Priors for Bayesian Neural Networks

by

Mark Robinson

B.Sc, University of Guelph 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Statistics)

we accept this thesis as conforming
to the required standard

The University of British Columbia

June 2001

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Statistics

The University of British Columbia
Vancouver, Canada

Date 06.26.2001

DE-6 (2/88)

Abstract

In recent years, Neural Networks (NN) have become a popular data-analytic tool in Statistics, Computer Science and many other fields. NNs can be used as universal approximators, that is, a tool for regressing a dependent variable on a possibly complicated function of the explanatory variables. The NN parameters, unfortunately, are notoriously hard to interpret. Under the Bayesian view, we propose and discuss prior distributions for some of the network parameters which encourage parsimony and reduce overfit, by eliminating redundancy, promoting orthogonality, linearity or additivity. Thus we consider more senses of parsimony than are discussed in the existing literature. We investigate the predictive performance of networks fit under these various priors. The Deviance Information Criterion (DIC) is briefly explored as a model selection criterion.

Contents

List of Figures

Acknowledgements

I would like to thank my supervisor, Dr. Paul Gustafson, for his support and encouragement throughout the process of producing this thesis. Also, thanks are owed to Dr. Harry Joe for his helpful comments on the thesis and for providing programming tips along the way.

My time here at UBC would not have been the same had it not been for the support of an incredible bunch of graduate students who helped me put things in perspective and who endured my high intensity. I graciously thank them for their support.

Lastly, I would like to thank my parents, brothers and friends for their encouragement and especially to Lynn for her love and support.

MARK ROBINSON

The University of British Columbia
June 2001

To coffee and all things real ...

Chapter 1

Introduction

In recent years, there has been considerable interest in the use of *neural networks* by both the computer science and statistical communities. In the computing community, applications are usually associated with artificial intelligence (e.g. pattern recognition). For the statistician, the methodology is used primarily for flexible (non-linear) regression as well as classification. There is considerable overlap in the two disciplines, especially in classification problems. Many reviews have been written from a statistical point of view (Cheng and Titterington, 1994; Ripley, 1994; Stern, 1996).

As the name suggests, neural networks initially were derived in an attempt to model the *parallel* processing capacity of the human brain. Computers have a remarkable ability to perform routine calculation tasks in a fraction of the time a human can. However, a human can perform tasks such as identifying faces or recognizing characters with little or no effort; a computer performing the same task may require a considerable amount of time and a sophisticated algorithm. Artificial neural networks (ANNs) — the mathematical constructs discussed in this thesis — attempt to harness these parallel capabilities to analyze experimental data.

In general, neural networks take a number of inputs (explanatory variables or covariates in statistical terminology) and via a collection of neurons (i.e. units which weight and sum a number of incoming variables to one output), generate a number of outputs (responses in statistical terms). For example, in a regression problem, the inputs may be some predictors x, which are used to calculate an output, y. The x and y can be scalar or vector quantities. In a classification problem,

the outputs y_i $(i = 1, \cdots, k)$ may represent the probability of being in the i-th group.

From a computer science viewpoint, these networks are to be learned[1] from the data (i.e. the network parameters are to be estimated). For this thesis, it is desired to work under the Bayesian framework, in which one thinks of each parameter as having a distribution which represents belief about the value of that parameter. We will see shortly that this is a difficult task.

1.1 Artificial Neural Networks

In the literature, ANNs go by many names: back-propagation NNs, feed-forward NNs and multilayer perceptron (MLP) NNs, all of which describe the same general structure.

The term back-propagations refers to the way that the derivatives are calculated for the optimization methods. Feed-forward refers to the forward directed links from the inputs to the outputs. The term MLP refers to the possibly multiple layers in the network which are connected by *neurons*.

Every ANN consists of an input layer which is eventually connected to an output layer using 1 or more *hidden* layers. In getting from the inputs to the outputs, each link in the network contributes a weight and every node in the hidden layer forces that linear combination through an activation function. This process is repeated for each of the hidden layers. Commonly, biases are added at each stage before being passed through an activation function. A three layer (1 input, 1 hidden, 1 output) neural network is depicted in Figure 1. For a three layer network, the output layer is computed as a function of a linear combination (l.c.) of non-linear functions of a linear combination of the inputs. The neural network depicted in Figure 1 can be written mathematically as:

$$y_k = \phi_0 \left\{ b_k + \overbrace{\sum_{h=1}^{H} v_{hk} \phi_h \underbrace{\left(a_h + \sum_{i=1}^{p} u_{ih} x_i \right)}_{\text{l.c. of inputs}}}^{\text{l.c. of non-linear functions}} \right\} \quad \text{for } k = 1, \cdots, q \tag{1.1}$$

where p is the number of input units, H is the number of hidden units, q is the number of output

[1]The learning discussed here is often called *supervised* learning, in that the data are to be classified into known groups. *Unsupervised* learning involves classifying into groups which are not previously specified.

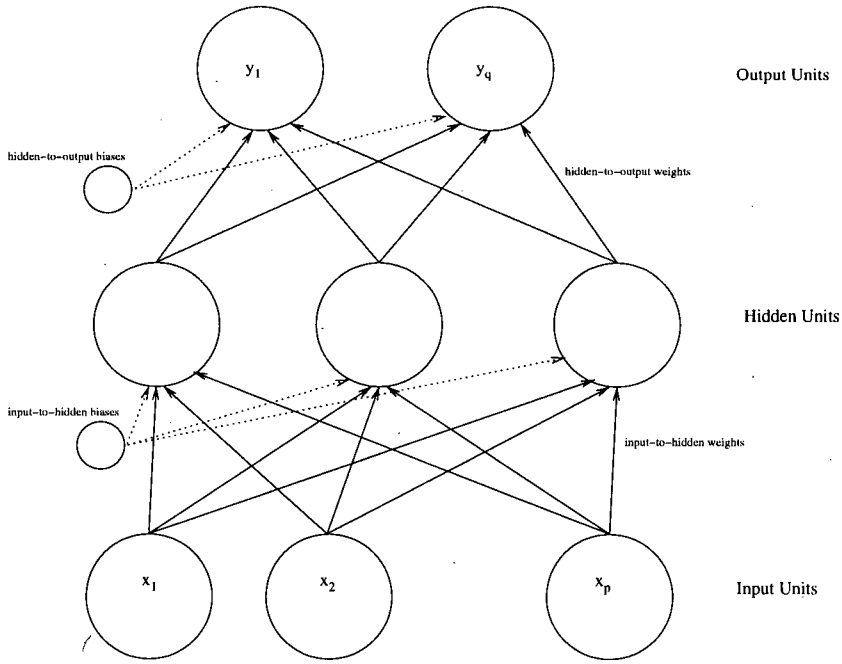


Figure 1.1: The Neural Network Architecture.

units and $\phi_0, \phi_1, \cdots, \phi_H$ are activation functions. Here, the v_{hk} represent the $H \times q$ *hidden-to-output weights* u_{ih} are the $p \times H$ *input-to-hidden weights*, b_k are the *output biases* and a_h are the *hidden biases*. There are also networks suggested which have *skip layer* connections, meaning there are direct links from the input to output layers which have corresponding weights. In this case, the

above model (1.1) becomes:

$$y_k = \phi_0 \left\{ b_k + \underbrace{\sum_{i=1}^{p} c_i x_i}_{\text{skip layer terms}} + \sum_{h=1}^{H} v_{hk} \phi_h \left(a_h + \sum_{i=1}^{p} u_{ih} x_i \right) \right\} \quad \text{for } k = 1, \cdots, q. \tag{1.2}$$

In most cases, the $\phi_1, \cdots, \phi_H$ are taken to be the same sigmoidal function (e.g. the logistic function, $(1+e^{-x})^{-1}$ or equivalently, the hyberbolic tangent function, $(e^x - e^{-x})(e^x + e^{-x})^{-1}$). In a regression problem, ϕ_0 is taken to be the identity function and in a binary classification problem, ϕ_0 is commonly taken to be the logistic function. For example, a regression network with one input, four hidden units and one output unit with a tanh activation function would relate the input x to the output y via the following model:

$$y = b + v_1 \tanh(a_1 + u_1 x) + v_2 \tanh(a_2 + u_2 x) + v_3 \tanh(a_3 + u_3 x) + v_4 \tanh(a_4 + u_4 x).$$

In general, a neural network is a tool for establishing an intricate non-linear relationship in either a regression problem or classification problem. A very detailed non-linear relationship can be described by models such as (1.1) or (1.2). In fact, it can be shown (although no attempt is made here) that a three layer ANN can approximate any function (on a compact domain) to a desired level (Hornik et al. 1989) by having an appropriate number of hidden nodes.

There are a few standard statistical methods which are special cases of the neural network. First of all, linear regression can be achieved with one hidden unit and identity functions for both ϕ_0 and ϕ_1. The linear logistic model can ascertained by using the identity function for ϕ_1 and the logistic function for ϕ_0. Other methods such as projection pursuit regression and radial basis functions are special cases as well (Chen and Titterington, 1994).

The competing statistical methods of ANNs are, for regression: multivariate adaptive regression splines (MARS; Friedman 1991), and the Bayesian version thereof; generalized additive models, non-parametric regression, smoothing techniques and, more recently, Bayesian regression modelling with interactions and smooth effects (B-WISE; Gustafson 2000). For classification, other statistical methods include (linear, quadratic, flexible) discriminant analysis and nearest-neighbour

methods. Classification and Regression Trees (Breiman 1984) also overlap many applications with neural networks.

Neural networks have been used quite extensively in a broad spectrum of disciplines. A small scan of applications includes: the papermaking industry (Edwards et al. 1999), the dairy industry (Macrossan et al. 1999) and meteorology and oceanography (Hsieh and Tang 1998).

1.2 The Bayesian Approach

In this thesis, I have chosen to take the Bayesian approach. The question arises: Why add an additional layer of complexity (computational or otherwise) to an already intricate problem? My reason is twofold. First, with the recent technology in Markov Chain Monte Carlo methods (see Chapter 3), Bayesian methods are becoming easier to implement. Second, the additional complexity also provides additional flexibility in terms of model capabilities and methods for inference. The rest of this section describes the Bayesian approach, as it applies to this thesis.

Frequentist (or classical or Fisherian) statisticians assume their data follows, at least approximately, some parametric model, from which they form the likelihood function (i.e. the joint distribution of the observed data but thought of as a function of the unknown parameters in the model). They find the parameters which maximize the likelihood, that is, the parameters which maximize the probability of seeing that data. They use elegant asymptotic theory to make inference on the parameters or functions thereof.

Bayesian statisticians attacking the same problem, assume the parameters in the model each have a distribution of their own. First, one's beliefs about these parameters are represented in a distribution called the *prior* distribution. Using the laws of probability, these beliefs are updated to the *posterior* distribution on which inferences are based. The prior distribution can allow for quite a flexible analysis, and therefore have a tremendous advantage over frequentist methods. Not surprisingly, this greatest benefit is also the greatest criticism, and rightfully so. Are humans able to reliable express their beliefs in a mathematical function? Likely not. In practice, it is extremely difficult to specify prior distributions. As will be illustrated later in the thesis, priors for neural networks are especially difficult to define because the parameters may not have simple

interpretations.

Quite generally, assume we have random variables $Y_1, \ldots, Y_n$ which generate data $y = (y_1, \ldots, y_n)$ from some parametric density $f(y_i|\theta)$ (sometimes written as $f_\theta(y_i)$). After we specify a prior distribution of the parameter θ, denoted by $p(\theta)$, we can appeal to the laws of probability and form the posterior distribution of θ in light of the observed data y, denoted by $p(\theta|y)$. The calculation is as follows:

$$\begin{aligned} p(\theta|y) &= \frac{f(y|\theta)p(\theta)}{\int f(y|\theta')p(\theta')d\theta'} \\ &\propto f(y|\theta)p(\theta). \end{aligned}$$

In many cases, the posterior distribution is not in a form easily recognizable as one of the standard distributions. Under certain general conditions, Markov Chain theory allows us to generate a Markov Chain which has the posterior distribution as its stationary distribution. Therefore, we follow a Markov chain for some period of time and monitor whether it converges. Assuming the chain has converged, we generate very large samples (i.e. Monte Carlo) from and summarize posterior attributes (or functions thereof) from these. This method of computation is called Markov Chain Monte Carlo (MCMC) for obvious reasons.

Also of importance, especially here in this thesis, is the concept of a *predictive* distribution. Suppose that we have a new observation y^* that we wish to make predictions for based on the previous observations. Again, using the laws of probability we can generate the conditional distribution of y^* given the observed data y, as follows:

$$p(y^*|y) = \int f(y^*|\theta)p(\theta|y)d\theta.$$

After determining the posterior distribution (perhaps by simulation), predictive inferences for this new observation can then be made.

1.3 Current Approaches to Bayesian Neural Networks

In this section, we discuss the recent methodology for analyzing neural networks under the Bayesian framework. Here, we discuss the work of

- Neal (1996)
- Rios Insua and Müller (1998); Müller and Rios Insua (1998)
- Andrieu, de Freitas and Doucet (1999); and,
- Lee (2000a, 2000b).

In each case, I keep the notation used in the original papers in order to form a distinction.

1.3.1 Neal

Similar to that in the previous sections, Neal defines the network with one hidden layer as:

$$\begin{aligned} f_k(x) &= b_k + \sum_j v_{jk} h_j(x) \\ h_j(x) &= \tanh\left(a_j + \sum_i u_{ij} x_i\right) \end{aligned}$$

where $f_k(x)$ denotes the k^{th} output. The biases for the output and hidden units are a_j and b_k, respectively. The weight from the i^{th} input unit to the j^{th} hidden unit is denoted by u_{ij}. The weight from hidden unit j to output unit k is denoted by v_{jk}. The tanh activation function is the same shape as the logistic function but ranges from -1 to 1 instead of 0 to 1.

Suppose w_{ij} is any of the network weights (either u or v above). Neal specifies priors on the weights hierarchically as follows:

$$\begin{aligned} w_{ij}|\sigma^2_{w_i} &\sim N(0, \sigma^2_{w_i}), \quad \text{and} \\ \sigma^2_{w_i} &\sim \Gamma^{-1}\left(\frac{\alpha_w}{2}, \frac{\alpha_w \omega_w}{2}\right) \end{aligned}$$

where Γ^{-1} represents the inverted Gamma distribution That is, if $X \sim \text{Gamma}(a, b)$, denoted $\Gamma(a, b)$, then $1/X \sim \text{Inverted-Gamma}(a, b)$, denoted $\Gamma^{-1}(a, b)$. Notice that there is a hyperparameter specified for each unit that the weight leaves from. Neal does, however, consider special cases where each hyperparameter is the same across the group. For example, if all the inputs were to known be on an equal footing, one could take $\sigma^2_{w_1} = \cdots = \sigma^2_{w_p} = \sigma^2_w$. The hyperparameters α_w and

ω_w are selected so as to assume very vague prior information regarding $\sigma^2_{w_i}$. Similarly, hierarchical priors are specified for the bias parameters.

An advantage of specifying a hyperparameter for the connections coming out of each input unit is, that if the weights of connections coming out of some input i are typically small, then the values of the parameter $\sigma^2_{w_i}$ would also be small, thus signifying little relevance to the targets. In this case, the impact of this input would be automatically downweighted. Mackay (1994) and Neal (1996) call this Automatic Relevance Determination (ARD).

For inference, Neal (1993) uses a MCMC method called hybrid Monte Carlo based on dynamical simulation techniques to make updates of the network weights and biases. Also, if the full conditional distributions of the hyperparameters are known, Gibbs sampling steps can be performed on them.

1.3.2 Rios Insua and Müller

Rios Insua and Müller (1998) define their model for a single output and p predictors as:

$$y_i = \sum_{j=1}^{M} \beta_j \, \psi(x_i^T \gamma_j) + \epsilon_i, \quad i = 1, \ldots, n$$

where

$$\epsilon_i \sim N(0, \sigma^2) \quad \text{and} \quad \psi(z) = \frac{1}{1 + e^{-z}},$$

and $x_i = (x_{i0}, x_{i1}, \ldots, x_{ip})$ and so that input-to-hidden biases are present, $x_{i0} \equiv 1$.

They define hierarchical priors similar to Neal, as follows:

1. $\beta_j | \mu_\beta, \sigma^2_\beta \sim N(\mu_\beta, \sigma^2_\beta)$ where $\mu_\beta \sim N(a_\beta, A_\beta)$ and $\sigma_\beta^{-2} \sim \Gamma(c_b/2, c_b C_b/2)$
2. $\gamma_j | \mu_\gamma, S_\gamma \sim N(\mu_\gamma, S_\gamma)$ where $\mu_\gamma \sim N(a_\gamma, A_\gamma)$ and $S_\gamma^{-1} \sim \text{Wish}(c_\gamma, (c_\gamma C_\gamma)^{-1})$
3. $\sigma^{-2} \sim \Gamma(s/2, sS/2)$

Their key observation is that conditional on the $\gamma = (\gamma_1, \ldots, \gamma_M)$ parameter, one simply has a standard linear model. As a result, the MCMC algorithm can be implemented efficiently by integrating out $\beta = (\beta_1, \ldots, \beta_M)$. Metropolis steps are done on γ after marginalizing over

β. Conditional on γ and the hyperparameters, β is multivariate normal. Similar to Neal, all hyperpriors are conjugate and can be sampled efficiently.

They extend their method to the variable architecture case, that is, where the number of hidden nodes is not fixed. Because there is nonnegligible uncertainty about whether a certain number of nodes is adequate, this is a particularly appealing approach. They develop a reversible-jump MCMC algorithm (Green 1995) in which a Markov Chain jumps randomly to different architectures, therefore having differing parameter spaces.

