Statistics and Computing **15**: 13–29, 2005

Bayesian neural networks for nonlinear time series forecasting

FAMING LIANG

Department of Statistics, Texas A&M University, College Station, TX 77843-3143, USA
fliang@stat.tamu.edu

Received April 2002 and accepted May 2004

In this article, we apply Bayesian neural networks (BNNs) to time series analysis, and propose a Monte Carlo algorithm for BNN training. In addition, we go a step further in BNN model selection by putting a prior on network connections instead of hidden units as done by other authors. This allows us to treat the selection of hidden units and the selection of input variables uniformly. The BNN model is compared to a number of competitors, such as the Box-Jenkins model, bilinear model, threshold autoregressive model, and traditional neural network model, on a number of popular and challenging data sets. Numerical results show that the BNN model has achieved a consistent improvement over the competitors in forecasting future values. Insights on how to improve the generalization ability of BNNs are revealed in many respects of our implementation, such as the selection of input variables, the specification of prior distributions, and the treatment of outliers.

Keywords: Bayesian model averaging, Bayesian neural network, evolutionary Monte Carlo, Markov Chain Monte Carlo, nonlinear time series forecasting

1. Introduction

Let y_t denote a univariate time series modeled by

$$y_t = f(\boldsymbol{x}_t) + \epsilon_t, \quad t = 1, 2, \ldots, n,$$

where $f(\cdot)$ is an unknown function, $\boldsymbol{x}_t = (y_{t-1}, \ldots, y_{t-p})$ is a vector of lagged values of y_t, and $\{\epsilon_t\}$ is iid noise with mean 0 and finite variance σ^2. In the context of time series, ϵ_t is often called an innovation or disturbance, and p is called the order of the autoregressive model. The determination of the function $f(\cdot)$ has been one of central topics in statistics for a long time.

An ARIMA(p, d, q) model (Box and Jenkins 1970) assumes $f(\cdot)$ is of the form,

$$z_t = (1 - B)^d y_t,$$
$$f(\boldsymbol{x}_t) = c + \sum_{i=1}^{p} a_i z_{t-i} - \sum_{j=1}^{q} b_j \epsilon_{t-j},$$

where p, d, and q are nonnegative integers; c, a_i's and b_i's are parameters; and B is the backshift operator such that $By_t = y_{t-1}$. The model identification, parameter estimation and model adequacy checking can be carried out by an iterative procedure (Box and Jenkins 1970). Once an ARIMA model is built, forecasts of future values are simply the conditional expectations, $E(y_{t+l} \mid y_1, \ldots, y_t)$ for $l = 1, 2, \ldots$, which minimize the mean squared prediction errors.

The ARIMA model works well for linear time series, however, it is not adequate for nonlinear time series. Two popular nonlinear models are the bilinear model (Subba Rao and Gabr 1984) and the threshold autoregressive model (Tong 1990). The bilinear model of order (p, q, r, s) assumes that $f(\cdot)$ is of the form

$$f(\boldsymbol{x}_t) = \sum_{i=1}^{p} a_i y_{t-i} + \sum_{j=1}^{q} b_j \epsilon_{t-j} + \sum_{i=1}^{r} \sum_{j=1}^{s} c_{ij} y_{t-i} \epsilon_{t-j}.$$

The threshold autoregressive model can be regarded as a piecewise linear model in which the linear relationship varies with the values of the process. For example, a self-exciting threshold autoregressive model SETAR$(k; p_1, \ldots, p_k)$ (Tong and Lim 1980) is defined by k different linear functions

$$f(\boldsymbol{x}_t) = \sum_{j=1}^{p_i} a_j^{(i)} y_{t-j}, \quad y_{t-d} \in R_i, \quad i = 1, \ldots, k,$$

where d is a fixed integer between 1 and $\max(p_1, p_2, \ldots, p_k)$, and R_i, $i = 1, \ldots, k$ form a partition of the real line, i.e., $\mathcal{R} = R_1 \cup R_2 \cup \ldots \cup R_k$. For these two nonlinear models, the model identification and parameter estimation can be carried out by, for example, maximizing the likelihood function.

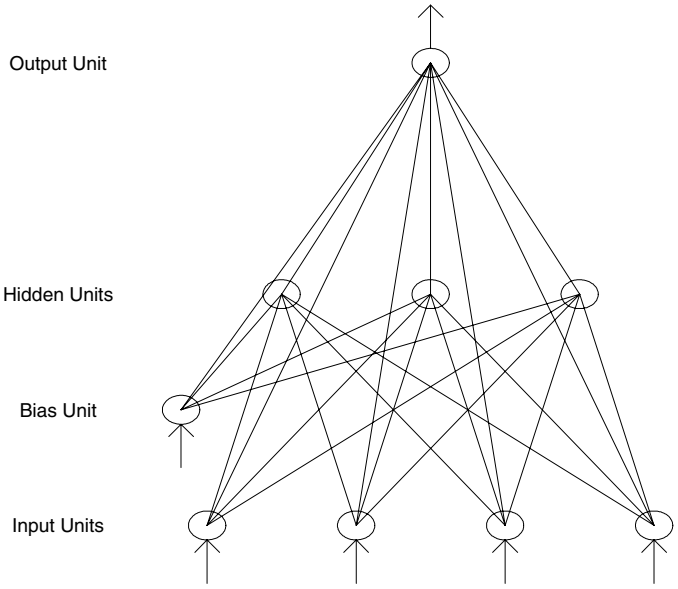

Fig. 1. *A fully connected one-hidden-layer feed-forward neural network with 4 input units, 3 hidden units and 1 output unit. The arrows indicate the direction of data feeding, where each hidden unit independently processes the values fed to it by units in the preceding layer and then presents its output to units in the next layer for further processing*

Although these models generally perform well, they have inherent limitations. First, without expertise it is possible to misspecify the function form of the most suitable model. Second, the models themselves are limited and may not be able to capture some kinds of nonlinear behavior. To alleviate these limitations, neural networks have been applied to modeling nonlinear time series by many authors, for example, Kang (1991), Hill, O'Connor and Remus (1996), Faraway and Chatfield (1998), and Park, Murray and Chen (1996). A one-hidden-layer feed-forward neural network (illustrated by Fig. 1) approximates $f(\cdot)$ by a function of the form

$$\hat{f}(\boldsymbol{x}_t) = \alpha_0 + \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{j=1}^{M} \beta_j \psi\left(\gamma_{j0} + \sum_{i=1}^{p} \gamma_{ji} y_{t-i}\right), \quad (1)$$

where M is the number of hidden units; α_0 denotes the bias of the output unit, α_i denotes the weight on the shortcut connection from the ith input unit to the output unit; β_j denotes the weight on the connection from the jth hidden unit to the output unit; γ_{j0} denotes the bias of the jth hidden unit, γ_{ji} denotes the weight on the connection from the ith input unit to the jth hidden unit; and $\psi(\cdot)$ is the activation function of the hidden units. Sigmoid and hyperbolic tangent functions are two popular choices for the activation function. Neural networks have two important advantages over the traditional statistical models. First, neural networks are universal approximators in that a neural network with linear output units can approximate any continuous function arbitrarily well on a compact set by increasing the number of hidden units (Cybenko 1989, Funahashi 1989, Hornik, Stinchcombe and White, 1989). Second, neural networks are able to estimate nonlinear functions, extract residual nonlinear elements, and at least partially transform the input data if needed (Hill, O'Connor and Remus 1996 and references therein).

For a given network structure, the weights can be determined by minimizing the sum of squares of the within-sample one-step ahead forecast errors, namely,

$$E = \sum_{t=p}^{n} (y_t - \hat{f}(\boldsymbol{x}_t))^2. \quad (2)$$

To avoid overfitting, (2) is often altered by a regularization term to

$$E = \sum_{t=p}^{n} (y_t - \hat{f}(\boldsymbol{x}_t))^2 + \tau SS_w, \quad (3)$$

where SS_w is the sum of squares of all weights, and τ is a smoothing parameter which can be determined by a cross-validation procedure. The minimization can be accomplished by the back propagation (Rumelhart, Hinton and Williams 1986), conjugate gradient, or any other optimization method. To make a distinction from Bayesian neural networks (BNNs) (Mackay 1992, Neal 1996) described below, henceforth, we will call a neural network with weights determined by minimizing (2) or (3) a traditional neural network (TNN).

BNNs were first proposed by Mackay (1992) and further developed by a number of authors, including Neal (1996), Müller and Rios Insua (1998), Marrs (1998), Holmes and Mallick (1998), Freitas and Andrieu (2000), Andrieu, Freitas and Doucet (2001), and Penny and Roberts (1999, 2000). BNNs are different from TNNs in two respects. First, the structure of BNN is variable, while the structure of TNN is fixed. Typically, the number of hidden units of BNN is subject to a prior distribution (Müller and Rios Insua 1998). Second, BNNs are trained by sampling from the joint posterior of the network structure and weights by MCMC methods. It avoids the problem of local minimum convergence, which is often encountered in TNN training.

In this study, we apply BNNs to time series analysis, and propose a Monte Carlo algorithm for BNN training. In addition, we go a step further in BNN model selection by putting a prior on network connections instead of hidden units as done by other authors. This allows us to treat the selection of hidden units and the selection of input variables uniformly. The model is compared to a number of competitors, including the Box-Jenkins model, bilinear model, threshold autoregressive model, and TNN model, on a number of popular and challenging data sets. Numerical results show that our BNN model has achieved a consistent improvement over the competitors in forecasting future values. In addition, insights on how to improve the generalization ability of BNNs are revealed in many respects of our implementation, such as the selection of input variables, the specification of prior distributions, and the treatment of outliers.

The remaining part of this article is organized as follows. In Section 2, we introduce the BNN model. In Section 3, we

describe the evolutionary Monte Carlo algorithm (Liang and Wong 2001) which will be used to sample from the posterior distribution of the BNN model. In Section 4, we present our numerical results on three time series data sets. In Section 5, we conclude the paper with a brief discussion.

2. Bayesian neural networks

2.1. *Bayesian neural network models*

In our BNN model, each connection is associated with an indicator function which indicates the effectiveness of the connection. The model can be written as

$$\hat{f}(\boldsymbol{x}_t) = \alpha_0 I_{\alpha_0} + \sum_{i=1}^{p} y_{t-i}\alpha_i I_{\alpha_i} + \sum_{j=1}^{M} \beta_j I_{\beta_j} \psi\left(\gamma_{j0} I_{\gamma_{j0}} + \sum_{i=1}^{p} y_{t-i}\gamma_{ji} I_{\gamma_{ji}}\right), \quad (4)$$

where I_ζ is the indicator function associated with the connection ζ. In this article, we set $\psi(z) = \tanh(z)$ for all examples to ensure that the output of a hidden unit is 0 if all connections to the hidden unit from input units have been eliminated. Thus, the hidden unit can be eliminated from the network without any effect on the network outputs. This is not the case for the sigmoid function, which will return a constant of 0.5 if the input is zero. Extra work is needed to make the constant be absorbed by the bias term if we want to delete the hidden unit from the network.

