Classification using Bayesian Neural Nets

Jan C. Bioch, Onno van der Meer, Rob Potharst
Department of Computer Science
Erasmus University Rotterdam
P.O. Box 1738, 3000 DR Rotterdam
The Netherlands
{bioch,vdmeer,robp}@cs.few.eur.nl
Technical Report eur-cs-95-09

Abstract

Recently, Bayesian methods have been proposed for neural networks to solve regression and classification problems. These methods claim to overcome some difficulties encountered in the standard approach such as overfitting. However, an implementation of the full Bayesian approach to neural networks as suggested in the literature applied to classification problems is not easy. In fact we are not aware of applications of the full approach to real world classification problems. In this paper we discuss how the Bayesian framework can improve the predictive performance of neural networks. We demonstrate the effects of this approach by an implementation of the full Bayesian framework applied to three real world classification problems. We also discuss the idea of calibration to measure the predictive performance.

1 Introduction

Due to the fact that neural networks are universal approximators, they are able to model non-linear regularities in the the data. On the other hand this often leads to the problem that a too-flexible network 'discovers' non-existent structures in the data. This is known as overfitting: a good fit on the training data and a poor generalization on the the test data. One of the advantages of the Bayesian approach is avoiding the problem of overfitting. It is also (theoretically) *not* necessary to split the data set in one or more training sets and test sets as is common in cross-validation used in the the standard approach. The Bayesian approach to neural network learning has recently been proposed by MacKay [Mac92a, Mac92b] and Neal [Nea92, Nea93a, Nea93b, Nea95]. The Bayesian approach to prediction has two aspects that can be used to reduce the risk of overfitting. Firstly, we need to specify a prior distribution for the network parameters, which expresses our beliefs about which values of the parameters would be likely. This belief is updated through the data, using a likelihood function, to give a posterior distribution for the network parameters. Secondly, predictions are based on all possible values of the network parameters,

weighted by this posterior distribution. This method deals explicitly with our uncertainty about the model parameters. Application of the Bayesian method should therefore lead to better balanced judgments about future observations. In general network parameters include the weights, number of hidden units, control parameters etc. However, in this paper we only consider the weights as the parameters of our model. Therefore a model in this paper corresponds to a point (weight-vector) in the weight space, and the set of all models to the weight space. A characteristic feature of (ideal) Bayesian predictions is that they are based on all models rather than on one model such as in the standard approach. Each model (set of weights) is weighted by its posterior probability. Therefore the output of a Bayesian network is obtained by averaging the outputs of the networks corresponding to all the points of the weight space. This is called integrating the posterior distribution over the weight space. This is clearly a demanding and non-trivial computation. The Bayesian approach to neural networks has been successfully applied by (e.g.) MacKay to regression problems. Since the computation of the posterior is difficult, MacKay uses Gaussian distributions to approximate the posterior. Another approach to calculate the posterior uses Monte Carlo methods and Markov chains. This sophisticated approach is proposed by Neal who demonstrates this method for a classification problem using a synthetic data set. For an account of all these matters we refer to the work of MacKay and Neal mentioned earlier. For background reading on Bayesian methods we can recommend [GCSR95, Ber85, BS94, BT73] or [Bre90].

In this paper we investigate the Bayesian approach to neural networks by implementing the full Bayesian approach for classification networks. This implementation is applied to three world data sets. The paper is organized as follows : Section 2 summarizes the standard approach to classification using feedforward neural networks. Section 3 gives a brief overview of the Bayesian framework for inference and prediction, and makes the comparison to the standard approach. Section 4 deals with the problem of implementing the calculations required by the Bayesian framework. Section 5 applies the Bayesian methodology to three data sets and discusses measuring predictive performance; we need to check that the estimated probabilities are *well calibrated*. Finally, section 6 concludes that the Bayesian framework can lead to better calibration and classification.

2 Standard Classification Networks

We will confine our discussion to feedforward networks with one hidden layer:

$$f(x,w) \quad = \quad l(w_o^b + \sum_h w_h g(w_h^b + \sum_i w_{ij} x_i)). \tag{1}$$

Here x is the input vector and the parameters w include the bias terms(w^b), input to hidden weights and hidden to output weights. The function $g()$ used in the hidden layer can be the logistic function or, for instance, the hyperbolic tangent function. For classification problems with two classes, it is convenient to use the logistic function

$$l(x) = \frac{1}{1+\exp(-x)}.$$

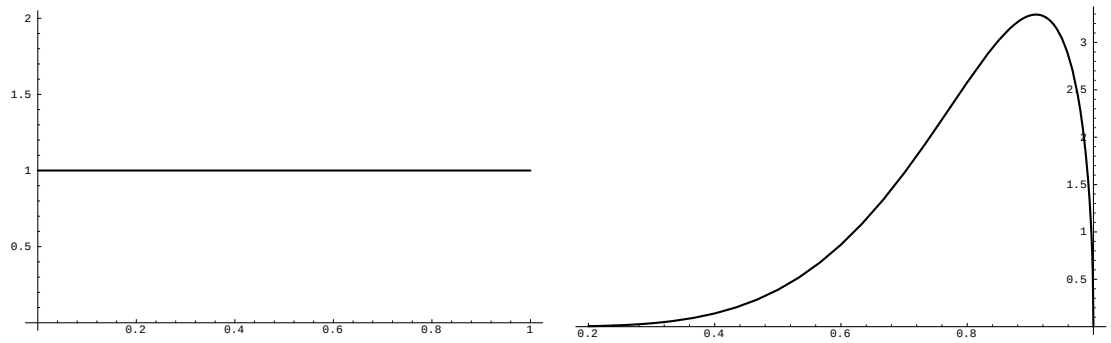
We assume we have a data set consisting of N input vectors $x_1, \ldots, x_N$ and corresponding targets y. The targets denote class–membership (0/1 for class 1 and 2). Training the neural network consists of finding weights that minimize the distance between outputs and targets. It is known that under some conditions the output $f(x, w)$ can be interpreted as the probability $P(y = 1|x, w)$. An appropriate error–function for a two–class problem is