1.3.3 Andrieu, de Freitas and Doucet

Andrieu, de Freitas and Doucet (1999) define a model with k hidden nodes and c outputs of the form

$$\mathbf{y}_t = \mathbf{b} + \boldsymbol{\beta}^T \mathbf{x}_t + \sum_{j=1}^{k} \mathbf{a}_j \phi(||\mathbf{x}_t - \boldsymbol{\mu}_j||) + \mathbf{n}_t, \quad t = 1, \ldots, N,$$

where $\mathbf{n}_t$ are zero-mean independent Gaussian errors with constant variances $\sigma_1^2, \ldots, \sigma_c^2$ for each of the c outputs and $\phi(\cdot)$ is a radial basis function (RBF). The μ_j parameters represent the radial centres. Common choices of the RBF may be:

- Linear: $\phi(\eta) = \eta$
- Thin plate spline: $\phi(\eta) = \eta^2 \log \eta$
- Multi-quadric: $\phi(\eta) = \sqrt{\eta^2 + \lambda^2}$
- Gaussian: $\phi(\eta) = \exp(-\lambda \eta^2)$.

The interpretations of the ensuing radial basis networks would presumably be quite different for these functions.

Andrieu, de Freitas and Doucet (1999) define a prior distribution over the parameters $(\mathbf{b}, \boldsymbol{\beta}, \mathbf{a}, \boldsymbol{\mu}, \sigma^2)$ as follows. *A priori*, the radial basis centres $\boldsymbol{\mu}_j$ are uniformly distributed over the space of where the input data lie, conditional on k. That is, they are restricted to the hypercube with one corner at all the maximum values of each of the predictors and the opposite corner at all of the minimum values of the each of the predictors. In this respect, the prior is slightly

data dependent. Conditional on the centres, the collection of *regression* parameters $(\mathbf{b}, \boldsymbol{\beta}, \mathbf{a})$ are specified to have a $(1+d+k)$-dimensional Gaussian distribution centred at the origin with inverse variance-covariance matrix $\frac{1}{\delta_i^2} D_i^T D_i$ where D_i is the design matrix including columns for k radial basis functions, d covariates and an intercept. Hence, the prior penalizes the radial basis functions for being too close. The parameter k is also taken to be random and is assumed to have a truncated (i.e. $k < N - (d+1)$) Poisson distribution with parameter Λ. The truncation is necessary to avoid columns of D_i where are linearly dependent. To finalize the prior distribution, Jeffrey's noninformative prior is used for σ_i^2 (i.e. $p(\sigma_i^2) = \sigma_i^{-2}$), and vague inverted Gamma and Gamma priors are used for each of δ_i^2 and Λ.

To simplify the posterior distribution, the authors marginalize over $(\mathbf{b}, \boldsymbol{\beta}, \mathbf{a})$ (Gaussian), analogous to what is done in Rios Insua and Müller (1998).

Sampling of the posterior distribution with k fixed is similar to Rios Insua and Müller (1998). Metropolis-Hastings steps are performed sequentially for each hidden node $j = 1, \ldots, k$ on μ_j. Following this, the regression parameters $(\mathbf{b}, \boldsymbol{\beta}, \mathbf{a})$ can be sampled jointly from a multivariate normal distribution. Similarly, the variances σ_i^2 and δ_i^2 can be sampled from inverted Gamma distributions. The hyperparameter Λ can be sampled using Metropolis steps. Andrieu, de Freitas and Doucet (1999) extend this to the random k case by appealing to the reversible jump MCMC method discussed in Green (1995), like Rios Insua and Müller in the last section.

1.3.4 Lee

Most recently, Lee (2000a, 2000b) has introduced a non-informative prior distribution over the parameters $(\beta, \gamma, \sigma^2)$ to avoid the selection of a large number of hyperparameters. Lee implements a neural network model of the form

$$y_i = \beta_0 + \sum_{j=1}^{M} \beta_j \, \psi(x_i^T \gamma_j) + \epsilon_i, \quad i = 1, \ldots, n$$

where, as before $\epsilon_i \sim N(0, \sigma^2)$ and $\psi(z) = (1 + e^{-z})^{-1}$.

He selects a prior which puts equal density on all regions of the parameter space (except on the variance σ^2 where the log scale is used). A restricted version of the standard Bayesian linear

regression prior ($p(\beta, \gamma, \sigma^2) \propto \sigma^{-2}$) is used. Unlike linear regression, this improper prior for neural networks gives rise to an improper posterior. Define $Z = \{z_{ij}\}$ where

$$z_{ij} = \frac{1}{1 + \exp(-\gamma_{j0} - \sum_h \gamma_{jh} x_{ih})}.$$

In the case of logistic activation functions, a restriction of the form

$$\Omega_n = \left\{(\beta, \gamma, \sigma^2) : |\gamma_{jh}| < C_n, |Z^T Z| > D_n\right\}$$

ensures linear independence of the logistic basis functions. Lee proves that posterior in this case is indeed proper and suggests values of C_n and D_n which work well in practice.

For MCMC sampling, Lee uses a streamlined version (i.e. because of fewer parameters) of the fixed-architecture Rios Insua and Müller algorithm.

Lee migrates to a discussion of model selection using Bayesian model averaging. Briefly, because there is uncertainty as to whether any given model is the correct one (or that there are multiple explanations of the data), it is of interest to combine and average over many candidate models. A review of this technique is given in Hoeting et al. (1999). Using Lee's notation, the predictive distribution of responses Y based on observed data D and q candidate models $M_1, \ldots, M_q$ is

$$P(Y|D) = \sum_i P(Y|D, M_i) P(M_i|D).$$

Clearly, this is a weighted average where the weights are the posterior probabilities of each model. Unfortunately, the term $P(M_i|D)$, sometimes known as the *evidence* is typically a difficult quantity to estimate.

Due to the potentially large number of possible models (subsets of the explanatory variables + number of hidden nodes), he introduces a Bayesian Random Searching (BARS) algorithm which randomly proposes jumps from model to model. Based on the Bayesian Information Criterion (BIC)[2], these steps are accepted or rejected. Assuming the models have equal prior probability, the posterior probabilities can be approximated as

$$P(M_i|D) = \frac{e^{BIC_i}}{\sum_j e^{BIC_j}}.$$

[2] $BIC_i = L_i - \frac{1}{2} p_i \log n$ where L_i is the maximized log-likelihood and p_i is the number of parameters in the model; BIC is an asymptotic approximation to the log Bayes factor under certain conditions.

After the BARS algorithm has found a reasonable subset of the model, a regular MCMC chain is run to fit a neural network on each. From the estimated posterior probabilites, predictions (or other posterior summaries of interest) can be averaged over numerous models.

1.4 Datasets Used in the Thesis

In this Section, I give a brief description of the three datasets which are analyzed in the thesis:

1. Ozone Data (Lee 2000b)
2. Tecator Data (Thodberg 1996)
3. Boston Housing Data (Neal 1996)

1.4.1 Ozone Data

This dataset consists of 330 observations of groundlevel ozone in the Los Angeles area, as well as the nine explanatory variables explained below

Covariate	Description
O3	ozone concentration
vh	altitude at which the pressure is 500 millibars
wind	wind speed (mph)
hum	humidity (%)
temp	temperature (degrees F)
ibh	temperature inversion base height (feet)
dpg	pressure gradient (mmHg)
ibt	inversion base temperature (degrees F)
vis	visibility (miles)
doy	day of year

Figure 1.2 gives a scatterplot of the data. One should note non-linearities of the response (e.g. O3 vs. doy) and predictors which are highly correlated (e.g. temp vs. ibt).

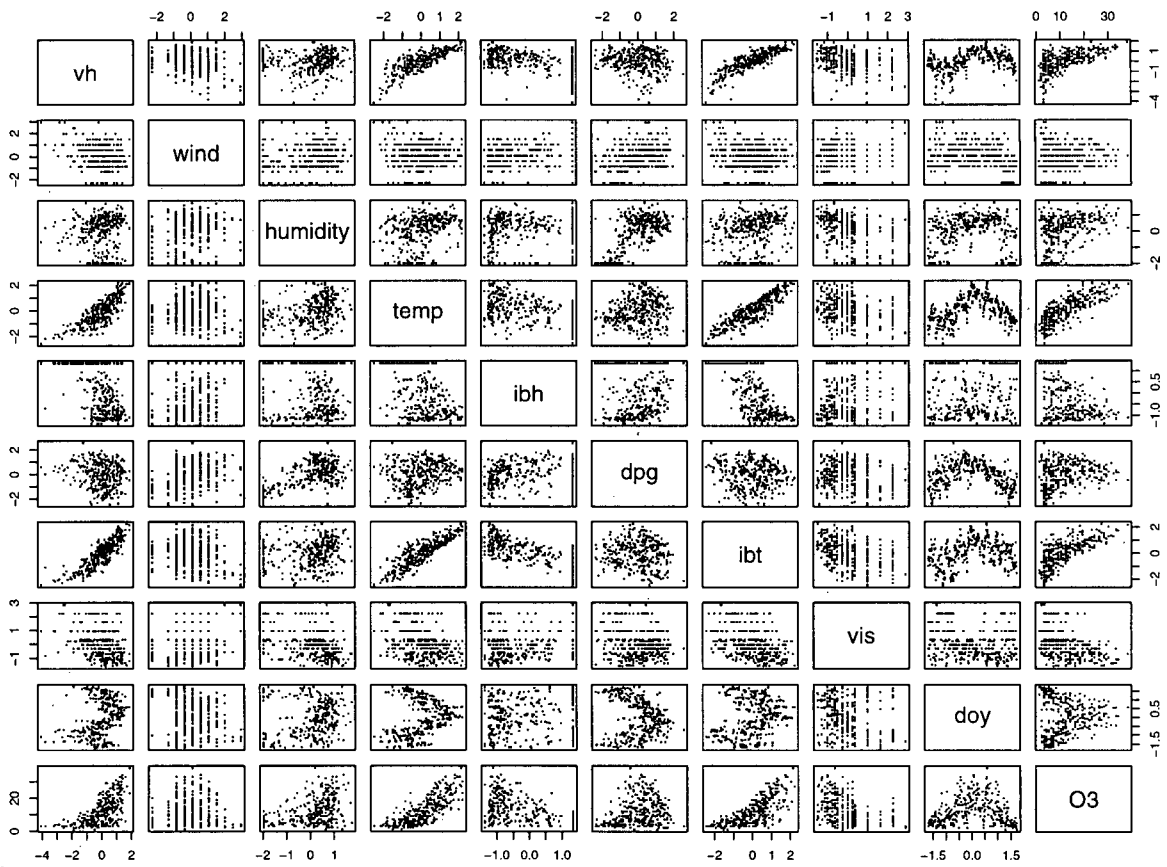


Figure 1.2: Scatter Plot of Ozone Data Set.

1.4.2 Tecator Data

The Tecator dataset involves predicting the fat content of meat based on spectrometer measurements at 100 absorbencies. Tecator is a Swedish company that manufactures the *Infratec Food and Feed Analyzer*, the instrument used to determine the absorbencies at 100 wavelengths in the region 850–1050 nm. The 10 variables PC1,...,PC10 are shown in Figure 1.3 as well as the fat content response, y. PC1,...,PC10 represent the first 10 principal components of the measurements at the 100 absorbencies.

1.4.3 Boston Housing Data

The Boston Housing dataset was introduced by Harrison and Rubinfeld (1978) and later analyzed in Neal (1996). Though the initial goal was not prediction, our results can be compared to Neal's. The data are comprised of observations on 14 items for 506 census tracts in the metropolitan Boston area. Our goal is to make predictions of the median house value using the other 13 covariates. A description of the variables is given below:

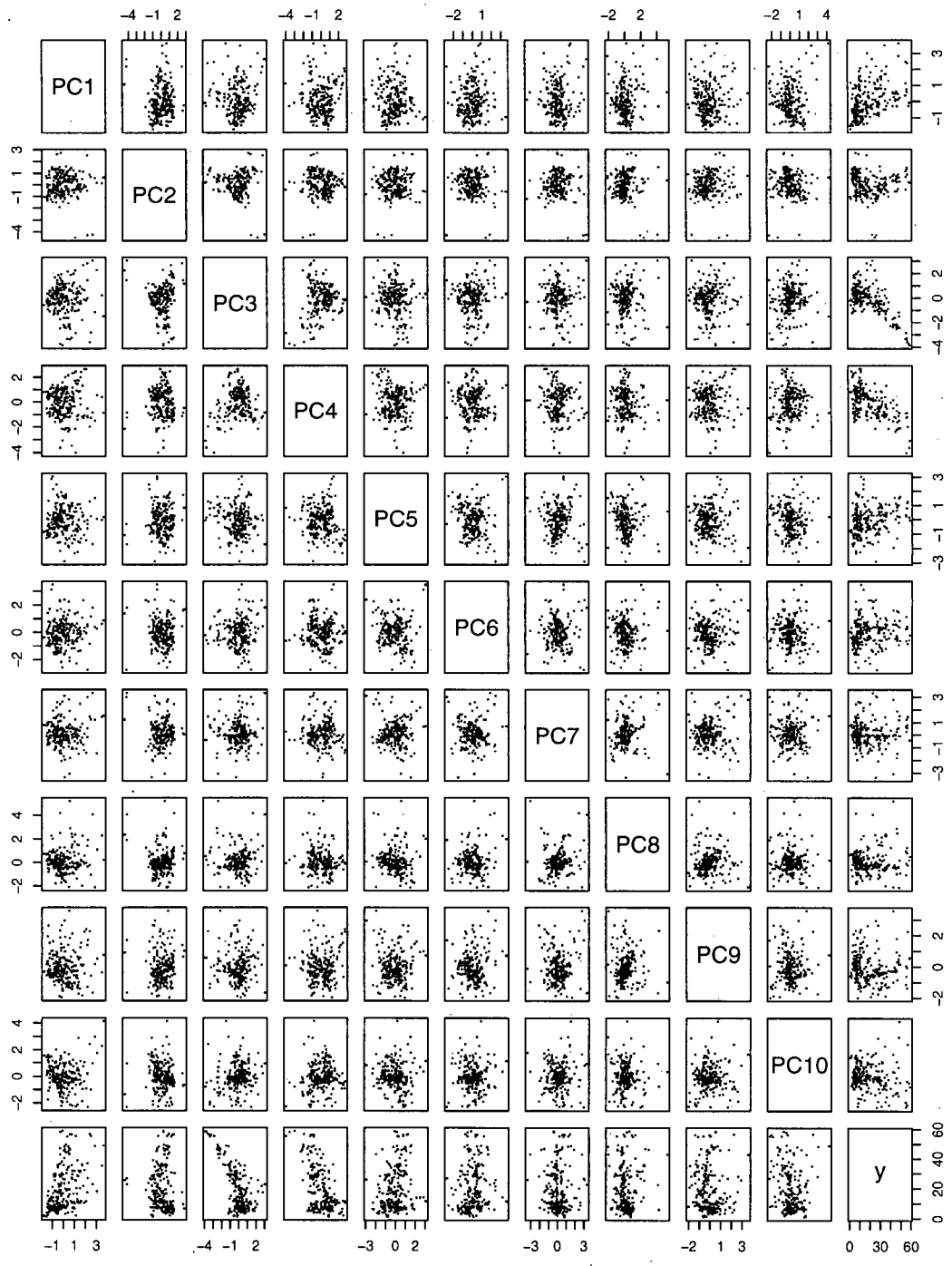


Figure 1.3: Scatter Plot of Tecator Data Set.

Covariate	Description
CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	nitric oxides concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centres
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per $10,000
PTRATIO	pupil-teacher ratio by town
B	1000(Bk - 0.63)$\hat{2}$ where Bk is the proportion of blacks by town
LSTAT	% lower status of the population
MEDV	median value of owner-occupied homes in $1000's

1.5 Outline and Scope of the Thesis

Though neural networks are capable of applications with multiple outputs, we limit ourselves here to a network of a single output unit. The methods described in the following chapters can easily be extended to this case. We also limit ourselves to the fixed architecture case, although we fit networks at differing numbers of hidden nodes in order to address the differences.

I now define the notation that will be used for the remainder of the thesis. We consider a neural network with k hidden nodes for regression of y on a p-dimensional covariate x as follows

$$y_i = \beta_0 + \sum_{j=1}^{k} \beta_j \, \psi(\alpha_j + \gamma_j^T x_i) + \epsilon_i, \quad i = 1, \ldots, n.$$

where the ϵ_i's are independent Gaussian with mean 0 and variance σ^2 and $\psi(\cdot)$ is the logistic function. The parameters $\alpha = (\alpha_1, \ldots, \alpha_k)$ and $\gamma = (\gamma_1, \ldots, \gamma_k)$ are referred to as the input-to-hidden biases and weights, respectively. The parameter $\beta = (\beta_0, \beta_1, \ldots, \beta_k)$ is comprised of a single

Figure 1.4: Scatter Plot of Boston Housing Data Set.

hidden-to-output bias (β_0) and the hidden-to-output weights $\beta_1, \ldots, \beta_k$. Commonly, β is referred to as the *regression* parameter. For notational convenience, I also define here $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$.

The thesis discusses and validates approaches to specifying the prior distribution over the neural network parameters based on geometrical properties. Chapter 2 discusses new prospects for prior distributions over the network parameters. Chapter 3 mentions the details of an MCMC sampling scheme. Empirical results of predictive performance for the three datasets are presented in Chapter 4. The last Chapter gives some concluding remarks and future directions, including a brief description of model selection and an analysis based on the Deviance Information Criterion (DIC). The implementation details and some auxiliary plots are given in an Appendix.