Let Λ be the vector consisting of all indicators in equation (4). Note that Λ specifies the structure of the network. Let $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_p)$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)$, $\boldsymbol{\gamma}_j = (\gamma_{j0}, \ldots, \gamma_{jp})$, $\boldsymbol{\gamma} = (\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_M)$, and $\boldsymbol{\theta} = (\boldsymbol{\alpha}_\Lambda, \boldsymbol{\beta}_\Lambda, \boldsymbol{\gamma}_\Lambda, \sigma^2)$, where $\boldsymbol{\alpha}_\Lambda$, $\boldsymbol{\beta}_\Lambda$ and $\boldsymbol{\gamma}_\Lambda$ denote the non-zero subsets of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, respectively. Thus, the model (4) is completely specified by the tuple $(\boldsymbol{\theta}, \Lambda)$. In the following we will denote a BNN model by $(\boldsymbol{\theta}, \Lambda)$, and re-denote $\hat{f}(\boldsymbol{x}_t)$ by $\hat{f}(\boldsymbol{x}_t, \boldsymbol{\theta}, \Lambda)$. To conduct a Bayesian analysis for model (4), we specify the following prior distributions: $\alpha_i \sim N(0, \sigma_\alpha^2)$ for $i = 0, \ldots, p$, $\beta_j \sim N(0, \sigma_\beta^2)$ for $j = 1, \ldots, M$, $\gamma_{ji} \sim N(0, \sigma_\gamma^2)$ for $j = 1, \ldots, M$ and $i = 0, \ldots, p$, $\sigma^2 \sim IG(\nu_1, \nu_2)$, where σ_α^2, σ_β^2, σ_γ^2 and λ are hyper-parameters to be specified by users (discussed below). The total number of effective connections is $m = \sum_{i=0}^{p} I_{\alpha_i} + \sum_{j=1}^{M} I_{\beta_j}\delta(\sum_{i=0}^{p} I_{\gamma_{ji}}) + \sum_{j=1}^{M}\sum_{i=0}^{p} I_{\beta_j} I_{\gamma_{ji}}$, where $\delta(z)$ is 1 if $z > 0$ and 0 otherwise. The model Λ is subject to a prior probability that is proportional to the mass put on m by a truncated Poisson with rate λ,

$$P(\Lambda) = \begin{cases} \dfrac{1}{Z}\dfrac{\lambda^m}{m!}, & m = 3, 4, \ldots, U, \\ 0, & \text{otherwise}, \end{cases}$$

where $U = (M+1)(p+1)+M$ is the number of connections of the full model in which all $I_\zeta = 1$, and $Z = \sum_{\Lambda\in\Omega} \lambda^m/m!$. Here we let Ω denote the set of all possible models with $3 \le m \le U$. We set the minimum number of m to three based on our views: neural networks are usually used for complex problems, and three has been a small enough number as a limiting network size. Furthermore, we assume that these prior distributions are independent *a priori*, and the innovation ϵ_t is distributed according to a normal distribution $N(0, \sigma^2)$. Thus, we have the following log-posterior (up to an additive constant),

$$\begin{aligned} &\log P(\boldsymbol{\theta}, \Lambda \mid \mathcal{D}) \\ &= \text{Constant} - \left(\frac{n}{2} + \nu_1 + 1\right)\log\sigma^2 - \frac{\nu_2}{\sigma^2} \\ &\quad - \frac{1}{2\sigma^2}\sum_{t=1}^{n}(y_t - \hat{f}(\boldsymbol{x}_t))^2 - \frac{1}{2}\sum_{i=0}^{p} I_{\alpha_i}\left(\log\sigma_\alpha^2 + \frac{\alpha_i^2}{\sigma_\alpha^2}\right) \\ &\quad - \frac{1}{2}\sum_{j=1}^{M} I_{\beta_j}\delta\left(\sum_{i=0}^{p} I_{\gamma_{ji}}\right)\left(\log\sigma_\beta^2 + \frac{\beta_j^2}{\sigma_\beta^2}\right) - \frac{1}{2}\sum_{j=1}^{M}\sum_{i=0}^{p} I_{\beta_j} I_{\gamma_{ji}} \\ &\quad \times\left(\log\sigma_\gamma^2 + \frac{\gamma_{ji}^2}{\sigma_\gamma^2}\right) - \frac{m}{2}\log(2\pi) + m\log\lambda - \log(m!). \quad (5) \end{aligned}$$

We note that an efficient way of sampling from (5) is to first simulate from the marginal distribution $P(\boldsymbol{\gamma}_\Lambda, \sigma^2, \Lambda \mid \mathcal{D})$, and then to sample $\boldsymbol{\alpha}_\Lambda$ and $\boldsymbol{\beta}_\Lambda$ conditional on the samples of $\boldsymbol{\gamma}_\Lambda$, σ^2, and Λ, since $\boldsymbol{\alpha}_\Lambda$ and $\boldsymbol{\beta}_\Lambda$ can be integrated out explicitly as illustrated in Dension *et al.* (2003). The theoretical basis of this sampling method is Rao-Blackwell's theorem (Casella and Berger, p. 342), which implies that the analytical integration is helpful in reducing the variance of Monte Carlo computation. In this article, we simulated from (5) directly, instead of simulating from $P(\boldsymbol{\gamma}_\Lambda, \sigma^2, \Lambda \mid \mathcal{D})$. Our sampling algorithm is more general. It is directly applicable to the cases where the activation function on the output unit is nonlinear and the innovations do not follow the normality assumption. In either of the above two cases, a closed form of $P(\boldsymbol{\gamma}_\Lambda, \sigma^2, \Lambda \mid \mathcal{D})$ is not available. In addition, our sampling algorithm avoids the problem of matrix inversion in computing $\boldsymbol{\alpha}_\Lambda$ and $\boldsymbol{\beta}_\Lambda$ in the stage of prediction.

For data preparation and hyperparameter settings, we have the following suggestions. To avoid some weights that are trained to be extremely large or small (in absolute value) to accommodate different scales of input and output variables, we suggest that all input and output variables be normalized before feeding to the networks. In all examples of this article, the data is normalized by $(y_t - \bar{y})/S_y$, where $\bar{y}$ and S_y denote the mean and standard deviation of the training data, respectively. Based on the belief that a network with a large weight variation usually has a poor generalization performance, we suggest that σ_α^2, σ_β^2, and σ_γ^2 should be set to moderate values to penalize a large weight variation. In this article, we set $\sigma_\alpha^2 = \sigma_\beta^2 = \sigma_\gamma^2 = 5$ for all examples. The setting should also be fine for the other problems. We put a non-informative prior on σ^2 and set $\nu_1 = \nu_2 = 0.05$ for all examples. The M and λ together control the network size. In practice, we can apply a cross-validation procedure to determine an optimal setting for them, or follow the suggestion of Weigend,

Huberman and Eumelhart (1990) to tune them to suitable values. Weigend, Huberman and Eumelhart (1990) suggest that the number of connections of a neural network should be about one tenth of the number of training patterns. In one of our examples, we assessed the influence of M and λ on network size, fitting, and forecasting performance. The numerical results suggest the performance of our BNN model is rather robust to the variation of M and λ, although the network size increases slightly as they increase.

Our BNN model is different from other BNN models existing in the literature in two important respects. First, in our BNN model, the input variables are selected automatically by sampling from the joint posterior of the network structure and weights. Second, the structure of our BNN model is usually sparse and its performance is less depending on the initial specification for the input patterns and the number of hidden units. It is sparse in the sense that only a small number of connections are active in the network. For the selection of input variables, a pioneer work has been done by Neal (1996) in the context of regression and classification, who specified the so-called automatic relevance determination (ARD) prior for weights. In the ARD prior, each input variable is associated with a hyperparameter that controls the magnitudes of the weights on the connections out of that input unit. Hence, the ARD prior works as a model selection mechanism for input variables with a high similarity to the indicator functions of our model. Comparing to the ARD prior, the indicator functions are more flexible. They control the effect of each connection separately, while the ARD prior controls the effects of all connections related with the same input unit jointly. Related works can also be found in Holmes and Dension (2002), which investigates the nonlinear basis function methods for the selection of input variables.

2.2. *Time series forecasting with BNN models*

Suppose that a series of samples, $(\boldsymbol{\theta}_1, \Lambda_1), \ldots, (\boldsymbol{\theta}_{\mathcal{N}}, \Lambda_{\mathcal{N}})$, have been drawn from the posterior (5). Let $\hat{y}_{t+1}$ denote the one step ahead forecast. One good choice of $\hat{y}_{t+1}$ is

$$\hat{y}_{t+1} = \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \hat{f}(\boldsymbol{x}_{t+1}, \boldsymbol{\theta}_i, \Lambda_i). \tag{6}$$

Following from standard MCMC results (Smith and Roberts 1993), we know

$$\hat{y}_{t+1} \to E(y_{t+1} \mid \mathcal{D}) \quad a.s., \quad \text{as } \mathcal{N} \to \infty.$$

However, for the multi-step case, it is not easy to obtain an unbiased forecast for neural networks or any other nonlinear models. In the following we give a multi-step ahead forecast which is unbiased and calculable with posterior samples. Since

$$\frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \hat{f}(\boldsymbol{x}_{t+h}, \boldsymbol{\theta}_i, \Lambda_i) \to E(y_{t+h} \mid \mathcal{D}), \quad a.s., \quad \text{as } \mathcal{N} \to \infty,$$

still holds for any $h > 1$, the resulting forecast will be unbiased and calculable if we replace the unknowns $y_{t+h-1}, \ldots, y_{t+1}$ included in $\boldsymbol{x}_{t+h}$ by their simulated realizations. A reasonable assumption is that $y_{t+1}, \ldots, y_{t+h-1}$ follow the same process than y_t. In Section 2.1, we have assumed that $y_t \sim N(f(\boldsymbol{x}_t), \sigma^2)$. An unbiased forecast at time $t+h$ can be generated in the following procedure.

1. For each sample $(\boldsymbol{\theta}_i, \Lambda_i)$, forecast y_{t+l} recursively for $l = 1, \ldots, h$ and repeat the process for $\mathcal{M}$ times, where

$$\begin{aligned} \hat{y}_{t+1}^{(i,j)} &= \hat{f}(y_{t-p+1}, \ldots, y_t, \boldsymbol{\theta}_i, \Lambda_i), \\ \hat{y}_{t+2}^{(i,j)} &= \hat{f}\big(y_{t-p+2}, \ldots, y_t, \hat{y}_{t+1}^{(i,j)} + e_{t+1}^{(i,j)}, \boldsymbol{\theta}_i, \Lambda_i\big), \\ &\vdots \end{aligned}$$

   where $(e_{t+1}^{(i,j)}, \ldots, e_{t+h-1}^{(i,j)})$, $i = 1, \ldots, \mathcal{N}$, $j = 1, \ldots, \mathcal{M}$, are future disturbances drawn from $N(0, \hat{\sigma}_i^2)$, where $\hat{\sigma}_i^2$ is an element of $\boldsymbol{\theta}_i$ and is itself an unbiased estimate of σ^2.
2. Average $\hat{y}_{t+h}^{(i,j)}$, $i = 1, \ldots, \mathcal{N}$, $j = 1, \ldots, \mathcal{M}$ to get the forecast

$$\hat{y}_{t+h}^{un} = \frac{1}{\mathcal{M}\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \sum_{j=1}^{\mathcal{M}} \hat{y}_{t+h}^{(i,j)}.$$

For a large value of $\mathcal{N}$, a reasonable choice of $\mathcal{M}$ is $\mathcal{M} = 1$ as used in this article.

Although $\hat{y}_{t+h}^{un}$ is unbiased, it often has a large variance due to the extra randomness introduced by the simulated future disturbances. In the following we propose an *ad hoc* forecast for y_{t+l} by setting the future disturbances to zero. That is,

1. For each sample $(\boldsymbol{\theta}_i, \Lambda_i)$, forecast y_{t+l} recursively for $l = 1, \ldots, h$ by the formula

$$\begin{aligned} \hat{y}_{t+1}^{(i)} &= \hat{f}(y_{t-p+1}, \ldots, y_t, \boldsymbol{\theta}_i, \Lambda_i), \\ \hat{y}_{t+2}^{(i)} &= \hat{f}\big(y_{t-p+2}, \ldots, y_t, \hat{y}_{t+1}^{(i)}, \boldsymbol{\theta}_i, \Lambda_i\big), \\ &\vdots \end{aligned}$$

2. Average $\hat{y}_{t+h}^{(i)}$, $i = 1, \ldots, \mathcal{N}$ to get the forecast

$$\hat{y}_{t+h}^{ad} = \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \hat{y}_{t+h}^{(i)}.$$

Although the forecast $\hat{y}_{t+h}^{ad}$ is biased, it often has a smaller mean squared prediction error (MSPE) than $\hat{y}_{t+h}^{un}$. The h-step MSPE for a general forecast $\hat{y}_{t+h}$ is defined as

$$\text{MSPE}(h) = \sum_{T=t}^{n-h} [y_{T+h} - \hat{y}_{T+h}]^2 / (n - h - t + 1),$$

which will be used to evaluate various forecasts in this article.

3. Computational implementation

Many Monte Carlo methods have been shown to be effective for training BNNs, for example, hybrid Monte Carlo (Neal 1996), reversible jump MCMC (Green 1995, Müller and Rios Insua 1998, Marrs 1998, Holmes and Mallick 1998, Andrieu, Freitas and Doucet 2000, 2001), sequential Monte Carlo (Freitas *et al*. 2001, Higdon, Lee and Bi 2002), and evolutionary Monte Carlo (Liang and Wong 2001). Although a comparison of the efficiencies of the above methods is interesting, it is not the focus of this article. In this article, we only want to show that BNN outperforms many competitors in forecasting nonlinear time series, although we also provide a Monte Carlo algorithm for BNN training. Theoretically, any Monte Carlo method, if it can sample from the posterior (5) effectively, will have the same forecasting performance as the evolutionary Monte Carlo (EMC) algorithm provided in this article.