$$D(w) = \sum_{i}^{N} y_i \log f(x_i, w) + (1 - y_i) \log(1 - f(x_i, w)) \tag{2}$$

For problems with more than two classes the targets are represented by binary vectors in which a single component is set to one to denote the correct class, and all the other components are set to zero. Also in this case the outputs of the trained network are interpreted as class probabilities, where it is common to use softmax units [Rip94] $\exp(x_o)/\sum_j \exp(x_j)$, to ensure the outputs sum to one. A popular way to estimate the errors of the outputs of the network is cross-validation. In this paper, however, we will only consider two-class problems.

3 Bayesian Predictive Inference

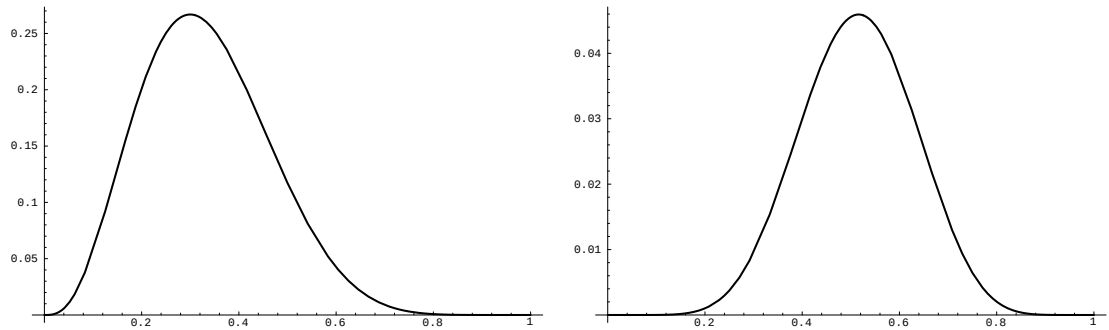
According to the Bayesian point of view, see for instance [Ber85], a researcher who wishes to gain knowledge about one or more parameters through some observations, starts with putting a prior distribution on the parameters. This prior distribution expresses a priori beliefs about the possible values the parameters can have. If there is no real prior information available about the parameter, this can be accommodated using a non–informative prior. For instance, if a parameter p can take on values between 0 and 1 a uniform prior can be used to express ignorance as to which values are more likely than others (See figure 1(a)). On the other hand, if it is believed that values near 0.8 are more probable a priori, a beta prior like (b) can be used. The



(a) Uniform prior

(b) Beta prior

Figure 1: Examples of prior distributions



(a) (b)

Figure 2: The posterior distributions corresponding to the priors in figure 1

next step in Bayesian inference is to specify a probabilistic model to express knowledge about the way the distribution depends on the parameters. For instance, if we perform n independent experiments, each with probability p of success, the probability of x successes is

$$\Pr(x|p) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}$$

Using Bayes' theorem, we can now calculate the *posterior* distribution of the parameter p, given the observations x

$$\Pr(p|x) = \frac{\Pr(x|p)\Pr(p)}{\int_0^1 \Pr(x|p)\Pr(p)dp}$$

The denominator is a normalizing constant and is usually ignored to give: posterior $\propto$ likelihood $\cdot$ prior. This probability distribution expresses the updated belief of the researcher after the experiment. For instance, if he had a uniform prior like in figure 1 and in ten experiments he observed three successes , the posterior distribution would look like depicted in figure two.

3.1 Likelihood functions

For classification with neural networks, the following model can be used. The output of a neural network (using the logistic function as discussed above) given an attribute vector x can be interpreted as the probability that the corresponding example belongs to class 1. A single example (x, y), where y denotes the correct class membership (either 0 or 1), has likelihood given the weights w :

$$f(x,w)^y (1 - f(x,w))^{1-y}$$

If we have n examples $(x_1, y_1), \ldots, (x_n, y_n)$, which we assume to be independent and identically distributed, we have the following likelihood function for the training data:

$$\Pr(y|w) = \prod_i^n f(x_i, w)^{y_i} (1 - f(x_i, w))^{1-y_i} \tag{3}$$

For neural network training it is convenient to use the logarithm of the posterior. The log–likelihood is in this case equal to the error–function $D(w)$ defined earlier, so the log posterior $E(w)$, is equal to $D(w) + \log \Pr(w)$, where $\Pr(w)$ is the prior over the weights.