Chapter 2

Considerations for the Prior Distributions

In this chapter, I discuss alternative specifications of the prior distribution over the network parameters. First, I motivate the topic with a simple example and then the priors that we use are introduced.

2.1 Motivation

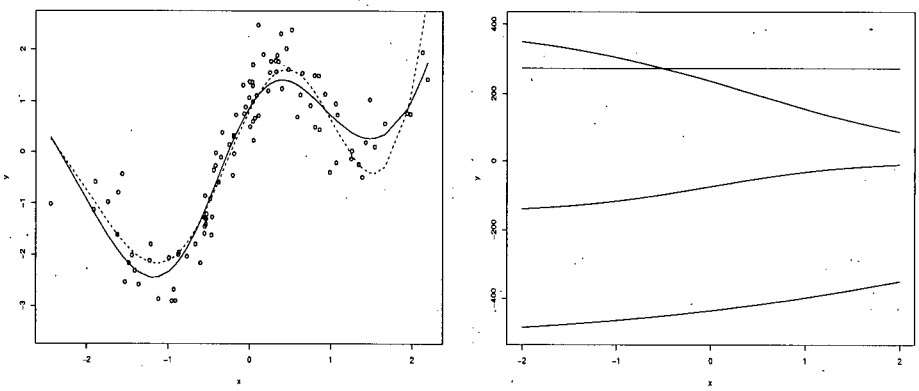
To motivate the discussion, consider a synthetic dataset of 200 observations which has the following true relationship

$$y = \cos(2.5x) + 1.5\sin(2.5x) + x + \epsilon,$$

where $\epsilon \sim N(0, \sigma^2 = .25)$. The left panel of Figure 2.1 shows the data as well as 2 neural network fits to the data using the methods of this thesis. The two fits were taken from 2 Markov Chains that were started at different initial values.

A neural network with 3 hidden nodes was used. The fit corresponding to the solid line in

Figure 2.1: Interpretation of Neural Network Parameters.



(a) Data and Neural Network Fits (b) Components of Neural Network

Figure 2.1(a) was the following

$$\begin{aligned}\hat{y}(x) &= 274.41 \\ &\quad -527.65\,\psi(+1.56 - 0.44x) \\ &\quad +394.28\,\psi(+0.41 - 0.83x) \\ &\quad -150.42\,\psi(-0.04 - 1.28x)\end{aligned}$$

where $\psi(\cdot)$ is the logistic function. Considering the scale that these parameters are on, it is interesting that, of all possible parameter combinations, the Markov Chain sampled this particular one. The right panel of Figure 2.1 shows the 3 separated logistic nodes in addition to the intercept term. The fit corresponding to the dotted line on panel (a) is

$$\begin{aligned}\hat{y}(x) &= +662.49 \\ &\quad -621.33\,\psi(-0.58 + 0.58x) \\ &\quad +24.87\,\psi(+0.73 + 1.97x) \\ &\quad -662.05\,\psi(+0.79 - 0.53x).\end{aligned}$$

Hence, there is no similarity for either the regression parameters or for the input-to-hidden weights.

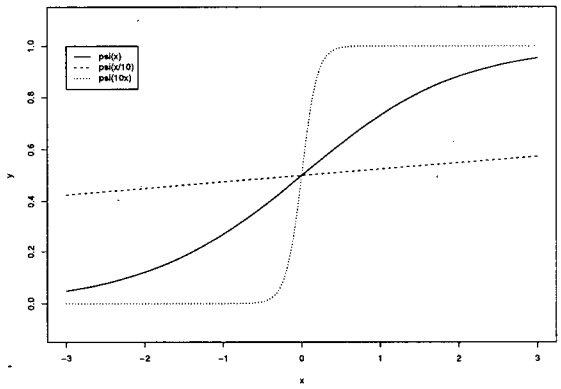


Figure 2.2: Examples of Logistic Nodes.

From both the numerical and the graphical displays, two central themes about neural networks are evident. First of all, the parameters would be difficult to interpret, and thus in a Bayesian framework, specification of a prior over them is also very difficult. Secondly, it is clear that many explanations of the data are possible, that is, there are many combinations of network parameters which bear no apparent resemblance, but give approximately the same relationship.

2.2 The Geometry of the Problem

For simplicity, consider a covariate x which is unidimensional. Consider the form of one hidden node as

$$y = \psi\left(\frac{x-\rho}{\nu}\right).$$

where ρ represents the center of the curvature and ν describes the degree of curvature. Figure 2.2 shows examples of logistic nodes. Here, we see the common S-shaped form of the logistic function. By changing the argument to $x/10$ and $10x$, the logistic node becomes effectively linear and effectively a step function over $x \in (-3, 3)$, respectively.

Due to this quite flexible setup, neural networks are able to capture quite an elaborate relationship.

2.3 Weight Decay

A common nuisance in neural network modelling is that of overfitting. That is, because the network is so versatile, it may end up fitting the random noise in the data instead of the true relationship. Given enough hidden nodes, the network may be capable of a fit with almost no error. Obviously, this would not generalize well for prediction.

Analogous to regularization in smoothing splines, *weight decay* penalizes for large values of the network weights ($\gamma_1, \ldots, \gamma_k$ in our notation), the values which correspond to sharper increases in the activation function. Smaller values of the weights correspond to activation functions which are closer to linearity. Thus, weight decay, if implemented appropriately, reduces oscillations and encourages a smoother response surface and can hopefully prevent overfit. Also a contributing factor to the degree of smoothness of the neural network output is the number of hidden nodes, k.

In the frequentist setting, a penalty term is added to the likelihood (and therefore can be referred to as penalized likelihood) which achieves weight decay. To illustrate, let

$$f(x_i) = \beta_0 + \sum_{j=1}^{k} \beta_j \, \psi(\alpha_j + \gamma_j^T x_i).$$

Assuming normally distributed errors, one would maximize (with respect to β and γ) the log-likelihood

$$l(\beta, \gamma) = \sum_{i=1}^{n} \frac{(y_i - f(x_i))^2}{\sigma^2} + \sum_{j=1}^{k} \frac{||\gamma_j||^2}{\tau^2}$$

where τ^2 determines the severity of the penalty. A tradeoff between fitting the training set well and having a generalizable model is necessary. Commonly, the choice of τ^2 is picked based on cross-validation experiments. Alternatively, a dataset is broken down into a training, validation and test set. The networks under different choices for τ^2 are fit on the training set and the fits of each are evaluated on the validation set. The value of τ^2 which gives the best fit is chosen and the resulting performance on the test set is reported.

In the Bayesian paradigm, the above approach is no different than specifying a prior on each γ_j. It would correspond to a normal prior centred at zero with variance τ^2 for each γ_{ij}.

2.4 Possibilities for the Prior

As mentioned in the previous section, the typical prior on γ_j which achieves weight decay is:

$$\gamma_j \sim N(0, \tau^2 I) \text{ for } j = 1, \ldots, p. \tag{2.1}$$

An appropriate value of τ^2 is generally not known a prior. In the Bayesian framework, this uncertainty is best captured in a prior distribution as is commonly done in the literature (Neal 1996, Rios Insua and Müller 1998). The standard approach, essentially for mathematical convenience, is to take an inverted gamma (Γ^{-1}) prior on τ^2. Neal discusses priors which disperse mass over a very broad range on the positive real line. For example, when analyzing the Boston Housing data, he uses parameters that give τ^{-2} a mean of 100 and a variance 200,000 for each of the input-to-hidden weight groups. Rios Insua and Müller use somewhat less disparate Γ^{-1} priors and give little guidance as to how they were selected.

Though Lee advocates a prior which in effect has no weight decay, it has been our experience that weight decay is necessary for reliable prediction in the Bayesian framework, especially in long runs of the Markov Chains (see Section 4.1).

For us, this motivates two further possibilities. First, we explore other senses of parsimony such as orthogonality of the weights γ_j or additivity of the hidden nodes. Second, we consider priors which encourage the parameters to reside in an *effective domain of interest.*

2.5 Parsimony Priors

Note that the prior on γ_j as in (2.1) combined with a $\Gamma^{-1}(a, b)$ hyperprior on the τ^2 parameter is mathematically equivalent to an unconditional Multivariate-t distribution for γ_j. That is,

$$p(\gamma_j) \propto \left(1 + \frac{1}{b}||\gamma_j||^2\right)^{-\frac{a+p}{2}}. \tag{2.2}$$

Consider a reparameterization of $(\alpha_j, \gamma_j) \longleftrightarrow (\mu_j, w_j, \lambda_j^2)$ given by

$$w_j = \frac{1}{||\gamma_j||}\gamma_j, \quad \mu_j = \frac{-\alpha_j}{||\gamma_j||} \text{ and } \lambda_j^2 = \frac{1}{||\gamma_j||^2}, \tag{2.3}$$

so that

$$\psi\left(\alpha_j + \gamma_j^T x\right) = \psi\left(\frac{w_j^T x - \mu_j}{\lambda_j}\right).$$

This helps one to think of desired geometrical properties of the unit vector w_j which promote model parsimony. A couple of possibilites follow.

2.5.1 Orthogonality of Input-to-Hidden Weights

Since one desires different hidden nodes to explain distinct components of the relationship, we may discourage w_i and w_j from pointing in the same direction. One possibility is to add a penalty for larger values of the absolute inner product between the all unit vectors w_i and w_j, perhaps in the same flavour as (2.2), such as

$$f_{\text{orth}}(w_1, \ldots, w_k) \propto \left(1 + \frac{1}{b}\frac{2}{k(k-1)}\sum_{i<j}|w_i^T w_j|\right)^{-\frac{k(a+p)}{2}}. \tag{2.4}$$

The term $\frac{2}{k(k-1)}\sum_{i<j}|w_i^T w_j|$ represents average absolute dot product amongst the $\frac{k(k-1)}{2}$ pairs of network weights. If all input-to-hidden weights are perfectly orthogonal (assuming $k < p$), then $f(w_1, \ldots, w_k) \equiv 1$ whereas if all input-to-hidden weights are linearly dependent, $f(w_1, \ldots, w_k) = (1 + b^{-1})^{-k(a+p)/2}$, a smaller value. Here, a and b determine the degree of penalization as they would for the weight decay prior.

Of course, in the Bayesian framework, really what is meant by adding a penalty is specifying a prior distribution. In this case, we have a prior which peaks in regions where the $(w_1, \ldots, w_k)$ are more orthogonal and has lowest density when they are close to the linear dependence.

2.5.2 Additivity of the Hidden Nodes

Yet another possibility for model parsimony is to encourage sparse linear combinations, where there are a small number of non-zero entries in the weight vector. In the limiting case, total *additivity* of the hidden nodes would result in a network of the form

$$f(x_i) = \sum_{j=1}^{k} \psi\left(\frac{x_{ij^*} - \mu_j}{\lambda}\right)$$

whereby the activation function acts on only one covariate at a time and, if $k < p$, then it acts on a subset of the original variables. If the true relationship is extremely non-linear in a particular covariate, it may require more than one hidden node. Also, because covariate j may not necessarily correspond with node j, the alternative definition x_{ij^*} represents a reshuffling of the x variables. A neural network of this type may mimic somewhat the flavour of generalized additive models (Hastie and Tibshirani 1990), except that ψ is fixed in advance.

To promote a prior distribution which achieves some degree of additivity, we may penalize unit vectors of the form $(\frac{1}{\sqrt{p}}, \ldots, \frac{1}{\sqrt{p}})$ in favour of vectors of the form $(0, \ldots, 0, 1, 0, \ldots, 0)$. This is achieved by defining a function, for example $g(u) = \sum_i |u_i| - 1$, and defining the prior as

$$f_{\text{add}}(w_1, \ldots, w_k) \propto \prod_{j=1}^{k} \left(1 + \frac{1}{b} g(w_j)\right)^{-\frac{(a+p)}{2}}. \tag{2.5}$$

The function $g(u)$ is maximized when $u = (\frac{1}{\sqrt{p}}, \ldots, \frac{1}{\sqrt{p}})$ corresponding to each covariate in a hidden node having equal weight. Since this is a scenario we wish to discount, the prior affords it the least mass. Accordingly, the prior gives largest mass when the weight is solely on one covariate. Again, the hyperparameters a and b determine the degree of penalization.

2.5.3 Selection of a and b

One approach for selecting the parameters a and b is that of a "factor of z" calculation. In the original weight decay prior on τ^2, the parameter a represents the degrees of freedom and $\frac{b}{a}I$ is the scale of the Multivariate-t distribution.

First of all, we pick a to be *small* to induce heavy tails (i.e. small number of degrees of freedom in a t-distribution). To select a value for b, we use the following. In the case of orthogonality, we favour total orthogonality over total linear dependence by a factor of z. That is,

$$\frac{f_{\text{orth}}(\text{total orthogonality})}{f_{\text{orth}}(\text{linear dependence})} = z$$

By the same approach, we favour total addivity over equal strength by a factor of z.

2.6 Effective Domain Prior

From the reparameterization above in (2.3), one can think of $w^T x$ as resting most often within a so-called *effective domain of interest* as defined by $(-\Delta(w), \Delta(w))$. So that this definition is tenable, the x's are prescaled (and precentred), a common procedure for models of this type.

One logical possibility, based on the standard deviation of the linear combination $w_j^T x$ and the notion of approximate normality (where 2 standard deviations covers most of the range of the data), may be

$$\Delta(w) = 2\sqrt{w_j^T R w_j}, \tag{2.6}$$

where R is the correlation matrix of x.

Consider the logistic activation function, $\psi(z) = (1 + e^{-z})^{-1}$. The function is well approximated by a function of the form

$$\psi_c(z) = \begin{cases} 0, & z < c, \\ z, & -c \leq z \leq c, \\ 1, & z > c. \end{cases}$$

Selecting $c = 3$ seems to work well, for example (see Figure 2.3). Large values of λ typically induce a more linear activation function, at least over the range of $\alpha_j + \gamma_j^T x_i$. The slope of this linear section is controlled to some extent by the λ parameter, but also by the β_j corresponding to the node. Hence, we have a redundancy.

In the following, we consider a restriction on λ and μ (in terms of $\Delta = \Delta(w)$) which should help alleviate this redundancy. Let $z = w^T x$ and consider, for simplicity, one hidden node. In this case, we are considering

$$y(z) = \beta_0 + \beta_1 \psi\left(\frac{z - \mu}{\lambda}\right),$$

where $z \in (-\Delta, \Delta)$.

Restriction on λ. Concerned mainly with the internal linear piece, suppose we focus on the situation where

$$\frac{\Delta - \mu}{\lambda} < c \text{ and } \frac{-\Delta - \mu}{\lambda} > -c$$

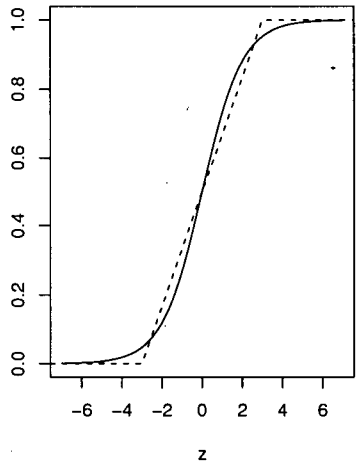


Figure 2.3: A Piecewise Linear Approximation to the Logistic Function.

which corresponds to

$$\lambda > \frac{\Delta - \mu}{c} \quad \text{and} \quad \lambda > \frac{\Delta + \mu}{c}.$$

In this situation, the function is well approximated by the linear central piece, leading to a function of the form

$$y(z) = \begin{cases} \beta_0 & z < -\Delta, \\ \beta_0 + \beta_1 \left(\frac{z-\mu}{\lambda}\right) & -\Delta \leq z \leq \Delta, \\ \beta_0 + \beta_1 & z > \Delta. \end{cases}$$

But, there are 4 parameters $(\beta_0, \beta_1, \mu, \lambda)$ available to characterize this function, which is an obvious overparameterization.

Restriction on μ. Similarly, we could also restrict our attention to the situation where $\mu < |\Delta|$. For values of μ outside this region, the majority of the curvature of the logistic function is missed (see figure 2.4) and, again we encounter a relationship that can be expressed in fewer parameters.

Hence, by restricting $\mu \in (-\Delta(w), \Delta(w))$ and $\lambda < \frac{\Delta(w)+|\mu|}{c}$, we can discount situations that have alternative representations. A parameter value outside this region will have a representative inside the region (or, on the boundary) which is able essentially to characterize the same functional

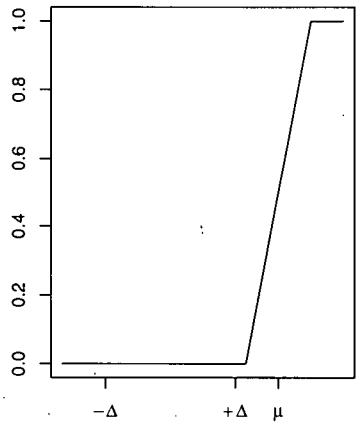


Figure 2.4: An Example of μ outside the *effective domain of interest.*

form. Thus, we may restrict ourselves to this region with little or no effect on the model. In practical terms, though, we only encourage the parameters to lie in this region by specifying a prior distribution which has greater density in these areas.

In our approach, we advocate the desired region of the parameters by specifying the prior distributions as follows:

$$w \sim \text{Uniform}, \tag{2.7}$$

$$\mu|w \sim N\left(0, \left\{\frac{\Delta(w)}{2}\right\}^2\right) \tag{2.8}$$

$$\lambda^2|\mu, w \sim \Gamma^{-1}\left(\frac{p}{2}, \left(\frac{p}{2}+1\right)\left\{\frac{\Delta(w)+|\mu|}{c}\right\}^2\right) \tag{2.9}$$

where the shape parameter of the Γ^{-1} mimics previous priors and the scale parameter gives a mode at the boundary, $(\frac{\Delta(w)+|\mu|}{c})^2$. Essentially, we are downweighting large values of λ for being redundant and small values of λ as weight decay. Note also this is a weakly data-dependent prior via the correlation matrix R. The prior is self-calibrating because the calculation of $\Delta(w)$ requires no subjective input.