EMC was originally designed for sampling from a distribution defined in a fixed dimensional space. In this article, we extend it to sample from a distribution defined in variable dimensional spaces. Specific mutation and crossover operators are designed for BNN models. The algorithm is described as follows.

Suppose that we want to sample from a distribution $f(\xi) \propto \exp(-H(\xi))$, where $H(\cdot)$ is called the energy function of ξ and it corresponds to the negative log-posterior in simulation from a posterior distribution. In EMC, a sequence of distributions $f_1(\xi), \ldots, f_N(\xi)$ are first constructed as follows: $f_i(\xi) \propto \exp\{-H(\xi)/t_i\}, i = 1, \ldots, N$, where t_i is called the temperature of $f_i(\cdot)$. The temperatures form a ladder $\boldsymbol{t} = (t_1, \ldots, t_N)$ with $t_1 > \cdots > t_N \equiv 1$. Issues related to the choice of the temperature ladder can be found in Liang and Wong (2000) and the references therein. Let ξ^i denote a sample from $f_i(\cdot)$. It is called an individual or a chromosome in terms of genetic algorithms. The N individuals, $\xi^1, \ldots, \xi^N$, form a population denoted by $\boldsymbol{z} = \{\xi^1, \ldots, \xi^N\}$, where N is called the population size. By assuming that all individuals of the same population are mutually independent, we have the following Boltzmann distribution for the population $\boldsymbol{z}$,

$$f(\boldsymbol{z}) \propto \exp\left\{-\sum_{i=1}^{N} H(\xi^i)/t_i\right\}. \tag{7}$$

The population is updated by mutation, crossover, and exchange operators (described in Appendix). The terms mutation and crossover root in genetic algorithms (Holland 1975, Goldberg 1989). Although they have a high similarity with those used in genetic algorithms, they have been modified such that they are reversible and usable as proposal functions for the Metropolis-Hastings algorithm (Metropolis *et al*. 1953, Hastings 1970). One iteration of EMC consists of the following steps.

- Apply the mutation or the crossover operator to the population with probability η and $1 - \eta$ respectively, where η is called the mutation rate.
- Try to exchange ξ^i with ξ^j for $N - 1$ pairs (i, j) with i being sampled randomly in $\{1, \ldots, N\}$ and $j = i \pm 1$ with probability $w_{i,j}$, where $w_{i,i+1} = w_{i,i-1} = 0.5$ for $1 < i < N$ and $w_{1,2} = w_{N,N-1} = 1$.

EMC is closely related with three algorithms, parallel tempering (Geyer 1991, Hukushima and Nemoto 1996), reversible jump MCMC (Green 1995), and the genetic algorithm (Holland 1975, Goldberg 1989). EMC roots in the genetic algorithm, but the acceptance of the updates are guided by the Metropolis-Hastings rule. So it falls into the class of MCMC methods. Both EMC and parallel tempering are multiple chain MCMC algorithms. The only significant difference between them is that EMC prescribes more general moves, such as swapping of individual parameters (partial state) between different chains, rather than simply swapping all parameters (whole state) of the two chains. But the extra moves have significantly accelerated the mixing of the Markov chain as shown later. We note that if $\eta = 1$, that is, only the mutation operator is performed, EMC is reduced to parallel tempering. If $\eta = 1$ and $N = 1$, EMC is reduced to reversible jump MCMC.

In EMC, the user-specified parameters include the population size N, mutation rate η, Metropolis step size κ, and temperature ladder $\boldsymbol{t}$. For all examples of this article, we set $N = 20$, $\eta = 0.6$, $\kappa = 0.25$, the highest temperature $t_1 = 20$, the lowest temperature $t_N = 1$, and the intermediate temperatures are equally spaced in inverse between t_1 and t_N. We choose the Metropolis step size such that the mutation operation has an acceptance rate ranged from 0.2 to 0.4 as suggested by Gelman, Roberts and Gilks (1996). We choose a slightly large value of η, since we notice that the crossover operation usually has a low acceptance rate, for example, it is around 0.065 for the Canadian lynx example. In general, the crossover operator provides a more global move in the parameter space, and thus, significantly accelerates the mixing of the Markov chain.

4. Time series forecasting

4.1. *Wolfer sunspot numbers*

We consider the annual sunspot numbers for the years 1700–1955 (Waldmeirer 1961). This data has been used by many authors to illustrate various time series models, for example, the ARIMA model (Box and Jenkins 1970), SETAR model (Tong and Lim 1980, Tong 1990), bilinear model (Gabr and Subba Rao 1981), and neural network model (Park, Murray and Chen 1996).

We follow Tong and Lim (1980) and Gabr and Subba Rao (1981) to use the first 221 observations for model building, and the next 35 observations for forecasting. To determine the input pattern for the BNN model, we first plot the partial autocorrelation function (PACF) of the training data. As suggested by Chatfield (2001, p. 223), the time plot and the PACF graph might give us some indication as to what lagged variables should be included as inputs. For example, we may include lag 12 for

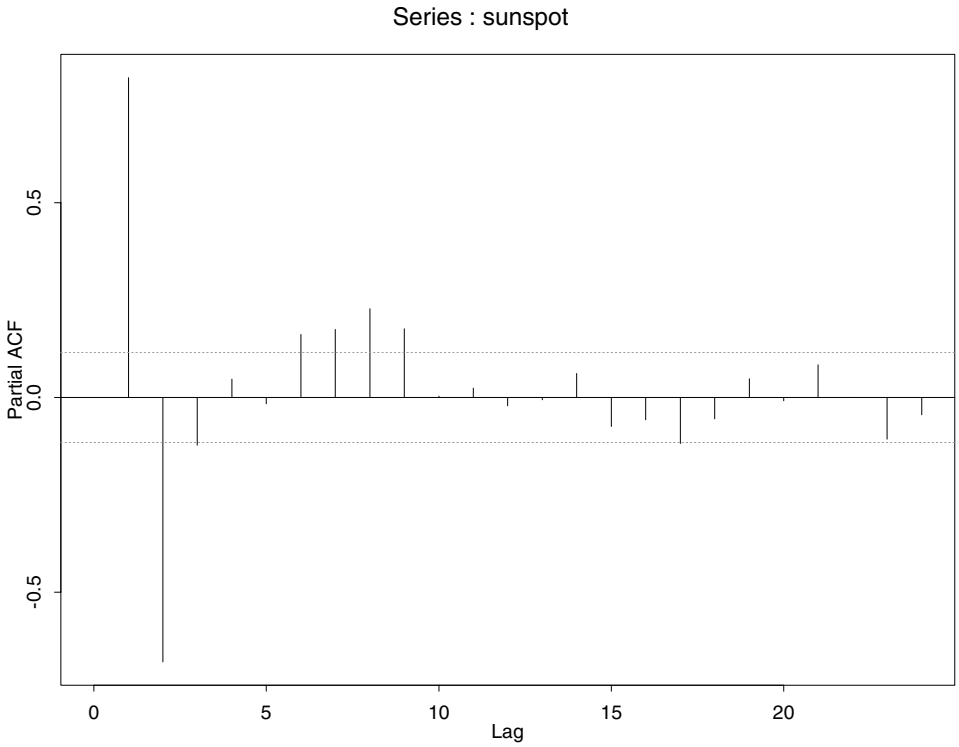

Fig. 2. *The PACF of the training data of the sunspot example*

seasonal data with large autocorrelation at lag 12. Figure 2 suggests $(y_{t-9}, \ldots, y_{t-1})$ be probably appropriate as an input pattern for this example. The later numerical results show that this input pattern is appropriate for this examle. The input patterns determined by this method are also appropriate for the other examples of this article.

4.1.1. Efficiency test

We first test the efficiency of EMC via comparisons with parallel tempering and reversible jump MCMC. In this experiment, we set the hyperparameter $\lambda = 10$ and the maximum number of hidden units $M = 5$. The full model (with 9 input units, 5 hidden units and 1 output unit) consists of 65 connections which should be large enough for this data set. In EMC, the weights of the BNN model were initialized as follows. They were first set to some random numbers drawn from the uniform distribution unif $(-0.1, 0.1)$, and then were updated for 500 iterations by the "Metropolis" moves. This produced a reasonable initial guess for the weights. Note that in the 500 iterations, no "death" or "birth" moves were performed. After the initialization process, the algorithm was run for 9100 iterations, and 700 samples were collected at the lowest temperature level with an equal time space. The CPU time used by the run was 44 s on a 2.8 GHZ computer (the same computer was used for all simulations of this paper). The overall acceptance rates of the mutation, crossover, and exchange moves are 0.35, 0.03, and 0.53, respectively. Although the acceptance rate of the crossover moves is low, it has made a significant improvement over its close relative, parallel tempering, in mixing of the Markov chain. Global moves often have a low acceptance rate in MCMC simulations. To reduce the huge within-run variance of the log-posterior values, we discarded the first 200 samples and used the following 500 samples to diagnose the convergence of the simulation. In parallel tempering, the weights of the BNN model were initialized as in EMC, and then the algorithm was run for 7000 iterations within the same CPU time as that used by EMC. Totally, 700 samples were collected in the same way as in EMC. The overall acceptance rates of the mutation and exchange moves are 0.35 and 0.53, respectively. As for EMC, we also discarded the first 200 samples, and used the remaining 500 samples to diagnose the convergence of the simulation. Note on average one iteration of parallel tempering costs a longer CPU time than that of EMC. This is because each individual is updated once in every iteration of parallel tempering, while only 40 percents of individuals are updated in one iteration of EMC if the crossover operation is performed. In reversible MCMC, the parameters were initialized as in EMC and then the algorithm was run for 140000 iterations within the same CPU time. Totally, 700 samples were collected in the same way as in EMC. The overall acceptance rate of the mutation moves is 0.28, which suggests that the algorithm has been implemented efficiently. Figure 3 shows ten convergence paths of the Gelman-Rubin statistic $\hat{R}$ (Gelman and Rubin 1992) for each algorithm, where each path was computed

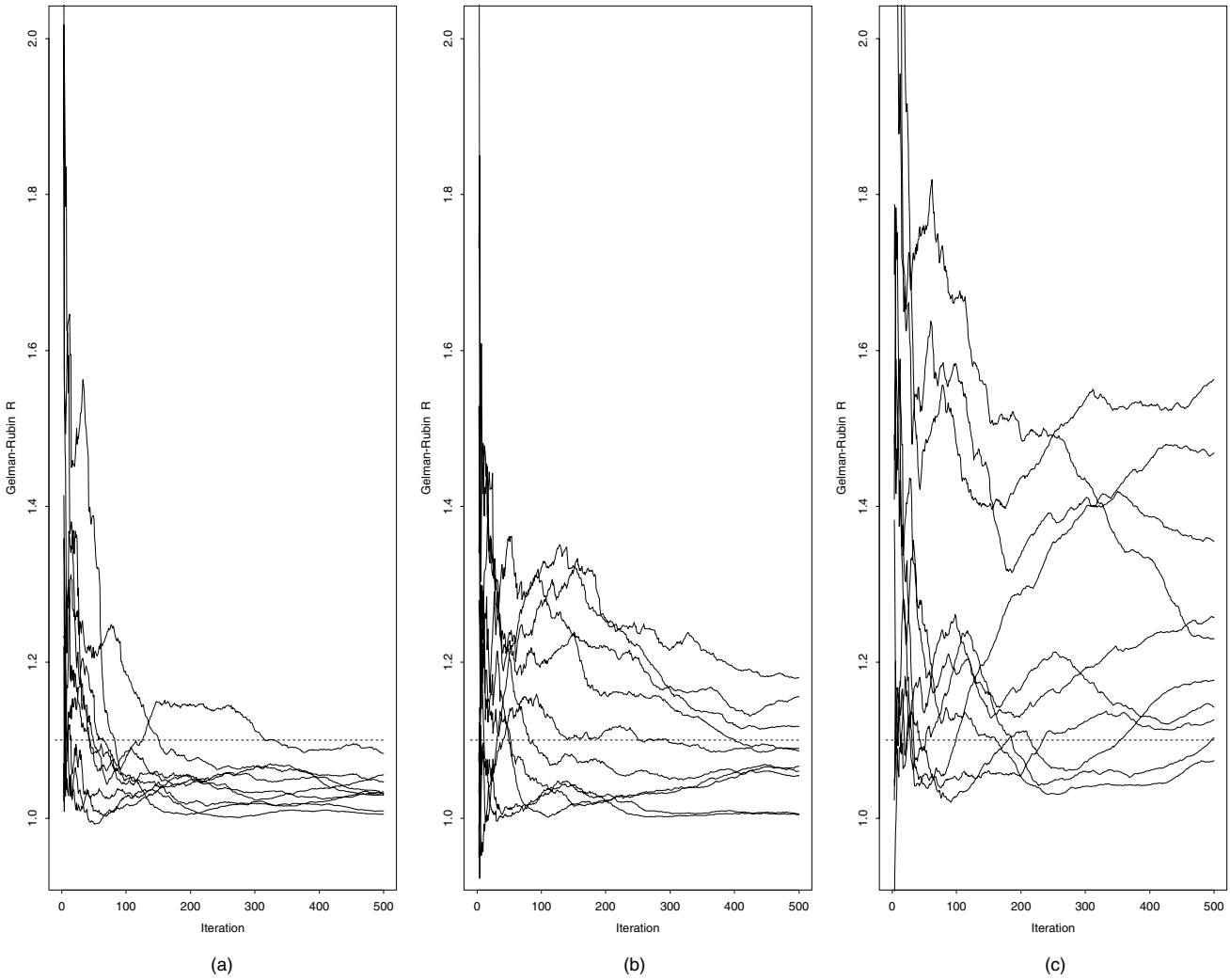

Fig. 3. *Convergence paths of the Gelman-Rubin statistic $\hat{R}$, where each path is computed with 5 independent runs. (a) EMC; (b) parallel tempering; (c) reversible jump MCMC*

with 5 independent runs (a total of 50 runs were made for each algorithm). In Fig. 3, we see that EMC has made a significant improvement in mixing of the Markov chain over both parallel tempering and reversible jump MCMC. For this example, EMC converges ($\hat{R} < 1.1$) very fast, usually within the first several thousands of iterations.