3.2 Predictive Distribution

For classification problems we are interested in the predictive distribution of a new example given its attribute vector and the training data:

$$\Pr(y_{n+1}|x_{n+1}, (x_1, y_1), \ldots, (x_n, y_n)) = \int \Pr(y_{n+1}|x_{n+1}, w) d \Pr(w|(x_1, y_1), \ldots, (x_n, y_n))$$

This predictive distribution is the full result of Bayesian inference. However, in many circumstances it is necessary to make a single–valued guess at the value of y_{n+1}. How this guess depends on the predictive distribution is determined by a loss–function, which expresses our judgment of the adverse effect of guessing $\hat{y}$ when the real value is y[Ber85]. The most widely used is squared–error loss, for which the best prediction for a test case is given by

$$\hat{y}_{n+1} = \int f(x_{n+1}, w) d \Pr(w|(x_1, y_1), \ldots, (x_n, y_n))$$

This corresponds to the mean output of the network, averaged [1] over the posterior distribution of the weights. Because of the complexity of neural network models the calculation of the required integrals can only be feasibly carried out using Markov Chain Monte Carlo numerical methods, as discussed in section 4.

If we compare this approach with the standard method of finding a minimum of the error function(= maximum likelihood estimate) and using this to make the predictions, we see that the standard approach ignores the uncertainty with respect to the weights and assumes all possible weights to be equally likely.

3.3 Priors for Neural Networks

A simple method that is often used to reduce the risk of overfitting, is adding a term to the cost function that penalizes (too) large weights [HKP91, KH91]:

$$E(w) = D(w) + \frac{1}{2} \lambda \sum_i^{n_w} w_i^2. \tag{4}$$

where n_w is the number of weights. This leads to the following update rule for gradient descent:

$$\Delta w_i \propto -\frac{\partial D(w)}{\partial w_i} - \lambda w_i \ , \tag{5}$$

[1] Note this results in a procedure quite similar to the combination of networks [LT93, TG95], where several networks are trained and the outputs of the networks are combined for the prediction.

which gives a weights the tendency to decay to zero, unless a larger value contributes to the reduction of the error. The tradeoff between keeping the weights small and minimizing the error is determined by the parameter λ, which is usually set to a fixed value, e.g. 0.001.

The weight decay term has a Bayesian interpretation[Mac92a, Mac92b] as a Gaussian prior distribution with zero mean and variance $1/2\lambda$. This means we consider small values for the weights to be a priori more likely than large weight. The region of values that receive a priori probability is determined by the variance of the distribution. Instead of fixing its value the Bayesian approach allows that we specify a *hyperprior* distribution for λ. This distribution is used to integrate out λ, removing it from the prior distribution. This means that the flexibility of the model will be automatically determined from the data.

Although there are several ways to implement the required hyperprior, we will focus here on a distribution that is conjugate to the prior for the weights and use a inverse–gamma hyperprior for the unknown variance [Ber85]:

$$\begin{aligned} \Pr(w) &= \int_0^\infty \lambda^{n_w/2} \exp(-\lambda \sum w_i^2) \frac{\lambda^{\alpha-1}}{\lambda_0^\alpha, (\alpha)} \exp(-\lambda/\lambda_0) d\lambda \\ &\propto (1 + \lambda_0 \sum w_i^2)^{(-\alpha - n_w)} \end{aligned} \tag{6}$$

where α and λ_0 are parameters of the hyperprior. The resultant prior distribution is a multivariate student t distribution, so,for instance, we should set $\alpha > 1$ to avoid a having prior with infinite variance[2].

Following the discussion in [BW91], we can examine the contribution of the prior during gradient descent:

$$\frac{-\partial \log \Pr(w)}{\partial w_i} = \frac{\lambda_0 n_w + 2\alpha}{\lambda_0 \sum_j w_j^2} w_i \tag{7}$$

This shows that integrating out λ implies that instead of having a fixed value, the variance can be determined from the data [Bre90]. In the above discussion the prior was assumed to apply to all weights in the network. It is clear from the form of the network that the weights can be divided into separate groups based on their function. For instance, MacKay used three groups : bias terms, input to hidden weights and hidden to output weights. This means that the prior has three separate terms, with n_w and other parameters set for each group.

3.4 Automatic Relevance Determination

This scheme can be extended further to what MacKay and Neal call the Automatic Relevance Determination prior [Nea95], where the input units belong to individual groups (this assumes the inputs are independent). For instance, for a network with five inputs and seven hidden units, the ARD prior would consist of the product of five terms based on equation 6 with $n_w = 7$ (see figure 3; although every input is connected to every hidden unit, this figure shows one of the five separate groups of the ARD prior, constisting of the seven input–to–hidden weights for input unit 3). Using this

[2]Setting $\alpha = 0$ and $\lambda_0 = 1$ mimics the effect of using $\Pr(\lambda) = 1/\lambda$, as suggested by [BW91] and others.

prior, we would expect irrelevant inputs to have relative small values for the input–to–hidden weights, compared to the relevant inputs, and not to be detrimental to the predictive performance. See [KM95, Chi95] for similar Bayesian variable selection methods applied to general regression models.