2.7 Priors on β and σ^2

In previous analyses of neural networks (Neal 1996; Rios Insua and Müller 1998), the prior on the β parameter is of the same hierarchical structure as the prior on γ_j. That is, weight decay is also applied to the β's. Such a prior would encourage *ridge regression.* Although ridge regression has a definite benefit in linear regression where the predictors are highly correlated, it is unclear whether it is useful in a neural network context. Thus, we follow Lee (2000a, 2000b) and take the standard non-informative prior for Bayesian linear regression, $p(\beta) \propto 1$. The posterior is proper despite using an improper prior.

The prior on σ^2 is also taken from Lee, who uses

$$p(\sigma^2) \propto \frac{1}{\sigma^2},$$

which is both non-informative and improper. The prior distributes equal mass over all regions for $\log \sigma^2$ and also results in a proper prior. Again, this is the standard non-informative prior (i.e. Jeffrey's prior) for Bayesian linear regression.

Chapter 3

Inference via Markov Chain Monte Carlo

In this chapter, I discuss the MCMC sampling scheme that was used to fit the neural networks. The reader is referred to the Appendix for the implementation details using S-plus and C.

3.1 The Posterior Distribution

To make inferences and predictions, we need to be able to sample from the posterior distribution, denoted hereafter by $p(\beta, \alpha, \gamma, \sigma^2|x, y)$. The posterior distribution can be expressed as

$$p(\beta, \alpha, \gamma, \sigma^2|x, y) = p(y|x, \beta, \alpha, \gamma, \sigma^2) \cdot p(\beta, \alpha, \gamma, \sigma^2),$$

where $p(y|x, \beta, \alpha, \gamma, \sigma^2)$ is the likelihood and $p(\beta, \alpha, \gamma, \sigma^2)$ is the prior on the network parameters and hyperparameters. For notational convenience in what follows, consider the following definition. Let Z be a $n \times (p+1)$ design matrix with elements

$$z_{ij} = \begin{cases} 1 & \text{if } j = 1, \\ \psi(\alpha_{j-1} + \gamma_{j-1}^T x_i) & \text{otherwise.} \end{cases}$$

With this simplification, one can think of $(Z\beta)_i$ as the prediction for observation i.

In the regression context where the errors are assumed to be independent Gaussian (with constant variance σ^2), the posterior distribution is of the form

$$p(\beta, \alpha, \gamma, \sigma^2 | x, y) \propto \frac{1}{(\sigma^2)^{\frac{n}{2}}} \exp\left\{-\frac{1}{2\sigma^2}(y - Z\beta)^T(y - Z\beta)\right\} \cdot \frac{1}{\sigma^2} \cdot p(\alpha, \gamma). \tag{3.1}$$

The priors given in Chapter 2 (equations 2.7, 2.8 and 2.9) are not in the same parameterization as that above. Fortunately, I do not calculate the full distributions in the (α, γ) parameterization because, as I show later in the chapter, it is not necessary. The Jacobian required for this calculation cancels out in the Metropolis-Hasting acceptance ratio (see Section 3.4).

The MCMC Sampling Algorithm is discussed first and the details follow.

3.2 The MCMC Sampling Algorithm

The sampling of the posterior consists of the following steps:

1. Start with initial values for α, γ and β (e.g. Unit Gaussian random variables).
2. Draw σ^2 and β from their full conditional distributions (see Section 3.3).
3. For each $j = 1, \ldots, k$, perform Metropolis-Hastings (MH) steps on α_j, γ_j:
   - *i.* Propose a candidate $(\tilde{\alpha}_j, \tilde{\gamma}_j) \sim N((\alpha_j, \gamma_j), c^2 I_{p+1})$
   - *ii.* Accept $(\tilde{\alpha}_j, \tilde{\gamma}_j)$ with probability $\min\left(1, \frac{p(\beta, \tilde{\alpha}, \tilde{\gamma}, \sigma^2 | x, y)}{p(\beta, \alpha, \gamma, \sigma^2 | x, y)}\right)$ (see Section 3.4 for further details).
4. Repeat steps 2 and 3 for the desired number of iterations.

The parameter c can be adjusted to achieve a desired acceptance rate. It may also be beneficial to sample α_j and γ_j separately with corresponding tuning parameters c_1 and c_2 in order to optimize acceptance.

3.3 Full Conditional Distributions

3.3.1 Derivation for σ^2

The distribution of $\sigma^2|\alpha,\gamma,\beta$ can be written as

$$p(\sigma^2|\ldots) \propto \left(\frac{1}{\sigma^2}\right)^{\frac{n}{2}+1} \exp\left\{-\frac{1}{\sigma^2}\cdot\frac{(y-Z\beta)^T(y-Z\beta)}{2}\right\},$$

which can be readily identified as an inverted gamma distribution with shape parameter $\frac{n}{2}$ and scale parameter $\frac{SSE}{2}$ where SSE is the sum of squared deviations from the observations and the estimates, based on the current values of α,γ and β.

3.3.2 Derivation for β

The distribution of $\beta|\alpha,\gamma,\sigma^2$ can be written as:

$$\begin{aligned} p(\beta|\ldots) &\propto \exp\left\{-\frac{1}{2\sigma^2}(y-Z\beta)^T(y-Z\beta)\right\} \\ &\propto \exp\left\{-\frac{1}{2\sigma^2}\left[\beta^T Z^T Z\beta - 2\beta^T Z^T y\right]\right\}. \end{aligned}$$

From this representation, it is clear the full conditional distribution is multivariate normal with mean $(Z^TZ)^{-1}Z^Ty$ and variance-covariance matrix $\sigma^2(Z^TZ)^{-1}$.

3.4 Acceptance Ratio Calculation for the MH Step

The prior distribution on α and γ, as specified indirectly by (2.7, 2.8 and 2.9) is necessary for calculating the acceptance ratio of the MH Step. For what follows, assume $(\mu,w,\lambda,\Delta(w))$ is calculated from (α_j,γ_j) via the reparameterization in (2.3) and the calculation in (2.6). Similarly, $(\tilde{\mu}_j,\tilde{w}_j,\tilde{\lambda}_j,\tilde{\Delta}(w_j))$ can be calculated from the proposal $(\tilde{\alpha}_j,\tilde{\gamma}_j)$.

Define $f(\mu,w,\lambda,\Delta)$ for the hierarchical prior defined by (2.7, 2.8 and 2.9). Assume the probing density for the MH step is $q(\cdot,\cdot)$. The Jacobian of the transformation would appear as a term in both the prior (in the new parameterization) and in the probing density and would cancel

out in the acceptance ratio. Also, because we chose a symmetric prior (i.e. Gaussian), the $q(a, \tilde{a})$ and $q(\tilde{a}, a)$ terms, where a is the value at the current iteration and $\tilde{a}$ is a proposal, cancel out as well.

At last, one is left with an acceptance ratio of the form:

$$\min\left(1, \frac{f(\tilde{\mu}_j, \tilde{w}_j, \tilde{\lambda}_j, \tilde{\Delta}_j)}{f(\mu_j, w_j, \lambda_j, \Delta_j)} \cdot \frac{p(y|\tilde{\alpha}, \tilde{\gamma}, \beta, \sigma^2)}{p(y|\alpha, \gamma, \beta, \sigma^2)}\right)$$

where

$$f(\mu, w, \lambda, \Delta) = \frac{1}{\Delta} \exp\left[-2\left(\frac{\mu}{\Delta}\right)^2 - \frac{1}{\lambda^2}\left(\frac{\Delta + |\mu|}{c}\right)\left(\frac{p}{2} + 1\right)\right]\left[\frac{\Delta + |\mu|}{c\lambda}\right]^{\frac{p}{2}+1}.$$

3.5 Posterior Inference

After allowing sufficient time for the Markov Chains to converge, the next step is to make inferences from the posterior samples. Because our main interest is prediction, I discuss making a prediction for a *new* observation, say y^* based on predictors x^*.

Assume we have allowed the chain to converge and have taken a large number, N, samples thereafter, perhaps also thinning them out to reduce the autocorrelation. (This thinning is only a concern when computing simulation standard errors of the estimates, which is not the main focus here.) Let $(\beta^{(k)}, \alpha^{(k)}, \gamma^{(k)})$ represent the k^{th} sample from the posterior distribution. The prediction for this new observation y^* at x^* for sample k can be calculated as

$$\hat{y}^{(k)}(x^*) = \beta_0^{(k)} + \sum_{j=1}^{k} \beta_j^{(k)} \psi(\alpha_j^{(k)} + \gamma_j^{(k)} x^*).$$

If N is sufficiently large and the samples are sufficiently independent, then one would have an adequate approximation to the predictive distribution of $y(x^*)$ based on the samples $1, 2, \ldots, N$. To summarize in a point estimate, one might choose the mean of the predictive distribution at the point x^*. This quantity can be estimated by

$$E(Y|X = x^*) \approx \hat{y}(x^*) = \frac{1}{N} \sum_{k=1}^{N} \hat{y}^{(k)}(x^*).$$

For the analyses presented in the next section, I calculate a mean square error (MSE) for the training and the test datasets. Assuming n observations in the dataset, this calculation is the

average squared deviation, as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}(x_i))^2.$$

3.5.1 Importance Sampling

Importance sampling is a practical tool to take samples from one distribution but in calculating a summary statistic (i.e. expectation of some function), weight the estimates in order to simulate sampling from an alternative distribution. For our purposes, we wish to make predictions under the alternative parsimony priors but we wish to do it by weighting the samples obtained from the posterior based on the effective domain prior.

In general, suppose we have sampled $z_1, \ldots, z_m$ from a distribution with density π_1 but we want to estimate a function $g(Z)$ under an alternative distribution with density π_2. It is well known that

$$E_{\pi_2} g(Z) = E_{\pi_1}\left[g(Z)\frac{\pi_2(Z)}{\pi_1(Z)}\right].$$

In our case, we wish to explore whether the extra senses of parsimony provide any benefit *over and above* the effective domain prior. For that reason, the ratio $(\frac{\pi_2}{\pi_1})$ and thus the weights are simply the penalty terms (2.4 and 2.5), for the orthogonality and additivity priors, respectively. That is $\pi_2 = \pi_1 \cdot f_{\text{orth}}$ for the orthogonality prior and $\pi_2 = \pi_1 \cdot f_{\text{add}}$ for the additivity prior.

Therefore, to make a prediction under the orthogonality prior, one weights the sample as follows

$$\hat{y}_{\text{orth}}(x^*) = \frac{\sum_{k=1}^{N} h_k \; \hat{y}^{(k)}(x^*)}{\sum_{k=1}^{N} h_k},$$

where

$$h_k = f_{\text{orth}}(w_1^{(k)}, \ldots, w_k^{(k)}).$$

A similar procedure is used for the linearity prior. An estimate of the MSE under these new priors can then be assessed.

Chapter 4

Empirical Results

This chapter discusses the empirical results we have found for the datasets mentioned in Chapter 1. First, I discuss the approach we took to monitoring the Markov Chains and I follow with a discussion of the goals of the empirical evaluation. Finally, a description of the results on each of the datasets is given.

4.1 Assessing Convergence

In my experience, assessing the convergence of the Markov Chains for Bayesian Neural Networks is very difficult. To illustrate this point, I present what I know now to be a typical plot of the sampled values of the network parameters. It is well known that these posterior distributions are multimodal and so textbook-style convergence is literally unheard of.

Figure 4.1 shows plots of the sampled network parameters for a neural network with five hidden nodes on the Ozone data set. The prior distribution used for the neural network in Figure 4.1 is the non-informative prior of Lee (2000a, 2000b). The panels are labelled by the parameter they describe. The reader should notice that because there are a large number of γ_{ij} parameters in the network fit, a summary measure (i.e. length) is used for monitoring purposes. Perhaps the most relevant and useful summary measure is that of the standard deviation σ. Despite many possible explanations of the data through the network parameters, one would hope that the estimate of the

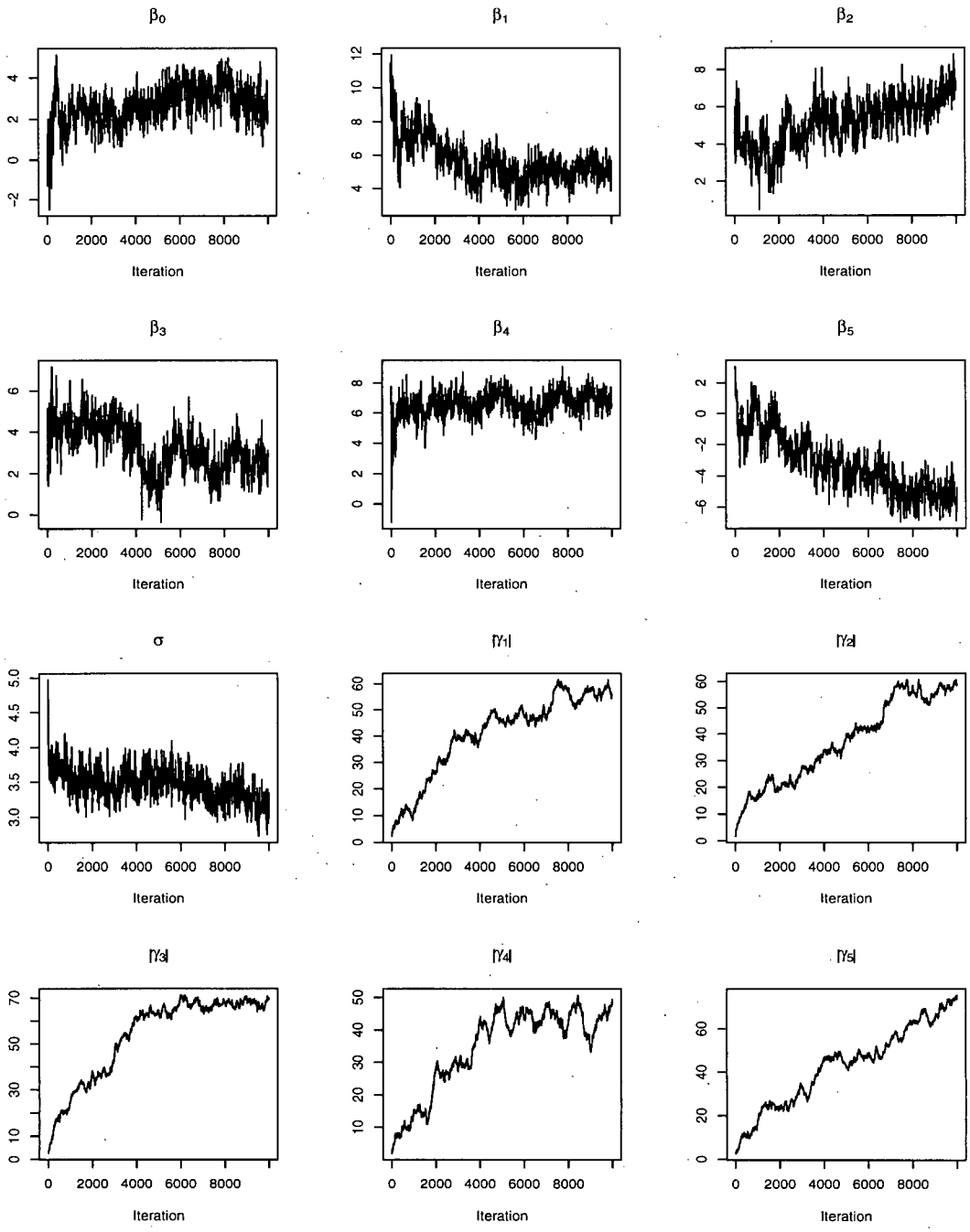


Figure 4.1: MCMC Plots for the Neural Network Parameters for 10000 iterations of a network with 5 hidden nodes on the Ozone dataset. Prior distribution from Lee (2000a, 2000b).

standard deviation remains stable.

Figure 4.2 shows a similar plot for a neural network with 5 nodes on the same data but using the *effective domain* prior discussed in this thesis. Although the scales are different in many occasions, a couple of comments are worth noting. Both networks are able to fit the data reasonably well, as suggested by the plots on σ. One potential disadvantage of the Lee prior is that under a prior with no weight decay, the size of each γ vector increase without bound. Even in longer runs of the chain (e.g. 50000), this gain in magnitude of each γ is still observed.

Therefore, for the monitoring of the Markov Chains, we simply use the MCMC plot of the standard deviation as our guide. To illustrate the stability of the neural network to the parameter σ, we made multiple independent runs of the chain and plot them together. Plots of σ for five independent fits of the network are shown in Figure 4.3 for the same dataset and prior as in Figure 4.2.

For the simulated sampling under the parsimony priors using importance sampling it is also important to monitor the importance weights to ensure that a small subset of the observations are not receiving the bulk of the weight and thus swaying the predictions accordingly. In general, the importance weights are well behaved. For example, Figure 4.4 shows importance weights for the penalty functions (2.4) and (2.5), corresponding to the orthogonality and additivity priors, respectively.

4.2 Goals

In the following sections of this chapter, I present a study of the performance of the methods suggested in the last chapter. The goals of the study are as follows:

- Demonstrate that the prior suggested in this thesis has the ability to capture the relationship adequately well.
- Examine the potential gains in using the parsimony priors.
- Evaluate the sensitivity of priors to the factor of z calculation.

