Numerical Integration by Cubature Formulae in Bayesian Neural Networks

M.C. van Wezel [a] W.A. Kosters [b]

[a] Erasmus University, Faculty of Economical Sciences, Department of Computer Science, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands. E-mail: mvanwezel@few.eur.nl

[b] Leiden Institute of Advanced Computer Science (LIACS), P.O. Box 9512, 2300 RA Leiden, The Netherlands. E-mail: kosters@liacs.nl

Abstract

We compare two methods for approximating the high-dimensional integrals encountered in Bayesian neural networks. The frequently used hybrid Monte Carlo method is contrasted with integration by cubature formulae. We combine the cubature integration method with a nonlinear transformation to make the approximation of the integral easier. An experimental comparison of both integration methods for our problem domain is given.

1 Introduction

Neural networks (NN's) are known as flexible models with the ability to approximate complex decision boundaries and highly nonlinear mappings. Datasets which are used to train neural networks are usually noisy and contain few samples and/or sparse regions. As a consequence, it is impossible to determine the "best" weights for a neural network. It seems more natural to generate a probability distribution over all possible weight vectors. This is exactly what is done in Bayesian neural networks. This probability distribution is based on a combination (by using Bayes' theorem) of a *prior* distribution, a probability distribution expressing our prior beliefs, and the *likelihood* expressing how likely the training data is given a weight vector.

Bayesian neural networks hold a great promise. By using Bayesian inference one is able to cope with the problem of over-fitting and one is forced to make distributional assumptions explicit. Bishop [1] mentions an impressive list of features of Bayesian neural networks. Unfortunately, all usefull quantities (e.g., predictions) take the form of a multidimensional integral, approximation of which is extremely difficult. In the recent past, various authors have attempted to approximate these integrals by using various techniques, among which Gaussian approximations [4] and hybrid Markov Chain Monte Carlo (MCMC) [5]. See Section 3 for a few more words on the latter technique.

In this paper, which is partly based on a chapter of [7], we will attempt to approximate the integrals by using so-called *cubature formulae* combined with a suitable transformation. A cubature formula can be considered to be a multidimensional form of a numerical quadrature formula. It is constructed in such a way that it integrates any polynomial up to a certain degree over a certain region exactly. An extensive list of cubature formulae is given in [6, 2]. Below (in Section 4), we will consider a specific cubature formula and we illustrate how to apply it to the Bayesian-NN problem. The integration problem at hand does not lend itself to the direct application of cubature formulae. To make the approximation easier, we transformed the integral using the multivariate transformation rule with a suitable nonlinear transformation. This transformation process is described in Section 5.

$\mathbf{w} = (w_1, \ldots, w_W)^T$	W-dimensional parameter vector of neural network, consisting of weights and biases.
$n : 1, \ldots, N$	Index for input-target data pairs.
$\mathbf{x}_n = (x_{n1}, \ldots, x_{nm})^T$	n-th input data vector (m-dimensional).
t_n	n-th target value (scalar).
$f(\mathbf{x}_n \vert \mathbf{w})$	Network output for input vector $\mathbf{x}_n$ given weight vector $\mathbf{w}$.

Table 1: Notations for MLP neural networks.

We ended up with an approximation scheme that requires an acceptable amount of computation time. An exprerimental evaluation of our method and the hybrid Markov Chain Monte Carlo method is given in Section 6. Unfortunately, our approximation method performs inferior to the hybrid Monte Carlo method in various respects. Nevertheless, we feel that it is justified to present our results in a paper because of the intrinsic importance of the problem domain.

2 The problem: expectations in Bayesian neural networks

In this section we briefly consider Bayesian inference applied to neural networks. The reader requiring more information on the subject is referred to [1]. We restrict ourselves to regression problems. See [1, 4] for a description of Bayesian neural networks for classification. The neural network type concerned is the well-known multilayer perceptron (MLP). The MLP-related notation that we will use in this paper is summarized in Table 1. For notational convenience we will only consider regression problems with one target variable, but the generalization to multiple targets is straightforward.