4.1.2. Parameter setting

In this section, we demonstrate how the optimal setting of M and λ can be determined by a cross-validation procedure for the BNN model. The training data was partitioned into two parts. The first part consists of the first 200 observations for model training and the second part consists of the next 21 observations for model validation. All cross settings were tried with $M = 4, 5, 6$ versus $\lambda = 5, 15, 25, 35$. For each setting, EMC was run for 10 times independently. Each run consists of 18000 iterations, where the first 500 iterations were used for the initialization process, the following 4500 iterations were discarded for the burn-in process, and the last 13000 iterations were used for inference. A total of 1000 samples were collected from the last 13000 iterations at the lowest temperature level with an equal time space. The CPU time used by a single run was about 78 s. The computational results are summarized in Table 1, which shows MSPE(1)'s and MSPE(2)'s for various settings of M and λ. Since the neural network model generally performs well in the later period of the forecast horizon (Hill *et al.* 1996, Kang 1991), we only need to control its performance in the early period by choosing suitable parameter values. For this example, the setting of $M = 5$ and $\lambda = 25$ was chosen by minimizing the sum of MSPE(1) and MSPE(2) in Table 1. The same procedure was also applied to other examples of this article. For simplicity, in the examples below we only report the chosen settings of M and λ, and omit the details of the cross-validation procedure.

Table 1. *Cross-validation results for BNN models: MSPE (1), MSPE (2) and their standard errors (the numbers in the parentheses) for the sunspot example, where each value was calculated based on 10 independent runs*

λ	5	15	25	35
$M = 4$	258.7 (3.7)	237.9 (8.0)	245.0 (7.7)	232.8 (6.3)
	255.0 (4.2)	239.5 (6.6)	262.8 (13.1)	255.8 (11.3)
$M = 5$	256.4 (5.4)	234.9 (4.2)	**231.1** (7.6)	238.1 (7.6)
	252.5 (8.1)	242.6 (9.9)	**242.3** (11.3)	260.7 (11.6)
$M = 6$	252.2 (5.5)	232.9 (5.4)	237.8 (7.9)	235.6 (6.5)
	263.6 (11.8)	248.8 (6.5)	265.3 (17.9)	257.5 (8.0)

4.1.3. Forecasting performance

The BNN model was applied to the above data set with $M = 5$ and $\lambda = 25$. EMC was run for 10 times independently. Each run consists of 31000 iterations. The first 500 iterations were used for the initialization process, the following 4500 iterations were discarded for the burn-in process, and the last 26000 iterations were used for inference. A total of 2000 samples were collected from the last 26000 iterations at the lowest temperature level with an equal time space. The CPU time used by a single run was about 130s. The maximum *a posteriori* (MAP) neural network structure sampled by EMC is displayed in Fig. 4. The other computational results are summarized in Table 2 together with the results reported in the literature.

For comparison, we first consider the Bayesian AR model averaging approach. Liang, Truong and Wong (2001) proposed an automatic prior setting for linear regression. They showed that the automatic prior results in an improved prediction performance over other priors, such as those used in Raftery, Madigan, and Hoeting (1997) and Fernández, Ley and Steel (2001). In addition, they showed that the resulting MAP model coincides with the minimum C_p model (Mallows 1973). Here we adopt the program and apply it to this data set with the maximum lag value $p = 9$ as in the BNN model. The program was run for 5 times independently. In each run 5000 samples were generated and were then used for prediction. This is the same for the other two examples of this article. The computational results are summarized in Table 2. Although the results will be different with different prior settings, more or less, the results presented here give us an impression of how the Bayesian AR model averaging approach performs on time series forecasting. See Barnett, Kohn and Sheather (1996) for an alternative method for Bayesian AR model estimation.

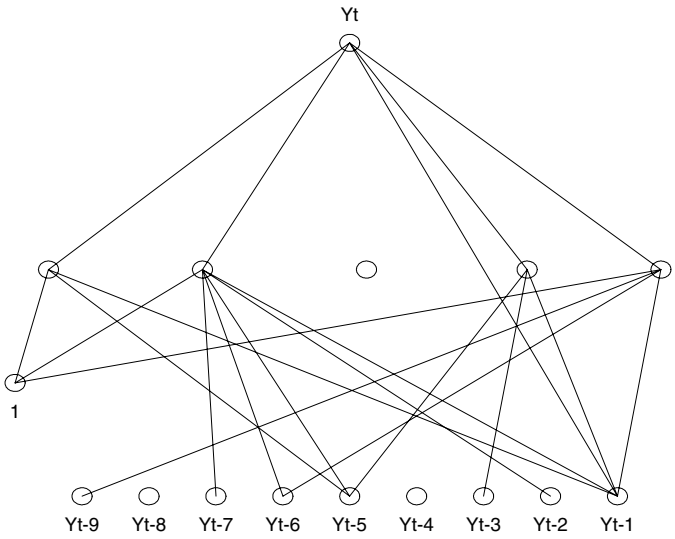

Fig. 4. *The maximum a posteriori (MAP) BNN structure sampled by EMC in a run with $M = 5$ and $\lambda = 25$*

The TNN model and a non-Bayesian non-neural nonparametric model were also applied to this data. The TNN software we used is available in Splus 6.1 by calling the command *nnet(·)*. To determine the number of hidden units and the smoothing parameter for the TNN model, a cross-validation experiment was also performed. The training data was partitioned in the same way as in the above cross-validation experiment for the BNN model. TNN models (with shortcut connections) were trained with all cross settings with $M = 2, 3, 4, 5, 6$ versus $\tau = 0$, 0.0001, 0.001, 0.01, 0.025, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5. The number of input units was fixed to 9 as in BNN models. For each setting, the TNN model was trained for 100 times independently with different starting values, and the averaged $MSPE(h)$'s were computed for $h = 1, \ldots, 6$. By minimizing $\sum_{h=1}^{6} MSPE(h)$, the setting $M = 6$ and $\tau = 0.5$ was chosen. We have also tried the criterion of minimizing $\sum_{h=1}^{2} MSPE(h)$ which was used for the BNN model. We found the models chosen by minimizing $\sum_{h=1}^{2} MSPE(h)$ are often overfitted to the training data and thus have a slightly worse forecasting performance than that reported in Table 2. This is also true for the other examples of this article. With the chosen setting, the TNN model was run for 100 times with the 221 observations as the training data. The forecasting performance on the last 35 observations is summarized in Table 2. The TNN models without shortcut connections were also tried. The results were not comparable with that shown in Table 2.

At last, we applied the kernel smoothing approach (Auestad and Tj∅stheim 1990, Härdle and Vieu 1992) to this data set. The s-step kernel predictor of y_{m+s} given $y_1, \ldots, y_m$ is defined by

$$\hat{R}_h(y_{m-p+1}, \ldots, y_m) = \frac{\sum_{i=p}^{m-s} y_{i+s} \prod_{j=p}^{1} K_h(y_{m-j+1} - y_{i-j+1})}{\sum_{i=p}^{m-s} \prod_{j=p}^{1} K_h(y_{m-j+1} - y_{i-j+1})},$$

where K_h is the Gaussian kernel, p is the order of the model, and the bandwidth h is selected to minimize

$$CV(h) = \frac{1}{T - p - s + 1} \sum_{i=p}^{T-s} [y_{i+s} - \hat{R}_i(y_{i-p+1}, \ldots, y_i)]^2,$$

Table 2. *Comparison of BNN and other time series models for the sunspot example, where MSE denotes the mean squared fitting error, "size" denotes the number of parameters, MSPE(h) denotes the mean squared h-step ahead prediction error for $h = 1, \ldots, 6$. In the BMA AR, BNN^{ad}, BNN^{un}, TNN columns, the numbers preceding to and included in the parentheses are the averaged MSPE (over 10 independent runs) and the standard errors of the average, respectively. BMA AR: Bayesian AR model averaging; BNN^{ad}: BNN models with the ad hoc predictor; BNN^{un}: BNN models with the unbiased predictor; TNN: traditional neural network models; Kernel: kernel smoothing approach. The results of Full AR, subset AR, SETAR, and bilinear models were taken from Gabr and Subba Rao (1981)*

Model	Full AR(9)	Subset AR	BMA AR	SETAR	Bilinear	BNN^{ad}	BNN^{un}	TNN	Kernel
MSE	199.27	203.21	201.43 (0.02)	153.71	124.33	124.74 (1.30)	124.74 (1.30)	162.94 (1.13)	78.51
size	10	4	3.56 (0.02)	19	11	27.18 (0.26)	27.18 (0.26)	76	–
MSPE(1)	190.89	214.1	211.07 (0.11)	148.205	123.77	142.03 (3.84)	142.87 (3.87)	171.99 (0.66)	76.80
MSPE(2)	414.83	421.4	414.26 (0.29)	383.9	337.54	347.85 (9.03)	346.44 (8.81)	407.71 (1.08)	317.30
MSPE(3)	652.21	660.38	646.14 (0.56)	675.59	569.79	509.60 (13.57)	513.99 (12.27)	607.18 (2.27)	549.14
MSPE(4)	725.85	716.08	698.23 (0.62)	773.51	659.047	482.21 (13.04)	521.47 (10.40)	615.52 (4.37)	779.73
MSPE(5)	771.04	756.39	736.81 (0.66)	784.27	718.866	470.35 (14.43)	561.94 (12.58)	617.24 (6.03)	780.16
MSPE(6)	–	–	756.98 (0.71)	–	–	468.18 (13.88)	577.79 (16.71)	578.15 (6.92)	736.03

with

$$\hat{R}_i(y_{i-p+1}, \ldots, y_i) = \frac{\sum_{k=p,k\neq i}^{T-s} y_{k+s} \prod_{j=p}^{1} K_h(y_{i-j+1} - y_{k-j+1})}{\sum_{k=p,k\neq i}^{T-s} \prod_{j=p}^{1} K_h(y_{i-j+1} - y_{k-j+1})}.$$

For this data set, we set $T = 221$. The value of p was determined by a cross-validation procedure as follows. With the same training data partition as for the BNN and TNN models, the settings $p = 1, \ldots, 12$ were tried. By minimizing either $\sum_{i=1}^{2} MSPE(i)$ or $\sum_{i=1}^{6}\sum_{i=1}^{6} MSPE(i)$, we always chose the same setting $p = 3$. The model was then re-trained with $p = 3$. The forecasting results are shown in Table 2.

Table 2 shows that BNN^{ad} is only inferior to the bilinear and kernel smoothing models for the one and two-step ahead forecasts. As the forecast horizon extends, it outperforms all other models. This result is consistent with the finding of Hill *et al.* (1996) and Kang (1991) that neural network models generally perform better in the latter period of the forecast horizon. We have three points to note for this table. First, BNN^{un} performs equally well as BNN^{ad} for the short term forecasts ($h \leq 3$). As the forecast horizon extends, the performance of BNN^{un} is deteriorated by the future disturbances. Comparing to the other unbiased predictors, full AR, subset AR, BMA AR, and kernel smoothing, it still has the best performance. The other two examples also suggest that BNN^{un} is worth of being highly recommended as an unbiased forecast. Second, for the one and two-step ahead forecasts, the kernel smoothing approach works the best, followed by the bilinear and BNN models. The predictors based on linear models perform less satisfactorily. When the true structure of a time series is complex, a simple parametric model may not be able to capture enough information for an accurate forecast. In this case, a suitable nonparametric model may perform well even for the short term forecasts. This point will be seen again in the next example. Third, neural network models usually have more parameters than any other linear or nonlinear parametric models. This is because the activation function is bounded and the effect of each connection on outputs is limited.