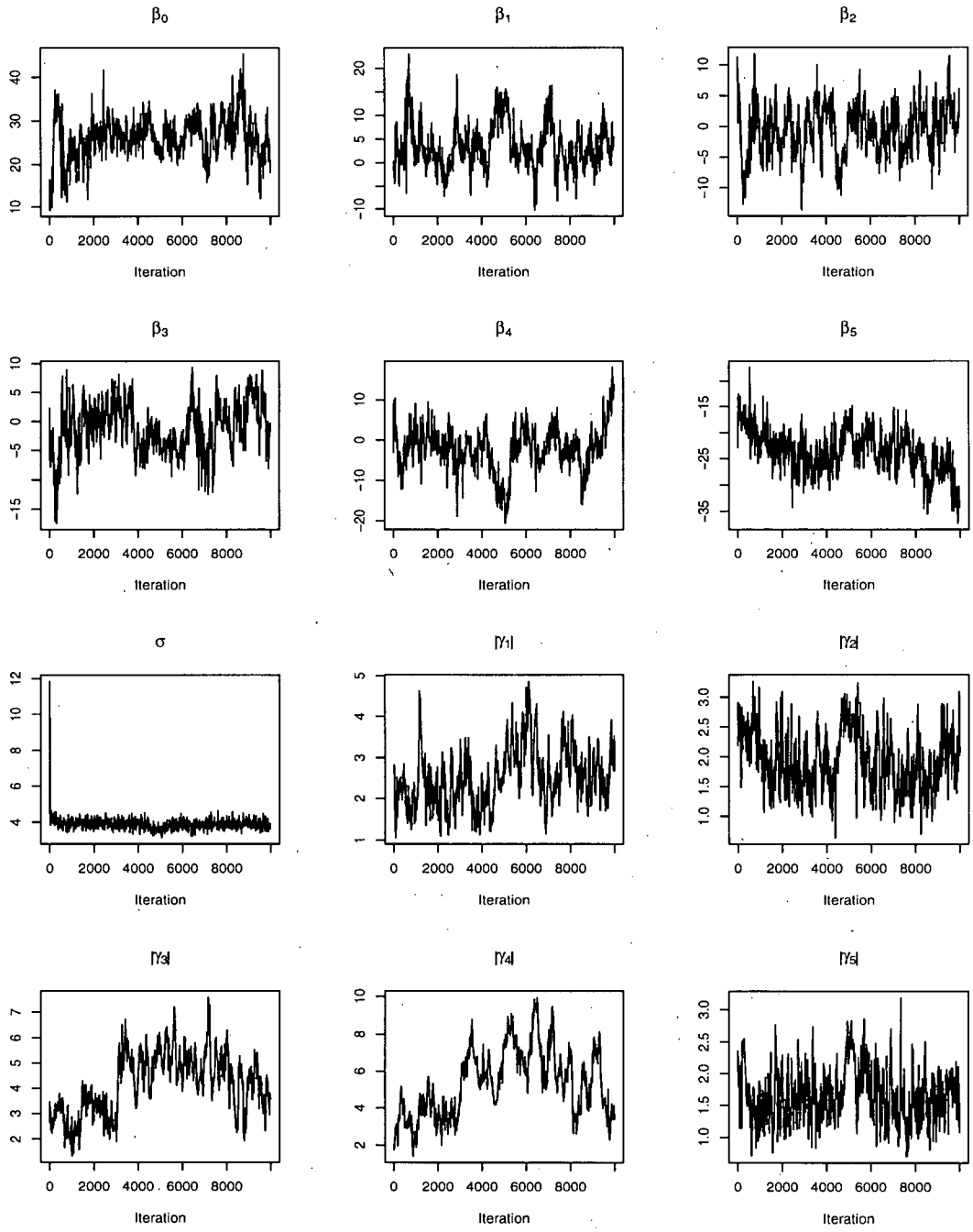


Figure 4.2: MCMC Plots for the Neural Network Parameters for 10000 iterations of a network with 5 hidden nodes on the Ozone dataset. Prior distribution is the *effective domain of interest.*

Figure 4.3: MCMC Plots of the σ Parameter as a diagnostic for convergence of the Markov Chain.

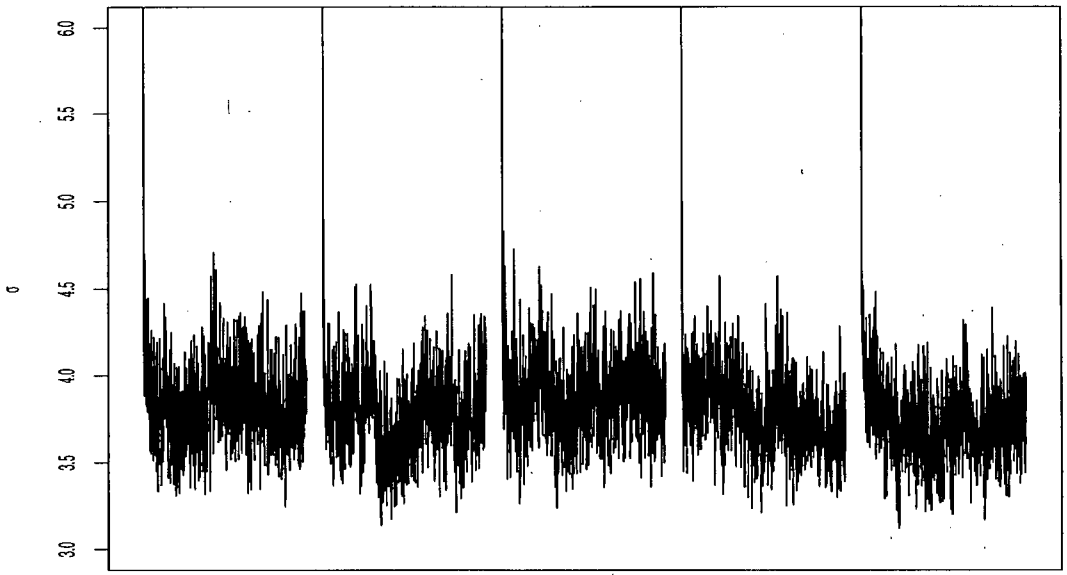
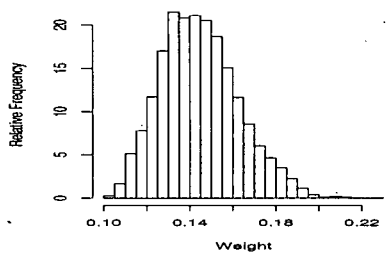
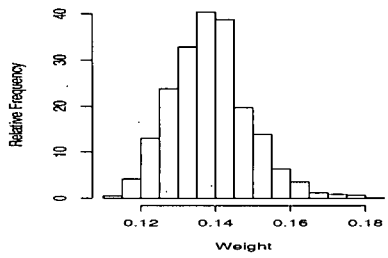


Figure 4.4: Importance Weights for the Parsimony Priors (Based on 10000 Samples).



(a) Orthogonality Penalty Function



(b) Additivity Penalty Function

Of course, there are many other factors which add extra variability to these assessments. For example, results may be highly dependent on the number of hidden nodes which are selected. Other sources of variability include the starting point of the chain or the split of the data which the network is fit and tested on.

4.3 Results on the Boston Housing Dataset

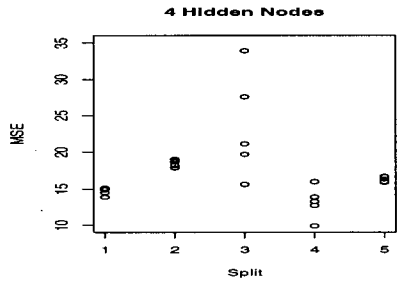
First of all, I present an illustration of the adequacy of the fit of a neural network under the *effective domain* prior for the Boston Housing dataset (A plot of the data was given in Figure 1.2).

For the analyses presented in this chapter, the networks were trained using a random split of the data of size 253, leaving another 253 observations in a validation dataset on which we present prediction results. From the monitoring of the σ parameter, as mentioned earlier, I chose a burnin of 20000 iterations which seems more than adequate. I arbitrarily chose a Monte Carlo sampling stage of 100000 iterations but thinning to take every 10-th sample. Predictions were made for the 253 observations in the validation dataset for each the 10000 samples and averaged to make a point estimate (i.e. posterior mean of the predictive distribution). MSEs were calculated as the average squared deviation between the estimated and observed response over the 253 observations.

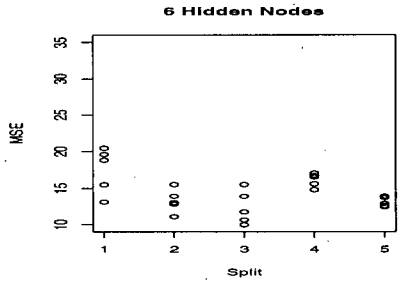
As suggested earlier, there are many potential sources of variability which may enter. For this reason, the analysis of the Boston housing data was repeated for: *i*) different random splits of the data; *ii*) runs of the Markov Chain from different starting points, and *iii*) on 4, 6, 8 and 10 hidden nodes.

For each number of hidden nodes, Figure 4.5 shows the results for five runs of the network fit on five randomly selected splits of the set. In most occasions, though not all, there exists more variation between different splits than between multiple runs on the same split. The straight line superimposed on each of these plots is the MSE presented in Neal (1996) for his neural network with eight hidden nodes and a Gaussian error distribution (MSE=13.7). In Neal's analysis, split of 253/253 was also used. It is clear from our analysis that there exists differences between the splits and it is very unlikely that I have chosen the same split as Neal. That said, our neural network is able to perform adequately well in terms of prediction.

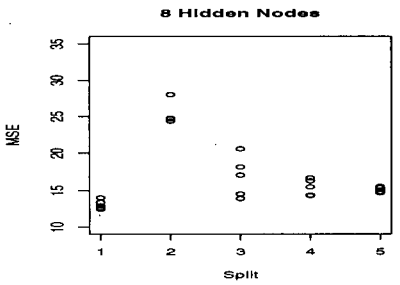
Figure 4.5: MSEs on the Validation Dataset (Boston Housing Data). Five runs from different initial values on five different splits of the data.



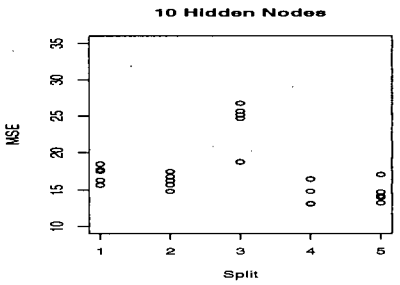
(a) 4 Hidden Nodes



(b) 6 Hidden Nodes

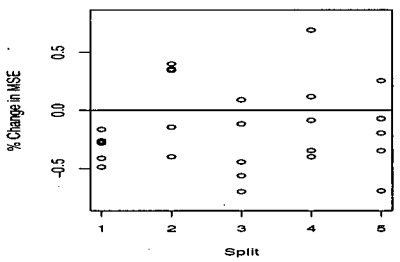


(c) 8 Hidden Nodes

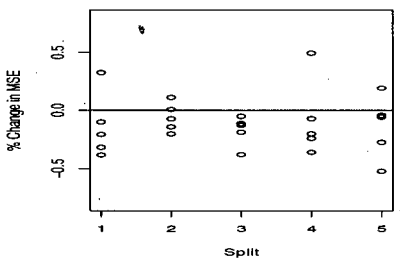


(d) 10 Hidden Nodes

Figure 4.6: Improvements (% Change in MSE) under Parsimony Priors (Boston Housing Data). Five runs from different initial values on five different splits of the data.



(a) 6 Nodes, Orthogonality Prior



(b) 6 Nodes, Additivity Prior

To evaluate the profit of the parsimony priors discussed in Chapter 2, we take the same approach of trying many runs of the chain on many splits. Figure 4.6 shows gains in MSE on the percentage scale (i.e. $\frac{MSE(\text{orthog. or addit.}) - MSE(\text{e.d.})}{MSE(\text{e.d.})}$) for a network with 6 hidden nodes. Here, points less than the superimposed line at zero represent a gain in performance. For completeness, the corresponding plots for 4, 8 and 10 hidden nodes are given in Appendix B.

It seems that there are gains, though modest, to be made by using the additivity and orthogonality priors. For 6 hidden nodes, average gains of 0.15% and 0.12% for the orthogonality and additivity priors were recorded, respectively. The plots in Figure 4.6 as well as those in the Appendix also suggest that when gains are made in one prior, they are also observed in the other.

For the Boston dataset, the selection of the hyperparameters for the prior used a *factor of* z calculation, as suggested in Section 2.5.3, with $z = 2$. In the next sections, we explore further the sensitivity of this choice.

4.4 Results on the Tecator Dataset

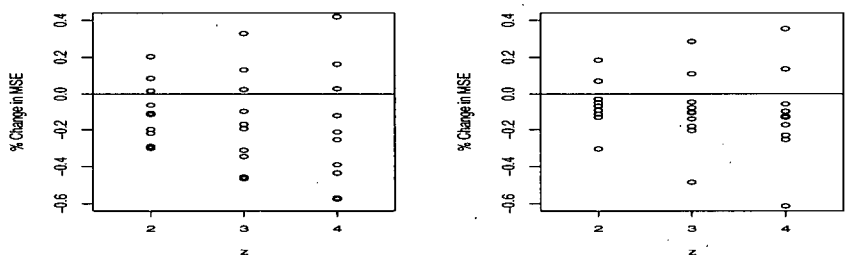
In this section, we discuss the results of our prior on the Tecator dataset (see Figure 1.3), previously analyzed with a neural network by Thodberg (1996).

From the previous analyses on the Tecator dataset, the training and validation set had already been chosen. Hence, there was no need to make random splits of the data. The predictive ability of our neural networks for this dataset was not as good as those presented in Thodberg (1996). For most of their networks, a model selection procedure was used to the choose a best model and that may explain the better performance. For example, he used a generalized prediction error procedure, derived from Akaike's Information Criterion or Mackay's evidence (Mackay 1992). Here, I focus on the potential gains by using the priors which encourage parsimony.

We wished to explore further the sensitivity of the predictive performance of the neural networks to the factor of z calculation. For our analysis, we chose z=2, 3 or 4.

Figure 4.7 shows 10 fits of the network with 4 hidden nodes on the given training set from different starting points at each of the three values of z suggested above. Not surprisingly, we observe that the gains (or losses) are exaggerated for large values of z. That is, if there are gains

Figure 4.7: Improvements (% Change in MSE) under Parsimony Priors (Tecator Data). Ten runs from different initial values.



(a) 4 Nodes, Orthogonality Prior

(b) 4 Nodes, Additivity Prior

to be made at $z = 2$, then those gains are accentuated with $z = 3$ or 4. One would expect, though, that there will be a point where a higher penalty will provide no further advantage.

As was done before, the corresponding graphs for 6, 8 and 10 hidden nodes are left until Appendix B.

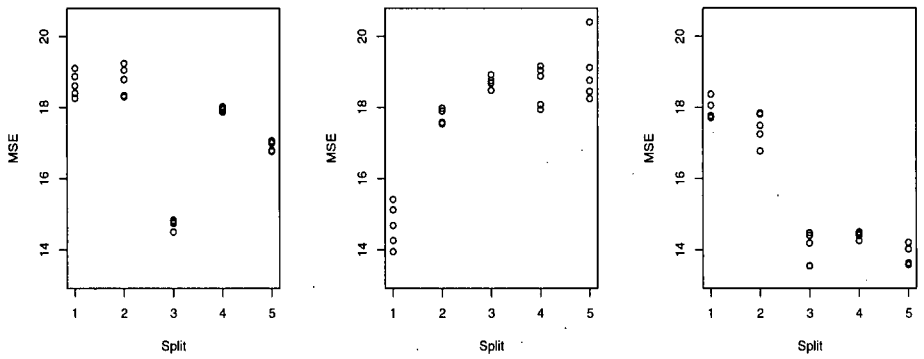
4.5 Results on the Ozone Dataset

In this section, I describe the neural networks fit on the Ozone dataset (see Figure 1.2).

Again, multiple splits of the dataset into training and validation subsets were done. Following Lee (2000a), I used a training set of size 200, leaving 130 observations in the validation set.

Figure 4.8 shows the MSEs on the 5 different fits of the network at each of 5 different random splits of the data for networks with 4, 6 and 8 hidden nodes. One may note the variation *between* different splits of the data is considerably higher than that *within* multiple runs of a split. The previous literature has compared different methods of analyzing these data by the multiple correlation coefficient, R^2. For example, a generalized additive model was able to achieve $R^2 = 0.80$ and a Lee neural network attained $R^2 = 0.79$. Again, our method does not perform quite as well ($R^2 = 0.76$) as any of those mentioned in Lee (2000a), but here we have not considered any model

Figure 4.8: MSEs on the Validation Dataset (Ozone Data). Five runs from different initial values on five different splits of the data.

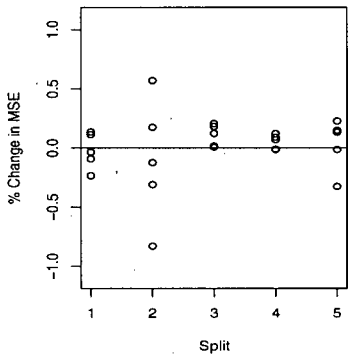


(a) 4 Hidden Nodes (b) 6 Hidden Nodes (c) 8 Hidden Nodes

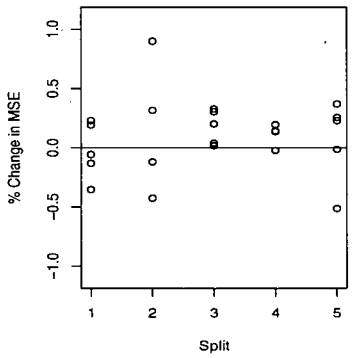
selection.