In MLP learning, the usual cost function that is minimized is $M(\mathbf{w}) = \alpha E(\mathbf{w}) + \beta E_w(\mathbf{w})$. Here, α and β are weighting factors. The first term is an error term and the second term is a regularization term used to prevent overfitting:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(f(\mathbf{x}_n|\mathbf{w}) - t_n)^2, \qquad E_w(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{W} w_i^2 \ . \tag{1}$$

The error term $E(\mathbf{w})$ can be regarded as minus the log-likelihood for a Gaussian noise model with standard deviation σ_o. The likelihood in this case is

$$P(t_1, \ldots, t_N|\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{w}, \sigma_o) = (\sigma_o\sqrt{2\pi})^{-N} \exp\left\{-\sum_{n=1}^{N}(t_n - f(\mathbf{x}_n|\mathbf{w}))^2/(2\sigma_o^2)\right\}.$$

Similarly, the term βE_w corresponds to a Gaussian distribution with mean 0 and variance $\sigma_w^2 = 1/\beta$, this time over the weights of the neural network. This distribution is a *prior* distribution for the weights:

$$P(\mathbf{w}|\sigma_w) = (\sigma_w\sqrt{2\pi})^{-W} \exp\left\{-\sum_{i=1}^{W} w_i^2/(2\sigma_w^2)\right\}. \tag{2}$$

Given the prior and the likelihood, Bayes' theorem can be applied, leading to the posterior probability distribution $\mathrm{pos}(\mathbf{w})$ of the weights (with log posterior $\mathrm{lpos}(\mathbf{w})$):

$$\begin{aligned} \mathrm{pos}(\mathbf{w}) &\propto \exp\left\{-\sum_{i=1}^{W} w_i^2/(2\sigma_w^2) - \sum_{n=1}^{N}(t_n - f(\mathbf{x}_n|\mathbf{w}))^2/(2\sigma_o^2)\right\} \\ \mathrm{lpos}(\mathbf{w}) &= -\sum_{i=1}^{W} w_i^2/(2\sigma_w^2) - \sum_{n=1}^{N}(t_n - f(\mathbf{x}_n|\mathbf{w}))^2/(2\sigma_o^2) \end{aligned} \tag{3}$$

The proportionality in Equation (3) can be used because the denominator serves merely as a normalizing constant. The parameter σ_w is an example of a *hyper-parameter*. A hyper-parameter is a parameter of a distribution of lower level parameters, in this case the weights and the biases of the neural network. A large value for σ_w represents a very broad prior distribution of the network weights, whereas a small value represents a very narrow distribution. The narrow distribution tends to correspond with strong regularization. The parameter σ_o is treated in the same way. By using hierarchical models, a posterior distribution for hyperparameters can be obtained (see [5]). In this case, a suitable prior is required for the hyperparameters. Since we are interested in the approximation capabilities of various integration algorithms, the inclusion of a second level of inference is not very relevant to us. Thus we use fixed values for σ_w and σ_o.

The posterior distribution can be used to obtain a predictive distribution of the target t_{N+1} given a new input vector $\mathbf{x}_{N+1}$

$$P(t_{N+1}|\mathbf{x}_1,\ldots,\mathbf{x}_{N+1},t_1,\ldots,t_N) = \int P(t_{N+1}|\mathbf{x}_{N+1},\mathbf{w},\sigma_o)\mathrm{pos}(\mathbf{w})\ d\mathbf{w}\,. \tag{4}$$

The mean and variance of t_{N+1} with respect to this distribution,

$$E[t_{N+1}] = \int f(\mathbf{x}_{N+1}|\mathbf{w})\mathrm{pos}(\mathbf{w})\ d\mathbf{w}\ , \tag{5}$$

$$\mathrm{var}[t_{N+1}] = \int f(\mathbf{x}_{N+1}|\mathbf{w})^2\mathrm{pos}(\mathbf{w})\ d\mathbf{w} - E[t_{N+1}]^2 + \sigma_o^2\,, \tag{6}$$

are interesting quantities as they can be seen as “single best guess” and “error bar”. Unfortunately, approximating these integrals is extremely difficult. In the remainder we will compare two approximation methods: hybrid MCMC and cubature formulae combined with a transformation. For other approximation methods, see [4, 3].

