An Incremental Bayesian Approach for Training Multilayer Perceptrons

Dimitris Tzikas[1,2] and Aristidis Likas[1,2]

[1] Biomedical Research Institute — FORTH
University Campus of Ioannina GR 45110, Ioannina, Greece
[2] Department of Computer Science
University of Ioannina, GR 45110, Ioannina, Greece

Abstract. The multilayer perceptron (MLP) is a well established neural network model for supervised learning problems. Furthermore, it is well known that its performance for a given problem depends crucially on appropriately selecting the MLP architecture, which is typically achieved using cross-validation. In this work, we propose an incremental Bayesian methodology to address the important problem of automatic determination of the number of hidden units in MLPs with one hidden layer. The proposed methodology treats the one-hidden layer MLP as a linear model consisting of a weighted combination of basis functions (hidden units). Then an incremental method for sparse Bayesian learning of linear models is employed that effectively adjusts not only the combination weights, but also the parameters of the hidden units. Experimental results for several well-known classification data sets demonstrate that the proposed methodology successfully identifies optimal MLP architectures in terms of generalization error.

1 Introduction

The *multilayer perceptron* (MLP) is a very popular neural network model for supervised learning problems. Assuming a *training set* $\{\boldsymbol{x}_n, t_n\}_{n=1}^N$, we can model the data generation process using a function y so that $t_n = y(\boldsymbol{x}_n) + \epsilon_n$, where ϵ_n is an error term. The MLP is a parametric form that is commonly used for the underlying function y. We are interested in the two-layer MLP which has one hidden layer with M hidden units and an output layer with a single unit. The output of such an MLP can be considered as a weighted linear model with respect to the hidden units:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}) + b = \sum_{j=1}^{M} w_j h_j(\boldsymbol{x}) + b, \tag{1}$$

where b is the output bias, $\boldsymbol{w} = (w_1, \ldots, w_M)^T$ are the weights, $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \ldots, h_M(\boldsymbol{x}))^T$ and the functions $h_j(\boldsymbol{x})$ are the outputs of the hidden units. Each hidden unit computes a weighted sum of the input $\boldsymbol{x} = (x_1, \ldots, x_d)^T$, which is then passed through an activation function. Here, the hyperbolic tangent function $tanh(z) =$

K. Diamantaras, W. Duch, L.S. Iliadis (Eds.): ICANN 2010, Part I, LNCS 6352, pp. 87–96, 2010.

$\frac{e^{2z}-1}{e^{2z}+1}$ is used, whose output ranges from -1 to $+1$. It must be noted that the sigmoid logistic function could also have been used. The weights of the j-hidden unit are denoted with $\boldsymbol{u}_j = (u_{j1}, \ldots, u_{jd})^T$ and we also assume a bias a_j:

$$h_j(\boldsymbol{x}) = \tanh\left(\boldsymbol{u}_j^T \boldsymbol{x} + a_j\right) = \tanh\left(\sum_{i=1}^{d} u_{ji} x_i + a_j\right). \tag{2}$$

Once the number M of hidden units is given, training of an MLP, i.e. estimating the parameters (weights $\boldsymbol{w}$, $\boldsymbol{u}$ and the biases a, b) of the network is relatively simple, because the derivatives of the MLP can be easily computed and general–purpose optimization algorithms, (e.g. quasi-Newton methods such as BFGS) can be effectively employed.

Multilayer perceptrons (with one hidden layer) have the property that they can approximate any function with arbitrary accuracy if a sufficient number of hidden units is used. Although this result is important, it must be noted that training MLPs with large numbers of hidden units usually leads to poor generalization performance. Therefore, in practice, best results are obtained when using the smallest number of hidden units that are sufficient to model the unknown function. Although some sampling-based Bayesian methods based on Markov Chain Monte Carlo have been proposed for tackling the MLP model selection problem [6], such methods have not achieved widespread use due to high computational complexity and the difficulty in deciding when to terminate the sampling procedure. For this reason the cross-validation approach is considered as the typical method used to estimate the number of MLP hidden units by considering several MLP architectures with different number of hidden units and selecting the network that exhibits the best cross-validation performance.