As in the previous sections, we wish to determine whether there exists any benefit in using the parsimony priors and also to observe how sensitive the analysis is to the calibration parameter, z. Figure 4.9 shows the gains for both parsimony priors on networks with 8 hidden nodes. Shown there are multiple runs from different splits of the data for $z = 2, 3$ and 4. The same observations can be made for the ozone dataset as for the previous datasets. That is, there are some gains to be made, but in general they are small. In most cases, the parsimony prior does little or no harm. Also, as was noticed earlier, increasing z has an effect of exaggerating the gains.

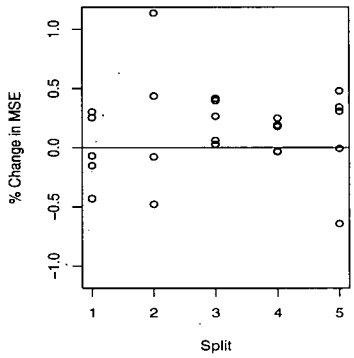
Figure 4.9: Improvements (% Change in MSE) under Parsimony Priors (Ozone Data). Five runs from different initial values on five different splits of the data at $z = 2, 3, 4$.



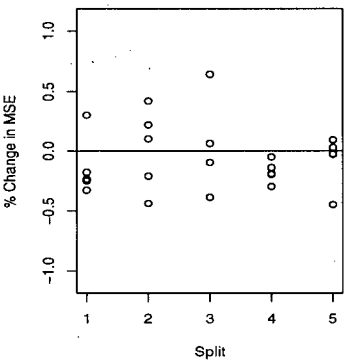
(a) Additivity Prior, z=2



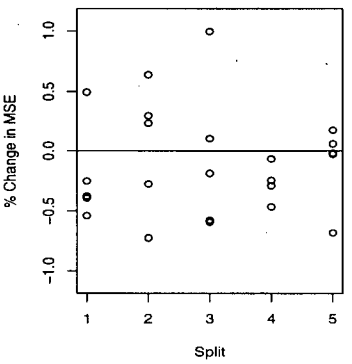
(b) Additivity Prior, z=3



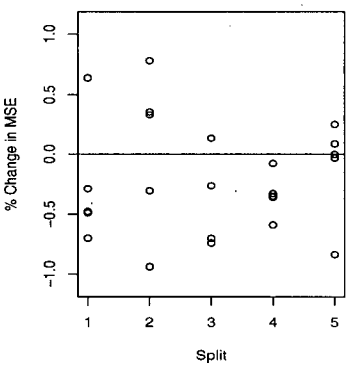
(c) Additivity Prior, z=4



(d) Orthogonality Prior, z=2



(e) Orthogonality Prior, z=3



(f) Orthogonality Prior, z=4

Chapter 5

Discussions and Further Work

Neural Network models for regression are extremely useful methods to have in a statistician's toolkit. Applying these methods from the Bayesian point of view, as was done here, provides the data analyst with a flexible and robust approach. Although we limited ourselves in this thesis to a univariate continuous response, the methods mentioned here easily generalize to a multivariate continuous response, a binary response or a categorical response.

In this thesis, we have discussed a novel method for specifying the prior distribution over the parameters for a neural network with logistic basis units. Briefly, our approach looks at the geometry of the logistic basis functions and specifies a prior which, on one end, applies weight decay, and on the other end, reduces redundancy by probabilistically eliminating parameters values which essentially overparameterize the model. This is accomplished by stochastically restricting the parameters to an *effective domain of interest*. Moreover, we introduce priors which encourage notions of parsimony, such as orthogonality of the input-to-hidden weights or additivity of the hidden nodes. These approaches were developed in Chapter 2.

Following the lead of Rios Insua and Müller (1998) and Lee (2000a, 2000b), methods for sampling the posterior distribution via MCMC are discussed in Chapter 3. Also mentioned is the method of importance sampling, which is used to assess the worth of the parsimony priors.

Chapter 4 describes and discusses a comprehensive comparison study of the priors discussed in Chapter 2. The study covered many extra sources of variability such as: the split of the data,

multiple fits from different starting points and the sensitivity of calibrating the prior. The study showed that the effective domain prior was able to capture the relationship with similar performance as the previous methods suggested by Neal, Rios Insua and Müller and Lee. An examination of the parsimony priors was conducted at the same time. For each of the three datasets, the orthogonality and additivity afforded only a small influence. In some cases, it appears to have shown a small gain in predictive performance and in others it showed little or no gain. There were no cases where the parsimony priors had an adverse effect on predictive performance.

In summary, the effective domain prior provides a valid and useful specification of the prior distribution over neural network parameters incorporating both weight decay and removing redundancy. The parsimony priors do not seem to provide much in the way of predictive performance, however, they also do little harm. Perhaps other senses of parsimony could prove useful.

Future Directions: Model Selection. Briefly mentioned in the literature review in Chapter 1, model selection is an important part of effectively using a neural network model for prediction. Model selection encompasses both the selection of an adequate network architecture (i.e. number of hidden nodes) and a selection of a reasonable subset of the explanatory variables.

In this thesis, one objective was to specify a prior distribution which indirectly encourages parsimonious models, and hopefully sidesteps some of the questions of model selection. However, the notions of additivity of the hidden nodes and orthogonality of the input-to-hidden weights were ineffective in providing major gains in predictive performance.

A recent development in the area of Bayesian model selection is the work of Spiegelhalter, Best and Carlin (1998). Since many model selection criteria (e.g. Akaike's Information Criterion (AIC), Bayesian Information Criterion, Minimum Description Length) are calculations based on the likelihood evaluated at the maximum likelihood estimator, it is unclear how to do the corresponding calculation in a Bayesian setting.

Spiegelhalter, Best and Carlin (1998) present an alternative strategy based on MCMC sampling output. They propose the *Deviance Information Criterion* (DIC), which is based on the

posterior distribution of the deviance statistic

$$D(\theta) = -2\log p(y|\theta) + 2\log f(y).$$

The second term assumes $f(y)$ to be a perfect predictor (saturated model) which gives probability 1 to each observation and therefore has no effect on model selection. Typically, the deviance is used to summarize the *goodness-of-fit* of the model. In the Bayesian setting, one can summarize the fit of the model by the posterior mean of the deviance. For neural networks for regression, $\bar{D}$ reduces to

$$\bar{D} = E_{\theta|y}\{D(\theta)\} \approx \sum_{i=1}^{M} \frac{(y - Z\beta)_{(i)}^T (y - Z\beta)_{(i)}}{\sigma_{(i)}^2}.$$

where M is the Monte Carlo sample size and $\theta|y$ denotes the posterior distribution of the parameters. Based on an analogy with AIC, the *effective number of parameters*, p_D is defined by

$$p_D = \bar{D} - D(E_{\theta|y}\{\theta\}) = \bar{D} - D(\bar{\theta}).$$

This has shown to be asymptotically non-negative (Spiegelhalter et al., 1998) As with other model selection criterion, the DIC is combined of a goodness-of-fit term and a penalty for complexity; it is defined as

$$\text{DIC} = \bar{D} + p_D.$$

DIC mimics the form of the AIC. The AIC adds a penalty term (for the number of parameters) to the deviance evaluated at the maximum likelihood estimator. Similar to other measures (e.g. AIC), smaller values of the DIC are preferred.

Spiegelhalter et al. (1998) make the argument that the DIC is a natural generalization of the AIC. However, they give the following warning for the use of the DIC: "We need to emphasise that we do not recommend that DIC be used as a strict criterion for model *choice* or as a basis for model averaging." DiCiccio et al. (1997) discuss many methods of computing Bayes factors, a problem that parallels model selection. They mention that their methods are effective in situations where the posterior distribution is well behaved, a situation which is not well suited to neural networks.

Since the DIC discussed above is an essentially a "for free" calculation once the Monte Carlo sample has been taken, we briefly explore its worth in the comparison of competing neural network

models. Our key consideration is the dependability of the estimate. Our knowledge of monitoring MCMC plots for neural networks would suggest that $\bar{D}$ will be stable, because it is composed of SSE, as well as the sample variance, σ^2. However, the deviance evaluated at the posterior mean of the parameters may be much less sound because of the multimodality. The following table shows a summary of the components of the DIC measure for repeated runs (on the same split) of a neural network on 4, 6 and 8 hidden nodes (on the Boston dataset).

Nodes	$\bar{D}$	$D(\bar{\theta})$	p_D	DIC
4	1684.183	1641.259	42.924	1727.107
4	1721.168	1657.078	64.09	1785.258
4	1668.159	1637.941	30.218	1698.377
4	1573.388	1531.038	42.35	1615.738
4	1729.801	1657.674	72.127	1801.928
6	1706.928	1678.629	28.299	1735.227
6	1992.409	1909.299	83.11	2075.519
6	1958.468	1908.796	49.672	2008.14
6	1882.927	1760.677	122.25	2005.177
6	1718.773	1640.565	78.208	1796.981
8	1886.049	1776.907	109.142	1995.191
8	2220.292	2135.306	84.986	2305.278
8	1873.486	1845.666	27.82	1901.306
8	1787.058	1838.826	-51.768	1735.29
8	2471.422	2542.731	-71.309	2400.113

As expected, one can see that the DIC criterion is quite variable even with respect to the starting value of the chain. Furthermore, variablility of the DIC from run to run increases as the number of nodes increases. It would seem as though the neural network with 4 hidden nodes provides a better model in this case. Figure 5.1 shows plots of DIC repeated for multiple splits. Again, one sees that the neural network with 4 hidden nodes provides the "better" model and it is clear that the DIC is much less stable for the model with 8 hidden nodes.

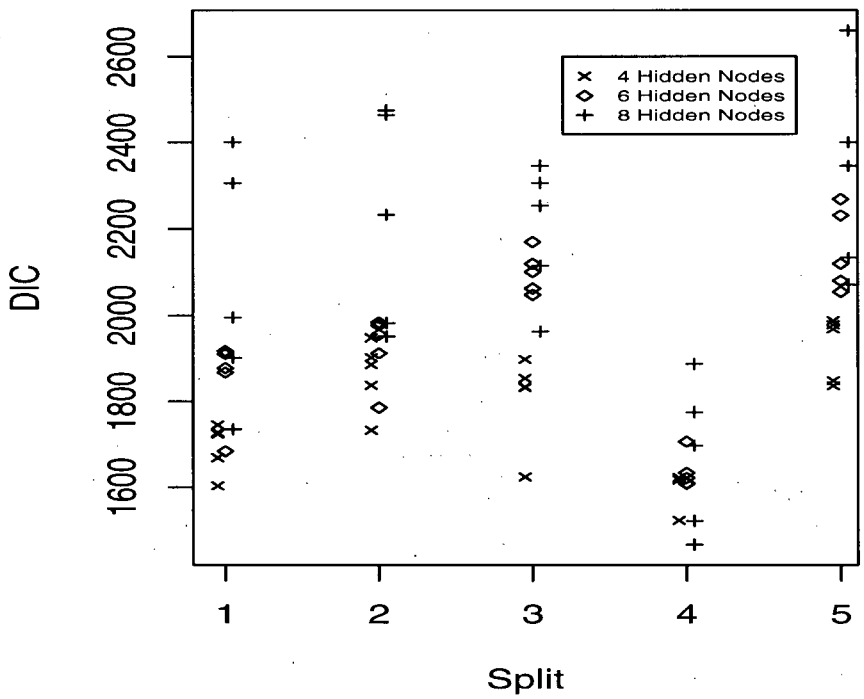


Figure 5.1: Plots of DIC for Neural Network Models with 4, 6 and 8 Hidden Nodes for Multiple Runs and Splits.

One alternative to model selection is that of Markov Chain Monte Carlo Model Composition (MC^3) (Raftery et al. 1997) whereby MCMC is performed on the model space. They apply this method in the case of linear regression models. Further work is required in the current domain to determine the feasibility for neural network models.

Appendix A: Implementation

The implementation of the neural networks combined the use of both programs written in C and R (see Ihaka and Gentleman 1996).

Taking advantage of the speed of a programming environment such as C, the MCMC sampling steps and the prediction calculations were performed in C.

These routines were called within R using the `.C` function. The outputs and results were plotted within R. Because of the similarity of R and S-plus, a similar implementation could be performed in S-plus.

Pages 52 to 54 show the C programs.

Pages 55 to 56 show the R programs.

```
#include "nn.h"

// for use with R:  gcc -fpic -c main.c sampling.c dist.c misc.c mse.c
//                  gcc -shared -o nn.linux.so main.o sampling.o dist.o misc.o mse.o

extern int acceptance[MAXV];
int violate1[MAXV], violate2[MAXV];

void nn(double *vBeta, double *vGamma, double *vSigma, int *M, int *N,
        int *p, int *burnin, int *thin, int *draws, double *vInputs,
        double *vTargets, double *c) {

  int i,j,k,ind1=0,ind2=0,ind3=0;
  double inputs[MAXN][MAXV], targets[MAXN], tmp, det;
  double ztemp[MAXN][MAXV], ztztemp[MAXV][MAXV], ztzinvtemp[MAXV][MAXV];
  double corrX[MAXV][MAXV];
  struct postSamp aSamp;
  time_t begin, middle, end;

  srandom(time(NULL));

  begin = time(NULL);

  // get info from R/Splus into C
  for(j=1; j<=*p; j++)
    for(i=1; i<=*N; i++) { inputs[i][j] = vInputs[ind1++]; }
  for(i=1; i<=*N; i++) {
    inputs[i][0] = 1.;
    targets[i] = vTargets[ind2++];
  }

  ind1=0; ind2=0;
  aSamp.M = *M; aSamp.p = *p; aSamp.N = *N;

  for(i=1; i<=*p; i++) for(j=1; j<=*p; j++) corrX[i][j] = -.01;
  setGlobals(*c);
  for(j=1; j<=*M+1; j++)
    aSamp.beta[j]=vBeta[ind1++];
  for(j=1; j<=*p+1; j++)
    for(k=1; k<=*M; k++)
      aSamp.gamma[j][k]=vGamma[ind2++];

  calcCorrMat(inputs,corrX,aSamp);

  aSamp.V = vSigma[ind3++];

  for(i=1; i<=*burnin; i++) {
    if( (i % 1000) == 0 ) {
        printf("%9d iterations completed.\n",i);
        for(j=1; j<=aSamp.M; j++) {
          violate1[j]=0; violate2[j]=0;
        }
    }
    aSamp = sampleAway(aSamp, inputs, targets, corrX);
  }
  middle = time(NULL);
  for(i=1; i<= (*draws * *thin); i++) {
    if( (i % 1000) == 0 ) {
        printf("%9d iterations completed.\n",i);
        for(j=1; j<=aSamp.M; j++) {
          violate1[j]=0; violate2[j]=0;
        }
    }
    aSamp = sampleAway(aSamp, inputs, targets, corrX);
    // dump the samples to the _vector_ for use in R or Splus
    if( (i % *thin) == 0 ) {
      for(j=1; j<=*M+1; j++) {   vBeta[ind1++]=aSamp.beta[j];   }
      for(j=1; j<=*p+1; j++) { for(k=1; k<=*M; k++) {
        vGamma[ind2++]=aSamp.gamma[j][k];}
      }
      vSigma[ind3++] = aSamp.V;
    }
  }
  end = time(NULL);
  printf("\n Acceptance Rates: ");
  for(i=1; i<=*M; i++) {
    tmp = (double) acceptance[i] / (double) (*burnin + *draws * *thin);
    printf(" %6.4f ", tmp );
  }
  printf("\n Times to Run: ");
  printf("burnin-%4.1f min.  draws-%4.1f min.", difftime(middle,begin)/60.,
                                                difftime(end,middle)/60.);
  printf("\n");

}
```

```
#include "nn.h"

// global variables
extern double ab, Ab, cb, cg, s, S, Cb, ag[MAXV], probeC;
extern double Cg[MAXV][MAXV], Ag[MAXV][MAXV];
extern int violate1[MAXV], violate2[MAXV];

struct postSamp sampleAway(struct postSamp samp, double x[][MAXV],
                           double y[], double corrX[][MAXV]) {

  double curGamma[MAXV][MAXV], gamma_tilda[MAXV], gammaJ[MAXV], probeV[MAXV][MAXV];
  double Zb[MAXN], z[MAXN][MAXV], sse=0., glen=0., tau=sqrt(5.), accept;
  double bv[MAXV][MAXV], zy[MAXV],bm[MAXV],ztz[MAXV][MAXV],ztzinv[MAXV][MAXV];
  double Z_tilda_b[MAXN], z_tilda[MAXN][MAXV], sse_tilda, glen_tilda, tot;
  double lp, lp_tilda;
  int i,j,k, M=samp.M,p=samp.p;

  // sample the variance (IG)
  calcZ(samp.gamma,x,z,ztz,samp,YES);
  multmv(z,samp.beta,samp.N,samp.M+1,Zb);
  for(i=1; i<=samp.N; i++) { sse += pow(y[i]-Zb[i],2); }
  samp.V = 1./rgamma( (double)samp.N / 2., 2./sse );

  // sample the beta (MVN)
  invertm(ztz,ztzinv,samp.M+1);
  for(i=1; i<=samp.M+1; i++)
    for(j=1; j<=i; j++) {
      bv[i][j] = samp.V*ztzinv[i][j];
      bv[j][i] = samp.V*ztzinv[i][j];
    }
  for(i=1; i<=samp.M+1; i++) {
    tot = 0.;
    for(j=1; j<=samp.N; j++) { tot += z[j][i]*y[j]; }
    zy[i] = tot;
  }
  multmv(ztzinv, zy, samp.M+1, samp.M+1, bm);  // bm = mean
  rmvn(bm,bv,samp.M+1,samp.beta);

  //  set probing var-cov matrix
  for(i=1; i<=p+1; i++)
    for(j=1; j<=p+1; j++)
      probeV[i][j] = pow(probeC,2)* Cg[i][j];

  // sample the gammas
  for(i=1; i<=M; i++) {

    // curGamma will get the proposal
    for(j=1; j<=p+1; j++)
      for(k=1; k<=M; k++)
        curGamma[j][k] = samp.gamma[j][k];
    // add small pertubation to current Gamma
    for(j=1; j<=p+1; j++) gammaJ[j] = curGamma[j][i];
    rmvn(gammaJ,probeV,p+1,gamma_tilda);
    for(j=1; j<=p+1; j++) curGamma[j][i] = gamma_tilda[j];

    calcZ(curGamma,x,z_tilda,ztz,samp,NO);
    multmv(z_tilda,samp.beta,samp.N,samp.M+1,Z_tilda_b);
    sse_tilda = 0.;  glen_tilda=0.;
    for(k=1; k<=samp.N; k++) sse_tilda += pow(y[k]-Z_tilda_b[k],2);
    lp_tilda = calcLogPrior( gamma_tilda, corrX, samp);

    calcZ(samp.gamma,x,z,ztz,samp,NO);
    multmv(z,samp.beta,samp.N,samp.M+1,Zb);
    sse = 0.;  glen=0.;
    for(k=1; k<=samp.N; k++) sse += pow(y[k]-Zb[k],2);
    lp = calcLogPrior( gammaJ, corrX, samp);

    accept = exp( 0.5*(sse-sse_tilda)/samp.V + (lp_tilda-lp) );

    if ( runif() < accept ) {
      for(j=1; j<=p+1; j++) samp.gamma[j][i]=gamma_tilda[j];
      acceptance[i]++;
    }

  }
  return(samp);
}

void calcZ (double gamma[][MAXV], double x[][MAXV], double z[][MAXV],
            double ztz[][MAXV], struct postSamp samp, int calcTrans) {
  int i,j,k;
  double tot=0.;

  // calculate Z matrix
  for(i=1; i<=samp.N; i++)
    for(j=1; j<=samp.M+1; j++) {
      tot = 0.;
      if (j==1) {
        z[i][j]=1.;
      } else {
        for(k=1; k<=samp.p+1; k++)
          tot+= gamma[k][j-1]*x[i][k-1];
          z[i][j] = 1./(1.+exp(-tot));
      }
    }
  // calculate Z'Z matrix
  if (calcTrans) {
    for(i=1; i<=samp.M+1; i++)
      for(j=i; j<=samp.M+1; j++) {
        tot=0.;
        for(k=1; k<=samp.N; k++)
          tot += z[k][j]*z[k][i];
        ztz[i][j] = tot;
        ztz[j][i] = tot;
      }
  }
}

void calcCorrMat( double x[][MAXV], double V[][MAXV], struct postSamp samp ) {
  int i,j,k;
  double tot;

  for(i=1; i<=samp.p; i++) {
    for(j=1; j<=samp.p; j++) {
      tot=0.;
      for(k=1; k<=samp.N; k++)
        tot += x[k][i]*x[k][j];
      V[i][j] = tot / (double) (samp.N-1);
    }
  }
}
```

