LETTER Communicated by Ralf Herbrich

An Effective Bayesian Neural Network Classifier with a Comparison Study to Support Vector Machine

Faming Liang
fliang@stat.tamu.edu
Department of Statistics, Texas A & M University, College Station, TX 77843, U.S.A.

We propose a new Bayesian neural network classifier, different from that commonly used in several respects, including the likelihood function, prior specification, and network structure. Under regularity conditions, we show that the decision boundary determined by the new classifier will converge to the true one. We also propose a systematic implementation for the new classifier. In our implementation, the tune of connection weights, the selection of hidden units, and the selection of input variables are unified by sampling from the joint posterior distribution of the network structure and connection weights. The numerical results show that the new classifier consistently outperforms the commonly used Bayesian neural network classifier and the support vector machine in terms of generalization performance. The reason for the inferiority of the commonly used Bayesian neural network classifier and the support vector machine is discussed at length.

1 Introduction

Support vector machines (SVMs) have recently been proposed as a technique for solving pattern classification problems (Boser, Guyon, & Vapnik, 1992; Cortes & Vapnik, 1995) and have rapidly gained in popularity due to their theoretical merits, computational simplicity, and excellent performance in real-world applications. The excellence is certainly based on a comparison with other classification tools, such as nearest neighbor, discriminant analysis, and neural networks. For example, Ding and Dubchak (2001) and Markowetz, Edler, and Vingron (2002) showed numerically that SVMs outperform neural networks and other classification tools in protein fold class prediction. Breiman (2002) commented that the excellent performance of SVMs is beginning to supplant the use of neural networks.

However, SVMs have two drawbacks despite their successes. One is that their decision boundary is determined only by support vectors. The boundary will be severely biased from the true one if some of the support vectors were statistical outliers. The support vectors often contain some extreme values of the populations. The low reliability of the extreme values often makes the decision boundary of SVMs not reliable at all. The other drawback is

that it does not take account of the uncertainty of data. As many researchers know, the methods that take account of data uncertainty—for example, the ensemble method (Perrone & Cooper, 1993; Rosen, 1996; Krogh & Vedelsby, 1995) and Bayesian model averaging (Raftery, Madigan, & Hoeting, 1997)—often have better generalization performance than methods that ignore data uncertainty.

One the other hand, we note that the inferiority of neural networks in classifications is due to implementations instead of their intrinsic capability. In particular, the implementations often suffer from the two difficulties. First is the necessity of a priori specification for the network structure, namely, the number of hidden units and input variables, assuming that the output variables have been determined by the problems. An inappropriate network structure often results in an overfitting or insufficient learning to the training data, and thus a poor generalization performance. The second difficulty is the lack of efficient training algorithms. The deterministic training algorithms such as backpropagation (Rumelhart, Hinton, & Williams, 1986), conjugate gradient, and their variants, usually converge to some local minima of the objective functions. This problem becomes more serious as the network size increases. Bayesian neural networks (BNNs) (Mackay, 1992; Neal, 1996) provide a natural framework for alleviating the above difficulties. First, the priors for the network structure and connection weights work naturally as a regulation term for network training. It controls the selection of input variables, the number of hidden units, and the range of the weight values. Second, Markov chain Monte Carlo (MCMC) methods are naturally used to train the networks, sampling from the joint posterior of the network structure and connection weights. It avoids the local minima problem encountered by the deterministic training algorithms. Third, the data uncertainty can be take into account, as Bayesian model averaging is naturally used for fitting and prediction with the MCMC outputs. For an overview of the BNN classifiers, see Thodberg (1996) and Husmeier, Penny, and Roberts (1999).

In this letter, we propose a new BNN classifier, different from that proposed in Neal (1996) in several respects, including the likelihood function, network structure, and prior specification. Under regularity conditions, we show that the decision boundary determined by the new classifier will converge to the true one. We also propose a systematic implementation for the new classifier. The tune of connection weights, the selection of hidden units, and the selection of input variables are unified by sampling from the joint posterior of the network structure and connection weights. The numerical results show that the new BNN classifier outperforms SVMs and other BNN classifiers in all examples in this letter. The reason for the inferiority of the other BNN classifiers and SVMs is discussed at length.

In section 2, we give a brief description of SVMs. In section 3, we describe the new BNN classifier and show that it will asymptotically converge to the true classification function under regularity conditions. In section 4,

we describe the numerical implementation for the new BNN classifier. In section 5, we present our numerical results with extensive comparisons to SVMs and other BNN classifiers. In section 6, we conclude with a brief discussion.

2 SVM Classifiers

In this section, we give a brief description of SVMs. Given the training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with explanatory variables $\boldsymbol{x}_i \in \mathbb{R}^p$ and the corresponding binary class labels $y_i \in \{-1, +1\}$, the SVM classifier is constructed in the form

$$y(\boldsymbol{x}) = \text{sign}\left[\sum_{j=1}^{m} w_j \psi_j(\boldsymbol{x}) + b\right] = \text{sign}[\boldsymbol{w}'\boldsymbol{\psi}(\boldsymbol{x}) + b], \tag{2.1}$$

where $\boldsymbol{\psi}(\boldsymbol{x}) = [\psi_1(\boldsymbol{x}), \ldots, \psi_m(\boldsymbol{x})]'$ denotes a set of nonlinear transformations from the input space to the feature space, and $\boldsymbol{w} = [w_1, \ldots, w_m]'$ is the set of linear weights connecting the feature space to the output space. The underlying assumption of equation 2.1 is that the data from two classes can always be separated by a hyperplane with an appropriate nonlinear mapping $\boldsymbol{\psi}(\cdot)$ to a sufficiently highly dimensional feature space. The nonlinear mapping $\boldsymbol{\psi}(\cdot)$ is related to the kernel function through

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{\psi}(\boldsymbol{x}_1)'\boldsymbol{\psi}(\boldsymbol{x}_2),$$

where the kernel $K(\cdot, \cdot)$ is symmetric and positive definite from Mercer's theorem (Mercer, 1909). Typically, one has the following choices for the kernel function: $K(\boldsymbol{x}, \boldsymbol{x}_i) = \boldsymbol{x}'\boldsymbol{x}_i$ (linear kernel); $K(\boldsymbol{x}, \boldsymbol{x}_i) = (\boldsymbol{x}'\boldsymbol{x}_i + 1)^d$ (polynomial kernel of degree $d \in \mathbb{N}$); and $K(\boldsymbol{x}, \boldsymbol{x}_i) = \exp\{-\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2/2\sigma^2\}$ (radial basis function (RBF) kernel). Note that the Mercer condition holds for all σ and d values in the RBF and polynomial kernels.

The linear weights in equation 2.1 are computed through minimizing the objective function

$$\begin{aligned} &\min_{w,\xi} J_{\text{SVM}}(\boldsymbol{w}, \boldsymbol{\xi}) = \frac{1}{2}\boldsymbol{w}'\boldsymbol{w} + C\sum_{i=1}^{N} \xi_i, \\ &\text{subject to } \begin{cases} y_i \boldsymbol{w}'\boldsymbol{\psi}(\boldsymbol{x}_i) \geq 1 - \xi_i, & i = 1, \ldots, N, \\ \xi_i \geq 0 & i = 1, \ldots, N, \end{cases} \end{aligned} \tag{2.2}$$

where ξ_i's are slack variables that are introduced to allow for some misclassification due to overlap distributions. The C is a user-specified positive real number. It controls the trade-off between the complexity of the machine and the number of nonseparable patterns. With the so-called Kuhn-Tucker construction (Fletcher, 1987; Bertsekas, 1995), the optimization problem, equa-

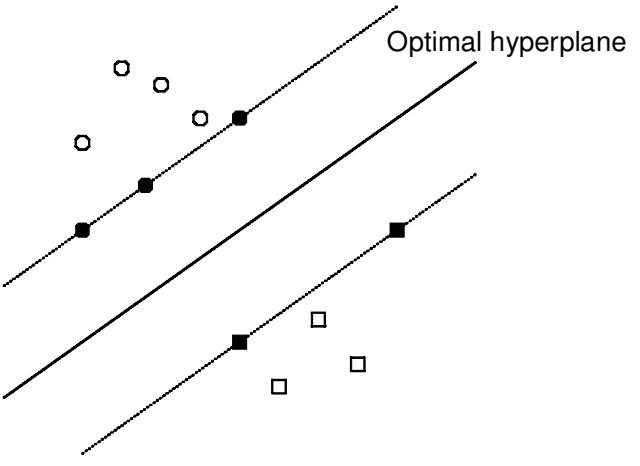

Figure 1: Illustration of the optimal hyperplane of SVMs for two linearly separable classes. The hyperplane is determined only by the support vectors, which are shown as the filled squares and circles.

tion 2.2, can be solved by solving its dual problem, and the resulting optimal classification hyperplane can then be expressed as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} y_i \hat{\alpha}_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b^*,$$

where $(\hat{\alpha}_1, \ldots, \hat{\alpha}_N)$ denotes the solution to the dual problem, and b^* is chosen so that $y_i f(\boldsymbol{x}_i) = 1$ for any i with $C > \hat{\alpha}_i > 0$. The decision rule for the classification is given by sign$[f(\boldsymbol{x})]$. The property of the optimal hyperplane can be illustrated by Figure 1. In this case, the two classes are linearly separable (a linear kernel is used), and the support vectors are those $\boldsymbol{x}_i$'s satisfying the condition

$$\hat{\boldsymbol{w}}' \boldsymbol{x}_i + b^* = +1 \quad \text{for } y_i = +1,$$
$$\hat{\boldsymbol{w}}' \boldsymbol{x}_i + b^* = -1 \quad \text{for } y_i = -1,$$

where $\hat{\boldsymbol{w}} = \sum_{j=1}^{N} y_j \hat{\alpha}_j \boldsymbol{x}_j$. The Kuhn-Tucker conditions imply that the following equalities hold:

$$\hat{\alpha}_i [y_i (\hat{\boldsymbol{w}}' \boldsymbol{x}_i + b^*) - 1] = 0, \quad i = 1, \ldots, N. \tag{2.3}$$

Therefore, only those $\hat{\alpha}_i$'s corresponding to the support vectors can be nonzero and thus contribute to the construction of the optimal hyperplane. In this sense, we say that the decision boundary of the SVMs is determined

locally by the support vectors, and the information contained in the other vectors is ignored in the boundary construction. If the support vectors unfortunately contain some statistical outliers, the resulting boundary may be severely biased from the true one. For linearly nonseparable cases, the optimal hyperplane is determined similarly, but with the slack variables being taken into account.