4.1.4. Sensitivity analysis

To assess the influence of λ on neural network size, fitting and forecasting performance, we fixed $M = 5$ and tried $\lambda = 5$, 10, 15, 20, 25, 30, and 35. For each value of λ, EMC was run for 10 times independently. The averaged network sizes, MSEs and MSPEs are shown in Fig. 5(a) and (b). Figure 5(a) shows the network size increases as λ increases. However, the increasing rate is decreasing as λ increases. Figure 5(b) shows in a wide range of λ, $10 \leq \lambda \leq 35$, BNN models have similar performances in fitting and forecasting. With $\lambda = 5$, the BNN model is clearly not sufficiently trained. The resulting fitting and forecasting performances are inferior to the other BNN models with large λ values. To assess the influence of M, we fixed $\lambda = 25$ and tried $M = 4$, 5, and 6. The results are summarized in Fig. 5(c) and (d). Figure 5(c) shows that the network size increases slightly as M increases even with the same λ. Figure 5(d) shows that all these three models have almost the same performances in fitting and short-term forecasting ($h < 3$), and the model with $M = 4$ is slightly inferior to the other two models in long-term forecasting ($h \geq 3$). This experiment suggests that a slightly large value of M is preferred in simulation.

To assess the influence of prior variances, we fixed $M = 5$ and $\lambda = 25$, and tried the settings with $\sigma_\alpha^2 = \sigma_\beta^2 = \sigma_\gamma^2 =$ 1, 2, 5, 10, and 100. For each setting, EMC was run for 10 times independently. The computational results are summarized in Fig. 6. It shows that the performance of BNN models is rather robust to the prior variance. However, the settings $\sigma_\alpha^2 = \sigma_\beta^2 = \sigma_\gamma^2 = 2$ and $\sigma_\alpha^2 = \sigma_\beta^2 = \sigma_\gamma^2 = 5$ perform the best. The reason is as follows. In general, a large prior variance results in a large posterior variance. Large weights usually have a strong effect on the network output. If the test data has a deviation from the training data, even if the deviation is very small, the network output will likely be far from the target as the deviation will be amplified or twisted by the large weights. On the other hand, if the

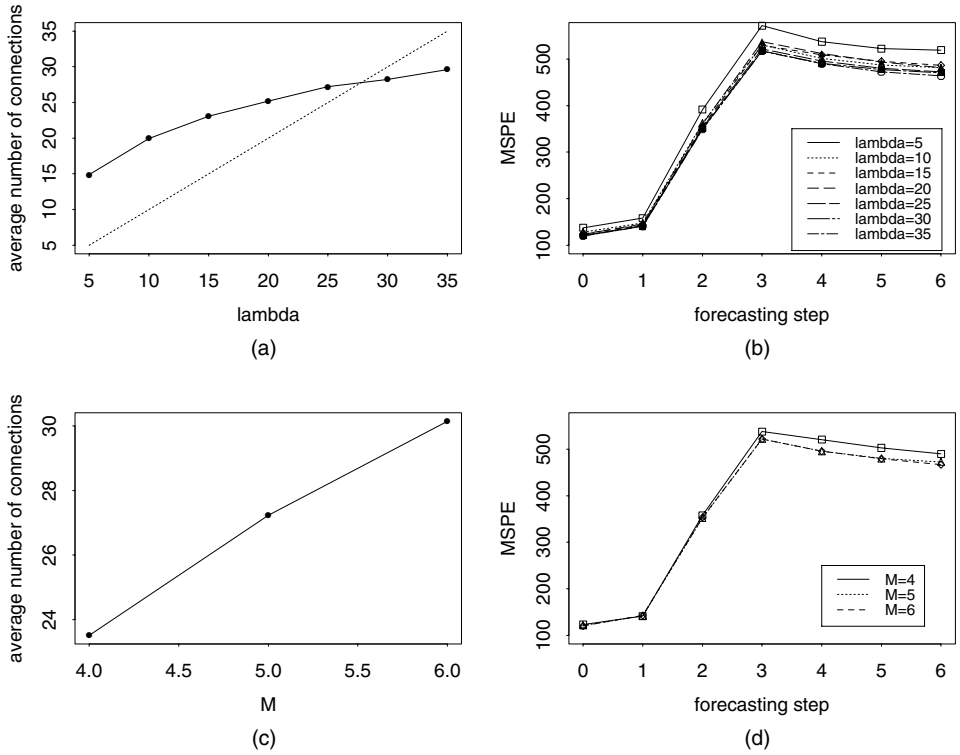

Fig. 5. *Assessment of the influence of λ and M on network size, and fitting and forecasting performance for BNN models, where MSPE(0) denotes the mean squared fitting error, i.e., MSE in Table 2*

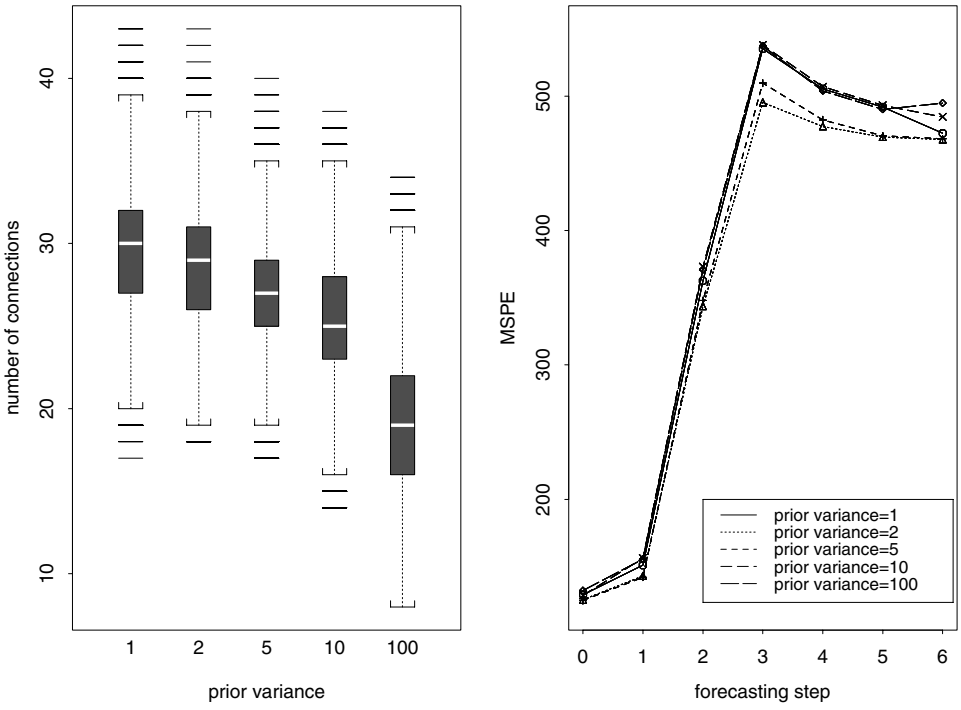

Fig. 6. *Assessment of the influence of prior variances on network size, and fitting and forecasting performance for BNN models, where MSPE(0) denotes the mean squared fitting error*

prior variance is too small, the data may not be able to be learned from sufficiently, and the resulting forecasting performance will also be poor. For this example, the small value $\sigma_\alpha^2 = \sigma_\beta^2 = \sigma_\gamma^2 = 1$ is as bad as the large value $\sigma_\alpha^2 = \sigma_\beta^2 = \sigma_\gamma^2 = 100$. This leads to our use of $\sigma_\alpha^2 = \sigma_\beta^2 = \sigma_\gamma^2 = 5$ for all examples of this article.

4.2. *Canadian lynx data*

This data set is the annual record of the numbers of Canadian lynx trapped in the Mackenzie river district of North West Canada for the years 1821–1934. It has a total of 114 observations. As the sunspot data, this data set has also been analyzed by many authors, including Tong and Lim (1980), Gabr and Subba Rao (1981), Nicholls and Quinn (1982), Lim (1987), and Tong (1990).

In this article, we follow Tong and Lim (1980) and Gabr and Subba Rao (1981) to consider the logarithmically transformed data (to the base 10), and use the first 100 observations for model building and the next 14 observations for forecasting. The PACF of the training data (Fig. 7(a)) shows that (y_{t-2}, y_{t-1}) or $(y_{t-11}, \ldots, y_{t-1})$ may be an appropriate input pattern for this example. Considering that the data set only consists of 100 observations, we suspect that the input pattern consisting of 11 variables may cause the network to be overfitted. Also, the PACF at lag 11 is only weakly significant. A weak significant correlation is often caused by random errors, especially when the data set is small. So we chose (y_{t-2}, y_{t-1}) as the input pattern. We note that Nicholls and Quinn (1982) analyzed this data set with an AR(2) model.

In the cross-validation experiment, we used the first 90 observations for model building and the following 10 observations for model validation. We tried all cross settings with $M = 7$, 8, 9 and $\lambda = 2.5$, 5, 10. The setting $M = 8$ and $\lambda = 5$ was chosen by minimizing $\sum_{h=1}^{2} MSPE(h)$. With this setting, EMC was run for 10 times independently. Each run consists of 22000 iterations, where the first 400 iterations were used for initialization, and the following 1600 iterations were discarded for the burn-in process. From the last 20000 iterations, a total of 2000 samples were collected at the lowest temperature level with an equal time space. The CPU time used by the run was about 58 s. Figure 7(b) shows the fitted and one-step ahead forecasting curves obtained in one run. The other results are summarized in Table 3.

For comparison, we also applied the Bayesian AR model averaging, TNN, and kernel smoothing approaches to this data set. In Bayesian AR model averaging, we set the maximum lag value $p = 12$ as for the full AR model. In the cross-validation experiment for the TNN model, we used the same data partition as that used for the BNN model, and tried all cross settings with $M = 3, \ldots, 10$ and $\tau = 0$, 0.0001, 0.001, 0.01, 0.1, ..., 0.8. We found the forecasting performance of the TNN model is rather robust to the variation of M given $\tau = 0.6$. The setting $M = 3$ and $\tau = 0.6$ was chosen by minimizing $\sum_{h=1}^{6} MSPE(h)$. With the chosen setting, TNN was run for 100 times again. The averaged MSPE's and the standard error of the average are shown in Table 3. Other criteria were also tried

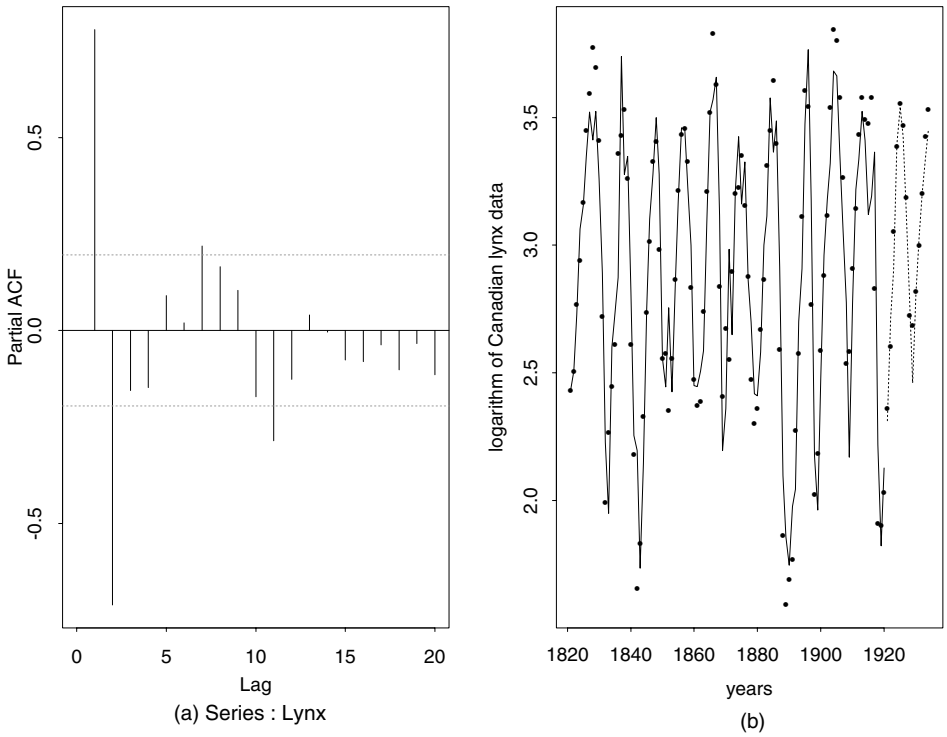

Fig. 7. *(a) The PACF of the logarithm of the Canadian lynx data. (b) The logarithmically transformed Canadian lynx data (points), the fitted curve (solid line), and the one-step ahead forecasting curve (dotted line). The latter two curves were obtained in one run of EMC with $M = 8$ and $\lambda = 5$*