In this paper, we propose a training methodology for the MLP with one hidden layer, that automatically estimates the appropriate number of hidden units and also learns the network parameters. The methodology is based on the sparse Bayesian linear model [1] and the underlying automatic relevance determination (ARD) principle that automatically determines the number of basis functions in linear models. We follow an incremental approach that starts with only one hidden unit and iteratively adds hidden units to the model. For each added hidden unit, optimal values for the weights and bias are estimated. In order to stop adding units when a sufficient number has been added to the model, we assume a sparse distribution for the weights of the output unit. This sparse prior distribution enforces the removal of hidden units that do not sufficiently contribute to the model, by setting the corresponding hidden to output connection weights equal to zero.

In the next section we describe the main idea of our method where the MLP is treated as a weighted linear combination of basis functions (hidden units), thus the incremental sparse Bayesian learning framework can be applied to estimate both the combination weights and the parameters of the hidden units. The sparsity enforcing prior imposed on the combination weights enforces many of them to become zero, thus the corresponding hidden units are removed from the network. Section 3 summarizes sparse Bayesian learning for linear models,

while Section 4 presents the proposed method for incremental Bayesian MLP training. Experimental results are presented in Section 5, while the last section provides conclusions and some directions for future work.

2 Automatic Model Selection for the Multilayer Perceptron

The multilayer perceptron of eq. (1) can be considered as a linear model, where the hidden units $h_j(\boldsymbol{x})$ play the role of basis functions. Linear models are very popular models, possibly because the weights $\boldsymbol{w}$ of the linear model can be computed very efficiently. Recent advances in sparse Bayesian modeling [1] allow for the automated estimation of the number of basis functions as follows: initially a model with many basis functions is assumed and then by imposing a sparse prior on the weights and ii) performing Bayesian inference of the weight values, we achieve pruning of basis functions that are not supported by the training data, i.e. the corresponding weights are found to be zero. This is very important, because by pruning irrelevant basis functions, an appropriate model is automatically selected. Following this aproach, we can use very flexible models with many basis functions, even if we have a small training set; irrelevant basis functions will be pruned and overfitting will be avoided.

A major shortcoming of the typical sparse Bayesian linear model is that basis functions are fixed and they have to be selected a priori. On the other hand, the hidden units of the MLP, which we treat as basis functions, contain parameters u_{ji}, a_j that are essential to be estimated. In this work, we employ an incremental Bayesian methodology [2], that simultaneously estimates the weights of the sparse linear model and parameters of its basis functions. This methodology, allows to treat a two-layer MLP as a sparse Bayesian linear model with adjustable basis function parameters. The weights of the connections from the hidden to the output layer correspond to the weights of the linear model and can be efficiently computed. Furthermore, imposing a sparse prior on these weights we could estimate the appropriate number of hidden units. The weights of the connections from input to the hidden layer of the MLP, correspond to parameters of the basis functions and could estimated simultaneously to the linear model weights using the methodology proposed in [2].

3 Sparse Bayesian Learning

3.1 Sparse Bayesian Linear Regression

In this section we briefly describe learning of sparse Bayesian linear models [1], which have the form of (1). We assume that the observations of the training set $\{\boldsymbol{x}_n, t_n\}_{n=1}^N$ have been corrupted with additive Gaussian noise with precision (i.e. inverse variance) β_n:

$$p(t_n|\boldsymbol{B}) = N(t_n|y(\boldsymbol{x}_n), \beta_n^{-1}). \tag{3}$$

where $\boldsymbol{B}$ is a diagonal matrix with elements $\beta_1, \ldots, \beta_N$. Assuming that the basis functions $h_j(\boldsymbol{x})$ are fixed, and defining the fixed 'design' matrix $\boldsymbol{\Phi} = (\boldsymbol{h}(\boldsymbol{x}_1), \ldots, \boldsymbol{h}(\boldsymbol{x}_N))^T$, with $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \ldots, h_M(\boldsymbol{x}))^T$, the likelihood of the observations can be written as:

$$p(\boldsymbol{t}|\boldsymbol{w}, \boldsymbol{B}) = \mathrm{N}(\boldsymbol{t}|\boldsymbol{\Phi}\boldsymbol{w}, \boldsymbol{B}). \tag{4}$$

In order to achieve sparse solutions, i.e. prune irrelevant basis functions, a Gaussian prior distribution with separate variance α_i^{-1} is assumed for each weight w_i:

$$p(\boldsymbol{w}|\boldsymbol{\alpha}) = \prod_{i=1}^{M} \mathrm{N}(w_i|0, \alpha_i^{-1}), \tag{5}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$. Moreover, we assume that each α_i is drawn from a Gamma distribution whose parameters are set to near zero values so as to be uninformative.