3 Hybrid Markov Chain Monte Carlo

This method was introduced by Neal [5]. We do not describe it in detail here. Essentially, the integrals (5) and (6) are approximated by an average of K values sampled according to the predictive distribution:

$$E[t_{N+1}] \approx \frac{1}{K}\sum_{i=1}^{K} f(\mathbf{x}_{N+1}|\mathbf{w}_i)\ , \tag{7}$$

$$\mathrm{var}[t_{N+1}] \approx \frac{1}{K}\sum_{i=1}^{K} f(\mathbf{x}_{N+1}|\mathbf{w}_i)^2 - (\frac{1}{K}\sum_{j=1}^{K} f(\mathbf{x}_{N+1}|\mathbf{w}_j))^2 + \sigma_o^2\ . \tag{8}$$

The points $\mathbf{w}_1,\ldots,\mathbf{w}_K$ are generated using an ergodic Markov Chain with the desired posterior as stationary distribution. Neal incorporates gradient information into the basic MCMC algorithm to achieve a sampling scheme that samples the whole posterior distribution. To keep the dependence of the sampled points as low as possible, the gradient vector of the error function with respect to the network weight vector is complemented with a “momentum” vector, which is used to simulate a trajectory from one sampled point to the next along the error landscape. This method is inspired by Hamiltonian dynamics, used to describe particle dynamics in physical systems. It has to be remarked that Neal does not show that the sampled points in weight space are indeed distributed according to the posterior density. This can be regarded a weakness of the method.

4 Cubature formulae and application to Bayesian neural networks

As stated earlier, a *cubature formula* is an integration formula that integrates all multidimensional polynomials up to a certain degree exactly. We denote a cubature formula by Q and its application to f by Qf. The highest degree of polynomials that are integrated exactly by the formula is called the *degree* of the formula. Cubature formulae are often derived for a specific region of $\mathbb{R}^N$, usually $[-1;1]^N$, but can be applied to affine regions by linear transformations. Thus, cubature formulae are derived to approximate

$$I = \int_{-1}^{1}\int_{-1}^{1}\cdots\int_{-1}^{1} f(x_1, x_2, \ldots, x_n) dx_1 dx_2 \ldots dx_n \approx Qf\,. \tag{9}$$

Cubature formulae often use the concept *fully symmetric sum.* A fully symmetric sum (FSS) is a summation over all permutations, including sign changes, of the arguments of a multidimensional function. For example, consider the application of a three dimensional function $f($. In this case the fully symmetric sum $\sum_{\text{FS}} f(\alpha, 0, 0)$ is:

$$f(\alpha,0,0) + f(0,\alpha,0) + f(0,0,\alpha) + f(-\alpha,0,0) + f(0,-\alpha,0) + f(0,0,-\alpha).$$

The tuple $(\alpha, 0, 0)$ is called the generator of the fully symmetric sum. In general a generator of the form $(\alpha, 0, \ldots, 0)$ generates a FSS with $2N$ terms. A generator with k α values followed by $N-k$ zeros, will generate a fully symmetric sum with $\binom{N}{k} 2^k$ terms.

In this paper, we use the cubature formula

$$I_5(f) = A_5 f(0,\ldots,0) + B_5 \sum_{\text{FS}} f(r,0,\ldots,0) + D_5 \sum_{\text{FS}} f(r,r,0,\ldots,0) \tag{10}$$

of degree 5. The coefficients A_5, B_5, D_5 and r can be found in [6]; they are obtained by solving nonlinear systems of equations.

If we want to integrate over a different hyperrectangle, say $B \subset \mathbb{R}^N$, than the standard integration region $\bar{B} = [-1;1]^N$ we can use a linear transformation $\phi : \bar{B} \to B$ given by $\phi(\mathbf{u}) = \mathbf{A}\mathbf{u} + \mathbf{b}$. In this case $J = \int_B f(\mathbf{u})\, d\mathbf{u} \approx |\det \mathbf{A}| I_5(f \circ \phi)$. Needless to say, we will attempt to perform an integration over a much larger hypercube than the $[-1;1]^N$ hypercube.

