, $f_w(h)$, the combined forecasting model is given as Eq. (28). The combination of short-term and long-term models can be extended to an ensemble of κ independent such VLM models. Assume the k-th VLM model has a h-step forecast $\hat{x}_{t+h}(k), k = 1, 2, \cdots, \kappa$, the final prediction by eVLM can be obtained as

$$\hat{x}(t+h) = \sum_{k=1}^{\kappa} \alpha_k(h) \hat{x}_{t+h}(k) \tag{35}$$

where $\alpha_k(h)$ is a function of horizon h and $\sum_{k=1}^{\kappa} \alpha_k(h) = 1$. The simplest version is simple averaging, i.e. $\alpha_1 = \alpha_2 = \cdots = \alpha_\kappa = 1/\kappa$. The number of independent models Z can be assigned to any positive integer, which is 5 in the experiments. To optimize the mixture model, $\alpha_k(h)$ can be set as $\alpha_k(h) = \frac{e^{-|k-h|}}{\sum_{k=1}^{\kappa} e^{-|k-h|}}$. The procedure of the eVLM model is summarized in Algorithm 1.

4. Experiments

The proposed VLM models and eVLM network have been evaluated and compared with various benchmark methods including the naive method,[38] exponential smoothing (ES),[39] regressive models and various neural networks, including support vector regression (SVR)[8] and the echo state network (ESN),[19] on several benchmark datasets, FX rates and weather data.

4.1. *Synthetic data*

The Mackey-Glass time series, a commonly used benchmark data, was evaluated on first. The second case study investigated on modeling and predicting the Lorenz series.

In these case studies, 10% of the training data was used as the validation set to terminate the training and to decide model orders.

There are several ways to measure prediction accuracy,[2,40] of which two mostly common ones were employed in the experiments. One is the normalized root-mean-square error (NRMSE)[32,33]

$$NRMSE = \sqrt{\frac{(1/N_{test})\sum_{t=1}^{N_{test}}(x_t - \hat{x}_t)^2}{(1/N_{test})\sum_{t=1}^{N_{test}}(x_t - \bar{x})^2}} \quad (36)$$

where N_{test} is the number of test examples, $\hat{x}_t$ and x_t are the predicted and actual values respectively, and $\bar{x}$ denotes the sample mean.

The Mackey-Glass time series was generated from the following time-delay differential equation,

$$\frac{dx_t}{dt} = \beta x_t + \frac{\alpha x_{t-\delta}}{1 + x_{t-\delta}^{10}} \quad (37)$$

where x_t is the sample time series at time t. An example of 2000 observations were generated with parameters set to $\alpha = 0.2$, $\beta = -0.1$ and $\delta = 17$, and the first 1800 points were used as the training set and the last 200 were employed as the test set.

The Lorenz time series were generated from the following system with parameters chosen as $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$

$$\frac{dx_t}{dt} = \sigma(y_t - x_t) \quad (38)$$

$$\frac{dy_t}{dt} = x_t(\rho - z_t) - y_t \quad (39)$$

$$\frac{dz_t}{dt} = x_t y_t - \beta z_t \quad (40)$$

where x_t, y_t and z_t are the values of time series at time t. An example of Lorenz time series, $x(t)$, of total 1000 points was generated. The first 900 data points were used for training and the rest for testing.

The actual and predicted test series on these two datasets are shown in Figs. 2 and 3. It is seen that the estimated values by the proposed method are closer to the actual values than others. The detailed NRMSE results (avergaed over H = 1, 2, ... 10) are presented in Tables 1 and 2. The compared benchmark methods performed multi-step forecast by the independent approach. Among these methods, the naive method had the highest NRMSE and ES had the similar performance to other neural networks. eVLM significantly outperformed all other multi-step forecasting methods with the lowest NRMSE. Although SOMAR and NGMAR performed better than other neural and learning methods, they were still inferior to eVLM.

The improved prediction performances are largely due to two factors. First, the probabilistic dependencies between neighboring time points are preserved and modeled in these VLMs and therefore beneficial for forecasting. Second, dynamic combination of short and long term models, as well as an ensemble of them in eVLM, further reduce the error in prediction in all prediction horizons. The proposed methods are effective and efficient for multi-step forecasting tasks.