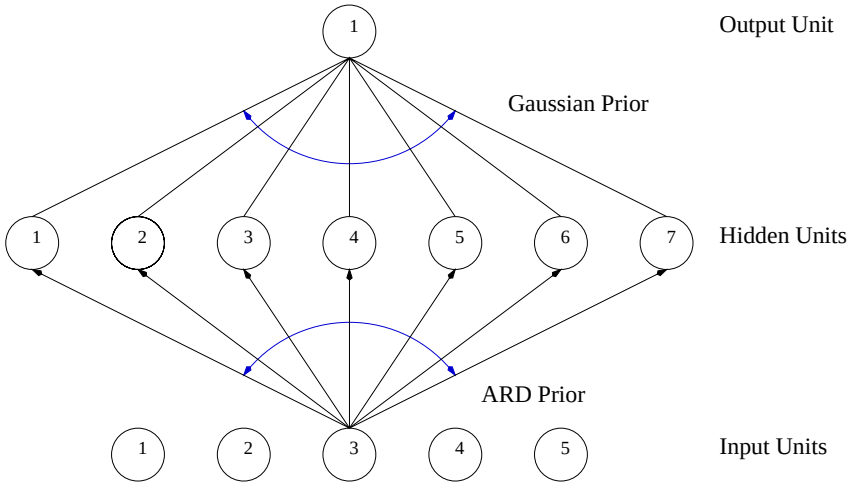


Figure 3: Automatic Relevance Determination prior

3.5 Infinite Networks

In the predictive approach [Rip96] a model is introduced solely to facilitate a probability assignment for unknown observables. Furthermore, as has been mentioned earlier, instead of using one model, we ideally average over all models. This would suggest using a model as flexible as is permissable by our computational resources. Neal [Nea95] shows that if a suitable prior is used, the number of hidden units,H can tend to infinity without causing the problems non–Bayesian methods would have with models this size. In fact, Neal shows that if $H \to \infty$ the complexity of the functions generated by the prior is independent of H. Neal establishes these result by using a Gaussian prior with fixed variance scaled by $1/\sqrt{H}$. For instance, in a network with seven hidden units we could use a prior with $\lambda = 0.01\sqrt{7}$(see figure 3).

4 Markov Chain Monte Carlo Methods

We have seen that the Bayesian approach requires an integration over the posterior distribution. Given the structure of neural networks and the size of the parameter space, this integration is clearly analytically intractable. At present the only feasible solution to this problem is the use of Markov Chain Monte Carlo (MCMC) methods. For a review of these methods see, for instance [Nea93b].

If we assume the square-error loss function, we need to calculate the average output of the neural network over the posterior distribution of the weights:

$$< f(x) >= \int f(x, w) \Pr(w|(x, y))dw. \tag{8}$$

MCMC methods approximate this integral by generating $I + M$ vectors $w_1, w_2, \ldots$ that form an ergodic Markov chain with stationary distribution $\Pr(w|(x,y))$. This sequence is then used to calculate

$$< f(x) >= \frac{1}{M} \sum_{t=I}^{I+M} f(x, w_t) \tag{9}$$

where I is the number of initial values we discard because the chain has not yet reached its stationary distribution.

Several different methods exist for producing the required Markov chain. Most of these methods operate by generating a random parameter vector using a proposal distribution, for instance a Gaussian or Cauchy distribution. For neural networks, randomly generating candidate states is likely to be a slow process, because the high–probability states of the posterior distribution occupy only a small amount of the total state space. In the context of neural networks it is much more convenient to use the readily available gradient information in generating new states. One implementation of this idea is the *Langevin* equation [Kro92] which takes the following form:

$$\Delta w_i \propto -\frac{\partial E(w)}{\partial w_i} + \eta_i \tag{10}$$

where $E(w)$ denotes the distribution we are interested in and η is Gaussian noise with mean zero. A disadvantage of this approach is that it results in a random walk.

A method that is well–suited for Bayesian calculations for neural networks and similar statistical applications is the *Hybrid Monte Carlo* (HMC) algorithm [DKPR87], which was introduced into the neural networks literature by Neal [Nea92, Nea93a]. This method uses stochastic and deterministic moves to generate a new state, and largely suppresses random walk behaviour. The HMC algorithm operates in an extended state space, where a momentum vector p is associated with the parameter vector w, and uses a "Hamiltonian" function:

$$H(p, w) = E(w) + \frac{1}{2}|p|^2 \tag{11}$$

where the second term stems from the analogy with physical kinetic energy.

One iteration of the HMC algorithm as used here (for a more general discussion see [DKPR87]) moves from a to state at time t_n to a new state at time t_{n+1} as follows :

1. *Stochastic move:* Draw a new momentum vector $p(t_n)$ from its stationary distribution
$$\Pr(p) = (2\pi)^{(-N/2)} \exp(-|p|^2/2),$$
where N is the dimension of the parameter space.

2. *Deterministic move:* Generate a candidate state by following Hamilton's dynamics from time t_n to time t_{n+1} :
$$\begin{aligned} \frac{\partial w}{\partial t} &= +\frac{\partial H}{\partial p} = p \\ \frac{\partial p}{\partial t} &= -\frac{\partial H}{\partial w} = -\frac{\partial E(w)}{\partial w} \end{aligned}$$

3. Accept the new state with probability $\min(1, \exp(-\Delta H))$ where

$$\Delta H = H(w(t_{n+1}), p(t_{n+1})) - H(w(t_n), p(t_n)).$$

This acceptance step compensates for any inaccuracies that are introduced in the discretization.

To implement the deterministic move of the HMC algorithm, we need to discretize the Hamiltonian dynamics; to maintain the validity of the method this discretization must preserve the volume of the phase–space. This is necessary to ensure that the Markov chain has posterior distribution as its stationary distribution. This requirement is met by symplectic integrators; in the following we will use such a Runge–Kutta–Nyström integrator. See appendix A for the details of the discretization.