When approximating the integrals (5) and (6) with I_5, the posterior density is sampled at the points necessary to evaluate I_5. To prevent numerical problems we multiplied the posterior density by a suitable constant $C = e^c$. The value for $c = \log C$ can for example be chosen to be equal to minus the value of the log posterior at a local minimum in the error landscape. For ease of notation write $\psi = e^{\text{lpos}\circ\phi + c}$. The approximations for integrals (5) and (6) are now

$$\begin{aligned} E[t_{N+1}] \approx \hat{E}[t_{N+1}] &= I_5((f\circ\phi)\psi), && (11)\\ \text{var}[t_{N+1}] \approx \widehat{\text{var}}[t_{N+1}] &= I_5((f\circ\phi)^2\psi) - \hat{E}[t_{N+1}]^2 + \sigma_o^2. && (12) \end{aligned}$$

Here, $((f\circ\phi)\psi)(\mathbf{x}) = f(\mathbf{x}_{N+1}|\phi(\mathbf{x}))\psi(\mathbf{x})$. Remember that $\phi(\mathbf{x})$ is a weight vector situated in our integration region B.

5 Adding a multivariate nonlinear transformation

It turned out in earlier experiments [7] that the direct application of cubature formulae to our problem is not a sensible approach. This can be explained by the fact that the posterior density is extremely low in most regions of the weight space compared to the value in the local optima, as can be illustrated by a simple example. From Equation (3) we see that the SSE (sum of squared errors) is a term in the exponent of the posterior distribution we wish to integrate. Suppose that for a certain weight vector, say $\mathbf{w}_a$, the SSE has a value of 2.0, and for another weight vector $\mathbf{w}_b$ it has a value of 202.0. These SSE values and their ratio are not unrealistic for a trained and untrained network. Now,

$$\frac{e^{-2}}{e^{-202}} = e^{200} \approx 7 \times 10^{86}\ . \tag{13}$$

If we disregard the other factors in the posterior distribution, this fraction tells us that the density of $\mathbf{w}_a$ is 7×10^{86} times larger than the density of $\mathbf{w}_b$. It makes little sense to evaluate expectations based on such extremely unlikely weight vectors as $\mathbf{w}_b$. However, this is exactly what happens during direct application of a cubature formula. No information is used about the location of the local optima in weight space; the formula imposes a "sampling pattern" and this pattern together with the linear transformation ϕ determines which points are sampled.

In this section we attempt to guide this sampling process in the right direction by making use of a suitable nonlinear transformation that maps large regions of low posterior probability onto small regions of high posterior probability. The integrals can be rewritten using the multivariate transformation theorem (which was already used implicitly with the linear transformation ϕ above). The general form of the multivariate transformation theorem can be used for any mapping $\varphi : \bar{A} \subset \mathbb{R}^M \to A \subset \mathbb{R}^M$ that is a continuously differentiable one-to-one mapping. Let $\varphi_\ell : \bar{A} \to \mathbb{R}, \ell = 1, \ldots, M$ denote the M components of φ. The Jacobian matrix of φ is denoted by $J_\varphi(\mathbf{x})$. Here, $\mathbf{x} = (x_1, \ldots, x_M)^T$ denotes a general parameter vector in the integration region and is not to be confused with a neural network input vector. Now, the multivariate transformation theorem is:

Theorem 1 *(Multivariate transformation theorem) Assume the Jacobian matrix $J_\varphi(\mathbf{x})$ is nonsingular on $\bar{A}$, i.e., $\det J_\varphi(\mathbf{x}) \neq 0$ for all $\mathbf{x} \in \bar{A}$. Then, for any integrable function $f : A \subset \mathbb{R}^M \to \mathbb{R}^M$, the composite mapping $|\det J_\varphi|(f \circ \varphi)$ is integrable on $\bar{A}$ and*

$$If = \int_A f(\mathbf{x}) d\mathbf{x} = \int_{\bar{A}} |\det J_\varphi(\mathbf{x})| f(\varphi(\mathbf{x})) d\mathbf{x} \ . \tag{14}$$

We know that the probability density is relatively large in the vicinity of local optima in weight space, i.e., weight vectors of trained networks. The transformation φ we use utilizes this fact by mapping the weight vectors evaluated in the cubature formulae to points in close vicinity to the local optima. Before we present the transformation we introduce some notation.