4.2. *Foreign exchange rates*

The proposed multi-step method was also evaluated on foreign exchange (FX) rate time series. FX rates (GBP/USD, GBP/EUR and GBP/JPY), downloaded from the PACIFIC exchange rate services, were used in this case.

Table 1. Average multi-step prediction performance of various models on Mackey–Glass ($H = 1, 2, \cdots, 10$)

Model	NRMSE	p-value
Naive	0.1325	**3.3e-8**
ES	0.0683	**1.7e-5**
ARMA	0.1247	**1.2e-7**
GARCH	0.1205	**2.6e-7**
MLP	0.0958	**8.4e-6**
SVR	0.0697	**6.3e-5**
ESN	0.0672	**4.1e-5**
SOM	0.1043	**3.0e-6**
NG	0.0775	**1.9e-4**
SOMAR	0.0695	**3.4e-4**
GSOMAR	0.0674	**2.6e-4**
NGMAR	0.0651	**8.5e-4**
VLM	0.0592	**0.0031**
eVLM	**0.0513**	N/A

Table 2. Average multi-step prediction performance of various models on Lorenz ($H = 1, 2, \cdots, 10$)

Model	NRMSE	p-value
Naive	0.1362	**1.3e-10**
ES	0.0870	**5.7e-8**
ARMA	0.1327	**2.4e-10**
GARCH	0.1189	**6.2e-10**
MLP	0.0885	**1.5e-7**
SVR	0.0851	**3.5e-6**
ESN	0.0768	**7.7e-5**
SOM	0.0932	**8.4e-8**
NG	0.0763	**2.3e-5**
SOMAR	0.0684	**9.4e-5**
GSOMAR	0.0661	**4.7e-4**
NGMAR	0.0635	**2.8e-4**
VLM	0.0569	**5.4e-3**
eVLM	**0.0464**	N/A

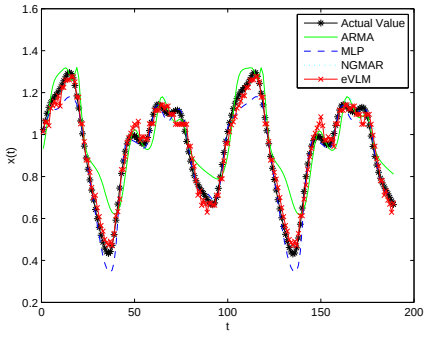

Figure 2. Actual and predicted test series for the Mackey-Glass time series.

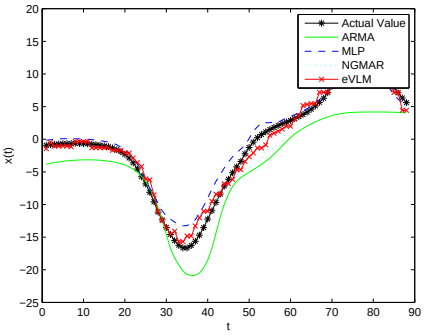

Figure 3. Actual and predicted test series for the Lorenz time series.

In addition to NRMSE, the correct prediction percentage (CPP) measure[33,41] was also adopted as a performance measure for making correct trend prediction (i.e. price going up or down). It is defined as,

$$CPP = \frac{Number\ of\ Correct\ Direction\ Predictions}{Total\ Number\ of\ Predictions} \tag{41}$$

The FX rate series used in the experiments include daily closing prices of GBP/USD, GBP/EUR and GBP/JPY over a period of 12 years (2004-2015). Each FX rate series was divided into training set (first 90% points) and test set (last 10% points).

Tables 3-5 show the averaged performances (NRMSE and CPP) over $H = 1, 2, \cdots, 10$ horizons and their t-test values. In each table, the first column represents the model used, the second and third columns are the NRMSEs and the corresponding p values of the t-test, and the fourth and fifth columns are the CPPs and their p values. As can be seen, the proposed methods significantly outperforms other methods in both NRMSE and CPP.

Fig. 4 plots the performance comparisons of various models including SVR,[8] ESN[19] and SOMAR[32] over all the horizons. The marked improvements of the proposed method over other methods on all horizons can be clearly seen.