5 Experiments

We applied the Bayesian methods discussed above on three realistic data sets. We focused on the following issues: does the Bayesian approach lead an improvement over standard methods in terms of better classification as well as better estimates of the probabilities, and is the risk of overfitting indeed reduced? We begin with a discussion on how to measure predictive performance. Details of the implementation and results are given for the three datasets.

5.1 Measuring Predictive Performance

There are two aspects of predictive performance that are important in the case of classification [MGR94]: the performance in assigning a new example to the correct class and the extent to which we can interpret the output of the network as the *conditional probability* that a new example will be in class 1 given its attribute vector. The discrepancies between predicted and actual outcome can be divided into two components: predictive bias(incorrect classification) and lack of calibration (over– or underestimating posterior probability). If the outputs of the network are well–calibrated we would expect the assigned probabilities to agree with the relative frequency of occurrence. For instance, we would expect 80 percent of the cases assigned probability 0.8 to occur. To measure calibration[Daw86], we group the cases in a test set according to the predicted probability $(0.0 - 0.1, \ldots, 0.9 - 1.0)$, and compare with the relative frequency of 1's in each of the groups. Although it is often informative to consider predictive bias and calibration separately, these two concept can be combined for a two class problem in the *logarithmic score* S, defined as $S(1, p(1|x)) = -\log p(1|x)$ and $S(0, p(1|x)) = -\log(1 - p(1|x))$. Notice that this corresponds to the likelihood function defined above. If we have little data available for testing, the observed error rate on the test set may be a poor estimate of the true error rate. If the output of the network is interpreted as a posterior probability, the following expression provides a more accurate estimate of the true error rate [Rip94]:

$$\frac{1}{N_{\text{test}}} \sum_{i \in \text{test set}} [1 - p(1|x_i)] \tag{12}$$

This is sometimes referred to as smoothing the error rate. Again this stresses the importance of obtaining good estimates of the posterior probabilities.

5.2 Preliminaries

It has been noted before that the parameters obtained by the standard training methods can lead to over–confident predictions. The full Bayesian predictive approach solves this problem by integrating over the weight space. Different classifiers correspond to different areas of the weight space. It is therefore important to obtain a representative sample from the posterior distribution. Using only a single run of the HMC algorithm will in general result in a sample from only one region of the weight space, as the modes of the posterior distribution are isolated, making jumps from one region to another unlikely. One solution to this problem is to run the algorithm several times starting with different initial values, and to combine these runs. In both experiments, we implemented the discretization of the Hamiltonian dynamics using an eighth–order explicit symplectic Runge–Kutta–Nyström integrator. This requires 17 evaluations of the gradient of the posterior distribution in the algorithm given in appendix A. We used the parameters as given by Okunbor and Lu[OL94]. To suppress the random walk tendencies, it is important to take steps as large as possible, while maintaining a high acceptance rate. If a small h is necessary to keep the rejection rate low, we can use the RKN a number of times in direct succession, to get the desired time–step.

Each data set was preprocessed to have inputs with zero mean and standard deviation one.

5.3 Pima–Indian

To gain insight into the effect of using Bayesian methods on the calibration of the outputs of neural networks, we use a relatively large data set that was previously investigated by Wahbda et al [WCWC93] in the context of smoothing spline models.

The data set was retrieved from the UCI Repository [MA92]. The data contains medical records from Pima–Indian women at least 21 years of age. This data set contains 8 predictor variables and the class label: Number of times pregnant, Plasma glucose concentration after 2 hours in an oral glucose tolerance test, Diastolic blood pressure (mm Hg), Triceps skin fold thickness (mm), 2-Hour serum insulin (mu U/ml), Body mass index (weight in kg/(height in m)^2), Diabetes pedigree function, Age (years). The class label is an indicator of a positive(1) test for diabetes between 1 and 5 years from the examination, or a negative(0) test for diabetes 5 or more years later.

The data was shown by Wahba et al to contain 16 instances with impossible attribute values, all of which were deleted, leaving 752 instances. Following Wahba et al, the dataset was split in a training set of 500 and a test set of 252 instances.

We used a network with 8 inputs, 16 hidden units and one output. The number of hidden units chosen was larger than necessary as indicated by initial experiments. This was done to test the effects on overfitting of the Bayesian approach. We used the Gaussian prior as described above, with three separate weight–groups. The parameters of the hyperprior were set to different values for the individual groups, based on the characteristics of each group. The following table summarizes the settings for the groups.