3 Bayesian Neural Network Classifiers

Suppose we have a group of data $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ are explanatory variables and $\boldsymbol{y}_i$ indicates the class membership. For problems of two classes, $\boldsymbol{y}_i$ is represented by a single digit, 0 or 1. For problems of multiple classes, $\boldsymbol{y}_i$ is represented by a binary vector, and class membership is indicated by the position of its only component of 1. A one hidden layer feedforward neural network model (illustrated by Figure 2) with weight eliminations approximates $\boldsymbol{y}_i$ by a group of q functions (corresponding to q output units) of the form

$$g_j(\boldsymbol{x}_i) = \phi_o\left[\alpha_{j0}I_{\alpha_{j0}} + \sum_{k=1}^{p} x_{ik}\alpha_{jk}I_{\alpha_{jk}} + \sum_{l=1}^{M}\beta_{jl}I_{\beta_{jl}}\phi_h\left(\gamma_{l0}I_{\gamma_{l0}} + \sum_{k=1}^{p} x_{ik}\gamma_{lk}I_{\gamma_{lk}}\right)\right], \quad j = 1, \ldots, q, \tag{3.1}$$

where α_{j0} denotes the bias of the jth output unit, α_{jk} denotes the weight on the shortcut connection from the kth input unit to the jth output unit; β_{jl} denotes the weight on the connection from the lth hidden unit to the jth output unit; γ_{l0} denotes the bias of the lth hidden unit, γ_{lk} denotes the weight on the connection from the kth input unit to the lth hidden unit; I_ζ is an indicator function that indicates the effectiveness of the connection ζ; M denotes the maximum number of hidden units specified by users; and $\phi_o(\cdot)$ and $\phi_h(\cdot)$ denote the activation functions of the hidden units and output units, respectively. Sigmoid and hyperbolic tangent functions are two popular choices for the activation functions. In this letter, we set $\phi_h(\cdot) = \tanh(\cdot)$ for all examples to ensure that the output of a hidden unit is 0 if all connections to the hidden unit from the input units have been eliminated, and thus the hidden unit can be eliminated from the network without any effect on the 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. The $\phi_o(\cdot)$ is set according to the problem under consideration. For the problems of binary classes, we set $\phi_o(\cdot)$ to the sigmoid function $\phi_o(z) = 1/(1 + \exp(-z))$, and for the problems of multiple classes, we set $\phi_o(\cdot)$ to the softmax function (Ripley, 1994) $\exp(z_i)/\sum_j \exp(z_j)$ to ensure the outputs sum to one. These settings also ensure the probability interpretation of the BNN outputs.

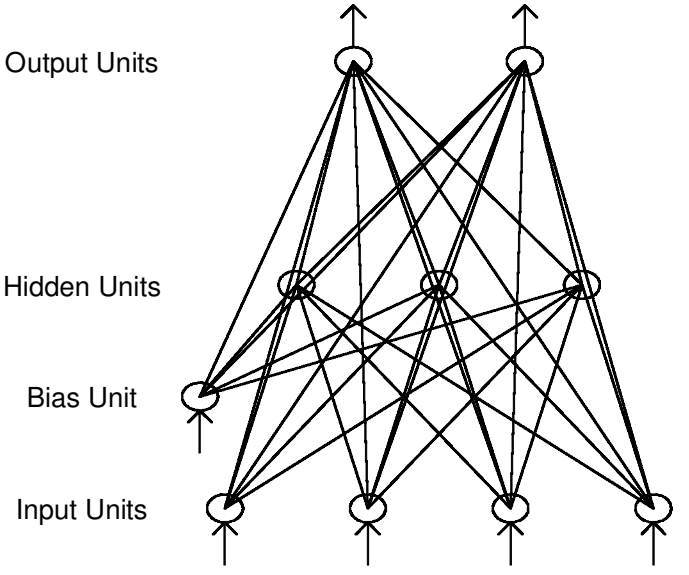

Figure 2: A fully connected one-hidden-layer feedforward neural network with four input units, three hidden units, and two output units. The arrows show the direction of data feeding, where each hidden unit independently processes the values fed to it by units in the preceding layer and then presents its output to units in the next layer for further processing.

For simplicity, in the following, we will suppress the bias terms α_{j0}, γ_{10}, $\ldots$, γ_{M0} by treating them as the weights on the connections exiting from an additional input unit with a constant input, say, $x_{i0} \equiv 1$. Let Λ be the vector consisting of all indicators of equation 3.1. Note that Λ specifies the structure of the network. Let $\boldsymbol{\alpha} = (\alpha_{jk})_{q\times(p+1)}$, $\boldsymbol{\beta} = (\beta_{jk})_{q\times M}$, $\boldsymbol{\gamma} = (\gamma_{jk})_{M\times(p+1)}$, and $\boldsymbol{\theta}_\Lambda = (\boldsymbol{\alpha}_\Lambda, \boldsymbol{\beta}_\Lambda, \boldsymbol{\gamma}_\Lambda)$, where $\boldsymbol{\alpha}_\Lambda$, $\boldsymbol{\beta}_\Lambda$, and $\boldsymbol{\gamma}_\Lambda$ denote the nonzero subsets of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$, respectively. Thus, the model 3.1 is completely specified by the tuple $(\Lambda, \boldsymbol{\theta}_\Lambda)$. For simplicity, in the following, we will denote the model by $\boldsymbol{\theta}_\Lambda$ only. To show the dependence of $g_j(\boldsymbol{x}_i)$ on the model, we will also write it as $g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda)$.

To conduct a Bayesian analysis for model 3.1, we first define the likelihood function as

$$P(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}_\Lambda) \propto \exp\{-\tau H(\boldsymbol{\theta}_\Lambda)\}, \tag{3.2}$$

where τ is a tunable parameter, and

$$H(\boldsymbol{\theta}_\Lambda) = \sum_{i=1}^{N}\sum_{j=1}^{q}[g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda) - y_{ij}]^2.$$

Note that this likelihood is not normalized. It can not be used directly for Bayesian inference, as its normalizing constant may contain the unknown parameters. With the same technique as that adopted by Sollich (2002), we avoid the problem by assuming an additional distribution for $\boldsymbol{x}$ such that the normalizing constant gets canceled out. If equation 3.2 is normalized as

$$P(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}_\Lambda) = \exp\{-\tau H(\boldsymbol{\theta}_\Lambda)\}/z_1(\boldsymbol{\theta}_\Lambda, \boldsymbol{x}),$$

where $z_1(\boldsymbol{\theta}_\Lambda, \boldsymbol{x})$ is chosen to normalize $P(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}_\Lambda)$ across all possible values of $\boldsymbol{y}$, then we choose

$$P(\boldsymbol{x} \mid \boldsymbol{\theta}_\Lambda) = Q(\boldsymbol{x}) z_1(\boldsymbol{\theta}_\Lambda, \boldsymbol{x})/z_2(\boldsymbol{\theta}_\Lambda),$$

where $Q(\cdot)$ is any positive function, and $z_2(\boldsymbol{\theta})$ is the normalizing constant. Thus, we have the joint distribution

$$\begin{aligned} P(\mathcal{D} \mid \boldsymbol{\theta}_\Lambda) &= P(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}_\Lambda) P(\boldsymbol{x} \mid \boldsymbol{\theta}_\Lambda) \\ &= \exp\{-\tau H(\boldsymbol{\theta}_\Lambda)\} Q(\boldsymbol{x})/z_2(\boldsymbol{\theta}_\Lambda), \end{aligned} \tag{3.3}$$

so $z_1(\boldsymbol{\theta}_\Lambda, \boldsymbol{x})$ is canceled out from the expression. More precisely, we can write $z_1(\boldsymbol{\theta}_\Lambda, \boldsymbol{x}) = \prod_{i=1}^N z_0(\boldsymbol{\theta}_\Lambda, \boldsymbol{x}_i)$, where $z_0(\boldsymbol{\theta}_\Lambda, \boldsymbol{x}_i)$ is the normalizing constant of the distribution,

$$P(\boldsymbol{y}_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda) = \exp\left\{-\tau \sum_{j=1}^{q} [g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda) - y_{ij}]^2\right\} / z_0(\boldsymbol{\theta}_\Lambda, \boldsymbol{x}_i),$$

corresponding to the ith observation; write $Q(\boldsymbol{x}) = \prod_{i=1}^N Q(\boldsymbol{x}_i)$; and write $z_2(\boldsymbol{\theta}_\Lambda) = [\int Q(\boldsymbol{x}_i) z_0(\boldsymbol{\theta}_\Lambda, \boldsymbol{x}_i)\, d\boldsymbol{x}_i]^N$, which depends on the sample size of the given data set.

We specify the following log-prior distribution for $\boldsymbol{\theta}_\Lambda$,

$$\begin{aligned} \log P(\boldsymbol{\theta}_\Lambda) = \text{Constant} &+ m_{\text{eff}} \log \lambda - \log(m_{\text{eff}}!) - \frac{m_{\text{eff}}}{2} \log(2\pi) \\ &- \frac{1}{2} \sum_{j=1}^{q} \sum_{k=0}^{p} I_{\alpha_{jk}} \left(\log \sigma_\alpha^2 + \frac{\alpha_{jk}^2}{\sigma_\alpha^2}\right) \\ &- \frac{1}{2} \sum_{j=1}^{q} \sum_{k=1}^{M} I_{\beta_j k} \delta\left(\sum_{l=0}^{p} I_{\gamma_{jl}}\right) \left(\log \sigma_\beta^2 + \frac{\beta_{jk}^2}{\sigma_\beta^2}\right) \\ &- \frac{1}{2} \sum_{j=1}^{M} \sum_{k=0}^{p} \delta\left(\sum_{l=1}^{q} I_{\beta_{lj}}\right) I_{\gamma_{jk}} \left(\log \sigma_\gamma^2 + \frac{\gamma_{jk}^2}{\sigma_\gamma^2}\right) \\ &+ \log z_2(\boldsymbol{\theta}_\Lambda), \end{aligned} \tag{3.4}$$

where $\delta(s)$ is 1 if $s > 0$ and 0 otherwise, m_{eff} is the total number of effective connections of $\boldsymbol{\theta}_\Lambda$, $m_{\text{eff}} = \sum_{j=1}^{q}\sum_{k=0}^{p} I_{\alpha_{jk}} + \sum_{j=1}^{q}\sum_{k=1}^{M} I_{\beta_{jk}}\delta(\sum_{l=0}^{p} I_{\gamma_{jl}}) + \sum_{j=1}^{M}\sum_{k=0}^{p}\delta(\sum_{l=1}^{q} I_{\beta_{lj}})I_{\gamma_{jk}}$. In the prior $P(\boldsymbol{\theta}_\Lambda)$, $z_2(\boldsymbol{\theta}_\Lambda)$ can be regarded as a weighting factor. Since the weighting factor depends on the sample size of the given data set, we may have some difficulty in probability interpretation. That is, if one constructs a distribution for a data set of size $N+1$ and then marginalizes over $\boldsymbol{x}_{N+1}$ and $\boldsymbol{y}_{N+1}$, one does not obtain the same distribution as if one had assumed from the start that there were only N points. As pointed out by Sollich (2002), "While this may seem objectionable on theoretical grounds, in most applications one has to deal with a single, given, data set—and thus a single value of N—and then this objection is irrelevant."

If the weighting factor were ignored, it is equivalent to assume the following independent priors for the network weights and structure: $\boldsymbol{\alpha}_\Lambda \sim N(0, \sigma_\alpha^2\mathbf{I}_\alpha)$, $\boldsymbol{\beta}_\Lambda \sim N(0, \sigma_\beta^2\mathbf{I}_\beta)$, and $\boldsymbol{\gamma}_\Lambda \sim N(0, \sigma_\gamma^2\mathbf{I}_\gamma)$, where $\mathbf{I}$ is an identity matrix with appropriate order and Λ is subject to a prior probability that is proportional to the mass put on m_{eff} by a truncated Poisson with rate λ,

$$\tilde{P}(\Lambda) = \begin{cases} \frac{1}{z_3}\frac{\lambda^{m_{\text{eff}}}}{m_{\text{eff}}!}, & m_{\text{eff}} = 3, 4, \ldots, U, \\ 0 & \text{otherwise}, \end{cases} \tag{3.5}$$

where $U = (p+1)(M+q) + Mq$ is the total number of connections of a model with all $I_\zeta = 1$, and $z_3 = \sum_{\Lambda\in\Omega}\lambda^{m_{\text{eff}}}/m_{\text{eff}}!$ Here we let Ω denote the set of all possible BNN structures with $3 \le m_{\text{eff}} \le U$. We set the minimum number of m_{eff} 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. The σ_α^2, σ_β^2, σ_γ^2, and λ are all hyperparameters to be specified by users (discussed below).