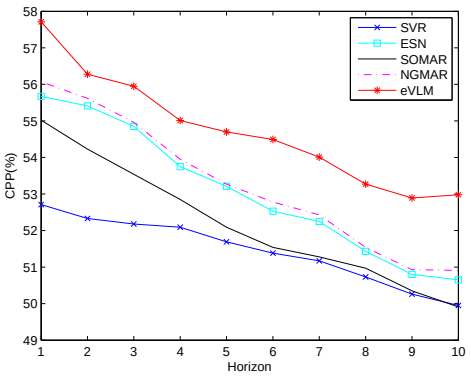

Figure 4. Performance comparisons (in CPP) of all the horizons by various models on GBP/USD.

Table 3. Multi-step prediction performance comparison of various models on GBP/USD ($h = 1, 2, \cdots, 10$)

Model	NRMSE	p-value	CPP(%)	p-value
Naive	0.3025	**1.7e-13**	49.37	**8.5e-14**
ES	0.1876	**2.5e-9**	51.75	**2.0e-8**
ARMA	0.2955	**1.6e-12**	49.68	**2.7e-12**
GARCH	0.2837	**2.7e-12**	49.92	**1.6e-12**
MLP	0.2367	**2.5e-10**	50.49	**8.5e-11**
SVR	0.1672	**8.7e-8**	51.43	**6.9e-8**
ESN	0.1353	**2.3e-4**	53.10	**8.4e-5**
SOM	0.2396	**4.9e-10**	50.22	**6.5e-12**
NG	0.1835	**3.5e-9**	50.63	**3.7e-10**
SOMAR	0.1533	**6.8e-7**	52.18	**2.0e-5**
GSOMAR	0.1418	**7.7e-7**	52.59	**1.1e-5**
NGMAR	0.1364	**8.1e-5**	53.25	**7.9e-5**
VLM	0.1193	**3.6e-4**	54.03	**1.5e-4**
eVLM	**0.1064**	N/A	**55.02**	N/A

Table 4. Multi-step prediction performance comparison of various models on GBP/EUR ($h = 1, 2, \cdots, 10$)

Model	NRMSE	p-value	CPP(%)	p-value
Naive	0.2055	**1.3e-11**	49.20	**8.1e-11**
ES	0.0931	**9.5e-7**	51.57	**3.9e-5**
ARMA	0.1964	**4.2e-10**	49.29	**7.8e-10**
GARCH	0.1882	**1.5e-10**	49.67	**3.4e-10**
MLP	0.1343	**8.6e-8**	50.14	**8.0e-9**
SVR	0.0878	**1.3e-6**	50.82	**1.9e-6**
ESN	0.0749	**5.3e-5**	52.22	**0.0011**
SOM	0.1402	**7.2e-9**	50.17	**7.3e-9**
NG	0.1085	**6.6e-8**	50.66	**6.5e-7**
SOMAR	0.0768	**4.9e-5**	51.39	**3.7e-4**
GSOMAR	0.0751	**6.5e-5**	51.84	**7.3e-4**
NGMAR	0.0740	**1.3e-4**	52.37	**0.0018**
VLM	0.0619	**0.0012**	52.84	**7.0e-4**
eVLM	**0.0526**	N/A	**53.61**	N/A

4.3. *Weather prediction competition*

The Weather Prediction Competition at International Joint Conference on Neural Networks IJCNN2015 (www.ijcnn.org) provided a good opportunity to test a prediction model on an extremely dynamic data (a local UK weather). The objective of this competition was to predict 5-day ahead a local weather data. It had attracted a number of entries to the competition, similar to the previous time series prediction competitions at the previous IJCNN.

The VLM method proposed in this paper was submitted to the competition and subsequently tested by the competition organizers. It was awarded as the Winner; and the result was announced at IJCNN2015 (http://www.ijcnn.org/assets/docs/ijcnn2015-awards.pdf). The experimental results are presented in Tables 4 and 5, in terms of the mean squared errors (MSE) in predicted maximum and minimum temperatures. The standard deviations of them are also presented, so are the t-test significance values (p-value).