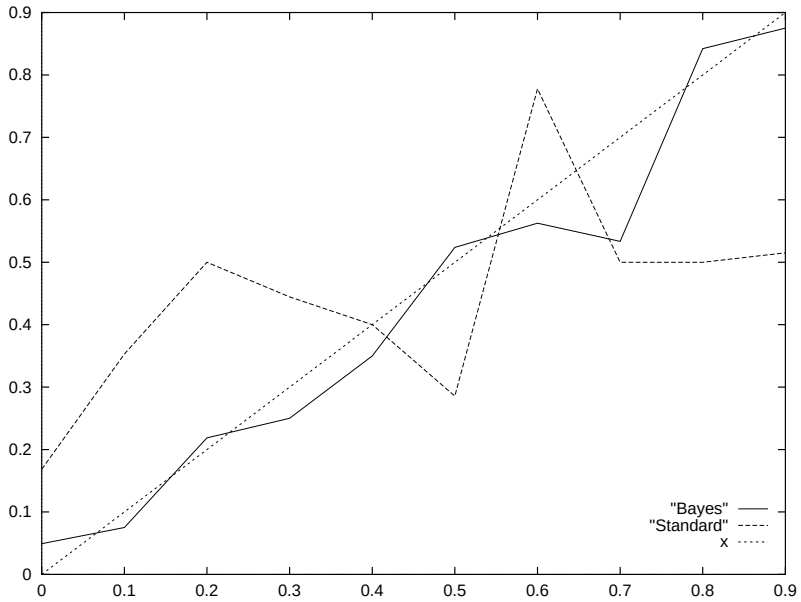


Figure 4: Calibration of neural network outputs

group	N_w	α	λ_0
input to hidden weights	128	2.0	0.03
hidden–biases	16	2.5	0.0001
hidden to output weights and biases	17	2.5	0.025

We used four separate runs of the HMC algorithm. For each run, the initial weights were randomly drawn from the Gaussian prior distributions. Each run started with an initial phase of 50 iterations with stepsize 0.5 to speed up reaching equilibrium. This was followed by 300 iterations of 10 successive RKN-steps with stepsize 0.01 to obtain a good approximation of the equilibrium state. Of the last 20 iterations of each run every other iteration was used as a basis for the predictions. A "standard" neural network with 8 hidden was trained using a variable metric algorithm and a weight–decay penalty (with λ set to 0.001).

Figure three gives the calibration curve of the Bayesian and standard neural network. Despite the relatively large training set the estimates of the standard network are quite poorly calibrated. The Bayesian network shows a marked improvement.

Wahba et al report a misclassification rate on the test set using a smoothing spline approach of 24 percent. The standard neural network had an error rate of 24.6 percent; the error rate for the Bayesian approach was 20.5 percent.

5.4 Low birth–weight

Hosmer and Lemeshow[HL89] give a dataset of 189 births at a US hospital, with an interest to predicting pregnancies that result in low birth weight. The following variables are available: age of mother in years, weight of mother, white/black/other, smoking during pregnancy(0/1), number of previous premature labors, history of

hypertension(0/1), has uterine irritability(0/1), number of physician visits in the first trimester, class label (0=normal / 1=low birth weight).

This dataset was studied by Ripley[Rip94], who detected 5 pairs of identical rows, one of each was subsequently removed. The remaining data was split in a training sample of 134 cases and a test set of 50 cases. The test set consists of 37 "normal" and 13 "low" cases.

The main objective in this case is to correctly identify the risk group. As the risk group is noticeable smaller than the normal group, the predictive approach would be expected to have a distinct effect for this data set. Ripley experimented with a crude form of the predictive approach, by training 20 standard networks and averaging over the outputs of these networks using Gaussian approximation.This did not result in better classification.

Ripley used networks with 2 and 6 hidden units, which give identical classification performance. As with the pima–indian data, we used a relatively large number of hidden units, 12 in this case. We used a similar grouping of the weights for the prior distribution as before. We used slightly different values for the hyperparameters for this data set.

group	N_w	α	λ_0
input to hidden weights	96	2.5	0.08
hidden–biases	12	2.0	0.001
hidden to output weights and biases	13	2.5	0.028

We used 10 independent runs of the HMC algorithm. As the posterior distribution seems too have a complex form for this problem we used a smaller stepsize than in the pima–indian experiment. Again we used an initial phase to approximate equilibrium, using 75 iterations with stepsize 0.4. This was followed by 400 iterations, using 5 successive applications of the RKN algorithm with stepsize 0.002. Every other iteration of the last 50 runs of each chain was used in making the predictions. Figure 5 shows the results for the Bayesian and standard networks. We repeated Ripley's experiments and included a standard network with 12 hidden units for comparison. The standard networks show a decrease in performance with increased size, especially if we use the smoothed error rate. The Bayesian network performs very well in comparison. This shows a quite significant improvement, especially in the risk group, where only two errors are made out of 13 cases (the neural networks reported by Ripley made four errors). The smoothed results are also much better, largely due to the improved estimates of the posterior probabilities.

We can also look at the logarithmic score for this problem to see the effect of combining the runs. The ten individual runs had the following logarithmic scores evaluated using the last iterations of the runs: 27.5, 28.3 31.1, 27.1, 29.1,27.6, 29.8, 28.1, 30.3, 29.6. The logarithmic score for the combined runs was 22.9. Following Madigan et al [MGR94], we can interpret this score as follows: the improvement of the combined runs over the best run is 4.2. Given that the test set contains 50 cases the improvement is $\exp(4.2/50) = 1.0876$ or roughly 9 percent.

method	Raw		Smoothed	
	normal	low	normal	low
neural net (2 hidden)	22	31	28	35
combined nets (2 hid)	24	31	33	38
neural net (6 hidden)	22	31	24	28
neural net (12 hidden)	30	36	34	38
Bayesian (12 hidden)	18	15	20	16