Table 3. *Comparison of BNN and other time series models for the Canadian lynx example. The notation is as defined in Table 2. The results of full AR, subset AR, SETAR, and bilinear models were taken from Gabr and Subba Rao (1981)*

Model	Full AR(12)	Subset AR	BMA AR	SETAR	Bilinear	BNN^{ad}	BNN^{un}	TNN	Kernel
MSE	.0358	.0378	.0382 (.00005)	.0415	.0223	.0455 (.00021)	.0455 (.00021)	.0567 (0.0)	.0313
Size	12	6	5.13 (.00050)	12	13	9.80 (0.16)	9.80 (.16)	15	–
MSPE(1)	.02549	.02233	.02295 (.00018)	.01448	.01331	.00851 (.00021)	.00858 (.00018)	.01770 (.0)	.02334
MSPE(2)	.07377	.07240	.07690 (.00067)	.0259	.04432	.01902 (.00063)	.01913 (.00065)	.06067 (.0)	.03984
MSPE(3)	.11612	.11992	.13395 (.00106)	.0329	.06282	.02520 (.00097)	.02738 (.00100)	.08597(.0)	.06292
MSPE(4)	.16121	.16873	.18683 (.00129)	.03744	.07657	.03077 (.00146)	.04047 (.00140)	.09645 (10^{-5})	.08259
MSPE(5)	.18488	.20211	.21903 (.00149)	.0481	.08596	.03409 (.00196)	.05770 (.00158)	.10541 (10^{-5})	.08082
MSPE(6)	.18560	.20690	.22381 (.00169)	.12268	.07562	.03068 (.00220)	.06662 (.00128)	.11041 (10^{-5})	.07979

for this example, but the resulting forecasting results are all inferior to that reported in Table 3. For the kernel smoothing approach, we tried $p = 1, \ldots, 12$ with the same cross-validation data partition as for the BNN and TNN models. By minimizing $\sum_{h=1}^{2} MSPE(h)$ or $\sum_{h=1}^{2} MSPE(h)$, we obtained the same setting $p = 11$. The corresponding forecasting results are given in Table 3.

Table 3 shows that BNN^{ad} outperforms all other models in forecasting future values for this example. We also tried the models with $\lambda = 10$ and 15. As expected, the fitting is slightly improved, but the forecasting is slightly deteriorated. For example, the MSPE(5)'s (the largest one among the six MSPE values) are 0.0421 and 0.0416 for the models with $\lambda = 10$ and $\lambda = 15$, respectively. Note that even with $\lambda = 10$ or 15, BNN^{ad} still offers a significant improvement over the other models shown in Table 3.

4.3. *Unemployment data of West Germany*

As the third example, we consider the case where the training data contains outliers. The data is the monthly number of people registered as unemployed in West Germany for the period January 1948–May 1980 (inclusive). The total number of observations is 389. This data was first analyzed by Gabr and Subba Rao (1981). We follow Gabr and Subba Rao to remove the trend and seasonality (12 month period) using the transformation $X_t = (1 - B)(1 - B^{12})Y_t$, where Y_t is the original series, and then work on the series X_t, which is plotted in Fig. 8(a). Here we regard the rapidly changed values as outliers and use this example to demonstrate how the series with outliers in the training region can be modeled by BNN models. For more discussions on outlier detection and treatments, see Barnett, Kohn, and Sheather (1996) and Gerlach, Carter and Kohn (1999).

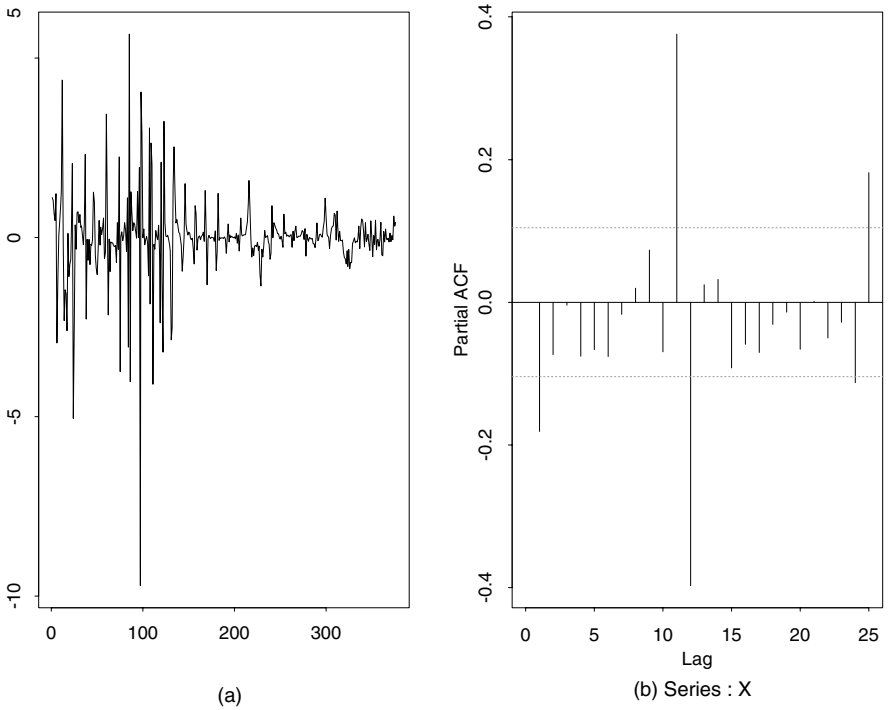

Fig. 8. *(a) The de-trended and deseasonalized unemployment data ($\times 10^5$) of West Germany from January 1948 to May 1980 (inclusive). (b) The PACF of the training part of the de-trended and deseasonalized unemployment data*

Table 4. *Comparison of BNN and other time series models for the unemployment example. The notation is as defined in Table 2. The results of full AR, subset AR, and bilinear models were taken from Gabr and Subba Rao (1981)*

Model	Full AR(12)	Subset AR	BMA AR	Bilinear	BNN^{ad}	BNN^{un}	TNN	Kernel
MSE	.81048	.81831	.85862 (.00035)	.36665	.54574 (.01431)	.54574 (.01431)	.85927 (.00493)	.92628
Size	12	5	3.25 (.04)	15	26.9 (0.47)	26.9 (.47)	41	–
MSPE(1)	.05111	.05039	.04692 (.00003)	.03690	.056145 (.00275)	.05643 (.00286)	.08073 (.00634)	.07796
MSPE(2)	.11254	.10981	.04886 (.00003)	.07598	.057374 (.00274)	.05982 (.00173)	.09990 (.01030)	.07783
MSPE(3)	.16619	.16269	.04618 (.00002)	.12446	.057520 (.00164)	.05830 (.00165)	.09937 (.01151)	.07787
MSPE(4)	.19934	.19677	.04777 (.00003)	.15105	.060659 (.00170)	.06152 (.00164)	.10108 (.01144)	.07739
MSPE(5)	.24927	.24865	.04997 (.00003)	.19703	.061332 (.00194)	.06418 (.00188)	.10124 (.01096)	.07752
MSPE(6)	–	–	.05239 (.00003)	–	.060309 (.00254)	.06721 (.00275)	.09967 (.01001)	.07741

We also follow Gabr and Subba Rao (1981) to use the first 365 observations for model building and the next 24 observations for forecasting. The PACF of the training data is shown in Fig. 8(b), which suggests that the input pattern $(X_{t-12}, \ldots, X_{t-1})$ may be appropriate for the BNN model. A cross-validation experiment was done with the first 341 observations for model building, and the next 24 observations for model validation. All cross-settings were tried with $M = 4, 5, 6$ and $\lambda = 10, 15, 20$. By minimizing $\sum_{h=1}^{2} MSPE(h)$, the setting $M = 4$ and $\lambda = 20$ was chosen. EMC was then run for 10 times independently with the chosen setting. Each run consists of 30000 iterations, where the first 500 iterations were used for weight initialization, the following 9500 iterations were discarded for the burn-in process, and 2000 samples were collected from the last 20000 iterations at the lowest temperature level with an equal time space. The CPU time used by a single run was about 188s. The results are summarized in Table 4.

We also applied the Bayesian AR model averaging, TNN, and kernel smoothing approaches to this data set. In Bayesian AR model averaging, p was set to 12 as in BNN models. A cross-validation experiment was also done for TNN models with $M = 2, \ldots, 6$, $\tau = 0, .0001, .001, .01, .1, 1, 2, 3, 4$, and 5, and the same training data partition as that used in the cross-validation experiment for the BNN model. A large value of τ tends to force all weights to 0, and thus a zero output for all input patterns. This approximately happens when $\tau > 5$ for this example. So the maximum value of τ we tried was 5. The setting $M = 2$ and $\tau = 5$ was chosen by minimizing $\sum_{h=1}^{6} MSPE(h)$. With the chosen setting, the TNN model was then trained for 100 times independently. The results are shown in Table 4. In the cross-validation experiment for the kernel smoothing approach, the settings $p = 1, \ldots, 12$ were tried. We have $p = 12$ by minimizing either $\sum_{h=1}^{2} MSPE(h)$ or $\sum_{h=1}^{6} MSPE(h)$. The forecasting results are shown in Table 4. BNN^{ad} outperforms all other predictors in Table 4, except for BMA-AR. Comparing the MSEs of BNN^{ad} and BMA-AR, we see that the BNN model may overfit to the data due to the presence of outliers in the training region.

To reduce the influence of outliers on BNN training, we suggest a heavy tail distribution, e.g., a student-t distribution, be used for modeling the innovations to prevent the data from being over fitted. For this example, we tried t-distributions with degrees of freedom equal to 5, 10, 20, and 50. For each t-distribution, we set $M = 4$ and $\lambda = 20$, and ran EMC for 10 times independently as for the models with the normality assumption. The forecasting results were given in Table 5. Comparing Tables 4 and 5, it is easy to see that the forecasting performance of BNN^{ad} has been significantly improved by modeling the innovations by the heavy-tail distributions. Just as expected, Table 5 also shows that MSE (fitting errors) increases and MSPE (forecasting error) decreases as the degree of freedom of the t-distribution increases. This suggests that a trade-off can be made between fitting and forecasting errors by choosing different degrees of freedom of the t-distribution for a time series with outliers presented in the training region.

Table 5. *Comparison of various student-t distributions for modeling the innovations for the unemployment example. The notation is as defined in Table 2*

Degree of freedom	t (5)	t (10)	t (20)	t (50)
MSE	.87695 (.00685)	.79409 (.01651)	.67668 (.01457)	.64042 (.00911)
Size	22.5 (.5)	24.2 (.5)	24.8 (.4)	25.1 (.5)
MSPE(1)	.04806 (.00111)	.04690 (.00135)	.04516 (.00142)	.05560 (.00264)
MSPE(2)	.04611 (.00098)	.04733 (.00184)	.04809 (.00181)	.05626 (.00286)
MSPE(3)	.04367 (.00045)	.04478 (.00140)	.05000 (.00220)	.05660 (.00364)
MSPE(4)	.04502 (.00074)	.04640 (.00208)	.05193 (.00269)	.05875 (.00296)
MSPE(5)	.04604 (.00046)	.04759 (.00186)	.05403 (.00339)	.05805 (.00314)
MSPE(6)	.04578 (.00038)	.04743 (.00189)	.05456 (.00388)	.05880 (.00345)

5. Discussion

In this article, we presented a systematic implementation for Bayesian neural networks in modeling and forecasting nonlinear time series. The selection of hidden units and the selection of input variables are unified by sampling from the joint posterior distribution of the weights and network structure. Insights into how to improve the generalization ability of BNN models are revealed in many respects of the implementation. A heavy tail distribution, such as a student t-distribution, is proposed to model the innovations for a time series with outliers presented in the training region. The numerical results show that BNN models consistently offer a significant improvement over the competitors in nonlinear time series forecasting.