The posterior distribution of the weights given the observations can be computed using Bayes's law:

$$p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{B}) = \frac{p(\boldsymbol{t}|\boldsymbol{w}, \boldsymbol{B})p(\boldsymbol{w}|\boldsymbol{\alpha})}{p(\boldsymbol{t}|\boldsymbol{\alpha}, \boldsymbol{B})}, \tag{6}$$

where $p(\boldsymbol{w}|\boldsymbol{a})$ is given by (5). It can be shown that the weight posterior distribution follows a Gaussian distribution [1]:

$$p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{B}) = N(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{7}$$

with

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{B}\boldsymbol{t}, \tag{8}$$

$$\boldsymbol{\Sigma} = (\boldsymbol{\Phi}^T\boldsymbol{B}\boldsymbol{\Phi} + \boldsymbol{A})^{-1}, \tag{9}$$

and $\boldsymbol{A} = \mathrm{diag}(\boldsymbol{\alpha})$.

When the basis functions contain adjustable parameters the linear model is very flexible and a stronger prior on $\boldsymbol{\alpha}$ is needed. Such a prior has been proposed in [3] and penalizes models with large number of 'effective' parameters. The prior depends on the trace of a so called 'smoothing matrix' $\boldsymbol{S} = \boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{B}$ as follows:

$$p(\boldsymbol{\alpha}) \propto \exp(-c\,\mathrm{trace}(\boldsymbol{S})), \tag{10}$$

where the sparsity parameter c provides a mechanism to control the amount of desired sparsity. When using specific values of the sparsity parameter c, some known model selection criteria are obtained [4]:

$$c = \begin{cases} 0 & \text{None,} \\ 1 & \text{AIC (Akaike information criterion),} \\ \log(N)/2 & \text{BIC (Bayesian information criterion),} \\ \log(N) & \text{RIC (Risk inflation criterion).} \end{cases} \tag{11}$$

When using this prior, the following update formulas for the weight precisions $\boldsymbol{\alpha}$ can be obtained [2]:

$$\alpha_i = \frac{\gamma_i}{\mu_i^2 - 2c\gamma_i \Sigma_{ii}}, \tag{12}$$

where μ and Σ are given from equations (8) and (9) respectively and $\gamma_i = 1 - \alpha_i \Sigma_{ii}$.

The learning algorithm iteratively applies the updates of $\boldsymbol{\alpha}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ until convergence. During those iterations, due to the sparse prior on the weights, some parameters α_i take very large values, thus the corresponding weights w_i are set to zero and the corresponding basis functions $h_i(\boldsymbol{x})$ are removed from the model. In this way automatic model selection is achieved.

3.2 Sparse Bayesian Classification

In this work for simplicity we only consider binary classification problems and assume that the outputs are coded so that $t_n \in \{0, 1\}$ [1]. Then, the likelihood of the training set is given by:

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}, \tag{13}$$

where $y_n = \sigma(y(\boldsymbol{x}_n|\boldsymbol{w}))$ with $\sigma(z)$ being the logistic sigmoid function. Using the Laplacian approximation, the classification problem can be mapped to a regression problem [1] with heteroscedastic noise $p(\epsilon_n) = \mathrm{N}(\epsilon_n|0, \beta_n)$. The noise precision is given by:

$$\beta_n = y_n(1 - y_n), \tag{14}$$

and the regression targets $\hat{\boldsymbol{t}} = (\hat{t}_1, \ldots, \hat{t}_N)^T$ are:

$$\hat{\boldsymbol{t}} = \boldsymbol{\Phi}\boldsymbol{w} + \boldsymbol{B}^{-1}(\boldsymbol{t} - \boldsymbol{y}), \tag{15}$$

where $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ and $\boldsymbol{B} = \mathrm{diag}(\beta_1, \ldots, \beta_N)$.