Figure 5: Error rate in percentages for the normal and low birthweight groups

5.5 The Sellers Data

This dataset is described in [BDV94] and consists of the scores of 69 salespeople on 6 scales, representing various psychological concepts such as 'adaption', 'rigidity', 'interpersonal control', etc. Independently, the salespeople were ranked by their managers as effective or less effective. The interest was focused on how well the sellers could be classified using only the psychological characteristics. The main problems encountered in the original analysis [BDV94] of these data using neural networks were the selection of the architecture and the optimal stopping of the gradient descent. As we have argued before both these problems stem from the use of maximum likelihood estimation. In the following we will demonstrate that these problems are automatically dealt with in the Bayesian framework. To demonstrate the effect of the prior distribution on the learning problem we will focus on Maximum a Posteriori (MAP) estimation, i.e. we find the mode of the posterior distribution and use this to make predictions. Although the proper approach would to use at least multiple modes and preferably a sample form one or more Markov Chain Monte Carlo chains, using MAP estimates allows us to focus on the role of the prior (compare for instance the statement of Zhu and Rohwer [ZR95] that priors for neural networks are often very weak, implying that the main benefits would stem from averaging over the posterior).

To allow comparisons with the original studies, we use the same 3-fold cross–validation as detailed in [BDV94]. For each partition (46 training/23 test) of the data we used the following architecture and parameters of the hyperpriors. Note that the values of the prior controlling the input to hidden weights so as to allow reasonably large values if necessary.

The network used had 6 inputs, 12 hidden units and 1 output unit. We used the likelihood function of eq 3. The prior used was as follows : the bias terms had a Gaussian prior with fixed variance, for the hidden to output weights we used the Neal's prior for infinite networks, and the input to hidden weights had a Automatic Relevance Determination prior based on the formula of eq 6. The following table gives the values for the parameters of the prior:

Input λ_0	0.003
Input α	2.5
Bias terms λ	0.001
Output units λ	$0.01\sqrt{12}$

The MAP estimate was found by training the network using a variable metric optimization method until convergence.

Result on the test set : 6 Errors, 7 Errors, 6 Errors. This gives an averaged error rate of 19/69 = 0.275. This is in agreement with the best results found using other methods.

Next, we examine the weights corresponding to the MAP estimate to see the effect of the prior. The following table shows how the Bayesian framework adapts the network to the data.

	1	2	3	4	5	6	7	8	9	10	11	12
B	0.02	0.02	0.02	6.49	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
I1	0.04	0.04	0.04	-5.14	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
I2	-0.01	-0.01	-0.01	0.57	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
I3	-0.00	-0.00	-0.00	-3.28	-0.03	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
I4	0.00	0.00	0.00	-0.12	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
I5	0.00	0.00	0.00	-2.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I 6	-0.02	-0.01	-0.01	-0.86	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
O	0.05	0.05	0.05	-5.18	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05

Of the 12 hidden units, apparently only one is actually used and the remaining 11 are kept from hurting the predictive performance by setting their corresponding weights to zero. The input–to–hidden weight for hidden unit 4 suggest that inputs 1,3 and 5 make the largest contribution to explaining the data, which is very similar to the result of [BDV94].

6 Conclusions

We have discussed how classification with neural network can benefit from Bayesian methods. For classification problems, better estimates of the posterior probabilities and lower misclassification rates may result. It is important that the predictions are based on multiple modes of the posterior. The technique of combining several runs gave good results.

Appendix A Discretization for Hybrid Monte Carlo

A widely used method to method to implement the discretization is the *leapfrog* algorithm. To move from the state at time t to the new state at a new time $t + \eta$ for a small η the leapfrog algorithm update w and p as follows :

$$p(t + \eta/2) = p(t) - \frac{\eta}{2}\nabla E(w(t)) \quad (13)$$

$$w(t + \eta) = w(t) + \eta p(t + \eta/2) \quad (14)$$

$$p(t + \eta) = p(t + \eta/2) - \frac{\eta}{2}\nabla E(w(t + \eta)) \quad (15)$$

To make a substantial move in a HMC–iteration, one typically applies the leapfrog step a large number (e.g. 1000) times with η set to as large a value as possible without causing a excessive number of rejected iterations. Since each leapfrog requires calculating the derivative of the log–posterior with respect to the weights, it would be interesting to reduce the number of steps needed to keep the acceptance rate high. One approach this problem is to use high–order symplectic integrators, such as the Runge–Kutta–Nyström method.

$B_1 = B_{17}$	0.33742256114297547454	$c_1 = 1 - c_{17}$	0.193820001820970477802
$B_2 = B_{16}$	0.53305963988632443229	$c_2 = 1 - c_{16}$	1.095939312371388830781
$B_3 = B_{15}$	-0.51659631611408440843	$c_3 = 1 - c_{15}$	1.100160082235691977733
$B_4 = B_{14}$	0.65600968737291831534	$c_4 = 1 - c_{14}$	0.103190884032448693119
$B_5 = B_{13}$	0.00406786380988635211	$c_5 = 1 - c_{13}$	0.623088159776038041926
$B_6 = B_{12}$	-0.85399694134759518427	$c_6 = 1 - c_{12}$	0.0851165384752928799728
$B_7 = B_{11}$	-0.00610812346737106375	$c_7 = 1 - c_{11}$	0.757862631166271977223
$B_8 = B_{10}$	0.21297732182168671589	$c_8 = 1 - c_{10}$	0.0114018784755162139177
B_9	0.26632861379051880890	c_9	0.50000000000000000000

Figure 6: The coefficients of the eighth–order RKN method

A s–stage Runge–Kutta–Nyström integrator operates as follows :

1. $W_0 = w(t_n)$, $P_1 = p(t_n)$.
2. for $i = 1, 2, \ldots, s$
   - $W_i = W_{i-1} + h(c_i - c_{i-1})P_i$
   - $P_{i+1} = P_i - hB_i\nabla E(W_i)$
3. $w(t_{n+1}) = W_s + h(c_{s+1} - c_s)P_{s+1}$, $p(t_{n+1}) = P_{s+1}$

where h is the step size, W and P denote the intermediate states, and c_i and B_i are coefficients that need to satisfy certain conditions to make the method symplectic. The coefficients for the RKN integrator as given by [OL94] are shown in figure 6.