Table 5. Multi-step prediction performance comparison of various models on GBP/JPY ($h = 1, 2, \cdots, 10$)

Model	NRMSE	p-value	CPP(%)	p-value
Naive	0.2183	**1.7e-11**	49.15	**2.3e-9**
ES	0.0974	**9.5e-7**	51.01	**5.6e-4**
ARMA	0.2131	**5.2e-11**	49.37	**7.3e-9**
GARCH	0.1992	**8.6e-11**	49.80	**1.6e-7**
MLP	0.1565	**1.1e-8**	50.13	**8.3e-7**
SVR	0.1067	**7.3e-8**	50.96	**1.3e-4**
ESN	0.0926	**8.4e-6**	51.54	**8.6e-4**
SOM	0.1509	**2.3e-8**	50.15	**6.9e-7**
NG	0.1326	**7.5e-8**	50.51	**3.2e-5**
SOMAR	0.1015	**9.6e-7**	51.29	**6.0e-5**
GSOMAR	0.0954	**5.3e-6**	51.40	**9.3e-5**
NGMAR	0.0912	**7.5e-6**	51.87	**0.0014**
VLM	0.0724	**0.0052**	52.40	**0.0065**
eVLM	**0.0617**	N/A	**53.11**	N/A

It is evident that the proposed methods outperform other neural networks as shown in the tables in terms of MSE and standard deviation (except ESN in some cases). The significant performance improvements show the importance of extracting probabilistic dependencies or relationship between multiple neighboring time points within the horizon in modeling and forecasting.

5. Discussions

In SOMAR or NGMAR, each node converges to an AR model in a mixture. When it is used for multistep prediction in the independent fashion, for a prediction horizon, H, each constructed AR model will be

$$x_{t+H} = w_{i,0}x_{t-l+1} + w_{i,1}x_{t-l+2} + \cdots + w_{i,l-1}x_t + \epsilon_{t+H} \tag{42}$$

where ϵ_{t+H} is white noise.

As can be seen, in this fashion, the model ignores the points between x_{t+H} and x_t. While in the eVLM models each horizon, h, within [1, H], and its equivalent AR model becomes

$$\begin{aligned} x_{t+h} = w_{i,0}x_{t-l+1} + \cdots + w_{i,l-1}x_t + \cdots \\ + w_{i,l+h-2}x_{t+h-1} + \epsilon_{t+h} \end{aligned} \tag{43}$$

and the output of the mixture model is

$$x_{t+h} = \sum_i \pi_i (w_{i,0}x_{t-l+1} + \cdots + w_{i,l+h-2}x_{t+h-1} + \epsilon_{t+h}) \tag{44}$$

That is, it uses all the available points, either observed or estimated. The latter are the equivalent estimated values by the lower horizon models but readily obtainable from neurons' weight vectors, containing the learned coefficients for all horizons. Moreover, the mixing is fully considered by employing the mixture of all component models rather than relying on only one local AR model. This naturally embeds regressive models of various horizons together and can enhance the predictions as demonstrated by the experimental results. Note that the mixing is carried out in a small neighborhood of the winner, therefore mixing weights are sparse comparing to the entire set of neurons.

6. Conclusions

In this paper, a generalized self-organizing mixture of autoregressive models of varied lengths has been introduced for multi-step prediction. It generalizes the previous self-organizing mixture of autoregressive models of the same order to a mixture of heterogeneous models and this further minimizes the accumulated errors along with various prediction steps. Such varied length models retain as much dependency information as possible among the points within the prediction horizon. These dependencies between consecutive future points serve the key role in improving the prediction accuracy. An ensemble of these varied length models can further reduce the prediction error and enhance the performance. The experiments conducted on various benchmark data, FX rates time series and weather data demonstrated the markedly improvements over the existing methods. The results also validate the efficiency of the proposed methods.

Table 6. Mean-squared-error (MSE) and standard deviation (σ) on weather data ($h = 1$)

Model	MinTmp	σ	p-value
MLP	4.37	0.57	**8.6e-6**
SVR	4.26	0.66	**3.1e-5**
ESN	4.05	0.51	**1.7e-4**
SOM	4.84	0.72	**8.5e-8**
NG	4.56	0.68	**2.3e-6**
VLM	3.97	1.51	**0.0012**
eVLM	**3.64**	**0.49**	N/A
Model	MinTmp	σ	p-value
MLP	4.97	0.71	**5.7e-6**
SVR	4.91	0.66	**8.3e-6**
ESN	4.85	**0.61**	**1.2e-4**
SOM	5.41	0.80	**2.6e-7**
NG	5.23	0.73	**9.4e-7**
VLM	4.84	0.82	**0.0028**
eVLM	**4.26**	0.62	N/A