Despite these successes, several issues still need to be cautious in practical applications of BNN models. The first issue is on the selection of input variables. We suggest the input variables be determined by examining the PACF of the training data. Although this method works well for all examples of this article, it is potentially inadequate, as it is an inherently linear technique. A general method can be designed based on the technique of Bayesian model averaging as follows.

(a) Choose a large enough value of p, and train a BNN model with all lag values $y_{t-p}, \ldots, y_{t-1}$ as input variables.
(b) Evaluate the probabilities $P_{in}(\sum_{j=1}^{M} I_{\gamma_{ji}} \geq 1 \mid \mathcal{D})$, for $i = 1, 2, \ldots, p$, where $P_{in}(\cdot|\cdot)$ is the posterior probability of the lag variable y_{t-i} being included in the model.
(c) Set a cut-off value for the posterior probabilities, and re-train the BNN model with the selected input variables.

The effectiveness of the method depends on many elements, for example, the number of hidden units and the hyperparameter setting. In fact, the above method has been partially implemented in this article. The p value we used is determined by the PACF. Our numerical results suggests the p value chosen by this method may be large enough for BNN models. As for the posterior probability $P_{in}(\cdot|\cdot)$ evaluation and the cut-off value setting, they are less important here, as we focus on forecasting or model averaging instead of model selection. We note that in general the input variables can also be determined by a cross-validation procedure.

The second issue is about network size. For BNN, a parsimonious structure is usually not pursued. In a parsimonious model, the weights are often trained to be extremely large (in absolute value) in order for the model to learn sufficiently from the training data. As explained before, this often results in a poor generalization ability. This point is easy to see in Fig. 6, where the models with prior variance 100 have a smaller network size (on average) than those with prior variance 2 and 5, but the latter models perform better in forecasting. A small weight variation is more important than a parsimonious structure for the generalization ability of BNN models. Of course, with the same weight variation, a parsimonious structure is still preferred for a good generalization ability.

The third issue is about the treatment of outliers. The unemployment example illustrates our treatment of outliers presented in the training region of the time series. For outliers presented in the forecasting region, a self-adjusted procedure for outliers (Chen and Liu 1993) may be applied to reduce their influence on forecasting.

A. Appendix

A.1. *Mutation*

In the mutation operator, a chromosome, say ξ^k, is selected at random from the current population $\boldsymbol{z}$. Then ξ^k is modified to a new chromosome $\xi^{k'}$ by one of the three types of moves, "birth", "death" and "Metropolis" (described below). The new population $\boldsymbol{z}' = \{\xi^1, \ldots, \xi^{k-1}, \xi^{k'}, \xi^{k+1}, \ldots, \xi^N\}$ is accepted with probability

$$\min\left\{1, \exp\{-(H(\xi^{k'}) - H(\xi^k))/t_k\}\frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})}\right\}, \tag{8}$$

where $T(\boldsymbol{z} \mid \boldsymbol{z}')/T(\boldsymbol{z}' \mid \boldsymbol{z})$ is the ratio of the transition probabilities.

Let S denote the set of effective connections of the current neural network ξ^k, and S^c be the complementary set of S. Let $m = \|S\|$ be the number of elements in S (recall the definition of m in Section 2). Let $P(m, \text{birth})$, $P(m, \text{death})$ and $P(m, \text{Metropolis})$ denote the proposal probabilities of the three types of moves for a network with m connections. For our examples, we set $P(m, \text{birth}) = P(m, \text{death}) = 1/3$ for $3 < q < U$, $P(U, \text{death}) = P(3, \text{birth}) = 2/3$, $P(U, \text{birth}) = P(3, \text{death}) = 0$, and $P(m, \text{Metropolis}) = 1/3$ for $3 \leq q \leq U$. In the "birth" move, a connection, say ζ, is first randomly chosen from S^c. If $\zeta \in \{\alpha_i : I_{\alpha_i} = 0\} \cup \{\gamma_{ji} : I_{\gamma_{ji}} = 0, \sum_{k=0}^{p} I_{\gamma_{jk}} \geq 1, 1 \leq j \leq M, 0 \leq i \leq p\}$, the move is called the type I "birth" move (illustrated by Fig. 9(a)). In this type of move, only the

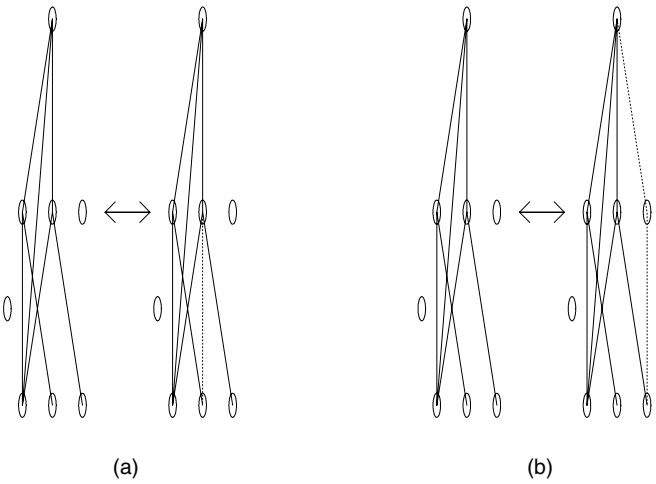

Fig. 9. *Illustration of the "birth" and "death" moves. The dotted lines denote the connections to add to or delete from the current network. (a) Type I "birth" ("death") move; (b) Type II "birth" ("death") move*

connection ζ is proposed to add to ξ^k with weight ω_ζ being randomly drawn from $N(0, \sigma^2_{\xi^k})$, where $\sigma^2_{\xi^k}$ is the sample variance of the effective weights of ξ^k. The resulting transition probability ratio is

$$\frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})} = \frac{P(m+1, \text{death})}{P(m, \text{birth})} \frac{U-m}{m'} \frac{1}{\phi(\omega_\zeta/\sigma_{\xi^k})},$$

where $\phi(\cdot)$ denotes the standard normal density, and m' denotes the number of deletable connections of the network $\xi^{k'}$. A connection , say υ, is deletable from a network if $\upsilon \in \{\alpha_i : I_{\alpha_i} = 1, 0 \le i \le p\} \cup \{\gamma_{ji} : I_{\gamma_{ji}} = 1, 1 \le j \le M, 0 \le i \le p\} \cup \{\beta_j : I_{\beta_j} = 1, \sum_{i=0}^p I_{\gamma_{ji}} = 1, 1 \le j \le M\}$.

If $\zeta \in \{\beta_j : I_{\beta_j} = 0, \sum_{i=0}^p I_{\gamma_{ji}} = 0, 1 \le j \le M\} \cup \{\gamma_{ji} : \sum_{k=0}^p I_{\gamma_{jk}} = 0, I_{\beta_j} = 0, 1 \le j \le M, 0, \le i \le p\}$, the move is called the type II "birth" move (illustrated by Fig. 9(b)), which is to add a new hidden unit. In this type of move, if $\zeta \in \{\beta_j : I_{\beta_j} = 0, \sum_{i=0}^p I_{\gamma_{ji}} = 0, 1 \le j \le M\}$, a connection, say ζ', is randomly chosen from the set of connections from input units to the corresponding hidden unit, and then ζ and ζ' are proposed to add to ξ^k together; if $\zeta \in \{\gamma_{ji} : \sum_{k=0}^p I_{\gamma_{jk}} = 0, I_{\beta_j} = 0, 1 \le j \le M, 0, \le i \le p\}$, the connection denoted by ζ' from the corresponding hidden unit to the output unit is also proposed to add to ξ^k together with ζ. The weights ω_ζ and $\omega_{\zeta'}$ are drawn independently from $N(0, \sigma^2_{\xi^k})$. The resulting transition probability ratio is

$$\frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})} = \frac{P(m+2, \text{death})}{P(m, \text{birth})} \frac{2(U-m)}{(1+1/(p+1))m'} \times \frac{1}{\phi(\omega_\zeta/\sigma_{\xi^k})\phi(\omega_{\zeta'}/\sigma_{\xi^k})}.$$

Once the "birth" proposal is accepted, the corresponding indicators are switched to 1.

In the "death" move, a connection, say ζ, is first randomly chosen from the set of deletable connections of ξ^k. If $\zeta \in \{\alpha_i : I_{\alpha_i} = 1, 0 \le i \le p\} \cup \{\gamma_{ji} : I_{\gamma_{ji}} = 1, \sum_{k=0}^p I_{\gamma_{jk}} > 1, 1 \le j \le M, 0 \le i \le p\}$, the move is called the type I "death" move (illustrated by Fig. 9(a)). In this type of move, only ζ is proposed to for deletion, and the resulting transition probability ratio is

$$\frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})} = \frac{P(m-1, \text{birth})}{P(m, \text{death})} \frac{m'}{U-m+1} \frac{\phi(\omega_\zeta/\sigma'_{\xi^k})}{1},$$

where m' denotes the number of deletable connections of ξ^k, and σ'_{ξ^k} denotes the sample standard deviation of the effective weights of ξ^k except for ζ. If $\zeta \in \{\gamma_{ji} : I_{\gamma_{ji}} = 1, \sum_{k=0}^p I_{\gamma_{jk}} = 1, \le j \le M, 0 \le i \le p\} \cup \{\beta_j : I_{\beta_j} = 1, \sum_{i=0}^p I_{\gamma_{ji}} = 1, 1 \le j \le M\}$, the move is called the type II "death" move (illustrated by Fig. 9(b)), which is to eliminate one hidden unit from the network. In this type of move, if $\zeta \in \{\beta_j : I_{\beta_j} = 1, \sum_{i=0}^p I_{\gamma_{ji}} = 1, 1 \le j \le M\}$, the only connection denoted by ζ' from some input unit to the corresponding hidden unit is also proposed for deletion together with ζ; if $\zeta \in \{\gamma_{ji} : I_{\gamma_{ji}} = 1, \sum_{k=0}^p I_{\gamma_{jk}} = 1, \le j \le M, 0 \le i \le p\}$, the connection denoted by ζ' from the corresponding hidden unit to the output unit is also proposed for deletion together with ζ. The resulting transition probability ratio is

$$\frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})} = \frac{P(m-2, \text{birth})}{P(m, \text{death})} \frac{(1+1/(p+1))m'}{2(U-m+2)} \times \frac{\phi(\omega_\zeta/\sigma'_{\xi^k})\phi(\omega_{\zeta'}/\sigma'_{\xi^k})}{1}.$$

Once the "death" proposal is accepted, the corresponding indicators are switched to 0.

In the "Metropolis" move, the weights are updated while keeping the indicator vector unchanged. In this type of move, a random direction d_{m+1} is first randomly drawn in the $(m+1)$-dimensional space, where $m+1$ is the total number of parameters of the current BNN model including the number of effective weights and the parameter σ^2, and then set

$$\xi^{k'} = \xi^k + \rho_k d_{m+1},$$

where $\rho_k \sim N(0, \tau_k^2)$, $\tau_k = \kappa\sqrt{t_k}$, and κ is a user specified scale parameter so-called the Metropolis step size. For this type of move, we have $T(\boldsymbol{z} \mid \boldsymbol{z}')/T(\boldsymbol{z}' \mid \boldsymbol{z}) = 1$, since the transition is symmetric in the sense that $T(\boldsymbol{z} \mid \boldsymbol{z}') = T(\boldsymbol{z}' \mid \boldsymbol{z}) = \phi(\rho_k/\tau_k)$. Although the above proposal works well in general, a more sophisticated one should take into account the difference of the scales of different connection groups, and propose separately for different groups, for example, the scales of $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ and $\boldsymbol{\gamma}$ may be different. Usually this will improve the mixing of the Markov chain.

A.2. *Crossover*

In the crossover operator, different offspring are produced by a recombination of parental chromosomes randomly selected from the current population. Suppose that ξ^i and ξ^j are chosen as parental chromosomes, new offspring $\xi^{i'}$ and $\xi^{j'}$ are generated as follows. First, an integer c is drawn randomly on the set $\{1, 2, \ldots, M\}$, then $\xi^{i'}$ and $\xi^{j'}$ are constructed by swapping the weights connected with hidden unit c between ξ^i and ξ^j. The hidden unit c is called the crossover unit. This operator is illustrated by Fig. 10. If there are ι crossover units, the operator is called the ι-unit crossover. In this article, only the 1-unit crossover operator is used. Comparing to the mutation operator, it is clear that the crossover operator provides a more global move in the space of network structures.