References

[BDV94] J.C. Bioch, M. van Dijk, and W. Verbeke. Neural networks : New tools for data analysis? In M. Taylor and P. Lisboa, editors, *Proceedings of Neural Networks Applications and Tools*, pages 28–38. IEEE Computer Society Press, 1994.

[Ber85] J.O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer Verlag, second edition, 1985.

[Bre90] G.L. Bretthorst. An introduction to parameter estimation using Bayesian probability theory. In P.F. Fougère, editor, *Maximum Entropy and Bayesian Methods*, pages 53–79. Kluwer Academic Publishers, 1990.

[BS94] J.M. Bernardo and A.F.M. Smith. *Bayesian Theory.* Wiley, 1994.

[BT73] G.E.P. Box and G.C. Tiao. *Bayesian Inference in Statistical Analysis.* Wiley, 1973.

[BW91] W.L. Buntine and A.S. Weigend. Bayesian back–propagation. *Complex Systems*, 5:603–641, 1991.

[Chi95] H. Chipman. Bayesian variable selection with related predictors. Technical Report Bayes–an/9510001, Graduate School of Business, University of Chigaco, 1995.

[Daw86] A.P. Dawid. Probability forecasting. In S.Kotz, N.L. Johnson, and C.B. Read, editors, *Encyclopedia of Statistical Sciences*, volume 7, pages 210–218. Wiley, 1986.

[DKPR87] S. Duane, A.D. Kennedy, B.J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195:216–222, 1987.

[GCSR95] A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin. *Bayesian Data Analysis.* Texts in Statistical Science. Chapman and Hall, 1995.

[HKP91] J.A. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation.* Addison–Wesley, 1991.

[HL89] D.W. Hosmer and S. Lemeshow. *Applied Logistic Regression.* Wiley, 1989.

[KH91] A. Krogh and J.A. Hertz. A simple weight decay can improve generalization. Technical report, The Niels Bohr Institute, 1991.

[KM95] L. Kuo and B. Malick. Variable selection for regression models. Technical report, Department of Statistics, University of Connectict, 1995.

[Kro92] A.S. Kronfield. Dynamics of Langevin simulations. Technical Report Fermilab–pub–92/133–T, Fermi National Accelerator Laboratory, 1992.

[LT93] M. LeBlanc and R. Tibshirani. Combining estimates in regression and classification. Technical report, University of Toronto, Department of Statistics, November 1993.

[MA92] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases and domain theories. Technical report, University of California, Department of Information and Computer Science, 1992.

[Mac92a] D.J.C. MacKay. The evidence framework applied to classification networks. *Neural Computation*, 4:720–736, 1992.

[Mac92b] D.J.C. MacKay. A practical Bayesian framework for backprop networks. *Neural Computation*, 4:448–472, 1992.

[MGR94] D. Madigan, J. Gavrin, and A.E. Raftery. Enhancing the predictive performance of Bayesian graphical models. Technical Report tr209, University of Washington, Department of Statistics, 1994.

[Nea92] R.M. Neal. Bayesian training of backpropagation networks by the Hybrid Monte Carlo Method. Technical Report CRG-TR-92-1, University of Toronto, Department of Computer Science, 1992.

[Nea93a] R.M. Neal. Bayesian learning via stochastic dynamics. In C.L. Giles, S.J. Hanson, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 475–482. Morgan Kaufmann, 1993.

[Nea93b] R.M. Neal. Probabilistic inference using Markov Chain Monte Carlo methods. Technical Report CRG–TR–93–1, University of Toronto, Department of Computer Science, 1993.

[Nea95] R.M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, 1995.

[OL94] D.I. Okunbor and E.J. Lu. Eighth–order explicit symplectic Runge–Kutta–Nystrom integrators. Technical Report CSC 94–21, University of Missouri–Rolla, September 1994.

[Rip94] B.D. Ripley. Flexible non–linear approaches to classification. In J.H. Friedman and H.Wechsler, editors, *From Statistics to Neural Networks. Theory and Pattern Recognition Applications*. Springer Verlag, 1994.

[Rip96] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[TG95] K. Tumer and J. Ghosh. Theoretical foundations of linear and order statistics combiners for neural pattern classifiers. Technical report, Department of Electrical and Computer Engineering, University of Texas, 1995.

[WCWC93] G. Wahba, G. Chong, Y. Wang, and R. Chappell. Soft Classification, a.k.a. Risk Estimation, via Penalized Log Likelihood and Smoothing Spline Analysis of Variance. Technical Report TR 899, University of Wisconsin, Madison Statistics Department, 1993.

[ZR95] H. Zhu and R. Rohwer. Information geometric measurements of generalisation. Technical Report NCRG/4350, Neural Computing Research Group, Aston University, July 1995.