3.3 Incremental Sparse Bayesian Learning

Notice that the computational cost of the sparse Bayesian learning algorithm is high for large datasets, because the computation of $\boldsymbol{\Sigma}$ in (9) requires $O(N^3)$ operations. A more computationally efficient incremental algorithm has been proposed in [5]. It initially assumes that $\alpha_i = \infty$, for all $i = 1, \ldots, M$, which corresponds to assuming that all basis functions have been pruned because of the sparsity constraint. Then, at each iteration one basis function may be either added to the model or re-estimated or removed from the current model. When adding a basis function to the model, the corresponding parameter α_i is set to the value that maximizes the likelihood.

[1] Multiclass problems can be solved using the one-vs-all approach, which builds only two class models.

More specifically, the method is based on the remark that the terms of the likelihood that depend on a single parameter α_i are [5]:

$$l_i = \frac{1}{2}\left(\log \alpha_i - \log(\alpha_i + s_i) + \frac{q_i^2}{\alpha_i + s_i}\right), \tag{16}$$

where

$$s_i = \boldsymbol{h}_i^T \boldsymbol{C}_{-i}^{-1} \boldsymbol{h}_i, \qquad q_i = \boldsymbol{h}_i^T \boldsymbol{C}_{-i}^{-1} \hat{\boldsymbol{t}}, \tag{17}$$

$\boldsymbol{h}_i = (h_i(\boldsymbol{x}_1), \ldots, h_i(\boldsymbol{x}_N))^T$ and $\boldsymbol{C}_{-i} = \boldsymbol{B} + \sum_{j \neq i} \alpha_j \boldsymbol{h}_j \boldsymbol{h}_j^T$. In regression we have $\hat{\boldsymbol{t}} = \boldsymbol{t}$ and usually $\boldsymbol{B} = \beta \boldsymbol{I}$, while in classification $\boldsymbol{B}$ and $\hat{\boldsymbol{t}}$ are given by (14) and (15) respectively.

Based on the above likelihood decomposition, in [2] the following update equation for α_i has been derived when the the sparsity prior $p(\boldsymbol{\alpha})$ of equation (10) is assumed:

$$\begin{aligned} &\alpha_i = \frac{s_i^2}{q_i^2 - (2c+1)s_i} && \text{if } q_i^2 > (2c+1)s_i, \\ &\alpha_i = \infty && \text{if } q_i^2 \leq (2c+1)s_i. \end{aligned} \tag{18}$$

The incremental training algorithm proceeds iteratively, by selecting at each iteration a basis function h_i (from a fixed pool of basis functions) and adding this basis function to the model if $q_i^2 > (2c+1)s_i$ or removing it otherwise. An important question that arises in the incremental algorithm is which basis function to select at each iteration. There are several possibilities, for example we could choose a basis function at random or with some additional computational cost, we could test several and select the one whose addition will cause the largest increase to the marginal likelihood. However in the above description we have made the assumption that the basis functions contain no adjustable parameters which is not convenient for the MLP case where the hidden units contain parameters to be learnt from the data. Such an approach is described next.

4 Incremental Bayesian MLP Learning

The proposed algorithm for incremental Bayesian MLP training is based on an extension [2] of the incremental method described above. This extension also allows for learning the parameters of the basis functions. In the MLP case, basis functions correspond to hidden units and we will use this term in the description that follows. More specifically, at each iteration we select the most appropriate hidden unit to add to the model as measured by the increment in the likelihood l_i. Therefore, in order to select a hidden unit for addition to the model we perform an optimization of the marginal likelihood with respect to the parameters of the hidden unit. Since we assume continuous parameters for the hidden units, continuous optimization methods should be employed, which

exploit the derivatives of the likelihood l_i with respect to the parameters of the hidden unit.