The dimension of our space was denoted by W, the number of parameters in the neural network. The set of locally optimal weight vectors is denoted $\{\mathbf{o}_i\}$, $i = 1, \ldots, O$, with $\mathbf{o}_i = (o_{i1}, \ldots, o_{iW})^T$, and $d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|$ denotes the Euclidean distance. $s(x)$ denotes the sigmoid function $s(x) = (1 + e^{\gamma x})^{-1}$. The parameter γ determines the steepness of the slope, and has a typical value of 500. This yields a very steep slope, which is necessary in our application. Note that $\frac{\partial}{\partial x} s(x) = -\gamma s(x)(1 - s(x))$. Furthermore we define the function $n(\mathbf{v}, \mathbf{o}_i)$, which indicates whether the optimum $\mathbf{o}_i$ is the *nearest* optimum for vector $\mathbf{v}$ as

$$n(\mathbf{v}, \mathbf{o}_i) = \prod_{j=1, j \neq i}^{O} c\,(\mathbf{v}, \mathbf{o}_i, \mathbf{o}_j), \tag{15}$$

where $c\,(\mathbf{v}, \mathbf{o}_i, \mathbf{o}_j)$ is a function that indicates whether optimum $\mathbf{o}_i$ is *closer* to $\mathbf{v}$ than optimum $\mathbf{o}_j$, defined by

$$c\,(\mathbf{v}, \mathbf{o}_i, \mathbf{o}_j) = s(d(\mathbf{v}, \mathbf{o}_i) - d(\mathbf{v}, \mathbf{o}_i)) \ . \tag{16}$$

In this section $\bar{B}$ denotes the standard region ($[-1;1]^W$), $\tilde{B}$ denotes the affine region, and as before B denotes the actual integration region. So, since $\phi : \bar{B} \to \tilde{B}$ and $\varphi : \tilde{B} \to B$, we have $\varphi \circ \phi : \bar{B} \to B$. Below, we use the following notational convention: $\mathbf{w} \in B, \mathbf{v} \in \tilde{B}, \mathbf{u} \in \bar{B}$.

The transformation $\varphi : \tilde{B} \to B$ that we use in our neural network application is

$$\varphi(\mathbf{v}) = \mathbf{w} = \mathbf{v} + \tau \sum_{i=1}^{O} n(\mathbf{v}, \mathbf{o}_i)\,(\mathbf{o}_i - \mathbf{v}). \tag{17}$$

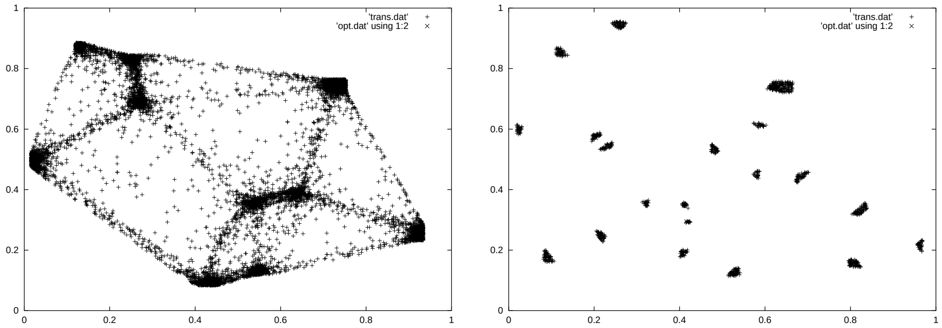

Figure 1: Example of Transformation (17) for the 2-dimensional case with 10 (left) and 20 (right) imaginary optima. The values of γ used to generate these figures were 50 (left) and 5000 (right).

Here, $\tau \in (0;1)$ is a parameter needed to prevent a singular Jacobian matrix [1] (see [7]). This transformation takes most points in the original space towards a position with an arbitrary small distance to the nearest optimum. So, we might say that all points are mapped onto the close proximity of the nearest local optimum. Examples of this transformation for the two dimensional case with 10 and 20 local optima are given in Figure 1. In this figure, only the transformed data are shown. The original data were uniformly distributed in the $[0;1]^2$ plane. It is clearly visible that all data points are located close to the optima after the transformation. The value of τ that was used in this example was 0.9; in our neural network it is typically 0.99, which results in a (desirably) stonger attraction of the local minima.