Table 7. Mean-squared-error (MSE) and standard deviation (σ) on weather data ($h = 5$)

Model	MinTmp	σ	p-value
MLP	4.93	0.72	**2.0e-7**
SVR	4.76	0.75	**3.7e-6**
ESN	4.38	**0.64**	**1.4e-5**
SOM	4.96	0.79	**3.9e-7**
NG	4.72	0.82	**9.5e-7**
VLM	4.19	0.75	**8.3e-5**
eVLM	**3.85**	0.66	N/A
Model	MaxTmp	σ	p-value
MLP	5.77	0.85	**2.0-7**
SVR	5.53	0.84	**5.5e-6**
ESN	5.24	0.70	**8.7e-6**
SOM	5.98	0.77	**6.3e-9**
NG	5.51	0.73	**4.4e-6**
VLM	5.03	0.76	**1.6e-4**
eVLM	**4.42**	**0.68**	N/A

The relationship between consecutive future points serve the key role in improving the prediction accuracy. An ensemble of these varied length models can further reduce the prediction error and en-

hance the performance. The experiments conducted on various benchmark data, FX rates time series and weather data demonstrated the markedly improvements over the existing methods. The results also validate the efficiency of the proposed methods.

Bibliography

1. B. D. Fulcher and N. S. Jones, Highly comparative feature-based time-series classification, *IEEE Transactions on Knowledge and Data Engineering* **26**(12) (2014) 3026-3037.
2. J. G. De Gooijer and R. J. Hyndman, 25 years of time series forecasting, *International Journal of Forecasting* **22** (2006) 443-473.
3. A. Sorjamaa, J. Hao, N. Reyhani, Y. Ji and A. Lendasse, Methodology for long-term prediction of time series, *Neurocomputing* **70** (2007) 2861-2869.
4. L. Zhang, W. -D. Zhou, P. -C. Chang, J. -W. Yang and F. -Z. Li, Iterated time series prediction with multiple support vector regression models, *Neurocomputing* **99** (2013) 411-422.
5. M. Marcellino, J. H. Stock and M. W. Watson, A comparison of direct and iterated multistep AR methods for forecasting macroeconomic time series, *Journal of Econometrics* **135** (2016) 499-526.
6. J. L. Rossello, M. L. Alomar, A. Morro, A. Oliver and V. Canals, High-density liquid-state machine circuitry for time-series forecasting, *International Journal of Neural Systems* **26**(5)(2016) 1-12.
7. H. Siqueira, L. Boccato, R. Attux, and C. Lyra, Unorganized machines for seasonal streamflow series forecasting, *International Journal of Neural Systems* **24**(3)(2014).
8. C. Cortes and V. Vapnik, Support-vector networks, *Machine Learning* **20**(3) (1995) 273-297.
9. T. V. Gestel, J. A. K. Suykens, D. E. Baestaens, A. Lambrechts, G. Lanckriet, B. Vandaele, B. De Moor and J. Vandewalle, Financial time series prediction using least squares support vector machines within the evidence framework, *IEEE Transactions on Neural Networks* **12**(4) (2001) 809-821.
10. S. Haykin, Neural Networks: A Comprehensive Foundation, 2004.
11. X. Peng, TSVR: An efficient Twin Support Vector Machine for regression, *Neural Networks* **23**(3) (2010) 365-372.
12. M. I. Szeliga, P. F. Verdes, P. M. Granitto and H. A. Ceccatto, Artificial neural network learning of nonstationary behavior in time series, *International Journal of Neural Systems* **13**(2) (2003) 103-109.
13. D. S. P. Salazar, P. J. L. Adeodato and A. L. Arnaud, Continuous Dynamical Combination of Short and Long-Term Forecasts for Nonstationary Time Series, *IEEE Transactions on Neural Networks and Learning Systems* **25** (2014) 241-246.
14. K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* **2**(5) (1989) 359-366.
15. M. Asaduzzaman, M. Shahjahan and K. Murase, Faster training using fusion of activation functions for feed forward neural networks, *International Journal of Neural Systems* **19**(6) (2009) 437-448.
16. I. M. Galvan, P. Isasi, R. Aler and J. M. Valls, A selective learning method to improve the generalization of multilayer feedforward neural networks, *International Journal of Neural Systems* **11**(2) (2001) 167-177.
17. A. G. Parlos, O. T. Rais and A. F. Atiya, Multi-step-ahead prediction using dynamic recurrent neural networks, *Neural Networks* **13** (2000) 765-786.
18. C. L. Giles, S. Lawrence and A. C. Tsoi, Noisy time series prediction using recurrent neural networks and grammatical inference, *Machine Learning* **44** (2001) 161-183.
19. H. Jaeger, Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach, Technical Report, Fraunhofer Institute AIS, St. Augustin-Germany, 2002.
20. S. F. Crone, M. Hibon and K. Nikolopoulos, Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction, *International Journal of Forecasting* **27**(3) (2011) 635-660.
21. D. Li, M. Han and J. Wang, Chaotic time series prediction based on a novel robust echo state network, *IEEE Transactions on Neural Networks and Learning Systems* **23**(5) (2012) 787-799.
22. L. Tian and A. Noore, Software reliability prediction using recurrent neural network with Bayesian regularization, *International Journal of Neural Systems* **14**(3) (2004) 165-174.
23. G. Simon, A. Lendasse, M. Cottrell, J. C. Fort and M. Verleysen, Time series forecasting: Obtaining long term trends with self-organizing maps, *Pattern Recognition Letters* **26**(12) (2005) 1795-1808.
24. H. Yin, Self-organising maps: Background, theories, extensions and applications, in *Computational Intelligence: A Compendium* (Springer, 2008), pp. 715-762.
25. T. Kohonen, Self-Organizing Maps, Berlin, Springer, 1995.
26. T. M. Martinetz, S. G. Berkovich and K. J. Schulten, 'Neural-gas' network for vector quantization and its application to time-series prediction, *IEEE Transactions on Neural Networks* **4** (4) (1993) 558-569.
27. J. Lampinen and E. Oja, Self-Organizing maps for spatial and temporal AR models, in *Proc. 6th SCIA*, Helsinki, Finland, 1989.
28. G. J. Chappell and J. G. Taylor, The temporal Kohonen map, *Neural Networks* **6**(3) (1993) 441-445.
29. T. Koskela, M. Varsta, J. Heikkonen and K. Kaski, Time series prediction using recurrent SOM with local linear models, *International Journal of Knowledge-Based Intelligent Engineering Systems* **2**

