Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method

Radford M. Neal

Technical Report CRG-TR-92-1
Connectionist Research Group
Department of Computer Science
University of Toronto

e-mail: radford@cs.toronto.edu

10 April 1992

Abstract. It is shown that Bayesian training of backpropagation neural networks can feasibly be performed by the "Hybrid Monte Carlo" method. This approach allows the true predictive distribution for a test case given a set of training cases to be approximated arbitrarily closely, in contrast to previous approaches which approximate the posterior weight distribution by a Gaussian. In this work, the Hybrid Monte Carlo method is implemented in conjunction with simulated annealing, in order to speed relaxation to a good region of parameter space. The method has been applied to a test problem, demonstrating that it can produce good predictions, as well as an indication of the uncertainty of these predictions. Appropriate weight scaling factors are found automatically. By applying known techniques for calculation of "free energy" differences, it should also be possible to compare the merits of different network architectures. The work described here should also be applicable to a wide variety of statistical models other than neural networks.

Introduction

The standard method of training backpropagation neural networks can be interpreted as an implementation of the statistical procedure of maximum likelihood estimation, or, when "weight decay" is used, of maximum penalized likelihood estimation. These procedures find a single "best" set of network parameter values (weights) based on a set of training cases. These parameter values are then used to make predictions for test cases.

The Bayesian approach to statistical inference requires, instead, that prediction for a test case be based on *all* possible values for the network parameters, weighted by the probability of each set of parameters values in light of the training data. Thus, while conventional training is an optimization problem, Bayesian training and prediction is an integration problem. The benefits of the Bayesian approach include the avoidance of "overfitting", an indication of the degree of uncertainty in the predictions, automatic selection of an appropriate scale for network weights, and the ability to sensibly compare the merits of different network architectures.

Recently, MacKay (1991,1992) and Buntine and Weigend (1991) have tackled the Bayesian training problem for backpropagation networks by assuming that the posterior parameter probabilities can be approximated by a Gaussian distribution around the mode. Although the empirical results reported by MacKay are impressive, it is of interest to investigate methods that do not make this assumption.

The Monte Carlo sampling method of Metropolis, *et al* (1953) can be applied to this problem. It is very slow, however, and does not make any use of the gradient information provided by the backpropagation algorithm. More promising is the "Hybrid Monte Carlo" technique of Duane, Kennedy, Pendleton, and Roweth (1987), which incorporates dynamical moves exploiting gradient information within the general framework of the Metropolis algorithm. These methods can be used in conjunction with the "simulated annealing" approach of Kirkpatrick, Gelatt, and Vecchi (1983), in order to avoid prolonged residency in bad local minima,

To begin, I describe the Bayesian formulation of the neural network training problem and previous work on solving this problem using the Gaussian approximation. Next, I present the Metropolis Monte Carlo algorithm and the Hybrid Monte Carlo method, and show how these can be adapted to incorporate simulated annealing. I then formulate the Bayesian training problem for backpropagation networks in a fashion appropriate to these methods. Empirical results on a problem from MacKay (1991, 1992) are presented. Finally, I discuss possible future developments, including the application of techniques developed for computing "free energy" differences to the problem of comparing the merits of different network architectures.

The neural network training problem

In this paper, I will deal only with neural networks used for regression, though the method should be applicable to classification networks as well. Assume we have a set of n independent training items, $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, each of which gives the value of an "output" vector, $\mathbf{y}_i$, associated with an "input" vector, $\mathbf{x}_i$. We also have the value of the input vector for an additional test item, $\mathbf{x}_{n+1}$. The task is to predict the corresponding output vector, $\mathbf{y}_{n+1}$. I assume that the entire training set is simultaneously available when making this prediction — i.e. the training is done in "batch", rather than "on-line", mode.

A neural network of some given architecture and with weight vector $\mathbf{w}$ defines a mapping from an input vector $\mathbf{x}$ to a predicted output vector $\widehat{\mathbf{y}}$ given by $\widehat{\mathbf{y}} = f(\mathbf{x}, \mathbf{w})$. Assuming a Gaussian noise model, the conditional probability distribution for the output vector given the input vector based on this mapping will be as follows:

$$P(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) \quad = \quad (2\pi\sigma^2)^{-\frac{D}{2}} \exp\left(-\frac{|\mathbf{y} - f(\mathbf{x}, \mathbf{w})|^2}{2\sigma^2}\right) \tag{1}$$

where D is the dimension of the output vector, and σ is the level of inherent noise in the outputs. (In this paper, I assume that the noise level is the same for all outputs, but accommodating separate noise levels presents no difficulty.)

The conventional maximum likelihood approach to neural network training is to use some gradient-based algorithm to find the weight vector, $\widehat{\mathbf{w}}$, that maximizes the degree of fit to the data, given by the log-likelihood $L(\mathbf{w})$, defined as

$$L(\mathbf{w}) \;=\; \sum_{i=1}^{n} \log P(\mathbf{y}_i \mid \mathbf{x}_i,\, \mathbf{w}) \;=\; -\sum_{i=1}^{n} \frac{|\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{w})|^2}{2\sigma^2} \;+\; C \tag{2}$$

where C does not depend on $\mathbf{w}$. The output noise level, σ, can also be estimated from the training data. Using this single weight vector, $\widehat{\mathbf{w}}$, the output for a test case with input vector $\mathbf{x}_{n+1}$ is predicted to be $\widehat{\mathbf{y}}_{n+1} = f(\mathbf{x}_{n+1}, \widehat{\mathbf{w}})$. The uncertainty in this prediction due to the inherent noise in the output is given by the estimate of σ, but the uncertainty resulting from error in the estimation of $\widehat{\mathbf{w}}$ is usually not accounted for.