In equation 3.5, we use $\tilde{P}(\Lambda)$ to indicate it is a probability distribution, but not the real prior probability we put on Λ. Similarly, we let $\tilde{\boldsymbol{\pi}}(\boldsymbol{\theta}_\Lambda \mid \Lambda)$ denote the prior density of $\boldsymbol{\theta}_\Lambda$ given structure Λ by ignoring the weighting factor. Thus, we have the following log posterior,

$$\begin{aligned} \log P(\boldsymbol{\theta}_\Lambda \mid \mathcal{D}) &= \log P(\mathcal{D} \mid \boldsymbol{\theta}_\Lambda) + \log P(\boldsymbol{\theta}_\Lambda) - \log Z(\mathcal{D}) \\ &= \log Q(\boldsymbol{x}) - \tau H(\boldsymbol{\theta}_\Lambda) + \log\tilde{P}(\Lambda) + \log\tilde{\boldsymbol{\pi}}(\boldsymbol{\theta}_\Lambda \mid \Lambda) \\ &= \text{Constant} - \tau H(\boldsymbol{\theta}_\Lambda) + m_{\text{eff}}\log\lambda - \log(m_{\text{eff}}!) \\ &\quad - \frac{m_{\text{eff}}}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{q}\sum_{k=0}^{p} I_{\alpha_{jk}}\left(\log\sigma_\alpha^2 + \frac{\alpha_{jk}^2}{\sigma_\alpha^2}\right) \end{aligned}$$

$$- \frac{1}{2} \sum_{j=1}^{q} \sum_{k=1}^{M} I_{\beta_j k} \delta \left(\sum_{l=0}^{p} I_{\gamma_{jl}} \right) \left(\log \sigma_\beta^2 + \frac{\beta_{jk}^2}{\sigma_\beta^2} \right)$$

$$- \frac{1}{2} \sum_{j=1}^{M} \sum_{k=0}^{p} \delta \left(\sum_{l=1}^{q} I_{\beta_{lj}} \right) I_{\gamma_{jk}} \left(\log \sigma_\gamma^2 + \frac{\gamma_{jk}^2}{\sigma_\gamma^2} \right), \tag{3.6}$$

where $Z(\mathcal{D})$ is the normalizing constant of the posterior.

The rationale of the likelihood function can be argued as follows. First, we rewrite $H(\boldsymbol{\theta}_\Lambda)$ as

$$H(\boldsymbol{\theta}_\Lambda) = \sum_{j=1}^{q} \sum_{i=1}^{N} [g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda) - y_{ij}]^2 = \sum_{j=1}^{q} H_j(\boldsymbol{\theta}_\Lambda).$$

For each $H_j(\cdot)$, we have

$$\begin{aligned} H_j(\boldsymbol{\theta}_\Lambda) &= \sum_{i=1}^{N} [g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda) - y_{ij}]^2 \\ &= N \left\{ \frac{N_0}{N} \frac{1}{N_0} \sum_{y_{ij}=0} [g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda) - 0]^2 + \frac{N_1}{N} \frac{1}{N_1} \sum_{y_{ij}=1} [g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda) - 1]^2 \right\}, \end{aligned}$$

where N_0 and N_1 denote the number of cases of $y_{ij} = 0$ and 1, respectively. As $N \to \infty$,

$$\begin{aligned} \lim_{N\to\infty} \frac{1}{N} H_j(\boldsymbol{\theta}_\Lambda) &= P(y_j = 0) \int [g_j(\boldsymbol{x}, \boldsymbol{\theta}_\Lambda)]^2 P(\boldsymbol{x} \mid y_j = 0)\, d\boldsymbol{x} \\ &\quad + P(y_j = 1) \int [g_j(\boldsymbol{x}, \boldsymbol{\theta}_\Lambda) - 1]^2 P(\boldsymbol{x} \mid y_j = 1)\, d\boldsymbol{x} \\ &= \int [g_j(\boldsymbol{x}, \boldsymbol{\theta}_\Lambda)]^2 P(\boldsymbol{x})\, d\boldsymbol{x} \\ &\quad - 2 \int g_j(\boldsymbol{x}, \boldsymbol{\theta}_\Lambda) P(y_j = 1 \mid \boldsymbol{x}) P(\boldsymbol{x})\, d\boldsymbol{x} \\ &\quad + \int P(\boldsymbol{x}, y_j = 1)\, d\boldsymbol{x} \\ &= \text{constant} + \int [g_j(\boldsymbol{x}, \boldsymbol{\theta}_\Lambda) - P(y_j = 1 \mid \boldsymbol{x})]^2 P(\boldsymbol{x})\, d\boldsymbol{x}, \end{aligned}$$

where the constant term is independent of $\boldsymbol{\theta}_\Lambda$. Thus, we have

$$\begin{aligned} \lim_{N\to\infty} \frac{1}{N} H(\boldsymbol{\theta}_\Lambda) &= \text{constant} \\ &\quad + \sum_{j=1}^{q} \int [g_j(\boldsymbol{x}, \boldsymbol{\theta}_\Lambda) - P(y_j = 1 \mid \boldsymbol{x})]^2 P(\boldsymbol{x})\, d\boldsymbol{x}, \end{aligned} \tag{3.7}$$

which means that the maximizer of $-\tau H(\boldsymbol{\theta}_\Lambda)$ minimizes the distance of the network output from the true classification function in the limit of infinite data. We note that Duda, Hart, and Stork (2001) have a similar derivation with the above.

For a given input pattern $\boldsymbol{x}_0$ drawn from the same distributions with the training patterns, the posterior mean of $g_j(\boldsymbol{x}_0)$ can be written as

$$E[g_j(\boldsymbol{x}_0) \mid \mathcal{D}] = \frac{\sum_{\Lambda\in\Omega} \tilde{P}(\Lambda) \int g_j(\boldsymbol{x}_0, \boldsymbol{\theta}_\Lambda) \exp\{-\tau H(\boldsymbol{\theta}_\Lambda)\} \tilde{\boldsymbol{\pi}}(\boldsymbol{\theta}_\Lambda \mid \Lambda)\, d\boldsymbol{\theta}_\Lambda}{\sum_{\Lambda\in\Omega} \tilde{P}(\Lambda) \int \exp\{-\tau H(\boldsymbol{\theta}_\Lambda)\} \tilde{\boldsymbol{\pi}}(\boldsymbol{\theta}_\Lambda \mid \Lambda)\, d\boldsymbol{\theta}_\Lambda}, \quad j = 1, \ldots, q. \tag{3.8}$$

The integrals in equation 3.8 can be approximated using the following formula (Laplace's method):

$$\int b(\theta) \exp\{-nh(\theta)\}\, d\theta = (2\pi/n)^{p/2} |\Sigma|^{1/2} \exp\{-nh(\hat{\theta})\} b(\hat{\theta}) \{1 + O(n^{-1})\}, \tag{3.9}$$

as $n \to \infty$, where $b(\cdot)$ is a general function that does not depend on n, $h(\theta)$ is a constant-order function of n as $n \to \infty$, p is the dimension of θ, $\hat{\theta}$ is the maximizer of $-h(\theta)$, and $\Sigma = (D^2 h(\hat{\theta}))^{-1}$ is the inverse of the negative Hessian matrix evaluated at $\hat{\theta}$.

For the general formulation, see Kass and Vaidyanathan (1992). By applying Laplace's method to the numerator of equation 3.8 with $b(\cdot) = g_j(\boldsymbol{x}_0, \boldsymbol{\theta}_\Lambda)\boldsymbol{\pi}(\boldsymbol{\theta}_\Lambda \mid \mathcal{D})$, we have

$$\begin{aligned}
&\sum_{\Lambda\in\Omega} \tilde{P}(\Lambda) \int g_j(\boldsymbol{x}_0, \boldsymbol{\theta}_\Lambda) \exp\{-\tau H(\boldsymbol{\theta}_\Lambda)\} \tilde{\boldsymbol{\pi}}(\boldsymbol{\theta}_\Lambda \mid \Lambda)\, d\boldsymbol{\theta}_\Lambda \\
&\quad \approx \sum_{\Lambda\in\Omega} \tilde{P}(\Lambda) (2\pi/N)^{m_{\text{eff}}} |\Sigma_\Lambda|^{1/2} \\
&\qquad \times \exp\left\{-N\frac{\tau}{N} H(\hat{\boldsymbol{\theta}}_\Lambda)\right\} g_j(\boldsymbol{x}_0, \hat{\boldsymbol{\theta}}_\Lambda) \tilde{\boldsymbol{\pi}}(\hat{\boldsymbol{\theta}}_\Lambda \mid \mathcal{D}) \\
&\quad \approx P(y_j = 1 \mid \boldsymbol{x}_0) \sum_{\Lambda\in\Omega} \tilde{P}(\Lambda) (2\pi/N)^{m_{\text{eff}}} |\Sigma_\Lambda|^{1/2} \\
&\qquad \times \exp\left\{-N\frac{\tau}{N} H(\hat{\boldsymbol{\theta}}_\Lambda)\right\} \tilde{\boldsymbol{\pi}}(\hat{\boldsymbol{\theta}}_\Lambda \mid \mathcal{D}),
\end{aligned} \tag{3.10}$$

where the first approximation follows from the Laplace formula, 3.9, and the second approximation follows from the following fact: As $N \to \infty$, it follows from equation 3.7 that $g_j(\boldsymbol{x}, \hat{\boldsymbol{\theta}}_\Lambda) = P(y_j = 1 \mid \boldsymbol{x}_0)$ holds for all $j = 1, \ldots, q$ at the maximizer of $-\tau H(\boldsymbol{\theta}_\Lambda)/N$. Here we assume that the number of hidden

units of Λ is sufficiently large to approximate the true classification function arbitrarily well with properly adjusted weights (White, 1992). $j = 1, \ldots, q$.

Similarly, by applying Laplace's method to the denominator of equation 3.8 with $b(\cdot) = \boldsymbol{\pi}(\boldsymbol{\theta}_\Lambda \mid \mathcal{D})$, we have

$$\sum_{\Lambda \in \Omega} \tilde{P}(\Lambda) \int \exp\{-\tau H(\boldsymbol{\theta}_\Lambda)\} \tilde{\boldsymbol{\pi}}(\boldsymbol{\theta}_\Lambda \mid \Lambda)\, d\boldsymbol{\theta}_\Lambda$$
$$\approx \sum_{\Lambda \in \Omega} \tilde{P}(\Lambda)(2\pi/N)^{m_{\text{eff}}} |\Sigma_\Lambda|^{1/2} \exp\left\{-N\frac{\tau}{N} H(\hat{\boldsymbol{\theta}}_\Lambda)\right\} \tilde{\boldsymbol{\pi}}(\hat{\boldsymbol{\theta}}_\Lambda \mid \mathcal{D}). \quad (3.11)$$

Following from equations 3.10 and 3.11 and the approximation accuracy ($O(n^{-1})$) of Laplace's method, we have

$$E[g_j(\boldsymbol{x}_0) \mid \mathcal{D}] \longrightarrow P(y_{0j} = 1 \mid \boldsymbol{x}_0), \quad j = 1, \ldots, q, \quad (3.12)$$

for a given τ as $N \to \infty$. This result will be further confirmed by our numerical results in section 5. The immediate implication of equation 3.12 is that if the posterior distribution, equation 3.6, can be sampled from by an MCMC method, then $P(y_{0j} = 1 \mid \boldsymbol{x}_0)$ can be estimated by

$$\widehat{E[g_j(\boldsymbol{x}_0 \mid \mathcal{D})]} = \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} g(\boldsymbol{x}_0, \boldsymbol{\theta}_{\Lambda,i}), \quad (3.13)$$

where $\boldsymbol{\theta}_{\Lambda,1}, \boldsymbol{\theta}_{\Lambda,2}, \ldots, \boldsymbol{\theta}_{\Lambda,\mathcal{N}}$ denote the posterior samples from an MCMC run. Following from standard MCMC results (Smith & Roberts, 1993) and equation 3.12, we have

$$\widehat{E[g_j(\boldsymbol{x}_0 \mid \mathcal{D})]} \longrightarrow P(y_{0j} = 1 \mid \boldsymbol{x}_0) \quad a.s. \quad \text{for all } j = 1, \ldots, q \quad (3.14)$$

for a given τ as $N \to \infty$ and $\mathcal{N} \to \infty$. Hence, the estimator, equation 3.13, is consistent conditional on the given data set $\mathcal{D}$, and we can state that with a sufficient number of training samples and an effective MCMC sampler, the true classification function can be approximated by equation 3.13 with a sufficient number of posterior samples. Thus, if $\boldsymbol{x}_0$ is a new sample point, its class membership can be naturally predicted (based on the given sample $(\boldsymbol{x}, \boldsymbol{y})$) by the values of $\widehat{E[g_j(\boldsymbol{x}_0 \mid \mathcal{D})]}$'s. For the problems of binary classes, it will be classified as class 1 if $\widehat{E[g_1(\boldsymbol{x}_0 \mid \mathcal{D})]} \geq 0.5$ and class 0 otherwise. For the problems of multiple classes, it will be classified as class κ if $\kappa = \arg\max_{1 \leq j \leq q} \widehat{E[g_j(\boldsymbol{x}_0 \mid \mathcal{D})]}$. Considering the difficulty of probability interpretation caused by the data-size dependent prior, the above prediction procedure can be regarded as an approximation to a proper Bayesian prediction. But it really works well in practice, as shown in this letter.