(1997) 60-68.
30. T. Voegtlin, Recursive self-organizing maps, *Neural Networks* **15**(8-9) (2002) 979-991.
31. H. Ni and H. Yin (2008), Self-organising mixture autoregressive model for non-stationary time series modelling, *International Journal of Neural Systems* **18**(6) (2008) 469-480.
32. H. Ni and H. Yin, A self-organizing mixture autoregressive network for FX time series modelling and prediction, *Neurocomputing* **72**(16-18) (2009) 3529-3537.
33. Y. Ouyang and H. Yin, A neural gas mixture autoregressive network for modelling and forecasting FX time series, *Neurocomputing* **135** (2014) 171-179.
34. H. Yin and N. M. Allinson, Self-organizing mixture networks for probability density estimation, *IEEE Transactions on Neural Networks* **12**(2) (2001) 405-411.
35. C. S. Wong and W. K. Li, On a mixture autoregressive model, *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* **62**(1) (2000) 95-115.
36. H. Yin and H. Ni, Generalized Self-Organizing Mixture Autoregressive Model for Modeling Financial Time Series, in *Proceedings of the International Conference on Artificial Neural Networks*, Limassol, Cyprus, 2009.
37. S. Ben Taieb, A. Sorjamaa and G. Bontempi, Multiple-output modeling for multistep-ahead time series forecasting, *Neurocomputing* **73** (2010) 1950-1957.
38. P. Goodwin, Using naive forecasts to assess limits to forecast accuracy and the quality of fit of forecasts to time series data, (October 26, 2014). Available at SSRN: http://ssrn.com/abstract=2515072.
39. J. F. Muth, Optimal properties of exponentially weighted forecasts, *Journal of the American Statistical Association* **55**(290) (1960) 299-306.
40. R. J. Hyndman and A. B. Koehler, Another look at measures of forecast accuracy, *International Journal of Forecasting* **22** (2006) 679-688.
41. A. Skabar, Direction-of-change financial time series forecasting using a similarity-based classification model, *Journal of Forecasting* **32**(5) (2013) 409-422.