The incremental algorithm performs three operations at each iteration; it first attempts to add a hidden unit to the model and adjusts its parameters, then updates all parameters of the current model and finally removes any hidden units that no longer contribute to the model. The algorithm is summarized in by following steps:

1) Select a Hidden Unit to Add to the Model. In order to add a hidden unit i, we maximize the likelihood l_i with respect to its parameters θ_{ik} (weights and bias). We can perform this maximization using a continuous numerical optimization method. The required derivatives are computed as [2]:

$$\frac{\partial l_i}{\partial \theta_{ik}} = -\left(\frac{1}{\alpha_i + s_i} + \frac{q_i^2 + c\alpha_i}{(\alpha_i + s_i)^2}\right) r_i + \frac{q_i}{\alpha_i + s_i}\omega_i, \tag{19}$$

where

$$r_i \equiv \frac{1}{2}\frac{\partial s_i}{\partial \theta_{ik}} = \boldsymbol{h}_i^T \boldsymbol{C}_{-i}^{-1} \frac{\partial \boldsymbol{h}_i}{\partial \theta_{ik}}, \qquad \omega_i \equiv \frac{\partial q_i}{\partial \theta_{ik}} = \boldsymbol{t}^T \boldsymbol{C}_{-i}^{-1} \frac{\partial \boldsymbol{h}_i}{\partial \theta_{ik}}. \tag{20}$$

Notice that since we use a local optimization method (in our case the quasi-Newton BFGS), we can only attain a local maximum of the marginal likelihood, which depends on the initialization. For this reason, in order to add a hidden unit we perform this maximization several times, each time with different initialization and then we keep the parameters that correspond to the best solution. Note that this optimization is not computationally demanding, since it involves only the parameters related to a hidden unit and not all the parameters of the current network model.

2) Optimize Current Model. Although we optimize the parameters of each hidden unit at the time that we add it to the model, it is possible that the optimal values for the already existing network parameters will change, because of the addition of the new hidden unit. For this reason, after the addition of a hidden unit, we further optimize the parameters α_i and $\boldsymbol{\theta}_i$ of all hidden units of the current model. Again, this optimization is performed using a continuous numerical optimization method (BFGS), and it is usually very efficient because the starting values are usually very close to the optimal.

4) Remove Hidden Units. After updating the hyperparameters $\boldsymbol{\alpha}$ of the current model, it is possible that some of the existing hidden units will no longer contribute to the model. This happens because of the sparsity prior, which allows only few of the basis functions (i.e. hidden units) to be used in the estimated model. For this reason, we remove from the model those hidden units that no longer contribute to the model, specifically those with $\alpha_i > 10^{12}$. Note that α_i is the inverse variance of the weight w_i which follows a zero mean Gaussian

distribution. Thus a large α_i value implies an almost zero variance, thus the weight w_i is equal to its mean which is zero and the corresponding unit is removed from the model. The removal of hidden units is important, not only because we avoid the additional computational cost of updating their parameters, but also because we avoid possible singularities of the covariance matrices due to numerical errors in the updates.

5) Repeat Until Convergence. We assume that the algorithm has converged when the increment of the likelihood is negligible ($\Delta L < 10^{-6}$) for ten successive iterations.

5 Numerical Experiments

In this section we provide numerical experiments that demonstrate the effectiveness of the proposed method. We have considered several commonly used benchmark data sets[2] that are summarized in Table 1. The purpose of these experiments is to compare the performance of the proposed methodology against the typical approach where the number of hidden units is selected through cross-validation, and to demonstrate the ability of the proposed method to automatically determine the appropriate model complexity.

Table 1. Dataset Description

Dataset	patterns (N)	features (d)
wdbc	569	30
bupa	345	6
sonar	208	60
pima-diabetes	768	8
ionosphere	351	34

For each dataset, we use the Levenberg-Marquardt optimization method to train multilayer perceptrons with one hidden layer and number of hidden units M in the range from 1 to 10. We then apply the proposed methodology that automatically estimates the appropriate number of hidden units, using $c = log(N/2)$ that corresponds to the BIC (Bayesian information criterion). In order to evaluate the methods, we use 10-fold cross validation. Moreover, in order to account for the dependence of the typical MLP training algorithm on the initial values of its parameters, we train each MLP ten times, starting from different initial values. Then, the solution with the minimum error (on the training set) is evaluated on the test set.