The effect of the application of the multivariate transformation theorem to the computational effort needed for the evaluation of the cubature formulae is significant: instead of the mapping ψ_r we now use mapping ζ_r, defined by

$$\zeta_r = e^{\mathrm{lpos} \circ \varphi \circ \phi_r + c} \kappa_r, \tag{18}$$

where $\kappa_r(\mathbf{x}) = |\det J_\varphi(\phi_r(\mathbf{x}))|$. The expressions for the approximations of the integral and the expectations that contained the mapping ψ_r (i.e., Equations (11) and (12)) should be altered to incorporate this new mapping.

6 Experiments and results

We use one real life dataset and two artificial datasets in our experiments:
The marketing data set (`marketing`). This data set is concerned with predicting market shares for a brand of a fast moving consumer good. It was used in [7]. It contains 255 patterns with 6 inputs and 1 target. 150 randomly selected patterns were used for training, the rest for testing. The inputs represent marketing variables such as price and product-availability. The target represents the market share as a percentage. Knowing the functional relationship between these variables and the market share has obvious advantages.
Artificial data sets `ds2` and `ds5`. These data sets both contain 100 samples, of which 75 randomly selected were used for training, the rest for testing. Both input and output dimensionality are 1. Inputs were generated randomly (though not uniformly) in the $[-1;1]$ interval. The corresponding targets have a Gaussian distribution with $\sigma = 0.1$ for `ds2` and $\sigma = 0.02$ for `ds5`, around $f(x) = \sin(x)\, x^2$.

In our experiments we compared normal, gradient based neural network learning with Neal's hybrid MCMC method for Bayesian NN's and cubature-based Bayesian NN's. We used the following parameter values for all datasets. All networks had 5 hidden nodes. The ensemble size O used in the nonlinear transformation (17) was

[1]The derivation of the Jacobian matrix for Transformation (17) can be found in [7]

	normal neural network						cubature based Bayesian NN						MCMC based Bayesian NN					
	training data			test data			training data			test data			training data			test data		
dataset	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg
ds2	6.45	8.88	6.97	3.76	4.71	4.12	6.38	6.54	6.46	4.15	4.68	4.35	6.35	6.37	6.36	4.46	4.57	4.50
ds5	0.10	0.13	0.11	0.04	0.05	0.04	0.10	0.17	0.11	0.04	0.10	0.05	0.08	1.57	0.25	0.05	0.95	0.15
marketing	1.26	2.98	2.05	1.09	3.53	2.36	0.86	19.06	3.87	1.23	54.52	8.08	0.81	1.51	1.12	1.29	1.89	1.61

Table 2: Average, minimum and maximum error over 10 independent runs.

method	normal neural network			cubature based Bayesian NN			MCMC based Bayesian NN		
dataset	`ds2`	`ds5`	`marketing`	`ds2`	`ds5`	`marketing`	`ds2`	`ds5`	`marketing`
# parameters	16	16	41	16	16	41	16	16	41
experiment 1	1	1	5	133	182	1954	92	91	302
experiment 2	1	1	4	139	194	1735	93	92	304
experiment 3	0	1	20	135	191	2003	95	95	298

Table 3: Wall clock time in seconds required by the various 'training' methods.

100. The parameters for the hybrid MCMC algorithm were as follows: the number of leapfrog steps was 10000, the mass was 0.1, the stepsize used in the leapfrog method was 0.0002. The Markov chains all had 100 equilibrium states. The neural network specific parameters varied for the three datasets: for `marketing` and `ds2` we used $\sigma_d = 0.1$ and $\sigma_w = 2$. For `ds5` we used $\sigma_d = 0.02$ and $\sigma_w = 2$. Most parameter values were chosen ad-hoc: after some experiments these parameter values seemed to yield satisfying results, but it is quite possible that they are not the optimal values. Note that the length of final part of the Markov chain equals the number of networks used in the ensemble used in the nonlinear transformation: both are 100. This has the consequence that the combined predictions are computed with respect to an equal number of local optima in both methods.

The average, best and worst results obtained in 10 independent runs are shown in Table 2. The values shown represent SSE values on normalised data (i.e., normalised targets with unit standard deviation and mean 0). Some remarks can be made with respect to these results.

Considering the minimal error obtained in the 10 runs, we see that both Bayesian methods slightly outperform normal neural network training as far as the training data is concerned. For the test data, the conjugate gradient based training procedure yields the best results. Looking at the average error, we see that the cubature based method and the MCMC method are very close to one another. Only for dataset `marketing`, the cubature based method seems to be much worse than the MCMC based method. Only the average performance of the MCMC method on dataset `marketing` beats the average performance of normal training.