For data preparation and hyperparameter setting, we have the following suggestions. To avoid some weights that are trained to be extremely large or small to accommodate different scales of the input variables, we suggest that all input variables be normalized before feeding to the networks. The hyperparameters include σ_α^2, σ_β^2, σ_γ^2, λ, τ, and M. Based on the belief that a network with a large weight variation usually has a poor generalization performance, we suggest that $\sigma_\alpha^2, \sigma_\beta^2$, and σ_γ^2 should be set to moderate values to penalize a large weight variation. For example, we set $\sigma_\alpha^2 = \sigma_\beta^2 = \sigma_\gamma^2 = 5$ for all examples of this letter. The setting should also be fine for the other problems. For M, we suggest that it should be large enough such that the full model is able to approximate the true function well. Our experience shows that the classifier is almost independent of M as long as it has been large enough. The λ and τ together control the network size. Just like the C of SVMs, they control the trade-off between the network complexity and the training error. The larger are λ and τ, the larger are the network size and the smaller the training error. In practice, a cross-validation procedure can be applied to determine an optimal setting for them. In this article, we usually give them several settings for each example to show that the new classifier is not sensitive to the prior settings. A rule of thumb is to choose λ and τ such that the number of effective connections is roughly one-tenth of the total number of training patterns (Weigend, Huberman, & Rumelhart, 1990), although in practice, many successful neural networks employ more than this number of connections.

For comparison, we also give a brief description of the commonly used BNN classifier (Neal, 1996). For simplicity, hereafter we will call Neal's classifier the old BNN classifier, and call the one we have proposed the new BNN classifier. They are different in three respects:

1. Likelihood function. The old BNN classifier assumes the likelihood function

$$f^*(\mathcal{D} \mid \boldsymbol{\theta}_\Lambda) = \prod_{i=1}^{N} g_1(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda)^{y_i} (1 - g_1(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda))^{1-y_i} \tag{3.15}$$

for the problems of two classes, and the likelihood function

$$f^*(\mathcal{D} \mid \boldsymbol{\theta}_\Lambda) = \prod_{i=1}^{N} \prod_{j=1}^{q} g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda)^{y_{ij}} \tag{3.16}$$

for the problems of multiple classes. Figure 3 compares the term $-\tau H(\boldsymbol{\theta}_\Lambda)$ (which plays the role of log-likelihood function for the new BNN classifier) and the log-likelihood value of equation 3.15 for a specific input pattern with class label $y = 1$. It is clear that equation 3.15 puts a more severe penalty on the misclassified patterns than does

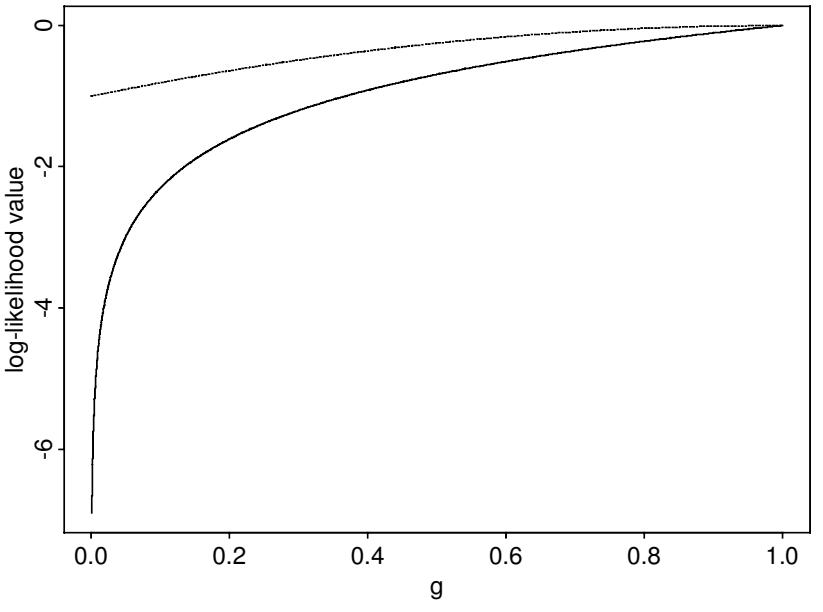

Figure 3: Comparison of $\tau H(\boldsymbol{\theta}_\Lambda)$ (dotted curve, $\tau = 1$) and the log-likelihood value of equation 3.15 (solid curve) for a specific input pattern $(\boldsymbol{x}_0, \boldsymbol{y}_0)$ with $\boldsymbol{y}_0 = 1$. The x-axis represents the network output $g(\boldsymbol{x}_0, \boldsymbol{\theta}_\Lambda)$, that is, the estimated probability $P(\boldsymbol{y}_0 = 1 \mid \boldsymbol{x}_0, \boldsymbol{\theta}_\Lambda)$.

equation 3.2. It implies that the old BNN classifier may have a tendency to overfit to the data, and thus may be inferior to the new one in generalization.

2. Prior specification. Neal (1996) developed a so-called automatic relevance determination (ARD) prior for the old BNN classifier. In the ARD prior, each input variable has associated with it a hyperparameter that controls the magnitudes of the weights on the connections out of that input unit.

3. Network structure. The old BNN classifier works on a prefixed structure. But the ARD prior works as a model selection mechanism for input variables. The hyperparameters in the prior control the magnitudes of the related weights, and thus control the effect of the input variables on output. See Neal (1996) for more descriptions for the ARD prior. In the new BNN classifier, each connection weight associates with an indicator variable. The resulting network structure is often sparse and depends less on the initial specification for the input variables and the number of hidden units.

The major difference between the new and old BNN classifiers is in likelihood instead of the prior specification and the network structure selection. In the latter two respects, they share some common features.

4 Computational Implementation

In this article, we propose an MCMC sampler, so called the tempered reversible jump MCMC (TRJ-MCMC), for simulation from the posterior of the BNN models. TRJ-MCMC is a combination of parallel tempering (Geyer, 1991; Hukushima & Nemoto, 1996) and reversible jump MCMC (RJMCMC) (Green, 1995) and has incorporated their advantages. It allows dimensional jumping of the systems, as in RJMCMC, but avoids being stuck in local energy minima by the simulations at high temperature levels as in parallel tempering. The algorithm is described as follows.

Suppose we would like to sample from a distribution $f(\boldsymbol{\xi}) \propto \exp(-\mathbb{E}(\boldsymbol{\xi}))$, where $\mathbb{E}(\cdot)$ denotes an energy function of $\boldsymbol{\xi}$ and it corresponds to the negative log-posterior value in a simulation from a posterior distribution. In TRJ-MCMC, a sequence of distributions $f_1(\boldsymbol{\xi}), \ldots, f_n(\boldsymbol{\xi})$ is first constructed: $f_i(\boldsymbol{\xi}) \propto \exp\{-\mathbb{E}(\boldsymbol{\xi})/t_i\}$, $i = 1, \ldots, n$, where t_i is called the temperature of $f_i(\cdot)$. The temperature sequence $\boldsymbol{t} = (t_1, \ldots, t_n)$ forms a ladder with $t_1 > \cdots > t_n \equiv 1$. Issues related to the choice of the temperature ladder can be found in Geyer and Thompson (1995) and Liang and Wong (2000). Let $\boldsymbol{\xi}^i$ denote a sample from $f_i(\cdot)$; we follow Liang and Wong (2000) in calling it an individual. The n individuals $\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^n$ form a population denoted by $\boldsymbol{z} = \{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^n\}$. We further assume that all individuals in the same population $\boldsymbol{z}$ are mutually independent, and they have the following joint density:

$$f(\boldsymbol{z}) \propto \exp\left\{-\sum_{i=1}^{n} \mathbb{E}(\boldsymbol{\xi}^i)/t_i\right\}. \tag{4.1}$$

TRJ-MCMC works by sampling from $f(\boldsymbol{z})$ directly. The samples of interest can be collected at the target temperature level. TRJ-MCMC consists of two operators, local updating and exchange, described in the following sections.

4.1 Local Updating. In the local updating operator, an individual, say $\boldsymbol{\xi}^k$, is selected at random from the current population $\boldsymbol{z}$. Then $\boldsymbol{\xi}^k$ is modified to a new individual $\boldsymbol{\xi}^{k'}$ by some type of moves (described below). The new population $\boldsymbol{z}' = \{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{k-1}, \boldsymbol{\xi}^{k'}, \boldsymbol{\xi}^{k+1}, \ldots, \boldsymbol{\xi}^n\}$ is accepted with probability $\min(1, r_l)$ according to the Metropolis-Hastings rule (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953; Hastings, 1970), and keeps $\boldsymbol{z}$

unchanged with the remaining probability, where

$$r_l = \frac{f(z')}{f(z)} \frac{T(z \mid z')}{T(z' \mid z)} = \exp\{-(\mathbb{E}(\xi^{k'}) - \mathbb{E}(\xi^k))/t_k\} \frac{T(z \mid z')}{T(z' \mid z)}, \tag{4.2}$$

and $T(\cdot \mid \cdot)$ denotes the transition probability between two populations.