A new population is constructed by replacing the parental chromosomes with the new offspring, and it is accepted with probability

$$\min\left\{1, \exp\{-(H(\xi^{i'}) - H(\xi^i))/t_i - (H(\xi^{j'}) - H(\xi^j))/t_j\} \frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})}\right\},$$

where $T(\boldsymbol{z}' \mid \boldsymbol{z}) = P(\xi^i, \xi^j|\boldsymbol{z})\, P(\xi^{i'}, \xi^{j'} \mid \xi^i, \xi^j)$, $P(\xi^i, \xi^j|\boldsymbol{z}) = 2/N(N-1)$ denotes the selection probability of (ξ^i, ξ^j) from population $\boldsymbol{z}$, and $P(\xi^{i'}, \xi^{j'} \mid \xi^i, \xi^j)$ denotes the generating probability of $(\xi^{i'}, \xi^{j'})$ from the parental chromosomes (ξ^i, ξ^j). The

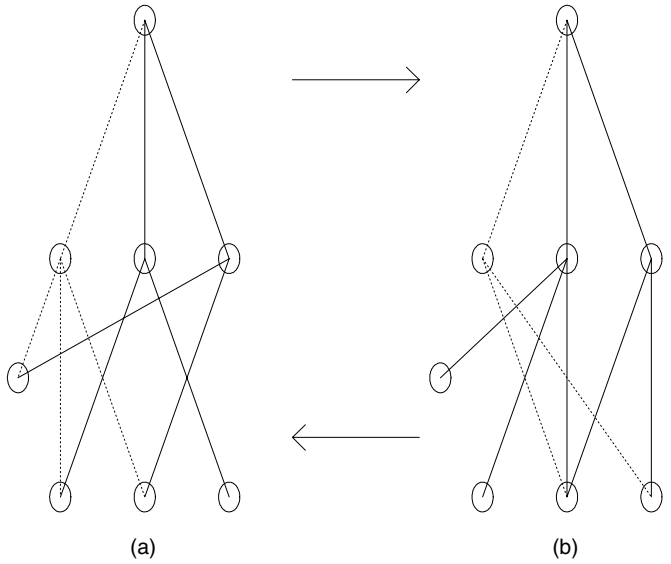

Fig. 10. *Illustration of the crossover operator. The dotted lines denote the connections to exchange between the two networks*

ι-unit crossover operator is symmetric, i.e., $P(\xi^{i'}, \xi^{j'}|\xi^i, \xi^j) = P(\xi^i, \xi^j|\xi^{i'}, \xi^{j'})$. Thus, we have $T(\boldsymbol{z} \mid \boldsymbol{z}')/T(\boldsymbol{z}' \mid \boldsymbol{z}) = 1$ for the crossover operator.

A.3. *Exchange*

Given the current population $\boldsymbol{z}$ and the attached temperature ladder $\boldsymbol{t}$, we try to make an exchange between ξ^i and ξ^j without changing the t's; that is, initially we have $(\boldsymbol{z}, \boldsymbol{t}) = (\xi^1, t_1, \ldots, \xi^i, t_i, \ldots, \xi^j, t_j, \ldots, \xi^N, t_N)$ and we propose to change it to $(\boldsymbol{z}', \boldsymbol{t}) = (\xi^1, t_1, \ldots, \xi^j, t_i, \ldots, \xi^i, t_j, \ldots, \xi^N, t_N)$. The new population is accepted with probability

$$\min\left\{1, \exp\left\{(H(\xi^i) - H(\xi^j))\left(\frac{1}{t_i} - \frac{1}{t_j}\right)\right\}\frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})}\right\}.$$

Typically, the exchange is only performed on two chromosomes with neighboring temperatures, i.e., $|i - j| = 1$. Let $p(\xi^i)$ be the probability that ξ^i is chosen to exchange with the other chromosome and $w_{i,j}$ be the probability that ξ^j is chosen to exchange with ξ^i, then $T(\boldsymbol{z}' \mid \boldsymbol{z}) = p(\xi^i)w_{i,j} + p(\xi^j)w_{j,i}$. Thus, $T(\boldsymbol{z}' \mid \boldsymbol{z})/T(\boldsymbol{z} \mid \boldsymbol{z}') = 1$ for the exchange operator.

Comment

The program is available by an request to the author.

Acknowledgments

The author thanks Professor Wayne Oldford, Professor Chuanhai Liu, Elaine Reed, the associate editor and three referees for their constructive comments and suggestions that have led to significant improvement of this article.

References

Andrieu C., Freitas N.D., and Doucet A. 2000. Reversible jump MCMC Simulated Annealing for Neural Networks. Uncertainty in Artificial Intelligence (UAI2000).

Andrieu C., Freitas N.D., and Doucet A. 2001. Robust full Bayesian learning for radial basis networks. Neural Computation 13: 2359–2407.

Auestad B. and Tjøstheim 1990. Identification of nonlinear time series: First order characterization and order determination. Biometrika 77: 669–687.

Barnett G., Kohn R., and Sheather S.J. 1996. Robust estimation of an autoregressive model using Markov chain Monte Carlo. Journal of Econometrics 74: 237–254.

Bishop C.M. 1995. Neural Networks for Pattern Recognition. Oxford University Press, Oxford.

Box G.E.P. and Jenkins G.M. 1970. Time Series Analysis, Forecast and Control, Holden Day, San Francisco.

Casella G. and Berger R.L. 2001. Statistical Inference, 2nd ed. Thomson Learning, Duxbury.

Chatfield C. 2001. Time-Series Forecasting. Chapman and Hall, London.

Chen C. and Liu L.M. 1993. Forecasting time series with outliers. Journal of Forecasting 12: 13–35.

Cybenko G. 1989. Approximations by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems. 2: 303–314.

Dension D., Holmes C., Mallick B., and Smith A.F.M. 2002. Bayesian Methods for Nonlinear Classification and Regression. Willey, New York.

Faraway J. and Chatfield C. 1998. Time series forecasting with neural networks: A comparative study using the airline data. Appl. Statist. 47: 231–250.

Fernández G., Ley E., and Steel M.F.J. 2001. Benchmark priors for Bayesian model averaging. Journal of Econometrics 100: 381–427.

Freitas N. and Andrieu C. 2000. Sequential Monte Carlo for Model Selection and Estimation of Neural Networks. ICASSP2000.

Freitas N., Andrieu C., Hjen-Srensen P., Niranjan M., and Gee A. 2001. Sequential Monte Carlo methods for neural networks. In: Doucet A., de Freitas N., and Gordon N. (Eds.), Sequential Monte Carlo Methods in Practice, Springer-Verlag.

Funahashi K. 1989. On the approximate realization of continuous mappings by neural networks. Neural Networks 2: 183–192.

Gabr M.M. and Subba Rao T. 1981. The estimation and prediction of subset bilinear time series models with applications. Journal of Time Series Analysis 2: 155–171.

Gelman A., Roberts R.O. and Gilks W.R. 1996. Efficient Metropolis jumping rules. In Bernardo J.M.: Berger, J.O., Dawid, A.P., and Smith A.F.M. (Eds.), Bayesian Statistics 5. Oxford University Press, New York.

Gelamn A. and Rubin D.B. 1992. Inference from iterative simulation using multiple sequences (with discussion). Statist. Sci. 7: 457–472.

Gerlach G., Carter C.K., and Kohn R. 1999. Diagnostics for time series analysis. Journal of Time Series Analysis.

Geyer C.J. 1991. Markov chain Monte Carlo maximum likelihood. In: Keramigas E.M. (Ed.), Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface pp. 153–163. Interface Foundation, Fairfax Station.

Goldberg D.E. 1989. Genetic Algorithms in Search, Optimization, & Machine Learning. Addison Wesley.

Green P.J. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. Biometrika 82: 711–732.

Härdle W. and Vieu P. 1992. Kernel regression smoothing of time series. Journal of Time Series 13: 209–232.

Hastings W.K. 1970. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57: 97–109.

Higdon D., Lee H., and Bi Z. 2002. A Bayesian approach to characterizing uncertainty in inverse problems using coarse and fine-scale information. IEEE Transactions on Signal Processing 50: 389–399.

Hill T., O'Connor M., and Remus W. 1996. Neural network models for time series forecasts. Management Science 42: 1082–1092.

Holland J.H. 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Holmes C.C. and Denison D. 2002. A Bayesian MARS classifer. Machine Learning, to appear.

Holmes C.C. and Mallick B.K. 1998. Bayesian radial basis functions of variable dimension. Neural Computation 10: 1217–1233.

Hornik K., Stinchcombe M., and White H. 1989. Multilayer feedforward networks are universal approximators. Neural Networks 2: 359–366.

Hukushima K. and Nemoto K. 1996. Exchange Monte Carlo method and application to spin glass simulations. J. Phys. Soc. Jpn. 65 1604–1608.

Kang S. 1991. An investigation of the use of feedforward neural networks for forecasting. Ph.D. Dissertation, Kent State University, Kent, Ohio.

Liang F., Truong Y.K., and Wong W.H. 2001. Automatic Bayesian model averaging for linear regression and applications in Bayesian curve fitting. Statistica Sinica 11: 1005–1029.

Liang F. and Wong W.H. 2000. Evolutionary Monte Carlo: Applications to C_p model sampling and change point problem. Statistica Sinica 10: 317–342.

Liang F. and Wong W.H. 2001. Real parameter evolutionary Monte Carlo with applications in Bayesian mixture models. J. Amer. Statist. Assoc. 96: 653–666.

Lim K.S. 1987. A comparative study of various univariate time series models for Canadian lynx data. Journal of Time Series Analysis 8: 161–176.

MacKay D.J.C. 1992. A practical Bayesian framework for backprop networks. Neural Computation 4: 448–472.

Mallows C.L. 1973, Some comments on C_p. Technometrics 15: 661–676.

Marinari E. and Parisi G. 1992. Simulated tempering: A new Monte Carlo scheme. Europhysics Letters 19: 451–458.

Marrs A.D. 1998. An application of reversible-jump MCMC to multivariate spherical Gaussian mixtures. In Advances in Neural Information Processing Systems 10. Morgan Kaufmann. San Mateo, CA, pp. 577–583.

Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., and Teller E. 1953. Equation of state calculations by fast computing machines. Journal of Chemical Physics 21: 1087–1091.

Müller P. and Insua D.R. 1998. Issues in Bayesian analysis of neural network models. Neural Computation 10: 749–770.

Neal R.M. 1996. Bayesian Learning for Neural Networks. Springer-Verlag, New York.

Nicholls D.F. and Quinn B.G. 1982. Random Coefficient Autoregressive Models: An Introduction. Springer-Verlag, New York.

Park Y.R., Murray T.J., and Chen C. 1996. Predicting sunspots using a layered perceptron neural network. IEEE Trans. Neural Networks 7: 501–505.

Penny W.D. and Roberts S.J. 1999. Bayesian neural networks for classification: How useful is the evidence framework? Neural Networks, 12: 877–892.

Penny W.D. and Roberts S.J. 2000. Bayesian methods for autoregressive models. In: Proceedings of Neural Networks for Signal Processing, Sydney, Dec. 2000.

Raftery A.E., Madigan D., and Hoeting J.A. 1997. Bayesian model averaging for linear regression models. J. Amer. Statist. Assoc. 92: 179–191.

Rumelhart D., Hinton G., and Williams J. 1986. Learning internal representations by error propagation. In: Rumelhart D. and McClelland J. (Eds.), Parallel Distributed Processing. MIT Press, Cambridge, pp. 318–362.

Rumelhart D. and McClelland J. 1986. Parallel Distributed Processing. MIT Press, Cambridge.

Smith A.F.M. and Roberts G.O. 1993. Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods (with discussion). J. Royal Statist. Soc. B, 55: 3–23.

Subba Rao T. and Gabr M.M. 1984. An Introduction to Bispectral Analysis and Bilinear Time Series Models. Springer, New York.

Tong H. 1990. Non-Linear Time Series: A Dynamical System Approach. Oxford University Press, Oxford.

Tong H. and Lim K.S. 1980. Threshold autoregression, limit cycles and cyclical data (with discussion). J. R. Statist. Soc. B 42: 245–292.

Waldmeirer M. 1961. The Sunspot Activity in the Years 1610–1960. Schultheses.

Weigend A.S., Huberman B.A., and Rumelhart D.E. 1990. Predicting the future: A connectionist approach. Int. J. Neural Syst. 1: 193–209.

Weigend A.S., Rumelhart D.E., and Huberman B.A. 1991. Generalization by weight-elimination with application to forecasting. In: Advance in Neural Information Processing Systems 3, Morgan Kaufmann, San Mateo, CA., pp. 875–882.