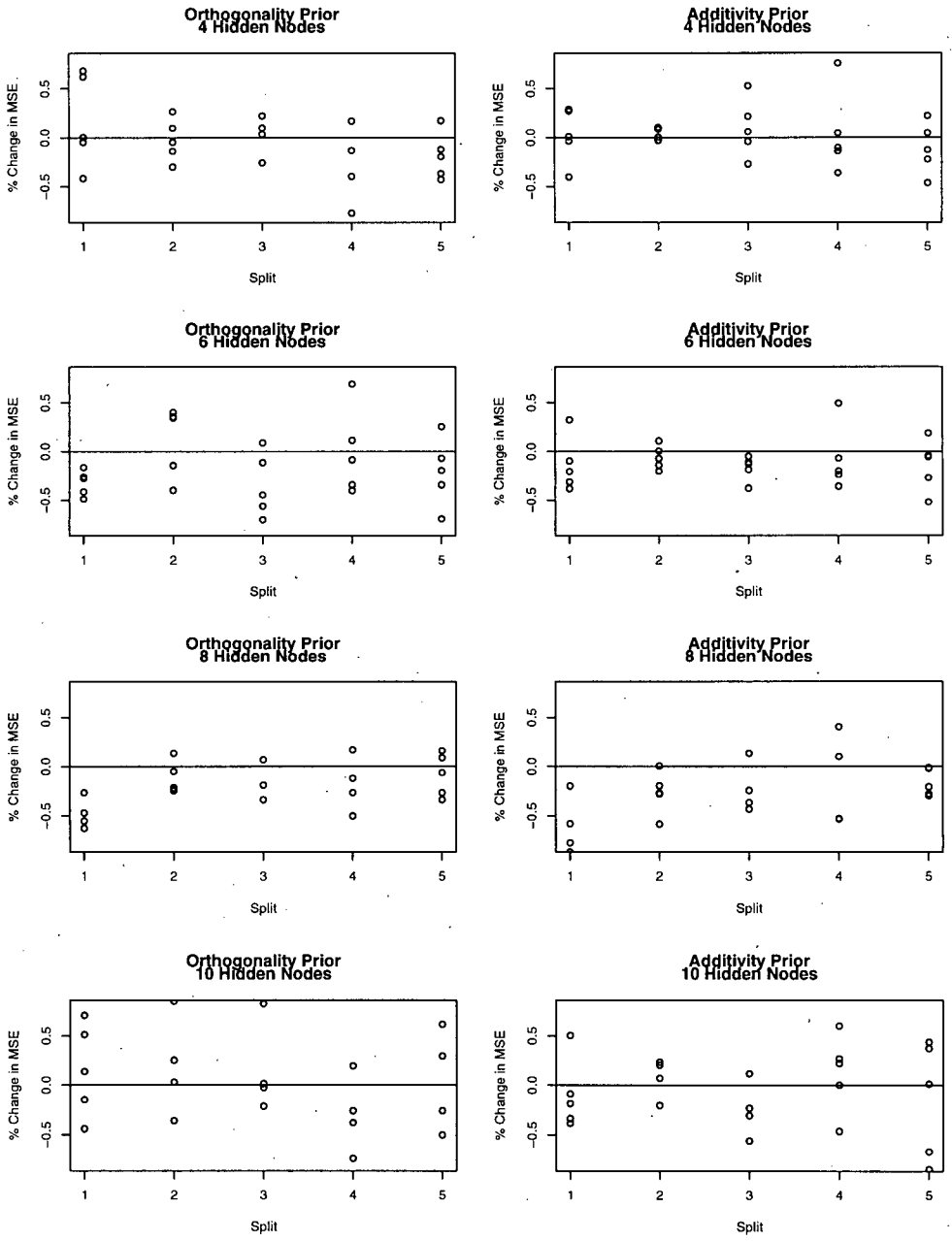
```
#include "nn.h"

void myest(double *vBeta, double *vGamma, int *M, int *N, int *p,
           double *vInputs, double *vTargets, double *estim, double *mse,
           double *D, double *v) {

  int i,j,k, ind=0;
  double inputs[MAXN][MAXV], targs[MAXN], myVect[MAXN], tot1, tot2;
  double z[MAXV][MAXV], ztz[MAXV][MAXV], Zb[MAXN];
  double curBeta[MAXV], curGamma[MAXV][MAXV], ests[MAXN], sse=0.;
  time_t begin, middle, end;
  struct postSamp samp;

  samp.N = *N; samp.M = *M; samp.p = *p; samp.V = *v;

  // get info from R/Splus into C
  for(i=1; i<=samp.N; i++) {
    inputs[i][0] = 1.;
    targs[i] = vTargets[i-1];
  }
  for(j=1; j<=*p; j++) for(i=1; i<=samp.N; i++) inputs[i][j] = vInputs[ind++];
  // get current beta,gamma
  for(j=0; j<=*M; j++) curBeta[j+1] = vBeta[j];
  for(j=1; j<=*p+1; j++) for(k=1; k<=*M; k++) curGamma[j][k] = vGamma[(j-1)*(*M)+k-1

  calcZ(curGamma,inputs,z,ztz,samp,NO);
  multmv(z,curBeta,samp.N,samp.M+1,Zb);

  for(i=1; i<=samp.N; i++) {
    sse += pow(vTargets[i-1]-Zb[i],2.);
    estim[i-1] = Zb[i];
  }
  *mse = sse / (double) (samp.N);
  *D = (double) samp.N * log( 2 * 3.14159265359 * samp.V ) + sse / samp.V;


}
```

```
nn<-
function(burnin = 1000, draws=100, thin=10, M = 4, data, c = 0.05)
{
        p <- ncol(data$x)
        N <- nrow(data$x)
        vBeta <- rep(-95., (M+1) * (draws+1))
        vBeta[1:(M+1)]<-rnorm(M+1)
        print(p); print(M); print(draws)
        vGamma <- rep(-95., (p + 1) * (M) * (draws+1))
        vGamma[1:(M*(p+1))]<-rnorm(M*(p+1))
        xx<-matrix(vGamma[1:(M*(p+1))],p+1,M,byrow=T)
        vSigma <- rep(-95., draws+1)
        vSigma[1]<-1.0
        z <- .C("nn",
                vBeta = as.double(vBeta),
                vGamma = as.double(vGamma),
                vSigma = as.double(vSigma),
                M = as.integer(M),
                as.integer(N),
                as.integer(p),
                as.integer(burnin),
                as.integer(thin),
                as.integer(draws),
                as.double(data$x),
                as.double(data$y),
                as.double(c))
        return(list(M = M, N = N, p = p,
                    beta = t(matrix(z$vBeta[(M+2):((M+1)*(draws+1))],
                                    nrow = M+1)),
                    gamma = t(matrix(z$vGamma[(M*(p+1)+1):((p+1)*(M)*(draws+1))],
                                    nrow = M * (p + 1))),
                        s = z$vSigma[2:(draws+1)])      )
}
```

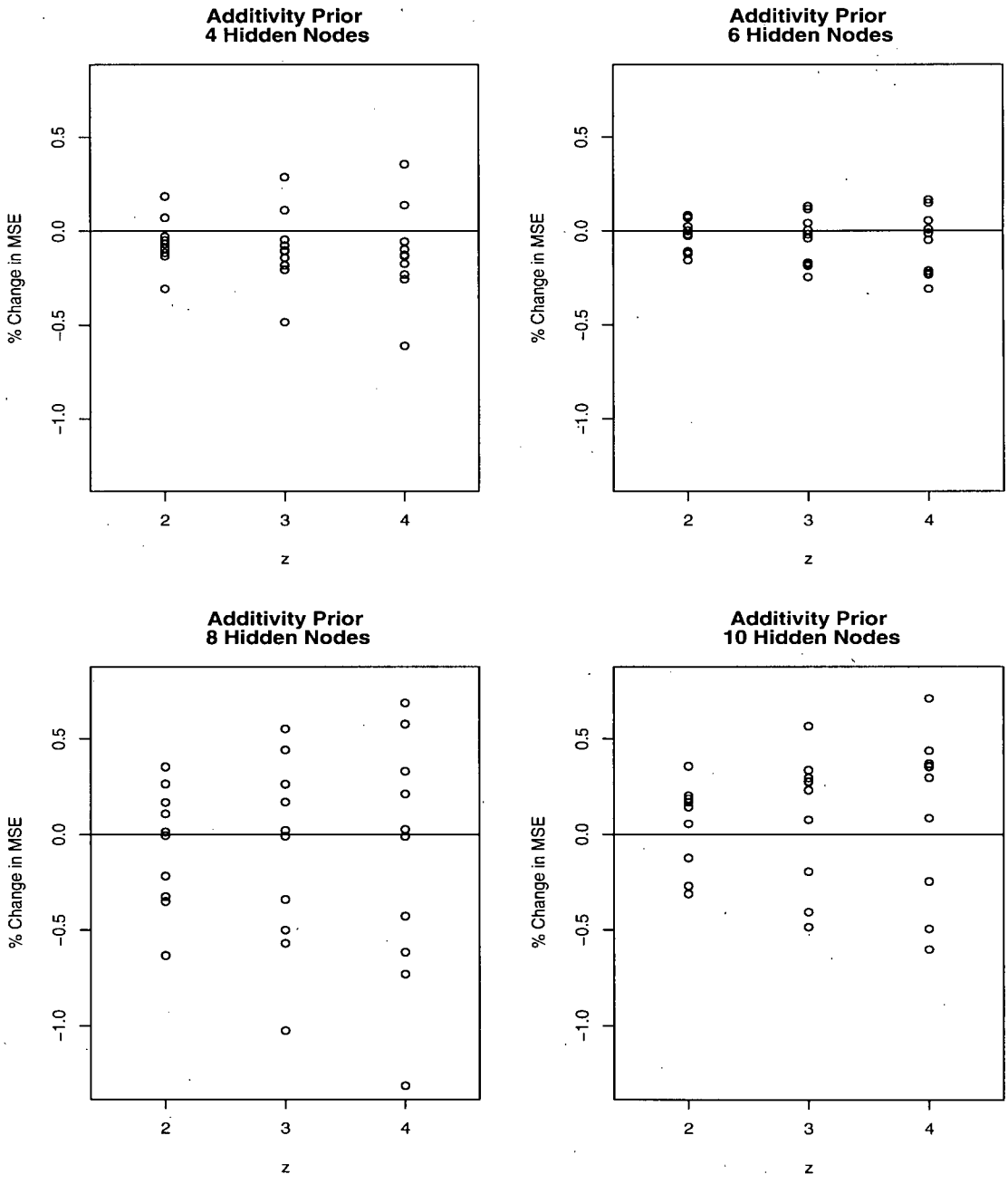
```
calc.mse1<-
function(data.fit=stop("No dataset specified"), data.valid,
         chain=stop("No MC specified"),a=6,factor.z=2)
{
        n.f <- nrow(data.fit$x)
        if(!missing(data.valid)) { n.v <- nrow(data.valid$x) }
        else { n.v <- 1 }
        b<- c(1,sqrt(chain$p)-1) / ( factor.z^(2/(a+chain$p))-1 )
        print(round(b,5))
        draws<-dim(chain$beta)[1]
        sqerrors<-rep(NA,draws)
        sqerrors.valid<-rep(NA,draws)
        wgt1<-rep(NA,draws)
        wgt2<-rep(NA,draws)
        myD<-rep(NA,draws)
        myD.valid<-rep(NA,draws)
        unitv <-matrix( rep(NA, chain$M*chain$p), nrow=chain$p )
        estimates.f <- matrix(rep(NA, n.f * draws), nrow = n.f)
        estimates.v <- matrix(rep(NA, n.v * draws), nrow = n.v)
        #calculate estimates from posterior samples
        #-----------+------------------------------
        for(j in 1:draws) {
                draw.myest <- myest.c(data.fit, chain, iter=j)
                estimates.f[, j] <- draw.myest$y
                sqerrors[j]<- draw.myest$mse
                myD[j]<- draw.myest$D
                if(!missing(data.valid)) {
                        draw.valid <- myest.c(data.valid, chain, iter=j)
                        estimates.v[, j] <- draw.valid$y
                        sqerrors.valid[j]<- draw.valid$mse
                        myD.valid[j]<- draw.valid$D
                }
                #calculate weights for importance sampling
                #-----------------------------------------
                cur.gamma <- R.get.gamma(chain,j)[-1,]
                lengths<-apply(cur.gamma,2,sumsq)
                #wgt1[j]<-prod((1+lengths/b[1])^( -(a+chain$p)/2 ))
                for(i in 1:chain$M) { unitv[,i]<-cur.gamma[,i]/sqrt(lengths[i])}
                sum.dots<-0
                for(i in 1:(chain$M-1)) {
                  for(k in (i+1):chain$M) {
                    sum.dots<-sum.dots+sum( abs( unitv[,i]*unitv[,k] ) )
                } }
                wgt1[j]<-prod((1+(2/(chain$M*(chain$M-1))*sum.dots)/b[1])^
                              ( -chain$M*(a+chain$p)/2 ))
                sums<-apply(unitv,2,sumabs)-1
                wgt2[j]<-prod((1+sums/b[2])^( -(a+chain$p)/2 ))
        }
        #print out MSE
        #-------------
        pt.estimate.f.0 <- apply(estimates.f, 1, mean)
        pt.estimate.f.1 <- apply(estimates.f, 1, weighted.mean, w=wgt1)
        pt.estimate.f.2 <- apply(estimates.f, 1, weighted.mean, w=wgt2)
        cat("MSE (based on ", draws, " samples):  ")
        fit.mse.0 <- mean( (data.fit$y-pt.estimate.f.0)^2 )
        fit.mse.1 <- mean( (data.fit$y-pt.estimate.f.1)^2 )
        fit.mse.2 <- mean( (data.fit$y-pt.estimate.f.2)^2 )
        cat(round(c(fit.mse.0,fit.mse.1,fit.mse.2), 6), "\n")
        dummy.chain<-list(beta=matrix(apply(chain$beta,2,mean),nrow=1),
                          gamma=matrix(apply(chain$gamma,2,mean),nrow=1),
                          s=mean(chain$s),M=chain$M,p=chain$p,N=chain$N)
        print(dummy.chain)
        DIC<- 2*mean(myD) - (myest.c(data.fit, dummy.chain, iter=1)$D)
        cat("DIC (test): ", round(DIC, 6), "\n")
        if(!missing(data.valid)) {
                pt.estimate.v.0 <- apply(estimates.v, 1, mean)
                pt.estimate.v.1 <- apply(estimates.v, 1, weighted.mean,w=wgt1)
                pt.estimate.v.2 <- apply(estimates.v, 1, weighted.mean,w=wgt2)
                cat("MSE (based on ", draws, " samples):  ")
                val.mse.0 <- mean( (data.valid$y-pt.estimate.v.0)^2 )
                val.mse.1 <- mean( (data.valid$y-pt.estimate.v.1)^2 )
                val.mse.2 <- mean( (data.valid$y-pt.estimate.v.2)^2 )
                cat(round(c(val.mse.0,val.mse.1,val.mse.2), 6), "\n")
                DIC.valid<- 2*mean(myD) - (myest.c(data.valid, dummy.chain, iter=1)$
                cat("DIC (validation): ", round(DIC.valid, 6), "\n")
        }
        return(list(x.fit = data.fit$x, y.fit = pt.estimate.f.0,
              sqerrors = sqerrors,sqerrors.valid = sqerrors.valid,
              mse=c(fit.mse.0,fit.mse.1,fit.mse.2),
              mse.valid=c(val.mse.0,val.mse.1,val.mse.2), wgts=cbind(wgt1,wgt2) ))
}
```