The move types we used for this article are "birth," "death," and "Metropolis" as in Green (1995). 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 $m_{\text{eff}} = \|S\|$ be the number of elements in S (recall the definition of m_{eff} in section 2); thus, $\|S^c\| = U - m_{\text{eff}}$. Let $P(m_{\text{eff}}, \text{birth})$, $P(m_{\text{eff}}, \text{death})$, and $P(m_{\text{eff}}, \text{Metropolis})$ denote the proposal probabilities of the three types of moves for a network with m_{eff} connections. In our examples, we set $P(m_{\text{eff}}, \text{birth}) = P(m_{\text{eff}}, \text{death}) = 1/3$ for $3 < q < U$, $P(U, \text{death}) = P(3, \text{birth}) = 2/3$, $P(U, \text{birth}) = P(3, \text{death}) = 0$, and $P(m_{\text{eff}}, \text{Metropolis}) = 1/3$ for $3 \le q \le U$. For convenience, in the remaining part of this section, we will let i be an index of the output units, j be an index of the hidden units, and l be an index of the bias unit and input units. Their respective ranges are $1 \le i \le q$, $1 \le j \le M$ and $0 \le l \le p$. In the "birth," move, a connection, say, ζ, is first uniformly chosen from S^c. If $\zeta \in \{\alpha_{il} : I_{\alpha_{il}} = 0\} \cup \{\beta_{ij} : I_{\beta_{ij}} = 0, \sum_{u=1}^{q} I_{\beta_{uj}} \ge 1\} \cup \{\gamma_{jl} : I_{\gamma_{jl}} = 0, \sum_{u=0}^{p} I_{\gamma_{ju}} \ge 1\}$, the move is called the type I "birth" move (illustrated by Figure 4a). In this type of move, only the connection ζ is proposed to add to ξ^k with weight ω_ζ being randomly drawn from $N(0, \sigma^2_{\xi^k})$, where $\sigma^2_{\xi^k} = \sum_{i=1}^{m_{\text{eff}}} (w_i - \bar{w})^2/(m_{\text{eff}} - 1)$, $w_1, \ldots, w_{m_{\text{eff}}}$ denote the effective weights of ξ^k, and $\bar{w} = \sum_{i=1}^{m_{\text{eff}}} w_i/m_{\text{eff}}$. The resulting transition probability ratio is

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

where $\phi(\cdot)$ denotes the standard normal density, and m'_{del} denotes the number of deletable connections of the network $\xi^{k'}$. A connection, say, υ, is deletable from a network if $\upsilon \in \{\alpha_{il} : I_{\alpha_{il}} = 1\} \cup \{\beta_{ij} : I_{\beta_{ij}} = 1, \sum_{u=1}^{q} I_{\beta_{uj}} > 1\} \cup \{\beta_{ij} : I_{\beta_{ij}} = 1, \sum_{u=1}^{q} I_{\beta_{uj}} = 1, \sum_{u=0}^{p} I_{\gamma_{ju}} = 1\} \cup \{\gamma_{jl} : I_{\gamma_{jl}} = 1, \sum_{u=0}^{p} I_{\gamma_{ju}} > 1\} \cup \{\gamma_{jl} : I_{\gamma_{jl}} = 1, \sum_{u=1}^{q} I_{\beta_{uj}} = 1, \sum_{u=0}^{p} I_{\gamma_{ju}} = 1\}$.

If $\zeta \in \{\beta_{ij} : I_{\beta_{ij}} = 0, \sum_{u=1}^{q} I_{\beta_{uj}} = 0, \sum_{u=0}^{p} I_{\gamma_{ju}} = 0\} \cup \{\gamma_{jl} : \sum_{u=1}^{q} I_{\beta_{uj}} = 0, \sum_{u=0}^{p} I_{\gamma_{ju}} = 0\}$, the move is called the type II "birth" move (illustrated by Figure 4b), which is to add a new hidden unit. In this type of move, if $\zeta \in \{\beta_{ij} : I_{\beta_{ij}} = 0, \sum_{u=1}^{q} I_{\beta_{uj}} = 0, \sum_{u=0}^{p} I_{\gamma_{ju}} = 0\}$, a connection, say, ζ', is uniformly chosen from the set of connections from input units to the corresponding hidden unit; otherwise, ζ' is uniformly chosen from the set of connections from the corresponding hidden unit to the output units, and then ζ and ζ' are proposed to add to ξ^k together. The weights ω_ζ and $\omega_{\zeta'}$ are

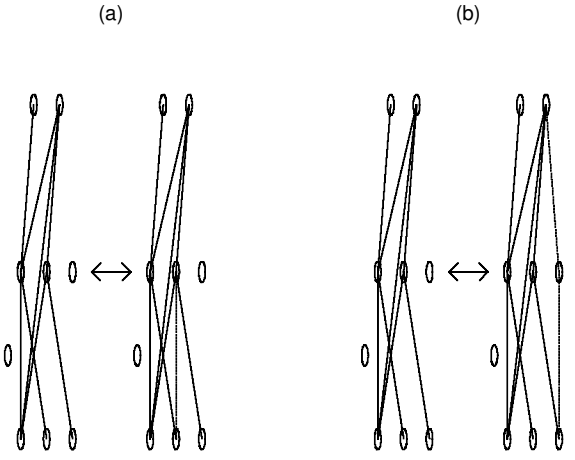

Figure 4: Illustration of the "birth" and "death" moves. The dotted lines denote the connections to add to or delete from the current network. (a) Type I "birth" ("death") move: There is only one connection to add to (from left to right) or delete from (from right to left) the current network. (b) Type II "birth" ("death") move: There are two connections to add to (from left to right) or delete from (from right to left) the current network simultaneously. The two connections must be the only two connections associated with that hidden unit; one must be from input units to that hidden unit, and the other one must be from that hidden unit to output units.

drawn independently from $N(0, \sigma^2_{\xi^k})$. The resulting transition probability ratio is

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

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

In the "death" move, a connection, say, ζ, is first uniformly chosen from the set of deletable connections of $\boldsymbol{\xi}^k$. If $\zeta \in \{\alpha_{il} : I_{\alpha_{il}} = 1\} \cup \{\beta_{ij} : I_{\beta_{ij}} = 1, \sum_{u=1}^{q} I_{\beta_{uj}} > 1\} \cup \{\gamma_{jl} : I_{\gamma_{jl}} = 1, \sum_{u=0}^{p} I_{\gamma_{ju}} > 1\}$, the move is called the type I "death" move (illustrated by Figure 4a). In this type of move, only ζ is proposed for deletion, and the resulting transition probability ratio is

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

where m_{del} denotes the number of deletable connections of $\boldsymbol{\xi}^k$, and σ'_{ξ^k} denotes the sample standard deviation of the connection weights of $\boldsymbol{\xi}^k$ except

ζ. If $\zeta \in \{\beta_{ij} : I_{\beta_{ij}} = 1, \sum_{u=1}^{q} I_{\beta_{uj}} = 1, \sum_{u=0}^{p} I_{\gamma_{ju}} = 1\} \cup \{\gamma_{jl} : I_{\gamma_{jl}} = 1, \sum_{u=1}^{q} I_{\beta_{uj}} = 1, \sum_{u=0}^{p} I_{\gamma_{ju}} = 1\}$, the move is called the type II "death" move (illustrated by Figure 4b), which is to eliminate one hidden unit from the model. In this type of move, if $\zeta \in \{\beta_{ij} : I_{\beta_{ij}} = 1, \sum_{u=1}^{q} I_{\beta_{uj}} = 1, \sum_{u=0}^{p} I_{\gamma_{ju}} = 1\}$, the only connection denoted by ζ' from some input unit to the corresponding hidden unit is also proposed for deletion together with ζ; otherwise, the connection denoted by ζ' from the corresponding hidden unit to some output unit is proposed for deletion together with ζ. The resulting transition probability ratio is

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

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 article, we implemented the move in a Metropolis-within-Gibbs sampler (Müller, 1993; Robert & Casella, 1999). In any step of the Gibbs sampler with a difficulty sampling from the conditional distribution, substitute a one-step Metropolis-Hastings move. Let the network weights be arranged first into a vector, $\boldsymbol{\xi}^k = (\xi_1^k, \ldots, \xi_{m_{\text{eff}}}^k)$, by some fixed order. Let $\boldsymbol{\xi}_{[a,b]}^k = (\xi_a^k, \xi_{a+1}^k, \ldots, \xi_b^k)$ and $\boldsymbol{\xi}_{-[a,b]}^k$ denote all elements of $\boldsymbol{\xi}^k$ except the set $\boldsymbol{\xi}_{[a,b]}^k$. Let κ be the maximum block size allowed by the move. The weights are updated iteratively as follows:

- Update $\boldsymbol{\xi}_{[1,\kappa]}^k$ by one step of Metropolis-Hastings (MH) simulation from the conditional density $f(\boldsymbol{\xi}_{[1,\kappa]}^k \mid \boldsymbol{\xi}_{-[1,\kappa]}^k)$.
- Update $\boldsymbol{\xi}_{[\kappa+1,2\kappa]}^k$ by one step of MH simulation from the conditional density $f(\boldsymbol{\xi}_{[\kappa+1,2\kappa]}^k \mid \boldsymbol{\xi}_{-[\kappa+1,2\kappa]}^k)$.
- .....
- Update $\boldsymbol{\xi}_{[(r-1)\kappa+1,m_{\text{eff}}]}^k$ by one step of MH simulation from the conditional density $f(\boldsymbol{\xi}_{[(r-1)\kappa+1,m_{\text{eff}}]} \mid \boldsymbol{\xi}_{-[(r-1)\kappa+1,m_{\text{eff}}]})$.

The first block is updated as follows. First, a random direction d_1 is uniformly drawn from the κ-dimensional space. Then set

$$\boldsymbol{\xi}_{[1,\kappa]}^{k'} = \boldsymbol{\xi}_{[1,\kappa]}^k + \rho_k d_1,$$

where ρ_k is a random variable drawn from a gaussian distribution with mean 0 and variance ηt_k. The η is a user-specified parameter, the so-called Metropolis step size. It controls the acceptance rate of the move. The $\boldsymbol{\xi}_{[1,\kappa]}^{k'}$ is accepted according to the Metropolis-Hastings rule. Once it is accepted, the current $\boldsymbol{\xi}^k$ will be updated immediately by replacing $\boldsymbol{\xi}_{[1,\kappa]}^k$ by $\boldsymbol{\xi}_{[1,\kappa]}^{k'}$. 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 $T(\boldsymbol{z} \mid \boldsymbol{z}') = T(\boldsymbol{z}' \mid \boldsymbol{z})$.

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

$$r_e = \frac{f(\boldsymbol{z}')}{f(\boldsymbol{z})} \frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})}$$
$$= \exp\left\{ (H(\boldsymbol{\xi}^i) - H(\boldsymbol{\xi}^j)) \left(\frac{1}{t_i} - \frac{1}{t_j} \right) \right\} \frac{T(\boldsymbol{z} \mid \boldsymbol{z}')}{T(\boldsymbol{z}' \mid \boldsymbol{z})}. \tag{4.3}$$

Typically, the exchange is performed on only two states with neighboring temperatures, $|i - j| = 1$. If $p(\boldsymbol{\xi}^i)$ is the probability that $\boldsymbol{\xi}^i$ is chosen to exchange with the other state and $w_{i,j}$ is the probability that $\boldsymbol{\xi}^j$ is chosen to exchange with $\boldsymbol{\xi}^i$, we have $T(\boldsymbol{z}' \mid \boldsymbol{z}) = p(\boldsymbol{\xi}^i) w_{i,j} + p(\boldsymbol{\xi}^j) w_{j,i}$. Thus, $T(\boldsymbol{z}' \mid \boldsymbol{z}) / T(\boldsymbol{z} \mid \boldsymbol{z}') = 1$ in equation 4.3.

4.3 Algorithm. With the operators described above, one iteration of TRJ-MCMC consists of the following two steps.

1. Try to update each of the individuals of the current population by the local updating operator.

2. Try to exchange $\boldsymbol{\xi}^i$ with $\boldsymbol{\xi}^j$ for $n - 1$ pairs (i, j) with i being sampled uniformly on $\{1, \ldots, n\}$ 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 $w_{1,2} = w_{n,n-1} = 1$.

In this algorithm, the user-specified parameters include the population size n, the temperature ladder $\boldsymbol{t}$, block size κ, and the Metropolis step size η. For parameter setting, we have the following suggestions. First, set the temperature ladder carefully. Set the highest temperature such that almost all local updating moves can be accepted, and set the intermediate temperatures such that the exchange operations have an almost equal acceptance rate for all neighboring levels. For all examples of this article, we set the temperature ladder to be a series of equal harmonic difference, that is, $1/t_{i+1} - 1/t_i = c$ for $i = 1, 2, \ldots, n - 1$, where c is a constant. Then set κ to a moderate value. It may range from tens to 100 in our experience. In this article, we set $\kappa = 50$ for all examples. Next, set η such that the Metropolis moves have a moderate acceptance rate, for example, 0.2 to 0.4, as suggested by Gelman, Roberts, and Gilks (1996).