[2] These datasets can be obtained from the UCI Machine Learning Repository, at `http://archive.ics.uci.edu/ml/`

The average error values (using 10-fold cross validation) for all datasets are reported in Table 2 and Table 3. In Table 2 bold values indicate the best result for each data set using the typical training approach. It is clear that for every data set the proposed method provides an estimate of M (Table 3) that is nearly equal to the best performance result of Table 2. This indicates that our method successfully estimates the number M of hidden units that define the MLP architecture with the minimum cross-validation error. Furthermore, the error rates obtained using the proposed method are comparable and in some cases superior to the error rates of the best MLP in Table 2.

Table 2. Average classification error rates using MLP with M hidden units

M	wdbc	bupa	sonar	pima-diabetes	ionosphere
1	**3.34**	35.68	22.57	23.18	12.83
2	4.05	31.62	24.45	23.56	14.24
3	3.87	32.19	**19.69**	**23.04**	**11.69**
4	3.87	**30.73**	19.74	25.25	16.51
5	3.87	32.46	26.88	24.35	15.35
6	4.22	32.77	23.48	25.25	12.23
7	3.16	35.10	25.40	25.51	18.52
8	4.75	35.06	22.12	27.08	17.97
9	3.87	36.23	20.60	24.74	15.67
10	4.22	32.71	21.67	26.29	18.82

Table 3. Average classification error rates and estimated number of hidden units M using the proposed method

	wdbc	bupa	sonar	pima-diabetes	ionosphere
M	1.40	3.10	3.00	1.00	3.30
Error	2.81	29.32	19.24	23.18	11.97

6 Conclusions

We have proposed a methodology to automatically obtain an effective estimation of the number of hidden units of the multilayer perceptron. The methodology is based on treating the MLP as a linear model, whose basis functions are the hidden units. Then, we use a sparse Bayesian prior on the weights of the linear model that enforces irrelevant basis functions (equivalently unnecessary hidden units) to be pruned from the model. In order to train the proposed model, we use an incremental training algorithm which at each iteration attempts to add a hidden unit to the network and adjusts its parameters assuming a sparse Bayesian learning framework. Numerical experiments using well-known classification data

sets demonstrate the effectiveness of the proposed method in providing low complexity networks exhibiting low generalization error.

We consider that the proposed method provides a viable solution to the well-studied problem of estimating the number of hidden units in MLPs. Therefore, is our aim to perform more extensive experiments with this method considering not only classification, but also regression data sets. In addition we plan to compare the method against other classification or regression models like SVM, RVM and Gaussian Processes. Finally, it would be interesting to extend the method in order to be used for training MLPs with two hidden layers.

Acknowledgments

This work was part funded by the European Commission (Project ARTREAT: Multi-level patient-specific artery and atherogenesis model for outcome prediction, decision support, treatment, and virtual hand-on training, FP7-224297 - Large-scale Integrating Project).

References

1. Tipping, M.E.: Sparse Bayesian learning and the relevance vector machine. Journal of Machine Learning Research 1, 211–244 (2001)
2. Tzikas, D., Likas, A., Galatsanos, N.: Sparse bayesian modeling with adaptive kernel learning. IEEE Transactions on Neural Networks 20(6), 926–937 (2009)
3. Schmolck, A., Everson, R.: Smooth relevance vector machine: a smoothness prior extension of the RVM. Machine Learning 68(2), 107–135 (2007)
4. Holmes, C.C., Denison, D.G.T.: Bayesian wavelet analysis with a model complexity prior. In: Bernardo, J.M., Berger, J.O., Dawid, A.P., Smith, A.F.M. (eds.) Bayesian Statistics 6: Proceedings of the Sixth Valencia International Meeting. Oxford University Press, Oxford (1999)
5. Tipping, M.E., Faul, A.: Fast marginal likelihood maximisation for sparse Bayesian models. In: Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (2003)
6. Neal, R.M.: Bayesian Learning for Neural Networks. Lecture Notes in Statistics, vol. 118. Springer, Heidelberg (1996)