Appendix B: Plots

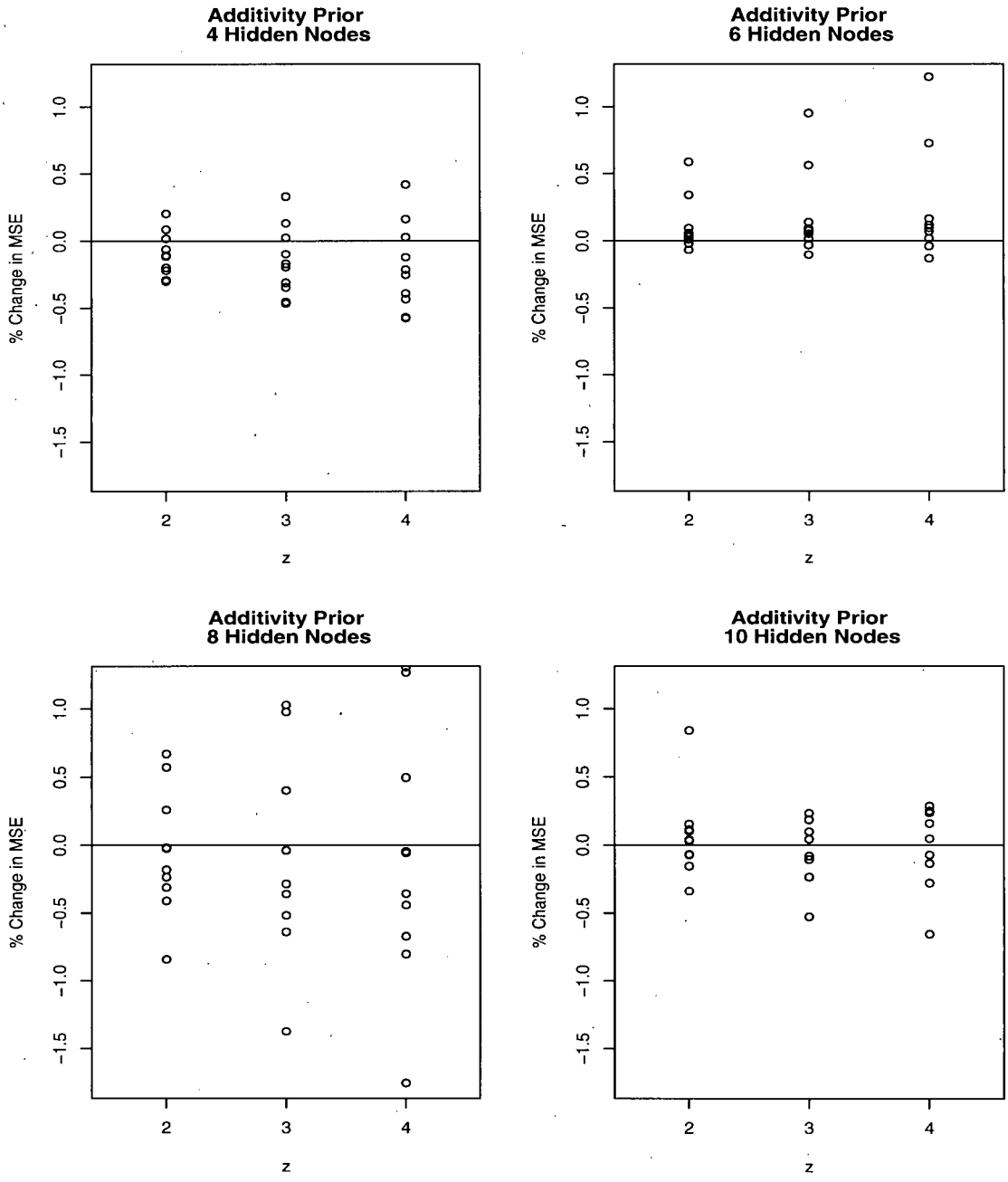
In the following pages, the extra plots referred to in Chapter 4 are given.



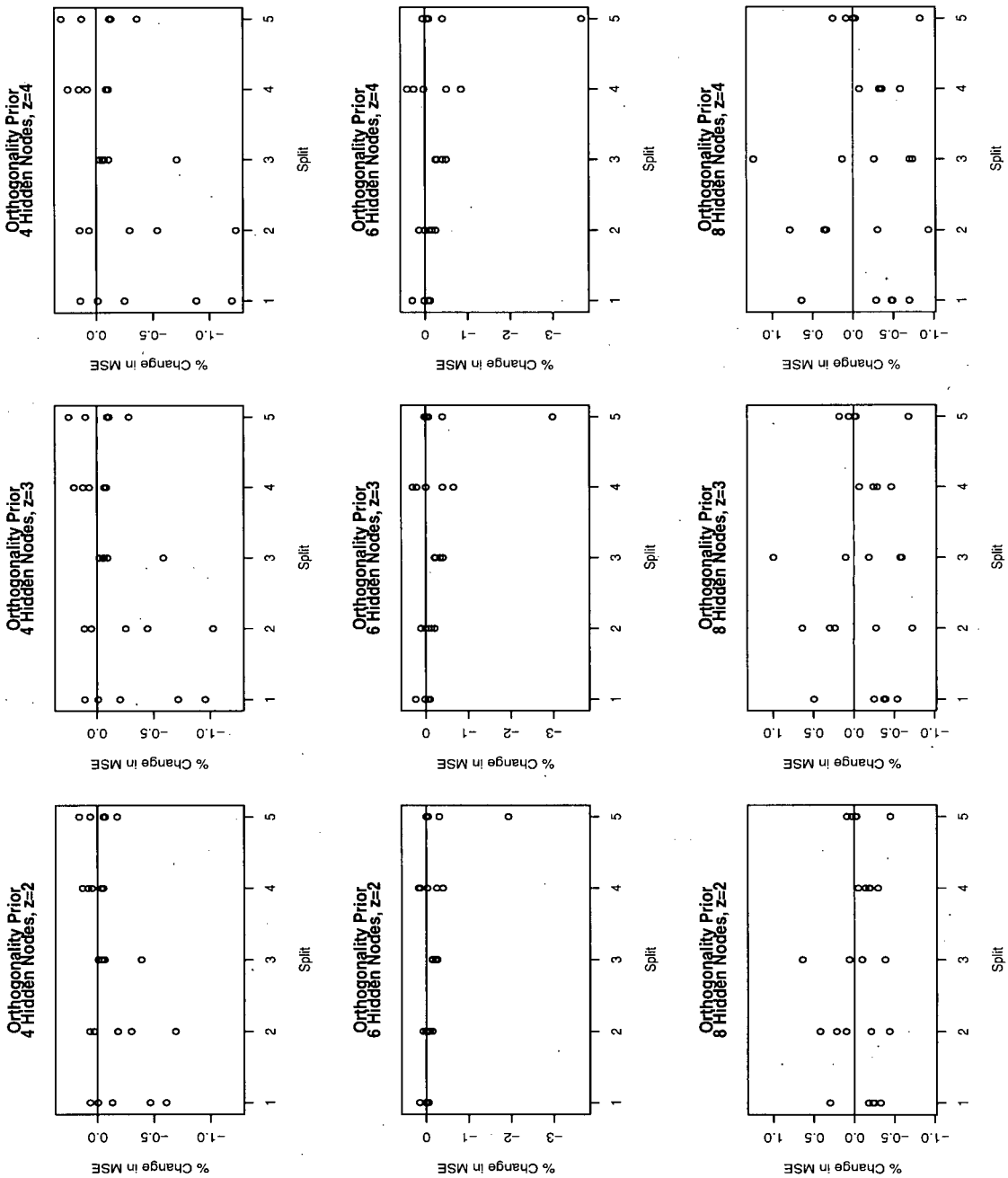
Boston Dataset.



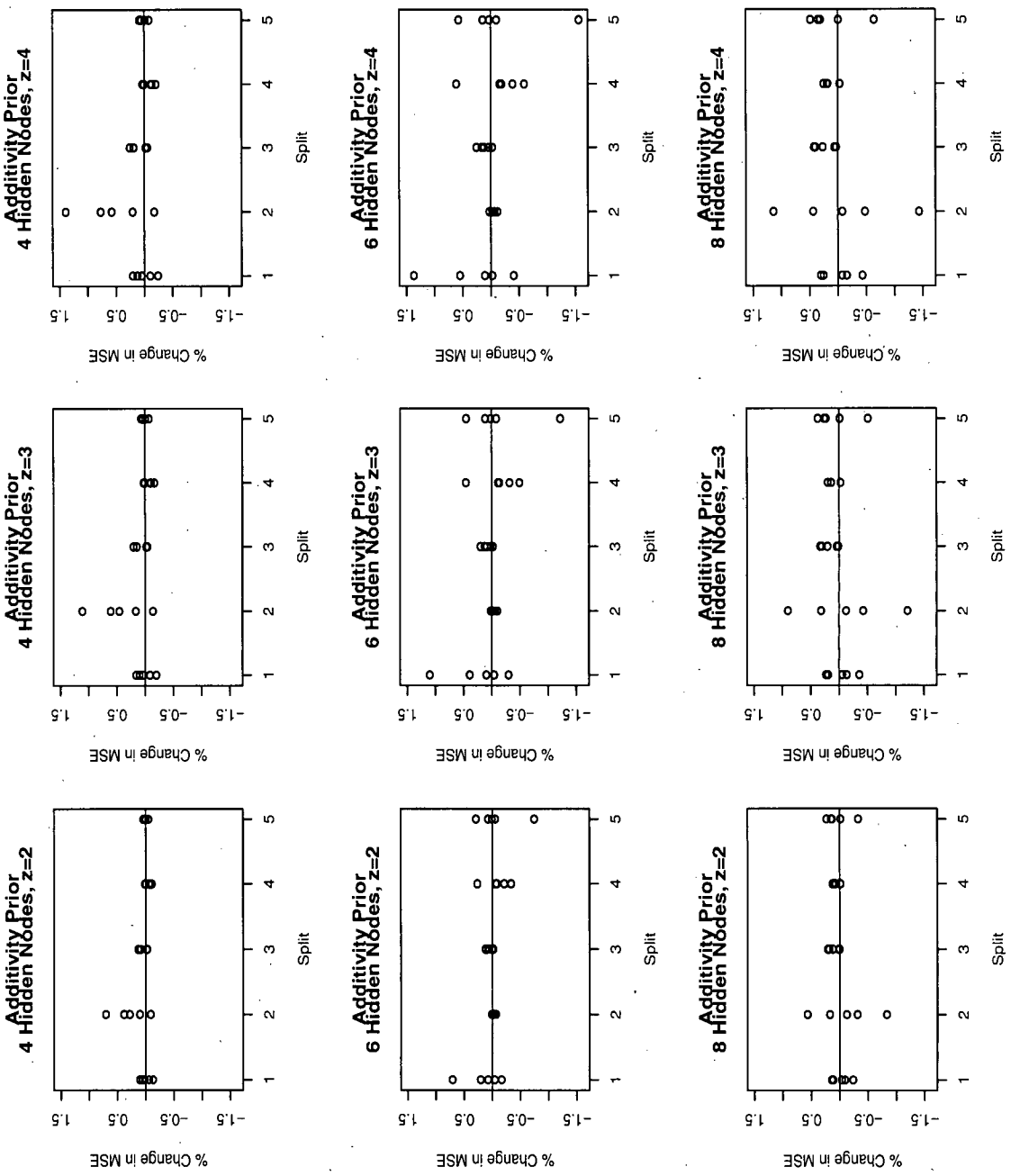
Tecator Dataset.



Tecator Dataset.



Ozone Dataset.



Ozone Dataset.

Bibliography

[1] Andrieu, C, de Freitas, J.F.G. and Doucet, A. (1999). Robust Full Bayesian Learning for Neural Networks. Technical Report 343, Engineering Department, Cambridge University.

[2] Breiman, L, Friedman, J.h., Olshen, R.A. and Stone, C.J. (1984) *Classification and Regression Trees* Monterey: Wadsworth and Brooks/Cole.

[3] Cheng, B. and Titterington, D.M. (1994). Neural Networks: A review from a statistical perspective (with discussion). *Statistical Science*, 9, 2-54.

[4] DiCiccio, T.J., Kass, R.E., Raftery, A.E. and Wasserman, L. (1997). Computing Bayes Factors by Combining Simulation and Asymptotic Approximations. *Journal of the American Statistical Association*, 92, 903-915.

[5] Edwards, P.J, Murray, A.F., Papadopoulos, G., Wallace, R., Barnard, J. and Smith, G. (1999). The Application of Neural Networks to the Papermaking Industry. *IEEE Transactions on Neural Networks* 10(6),1456-1464.

[6] Friedman, J.H. (1991). Multivariate adaptive regression splines (with discussion). *Annals of Statistics.* 19, 1-141.

[7] Green, P.J. (1995). Reversible jump Markov Chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4),711-32.

[8] Gustafson, P. (2000). Bayesian Regression Modelling with Interactions and Smooth Effects. *Journal of the American Statistical Association.* To appear.

[9] Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models.* London: Chapman and Hall.

[10] Harrison, D. and Rubinfeld, D.L. (1978). Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5, 81-102.

[11] Hoeting, J.A., Madigan, D., Raftery, A.E. and Volinsky, C.T. (1999). Bayesian Model Averaging: A Tutorial (with discussion)." *Statistical Science*, 14,4,382-417.

[12] Hornik, K, Stinchcombe, M. and White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5), 359-366.

[13] Hsieh, W.W and Tang, B. (1998). Applying Neural Networks to Prediction and Data Analysis in Meteorology and Oceanography. *Bulletin of the American Meteorological Society*, 79(9),1855-1870.

[14] Ihaka, R. and Gentleman, R. (1996) R: A Language for Data Analysis and Graphics. *Journal of Graphical and Computational Statistics*, 5, 299-314.

[15] Lee, H.K.H. (2000a). A Noninformative Prior for Neural Networks. Technical Report 00-04, Duke University, Institute of Statistics and Decision Sciences.

[16] Lee, H.K.H. (2000b). A Framework for Nonparametric Regression Using Neural Networks. Technical Report 00-32, Duke University, Institute of Statistics and Decision Sciences.

[17] Lee, H.K.H. (2000c). Model Selection for Neural Network Classification. Technical Report 00-18, Duke University, Institute of Statistics and Decision Sciences.

[18] Mackay, D.J.C (1992). The evidence framework applied to classification networks. *Neural Computation.* 4, 720-736.

[19] Mackay, D.J.C (1994). Bayesian Non-Linear Modelling for the Energy Prediction Competition. *ASHRAE Transactions*, 100, Part 2, 1053-1062. *Neural Computation.* 4, 720-736.

[20] Macrossan, P.E., Abbass, H.A., Mengersen, K., Towsey, M. and Finn, G. (1999). Bayesian Neural Network Learning for Prediction in the Australian Dairy Industry. In *Lecture Notes in Computer Science*, 1642, 395-406.

[21] Müller, P. and Rios Insua, D. (1998). Issues in Bayesian Analysis of Neural Network Models, *Neural Computation*, 10, 571-592.

[22] Neal, R.M. (1993). Probabilistic inference using Markov Chain Monte Carlo methods. *Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto.*

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

[24] Raftery, A.E., Madigan, D. and Hoeting, J.A. (1997). Bayesian Model Averaging for Linear Regression Models. *Journal of the American Statistical Association.* 437, 179-191.

[25] Rios Insua, D. and Müller, P. (1998). Feedforward Neural Networks for Nonparametric Regression. In *Practical Nonparametric and Semiparametric Bayesian Statistics* (Dey Dipak, Peter Müller, Debajyoti Sinha, editors) 181-193. Springer, New York.

[26] Ripley, B.D. (1994). Neural networks and related methods for classification. *Journal of the Royal Statistical Society B*, 56, 409-456.

[27] Spiegelhalter, D.J., Best, N.G. and Carlin, B.P. (1998). Bayesian deviance, the effective number of parameters, and the comparison of arbitrarily complex models. *Applied Statistics*, to appear.

[28] Stern, H.S. (1996). Neural networks in applied statistics. *Technometrics*, 38, 205-220.

[29] Thodberg, H.H. (1996). A Review of Bayesian Neural Networks with an Application to Near Infrared Spectroscopy. *IEEE Transactions on Neural Networks.* 7(1), 56-72.