5 Numerical Results

5.1 Two Simulated Examples. BNNs were first tested with two simulated examples with known optimal classification boundaries. Example 1 contains 50 pairs of training and test data sets, which are all mutually independent. Each training set consists of 50 independent and identically distributed (i.i.d.) samples drawn from $N_2(\boldsymbol{\mu}, \mathbf{I_2})$ (class 1) and 50 i.i.d. samples drawn from $N_2(-\boldsymbol{\mu}, \mathbf{I_2})$ (class 2); each test set consists of 500 i.i.d. samples drawn from $N_2(\boldsymbol{\mu}, \mathbf{I_2})$ and 500 i.i.d. samples drawn from $N_2(-\boldsymbol{\mu}, \mathbf{I_2})$, where $\boldsymbol{\mu} = (1.1, 1.1)'$ and $\mathbf{I_2}$ is a 2×2 identity matrix. It is clear that the optimal classification boundary of each data set is the straight line $y = -x$ in the two-dimensional Cartesian coordinates.

The new BNN classifier was first applied to the example with $M = 2$, $\tau = 0.1$ and a variety of λ's with values ranging from 10 to 25. Here we set $M = 2$, as the corresponding full model will have 11 connections and that should have been enough for this example to approximate the true classification boundary. The τ and λ were chosen by a 10-fold cross-validation procedure for 10 training sets; that is, each training set was randomly divided into 10 disjoint sets of equal size, the classifier was trained 10 times, each time with a different set held out as a validation set, and the quality of the parameter settings was assessed by averaging over the 100 errors (10 sets $\times$ 10-fold runs). In training, a temperature ladder of 10 levels was used with the highest temperature $t_1 = 5$ and the Metropolis step size $\eta = 5$. Each run consists of 6000 iterations. The first 1000 iterations were discarded for the burn-in process, and the followed 5000 iterations were used for inference. The results are summarized in Table 1. Here the values of τ are very small, as we expect the learned classification boundary to be an approximately straight line and not to be strongly affected by the samples near the boundary. While given τ, the influence of λ on the classifier is not significant. For the other examples, the τ and λ were determined similarly; we will skip the details of the cross-validation experiments and provide only the final settings.

In the formal training, we have the same MCMC setting as in the cross-validation training. For each data set, the classifier was run 10 times indepen-

Table 1: Cross-Validation Experiment for the First Simulated Example.

Parameter	$\tau = 0.05$		$\tau = 0.1$		$\tau = 0.15$		$\tau = 0.2$	
Setting	Training	Test	Training	Test	Training	Test	Training	Test
$\lambda = 10$	5.23	0.63	5.32	0.60	5.20	0.60	5.16	0.62
$\lambda = 15$	5.29	0.63	5.22	0.60	5.21	0.63	5.13	0.62
$\lambda = 20$	5.28	0.63	5.23	0.60	5.20	0.61	5.14	0.60
$\lambda = 25$	5.29	0.59	5.22	0.61	5.32	0.61	5.14	0.63

Note: The columns show the averaged 10-fold training and test errors for 10 data sets.

dently. Each run consists of 6000 iterations, and the last 5000 iterations were used for inference. The overall acceptance rates of the local updating moves and exchange operations are about 0.55 and 0.75, respectively. The training error (reported in the "Training" column in Table 2) was computed as the average of the 500 training errors (50 data sets $\times$ 10 runs), and the test error (reported in the "Test" column in Table 2) was computed as the average of the 500 test errors. The standard deviations of them (the figures in parentheses in Table 2) were computed in the formula $SD_e = \sqrt{\sum_{i=1}^{50} \widehat{\mathrm{var}}(\bar{e}_i)}/50$, where e indicates the error type, training error, or test error; $\bar{e}_i$ denotes the averaged error (over 10 runs) of the ith data set; and $\widehat{\mathrm{var}}(\bar{e}_i) = \frac{1}{(10)(9)} \sum_{j=1}^{10} (e_{ij} - \bar{e}_i)^2$. The same computational procedure for the training and test errors and their standard deviations was also applied to the old BNN classifier and other examples in this article.

For comparison, we also applied the old BNN classifier and SVMs to the same data sets. The software for the old BNN classifier was developed by R. M. Neal and was downloaded on-line from http://www.cs.toronto.edu/~radford (as suggested by a referee). Refer to Neal (1996) for more details of the software. For each data set, it was also run 10 times independently. Each run consists of 30 "iterations" and costs about the same CPU time on the same computer as that of the new BNN classifier. Note that "Neal's iteration" is different from ours; it contains many substeps, Gibbs sampling, heat bath sampling, and hybrid Monte Carlo. Fortunately, with 30 iterations, the results have been reliable enough for comparison. The numbers of hidden units we tried for the old classifier include 2, 3, 4, and 5. The results were summarized in Table 2. The SVM software we used is LIBSVM, which was developed by Chang and Lin (2001) and was downloaded on-line from http://www.csie.ntu.edu.tw /~cjlin. A linear kernel was first tried with different values of C ranging from 0.001 to 10. The computational results were also summarized in Table 2. Since SVMs do a deterministic computation and are very fast, they were run only one time for each data set, and the CPU time they cost was not recorded. The standard deviations of the averaged training and test errors can be regarded as 0. The other types of kernels, such as the RBF and polynomial kernels, were also tried for this example. Their generalization performances were all worse than that of the linear kernel. This is reasonable as the true boundary of this example is a linear function.

Table 2 shows that the new BNN classifier is superior to the old BNN classifier in test errors but inferior in training errors. This is consistent with our conjecture: The old BNN classifier will tend to overfit to the data and perform less well in generalization, since it puts a more severe penalty on the misclassified patterns. Table 2 also shows that with the correct kernel and the suitable value of C, SVM performs competitively with the new BNN classifier in generalization for this example. But for more complicated problems, our experiments show that SVM is usually inferior to the new

Table 2: Comparison of the New BNN Classifier, the Old BNN Classifier, and SVM for the First Simulated Example.

	Parameter Setting	CPU(s)	Training	Test
New BNN	$\lambda = 10$	47.1	5.426(0.026)	62.032(0.097)
	$\lambda = 15$	47.3	5.416(0.026)	61.708(0.100)
	$\lambda = 20$	47.5	5.388(0.023)	61.726(0.096)
	$\lambda = 25$	47.6	5.364(0.024)	61.706(0.099)
Old BNN	$M = 2$	39.9	5.206(0.026)	63.52(0.11)
	$M = 3$	47.6	5.192(0.029)	63.43(0.12)
	$M = 4$	55.3	5.130(0.030)	63.62(0.11)
	$M = 5$	62.6	5.166(0.031)	63.63(0.12)
SVM (linear)	$C = 0.001$	—	7.24	72.32
	$C = 0.01$	—	5.36	61.94
	$C = 0.10$	—	5.60	61.64
	$C = 1.0$	—	5.40	63.40
	$C = 10.0$	—	5.18	64.16

Notes: The "CPU(s)" column shows the total CPU time (in seconds) cost by 10 independent runs for each data set. The "Training" and "Test" columns show the averaged training and test errors and their respective standard deviations (in parentheses).

BNN classifier as shown below. Figures 5a, 5b, and 5c show the decision regions of the three classifiers for one of the 50 data sets. They were generated by the three classifiers with $\lambda = 15$, $M = 3$, and $C = 0.1$, respectively. The plots show again that for this example, the old BNN classifier has a tendency to overfit compared to the other two classifiers, and the new BNN classifier has a comparable performance with SVM, although the situation is not favorable to it. The true classification boundary of this example is linear, and the new BNN classifier approximates the linear boundary.

As a by-product, we show that the new BNN classifier has the ability to select relevant variables from the pool of explanatory variables. To show this, we retrain a training set randomly selected from the above 50 ones, but with one additional input variable x_3, which consists of 100 i.i.d. samples drawn from $N(0, 1)$ independent of x_1 and x_2. In training, we have the same MCMC parameter setting and the same network parameter setting as described above except for $\tau = 5$, $\lambda = 0.5$, and a total of 51,000 iterations. Here we set $\tau = 5$ and $\lambda = 0.5$ to drive the networks to have a very good approximation to the targets but with a very small network size, that is, a very small number of effective connections. This setting is based on our belief that a neural network with a very good approximation to the targets but with a minimum structure should include the most relevant, linearly or nonlinearly, variables as the input variables. Note that the resulting networks may have a poor generalization performance due to the extremeness (minimum structure and maximum approximation) we pursued in the

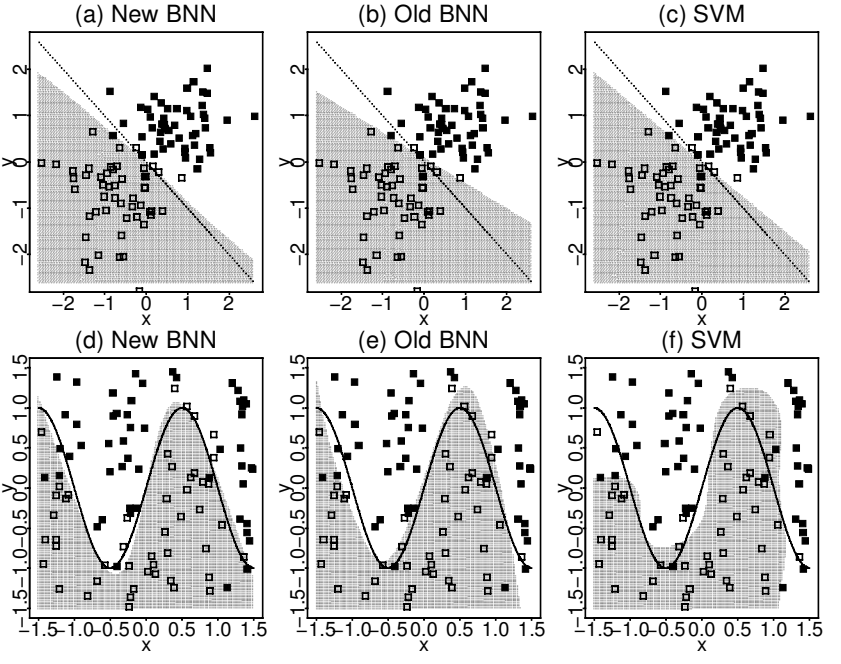

Figure 5: The decision regions learned by the three classifiers for two simulated data sets. The training data are shown as squares: the filled squares for class 1 and the open squares for class 2. (a–c) The decision regions of the three classifiers for one data set of the first simulated example. The dotted lines show the true classification boundaries of the data set. (d–f) The decision regions of the three classifiers for one data set of the second simulated example. The sine curves show the true classification boundaries of the data set.

training process. Under the above setting, we have $m_{\text{eff}} = 3.36$ on average, and a classification accuracy of 96% for the training set. Figure 6a shows the posterior probabilities of each of the connections of the full model, where the names of the bars show the corresponding input or hidden units. For example, starting from the left, the first, second, and third x_1's correspond to the connections from the input variable x_1 to the first hidden unit, the second hidden unit, and the output unit, respectively. The x_0's and x_2's can be explained in the same way. The h_1 and h_2 indicate the connections from the two hidden units to the output unit. The high posterior probabilities of the connections from x_1 and x_2 to the hidden units indicate the nonlinear relevance of x_1 and x_2 to the target variable. Figure 6b shows the averaged frequencies of the connections exiting from each of the input variables. The frequencies were computed based on the networks sampled from the pos-