The amount of time required for the application of the cubature based method is rather high. The cubature rule I_5 evaluates a W-dimensional function at $2W^2+1$ points. In our case the integrand includes a posterior density including a likelihood consisting of N (the number of data vectors) factors. Furthermore, for each sampled weight vector a non-linear transformation has to be applied. Altogether, this leads to considerable time consumption of the cubature based method. The wall clock time required by the various methods in three experiments for all datasets is shown in Table 3. It is clear that the cubature method is by far the most time-consuming.

It is interesting to observe how the integration methods perform when approximating the conditional variances (error bars). Note that the last term of Equation (6) is passed as a parameter to our integration methods. We are interested in the variance that results from the posterior weight distribution (the first two terms of Equation (6)). We created an artificial data set `eb.dat` in which the targets are given by a nonlinear function and the inputs are in the $[-2; 2]$ interval with a void in $[-0.25; 0.25]$. In this $[-0.25; 0.25]$ region we expect "large" errorbars. Figure 2 shows the resulting errorbars for both integration methods at tabulated inputs. These figures show that the cubature method is not suitable for the approximation

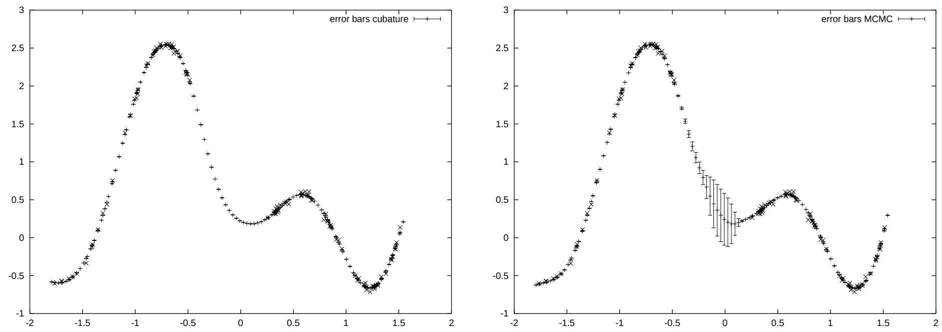

Figure 2: Errorbars yielded by cubature (left) and MCMC method (right) for artificial data set.

of error bars. The resulting error bars, which are hardly visible in the figure, are equally large in all regions of the input space. We found this to be due to the fact that the contributing networks were weighted according to their posterior probability, which resulted in all weight being assigned to the best performing network, and thus only σ_d contributed to the error bars. As a second remark: the size of the errorbars in the Monte Carlo method heavily depends on the value of parameter σ_w that is used (4 in our example).

7 Summary and Conclusions

We applied a cubature formula and a hybrid Monte Carlo method to the approximation of integrals in Bayesian neural networks. The cubature method included a transformation of the integral by the multivariate substitution theorem. We evaluated both methods on various datasets experimentally. The results were in favour of the Monte Carlo method. However, the results yielded by the cubature method are certainly acceptable and further research in this direction may bring improvement, in which case we would have a well founded, well working method as an alternative to hybrid Monte Carlo — an attractive outlook.

References

[1] C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon-Press, Oxford, 1995.

[2] R. Cools and P. Rabinowitz. Monomial cubature rules since 'Stroud' — A compilation. Technical Report Report TW 161, Department of Computer Science, Katholieke Universiteit, Leuven, Belgium, 1991.

[3] J. F. G. de Freitas, M. Niranjan, A. H. Gee, and A. Doucet. Sequential Monte Carlo methods to train neural network models. *Neural Computation*, 12(4):955–993, 2000.

[4] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[5] R. M. Neal. *Bayesian Learning for Neural Networks.* PhD thesis, Department of Computer Science, University of Toronto, 1995.

[6] A. H. Stroud. *Approximate Calculation of Multiple Integrals.* Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[7] M.C. van Wezel. *Neural Networks for Intelligent Data Analysis – Theoretical and experimental aspects.* PhD thesis, Leiden Institute of Advanced Computer Science (LIACS), Leiden University, March 2002.