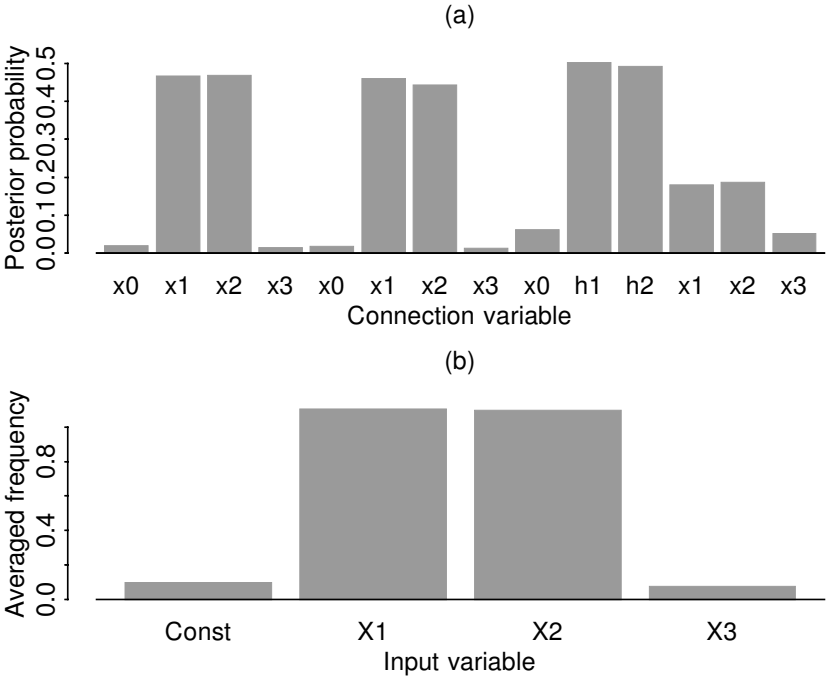

Figure 6: (a) The posterior probabilities of each of the connections of the full BNN model, which has three input units, two hidden units, and one output unit. (b) The averaged frequencies of the connections exiting from each of the input variables.

terior distribution. For example, the averaged frequency of x_1 is equal to the sum of the posterior probabilities of all connections out of it. The frequencies reflect the relative importance of each of the input variables for the explanation for the target. As we expected, Figure 6b shows that the role of x_3 in the BNN model is only comparable to a constant, and thus it is irrelevant to the target variable.

Example 2 also contains 50 pairs of training and test sets, which are all mutually independent. Each training set consists of 100 i.i.d. samples, and each test set consists of 1000 i.i.d. samples. They were generated as follows: (1) Draw a point $(x_1, x_2) \sim \text{unif}[-1.5, 1.5]^2$. (2) Compute $d = x_2 - \sin(\pi x_1)$. (3) If $d > 0$, that is, the point (x_1, x_2) is above the sine curve, with probability $\exp(-5|d|)$, the point is assigned to class 1 and with the remaining probability, it is assigned to class 2. Otherwise, with probability $\exp(-5|d|)$, the point is assigned to class 2, and with the remaining probability it is assigned to class 1. The data generation process implies that the optimal classification

Table 3: Comparison of the New BNN Classifier, the Old BNN Classifier, and SVM for the Second Simulated Example.

	Parameter Setting	CPU(s)	Training	Test
New BNN	$\lambda = 1.0e-5$	45.0	10.496(0.019)	124.342(0.075)
	$\lambda = 1.0e-4$	46.1	10.392(0.021)	124.846(0.096)
	$\lambda = 1.0e-3$	47.3	10.216(0.022)	124.902(0.088)
	$\lambda = 1.0e-2$	49.1	9.952(0.022)	125.068(0.152)
Old BNN	$M = 2$	40.1	9.822(0.068)	127.462(0.651)
	$M = 3$	47.4	9.190(0.063)	127.198(0.454)
	$M = 4$	54.9	9.066(0.054)	126.998(0.459)
	$M = 5$	62.5	9.022(0.078)	128.018(0.486)
SVM (RBF)	$C = 0.5$	—	9.72	153.72
	$C = 1.0$	—	8.78	147.02
	$C = 1.5$	—	8.32	144.82
	$C = 2.0$	—	7.62	144.00
	$C = 3.0$	—	7.16	144.96

Notes: The "CPU(s)" column shows the total CPU time (in seconds) cost by 10 independent runs for each data set. The "Training" and "Test" columns show the averaged training and test errors and their respective standard deviations (in parentheses).

boundary is the sine curve $y = \sin(\pi z)$, $z \in [-1.5, 1.5]$. Figure 5d shows a training set generated by the process together with the sine curve.

The new BNN classifier was applied to this example with $M = 3$, $\tau = 15$ and a variety of λ's ranging from 1.0e-5 to 1.0e-2. Here we set a relatively large value for τ, since we expect the learned classification boundary to be very curved. In training, a temperature ladder of 10 levels was used with $t_1 = 5$ and the Metropolis step size $\eta = 0.25$. As for example 1, each run consists of 6000 iterations. The first 1000 iterations were discarded for the burn-in process, and the following 5000 iterations were used for inference. The overall acceptance rates of the local updating moves and the exchange rates were about 0.25 and 0.6, respectively. The averaged training and test errors are reported in Table 3.

For comparison, we also applied the old BNN classifier and SVM to this example. In SVM, a RBF kernel was used with σ being the median of the Euclidean distances from each sample of class 1 to the nearest sample of class 2 as suggested by Brown et al. (2000). A variety of C values were tried ranging from 0.5 to 3. Table 3 shows that the new BNN classifier outperforms significantly the old BNN classifier and SVM in general for this example. Figure 5 compares the decision regions of the three classifiers for one training set of this example. They were generated by the three classifiers in one run with $\lambda = 1.0e-5$, $M = 4$, and $C = 2.0$, respectively. It shows that SVM cannot approximate well the true classification boundary, the old classifier overfits to the data, and the new classifier can approximate the

true classification boundary rather well, even though the true boundary is very curved.

5.2 Breast Cancer Classification. The Wisconsin Breast Cancer database (WBCD), collected by Dr. W. H. Wolberg at University of Wisconsin Hospitals, has 683 samples, which consist of visually assessed nuclear features of fine needle aspirates (FNAs) taken from patients' breasts. Each sample was assigned a 9-dimensional vector of diagnostic characteristics: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses. Each component is in the interval 1 to 10, with 1 corresponding to the normal state and 10 to the most abnormal state. The samples were classified into two classes, benign and malignant. The classification was confirmed by either biopsy or further examinations. (For a detailed description for the data set, see Mangasarian & Wolberg, 1990 or ftp://ftp.cs.wisc.edu/mathprog/cpo-dataset/ machine-learn/cancer1.) The data set has been used as a benchmark example for machine learning. Here we apply the new BNN classifier to this data set and compare its performance with that of the old BNN classifier and SVM. Fifty pairs of training and test sets were generated from the WBCD data set. Each training set consists of 342 samples uniformly drawn from the data set, and the corresponding test set consists of the remaining samples.

For the new BNN classifiers, we set $M = 3$, $\tau = 0.5$, $\lambda = 2.5, 5, 10, 20, 40$, $\eta = 5$, and employed a temperature ladder of 10 levels with $t_1 = 2.5$. The classifier was run 10 times independently for each pair of the training and test sets. Each run consists of 6000 iterations, where the first 1000 iterations were discarded for the burn-in process, and the following iterations were used for inference. For the old BNN classifier, we tried four network structures with $2, 3, 4$, and 5 hidden units, respectively. It was also run for 10 times independently for each pair of the training and test sets. For SVM, we tried both the linear and RBF kernels, and a wide range of C. Again, the parameter σ of the RBF kernel was set as the median of the Euclidean distances from each sample of class 1 to the nearest sample of class 2. The computational results are summarized in Table 4, which shows that the new BNN classifier outperforms significantly the other three classifiers in general for this example. Comparing to the new BNN classifier, the old BNN classifier has a tendency to overfit to the data.

Figure 7a shows the posterior probability of each of the connections of a new BNN classifier with $M = 5$, $\tau = 5$, and $\lambda = 0.1$ and a training set consists of the 683 samples. The averaged network size is $m_{\text{eff}} = 5.1$, and the classification accuracy is 97.7%. Figure 7a shows that the sampled network structures are dominated by the shortcut connections. This is consistent with the work of Bennett and Mangasarian (1992) that the data can be classified by a hyperplane with an accuracy rate of 97.7%. Figure 7b shows the averaged frequencies of each of the nine input variables. The constant term is also

Table 4: Comparison of the Four Classifiers: New BNN Classifier, Old BNN Classifier, SVMs with Linear Kernels, and SVMs with RBF Kernels, Breast Cancer Example.

	Parameter Setting	CPU(s)	Training	Test
New BNN	$\lambda = 2.5$	196.1	7.994(0.023)	9.134(0.032)
	$\lambda = 5.0$	201.5	7.846(0.026)	8.914(0.032)
	$\lambda = 10$	203.0	7.746(0.025)	8.874(0.033)
	$\lambda = 20$	203.2	7.776(0.027)	8.966(0.032)
	$\lambda = 40$	203.5	7.728(0.026)	9.054(0.030)
Old BNN	$M = 2$	164.5	7.670(0.064)	10.980(0.059)
	$M = 3$	195.4	7.256(0.087)	11.012(0.063)
	$M = 4$	229.2	6.994(0.101)	10.944(0.058)
	$M = 5$	262.8	6.972(0.096)	10.900(0.056)
SVM (linear kernel)	$C = 0.01$	—	9.96	10.26
	$C = 1/64$	—	9.92	10.10
	$C = 1/16$	—	9.76	10.30
	$C = 1/4$	—	8.96	10.68
	$C = 1.0$	—	8.50	11.04
SVM (RBF kernel)	$C = 0.1$	—	10.46	10.62
	$C = 0.5$	—	9.40	10.30
	$C = 1.0$	—	8.84	10.28
	$C = 1.5$	—	8.52	10.42
	$C = 2.5$	—	7.94	10.84

Notes: The "CPU(s)" column shows the total CPU time (in seconds) cost by 10 independent runs for each data set. The "Training" and "Test" columns show the averaged training and test errors and their respective standard deviations (in parentheses).

included in the plot for a reference. It indicates that the four most relevant diagnostic characteristics are clump thickness, uniformity of cell size, bare nuclei, and normal nuckeoli. With different settings for τ and λ, the results are similar.

5.3 Protein Structure Class Prediction. Finally, we consider one example with multiple classes. The majority of proteins of known structures can be classified into one of the following four structural classes: α, β, α/β, and $\alpha + \beta$, based on the percentages of their secondary structure components. The prediction of protein structure class has been one of important components of the overall protein structure prediction problem (Levitt & Chothia, 1976). The early studies on the subject reveal that the structure class of a protein might basically depend on its amino acid composition. The underlying physical mechanism of the correlation was explored by Bahar, Atilgan, Jemigan, and Erman (1997) and Chou (1999). A comprehensive review of the studies can be found in Chou (2001).

In this article, we restudied the problem with BNN classifiers. The data set we used was downloaded from http://www.nersc.gov/~cding/protein.

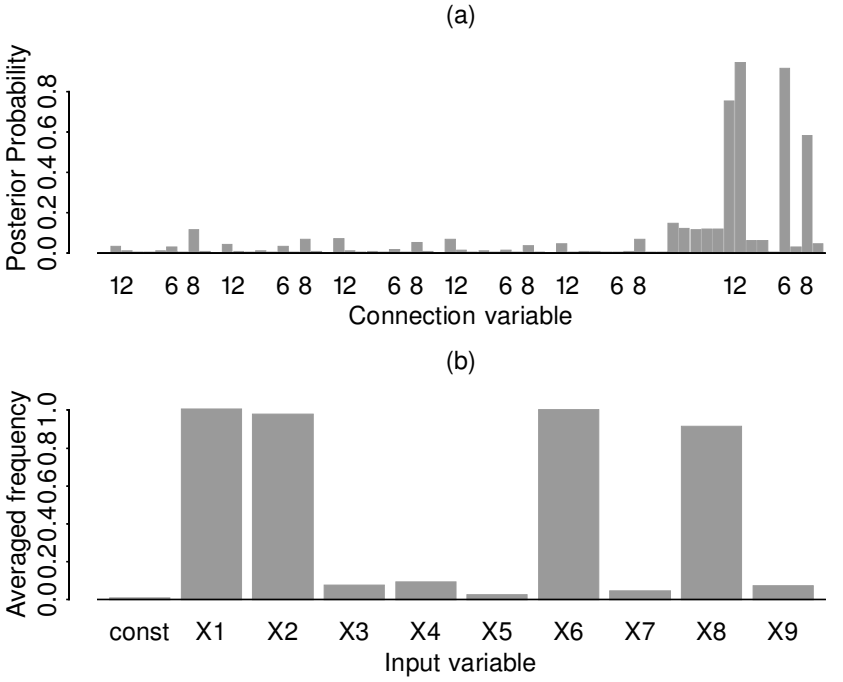

Figure 7: (a) The posterior probability of each of the connections of a full BNN model with $M = 5$, $\tau = 5$, and $\lambda = 0.1$. A training set consists of all the 683 samples. (b) The averaged frequencies of the connections out of each of the input variables. The nine input variables from x_1 to x_9 are clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses.

The training set consists of 313 proteins where two proteins have no more than 35% of the sequence identity for the aligned subsequences longer than 80 residuals. The test set consists of 385 proteins where all protein sequences have less than 40% identity with each other and less than 35% identity with the proteins of the training set. (For more details of the data set, see Ding & Dubchak, 2001.) In this example, the BNN classifiers have four output units and 21 input units, where the first 20 input units correspond to the compositions of the 20 amino acids and the last one corresponds to the length of the protein sequence. For the new BNN classifier, we set $M = 10$, $\tau = 2$, $\lambda = 30, 35, 40, 45$, $\eta = 3.0$, and employed a temperature ladder of 20 levels with $t_1 = 5$. For each λ, the classifier was run for 10 times independently. Each run consists of 200,000 iterations, and the first 2000 iterations were discarded for the burn-in process. The overall acceptance rates of the local updating and exchange operations were around

Table 5: Comparison of New BNN Classifier, Old BNN Classifier, and SVM, Protein Structure Class Prediction Example.

	Parameter Setting	CPU(m)	Training	Test	Frequency	Best
New BNN	$\lambda = 30$	164	19.3(0.63)	80.2(0.65)	9	76
	$\lambda = 35$	165	19.2(0.59)	80.8(0.65)	8	77
	$\lambda = 40$	165	18.9(0.53)	79.2(0.71)	9	75
	$\lambda = 45$	166	16.5(0.81)	80.6(0.67)	8	77
Old BNN	$M = 8$	137	18.8(4.34)	86.9(1.11)	1	81
	$M = 10$	162	3.9(1.92)	83.5(0.73)	3	80
	$M = 12$	181	3.6(2.10)	83.4(0.54)	3	81
	$M = 15$	209	9.5(3.60)	83.2(0.87)	5	79
SVM (RBF)	$C = 1.0$	—	31	83	—	—
	$C = 1.5$	—	20	85	—	—
	$C = 2.0$	—	15	86	—	—
	$C = 2.5$	—	7	88	—	—

Notes: The "CPU" column shows the CPU time cost by each run. The "Training" and "Test" columns show the averaged training and test errors over the 10 runs and their respective standard deviations (the figures in the parentheses). The "Frequency" column shows the number of runs (out of 10) with test error less than 83, the minimum test error of SVM. The "Best" column shows the minimum test error of the 10 runs.

0.3 and 0.25, respectively. For the old BNN classifier, we tried four different network structures with $M = 8, 10, 12$, and 15. It was also run 10 times independently for each structure. Each run consists of 5000 iterations, and the first 1000 iterations were discarded for the burn-in process. The computational results are summarized in Table 5. LIBSVM implemented the multiclass classification with the "one-against-one" approach (Knerr, Personnaz, & Dreyfus, 1990), in which $k(k-1)/2$ classifiers are constructed and each one trains data from two different classes. The classification made use of a voting strategy: each binary classification is considered to be a vote, and in the end, the point is designated to be in a class with the maximum number of votes. This approach has been shown to have a better performance than the "one-against-all" approach (Weston & Watkins, 1998; Platt, Cristianini, & Shawe-Taylor, 2000). (For more details on the implementation, see http://www.csie.ntu.edu.tw/∼cjlin.) For this example, we tried the RBF kernel and a variety of C ranging from 0.1 to 100. Table 5 shows the best four test errors and the associated C values and training errors. The other types of kernels were also tried, but their performances are not as good as that of RBF. This example shows that both BNN classifiers outperform SVMs in generalization performance, and the new BNN classifier outperforms the old one in whatever the minimum test error or the frequency of beating SVMs.

6 Conclusion

In this article, we introduced a new BNN classifier that is different from the old one in several respects, including the likelihood function, prior specification, and network structure. Under mild conditions, we show that the new BNN classifier will converge to the true classification function. Numerical results show that the new BNN classifier provides a consistent improvement over the old BNN classifier and SVMs in general performance for all examples of this article. The success of the new BNN classifier comes from two factors. First is an appropriate likelihood function, or equivalently, an appropriate penalty function for the misclassified patterns. In this article, we chose the likelihood function from the family $\{\exp(-\tau \sum_{i=1}^{N} \sum_{j=1}^{q} [g_j(\boldsymbol{x}_i, \boldsymbol{\theta}_\Lambda) - y_{ij}]^2) : \tau > 0\}$, with τ being determined by a cross-validation procedure in practice. Our results suggest that we may choose the likelihood function from a more broad likelihood family for a BNN model with further improved performance. Second is the use of global information of the data in the decision boundary determination, in contrast to the local determination of the decision boundary of SVMs by the support vectors. In this article, each pattern of the training set is equally treated in the training process to make sure that global information of the data is extracted by the BNN model. Further research on how to extract the global information from the data efficiently will be of interest. As a by-product, we also showed numerically that the new BNN classifier can identify the relevant linear or nonlinear variables from the pool of explanatory variables.

The program is available from the author by request.

References

Bahar, I., Atilgan, A. R., Jemigan, R. L., & Erman, B. (1997). Understanding the recognition of protein structural classes by amino acid composition. *Proteins: Struc., Func., and Genetics, 29*, 172–185.

Bennett, K. P., & Mangasarian, O. L. (1992). Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software, 1*, 23–34.

Bertsekas, D. P. (1995). *Nonlinear programming*. Belmont, MA: Athenas Scientific.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Fifth Annual Conference on Computational Learning Theory* (pp. 142–152). San Mateo, CA: Morgan Kaufmann.

Breiman, L. (2002). Statistical modeling—the two cultures. *Statistical Science, 16*, 199–215.

Brown, M. P. S., Grundy, W. N., Lin, D., Cristianini, N., Sugnet, C. W., Furey, T. S., M.A. Jr., & Haussler, D. (2000). Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proc. Natl. Acad. Sci, USA, 97*, 262–267.

Chang, C. C., & Lin, C. J. (2001). Training ν-support vector classifiers: Theory and algorithms. *Neural Computation, 13*, 2119–2147.

Chou, K. C. (1999). A key driving force in determination of protein structure classes. *Biochem. Biophys. Res. Commun., 264*, 216–224.

Chou, K. C. (2001). Review: Prediction of protein structural classes and subcellular location. *Current Protein and Peptide Science, 1*, 171–208.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning, 20*, 273–297.

Ding, C. H. Q., & Dubchak, I. (2001). Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics, 17*, 349–358.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

Fletcher, R. (1987). *Practical methods of optimization* (2nd ed.). New York: Wiley.

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

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

Geyer, C. J., & Thompson, E. A. (1995). Annealing Markov chain Monte Carlo with applications to ancestral inference. *J. Amer. Statist. Assoc., 90*, 909–920.

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

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

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

Husmeier, D., Penny, W. D., & Roberts, S. J. (1999). An empirical evaluation of Bayesian sampling with hybrid Monte Carlo for training neural network classifiers. *Neural Networks, 12*, 677–705.

Kass, R. E., & Vaidyanathan, S. (1992). Approximate Bayesian factors and orthogonal parameters, with applications to testing equality of two binomial proportions. *J. Royal Statist. Soc., B*, 129–144.

Knerr, S., Personnaz, L., & Dreyfus, G. (1990). Single-layer learning revisited: A stepwise procedure for building and training a neural network. In J. Fogelman (Ed.), *Neurocomputing: Algorithms, architectures and applications*. Berlin: Springer-Verlag.

Krogh, A., & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In J. D. Cowan, D. Touretzky, & J. Alspector (Eds.), *Advances in neural information processing systems, 7*. Cambridge, MA: MIT Press.

Levitt, M., & Chothia, C. (1976). Structural patterns in globular proteins. *Nature, 261*, 552–557.

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

MacKay, D. J. C. (1992). A practical Bayesian framework for backprop networks. *Neural Computation, 4*, 448–472.

Mangasarian, O. L., & Wolberg, W. H. (1990). Cancer diagnosis via linear programming. *SIAM News, 23*, 1–18.

Markowetz, F., Edler, L., & Vingron, M. (2002). *Support vector machines for protein fold class prediction* (Tech. Rep.). Berlin: Max-Planck-Institute for Molecular Genetics, Computational Molecular Biology.

Mercer, J. (1909). Functions of positive and negative type, and their connection with the theory of integral equations. *Transactions of the London Philospohical Society A, 209*, 819–835.

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

Müller, P. (1993). *Alternatives to the Gibbsa sampling scheme* (Tech. Rep.). Durham, NC: Institute of Statistics and Decision Sciences, Duke University.

Neal, R. M. (1996). *Bayesian learning for neural networks*. New York: Springer-Verlag.

Perrone, M. P., & Cooper, L. N. (1993). When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone (Ed.), *Neural networks for speech and image processing*. London: Chapman-Hall.

Platt, J. C., Cristianini, N., & Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification. In S. A. Solla, T. K. Leen, & K.-R. Müller (Eds.), *Advances in neural information processing systems, 12* (pp. 547–553). Cambridge, MA: MIT Press.

Raftery, A.E., Madigan, D., & Hoeting, J.A. (1997). Bayesain model averaging for linear regression models. *J. Amer. Statist. Assoc., 92*, 179–191.

Ripley, B. D. (1994). Flexible nonlinear approaches to classification. In J. H. Friedman & H. Wechsler (Eds.), *From statistics to neural networks: Theory and pattern recognition applications*. Berlin: Springer-Verlag.

Robert, C. P., & Casella, G. (1999). *Monte Carlo statistical methods*. New York: Springer-Verlag.

Rosen, B. E. (1996). Ensemble learning using decorrelated neural networks. *Connection Science, 3*, 373–384.

Rumelhart, D., Hinton, G., & Williams, J. (1986). Learning internal representations by error propagation. In D. Rumelhart & J. McClelland (Eds.), *Parallel distributed processing* (pp. 318–362). Cambridge, MA: MIT Press.

Smith, A. F. M., & 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.

Sollich, P. (2002). Bayesian methods for support vector machines: Evidence and predictive class probabilities. *Machine Learning, 46*, 21–52.

Thodberg, H. H. (1996). A review of Bayesian neural networks with an application to near infrared spectroscopy. *IEEE Transactions on Neural Networks, 7*, 56–72.

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

Weston, J., & Watkins, C. (1998). *Multi-class support vector machines* (Tech. Rep. CSD-TR-98-04). Egham, Surrey: Royal Holloway, University of London.

White, H. (1992). *Artificial neural networks: Approximation and learning theory*. Cambridge, MA: Blackwell.

Received July 22, 2002; accepted January 28, 2003.