The use of "weight decay" corresponds to maximum penalized likelihood estimation, using a penalized log-likelihood, $L'(\mathbf{w})$, typically defined as

$$L'(\mathbf{w}) \;=\; -\frac{|\mathbf{w}|^2}{2\omega^2} \;-\; \sum_{i=1}^{n} \frac{|\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{w})|^2}{2\sigma^2} \tag{3}$$

Here, the constant ω relates to the expected scale of the weights, and might be set by hand, or by some cross-validation procedure. The inclusion of the penalty term reduces the tendency of maximum likelihood estimation to "overfit" the data — i.e. to model the noise rather than the true regularities.

In the Bayesian approach to statistical prediction, one does not use a single "best" set of network weights, but rather integrates the predictions from all possible weight vectors over the posterior weight distribution, which combines information from the data with a prior bias toward more plausible weights. The best single-valued prediction for a test case with input $\mathbf{x}_{n+1}$ (assuming squared-error loss) is given by

$$\widehat{\mathbf{y}}_{n+1} \;=\; \int_{R^N} f(\mathbf{x}_{n+1}, \mathbf{w})\, P(\mathbf{w} \mid (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))\, d\mathbf{w} \tag{4}$$

where N is the dimension of the weight vector.

This integration of predictions is one way in which the Bayesian approach reduces overfitting (the other is the preference over weights embodied in the prior). With integration, a weight vector that fits the data only slightly better than others will contribute only slightly more to the prediction, rather than completely dominating, as happens with maximum likelihood estimation.

The single-valued prediction of equation (4) contains no indication of uncertainty. It also does not adequately represent situations where the output might plausibly be in any of several disjoint regions. A complete statement concerning the test case output is given by its predictive distribution:

$$\begin{aligned} P(\mathbf{y}_{n+1} \mid \mathbf{x}_{n+1},\, (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)) \;&=\; \int_{R^N} P(\mathbf{y}_{n+1} \mid \mathbf{x}_{n+1},\, (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n),\, \mathbf{w}) \\ &\qquad\qquad P(\mathbf{w} \mid (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))\, d\mathbf{w} \qquad (5) \\ &=\; \int_{R^N} P(\mathbf{y}_{n+1} \mid \mathbf{x}_{n+1},\, \mathbf{w}) \\ &\qquad\qquad P(\mathbf{w} \mid (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))\, d\mathbf{w} \qquad (6) \end{aligned}$$

where $P(\mathbf{y}_{n+1} \mid \mathbf{x}_{n+1},\, \mathbf{w})$ is given by equation (1), with σ assumed known for the moment. This distribution incorporates not only the uncertainty due to inherent noise, but also uncertainty in the weight vector, $\mathbf{w}$.

The posterior probabilities for weight vectors that are used above can be written as

$$
\begin{aligned}
P(\mathbf{w} \mid (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)) &= \frac{P(\mathbf{w})\, P((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n) \mid \mathbf{w})}{P((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))} \\
&= \frac{P(\mathbf{w})\, P(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \mathbf{x}_1, \ldots, \mathbf{x}_n,\, \mathbf{w})}{P(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \mathbf{x}_1, , \ldots, \mathbf{x}_n)} \qquad (7) \\
&= \frac{P(\mathbf{w}) \prod_{i=1}^{n} P(\mathbf{y}_i \mid \mathbf{x}_i,\, \mathbf{w})}{P(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \mathbf{x}_1, , \ldots, \mathbf{x}_n)} \qquad (8)
\end{aligned}
$$

(Note that the network attempts to model only the mapping from inputs to outputs, not the distribution of the inputs themselves.)

To complete the Bayesian formulation of the problem, a prior distribution for the network weights is required. One possibility, analogous to the penalty term in equation (3), is

$$
P(\mathbf{w}) \;=\; (2\pi\omega^2)^{-\frac{N}{2}} \exp\left(-\frac{|\mathbf{w}|^2}{2\omega^2}\right) \qquad (9)
$$

The expected scale of the weights is given by ω, assumed for the moment to be set by hand.

The high-dimensional integrals required to make the predictions given by equations (4) and (6) are analytically intractable and difficult to compute numerically. MacKay (1991,1992) and Buntine and Weigend (1991) have approached this problem by assuming that the posterior weight distribution (equation (8)) can be approximated by a multivariate Gaussian around its mode (actually, around whichever of its many modes has been found by an optimization procedure). If one also assumes that the network function, $f(\mathbf{x}_{n+1}, \mathbf{w})$, is approximately linear with respect to $\mathbf{w}$ in the vicinity of this mode, then the integrals in equations (4) and (6) can feasibly be computed — in fact, the predictive distribution for $\mathbf{y}_{n+1}$ (equation (6)) will be another multivariate Gaussian.

MacKay also uses the Gaussian approximation to calculate the denominator in equation (8):

$$
P(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \mathbf{x}_1, \ldots, \mathbf{x}_n) \;=\; \int_{R^N} P(\mathbf{w}) \prod_{i=1}^{n} P(\mathbf{y}_i \mid \mathbf{x}_i,\, \mathbf{w})\, d\mathbf{w} \qquad (10)
$$

This quantity gives the evidence provided by the training data in favour of the particular model employed, and can be used to select between different modes found by the optimization procedure, between different weight scales, ω, between different noise levels, σ, and between network architectures with different numbers of hidden units and different connectivities.

The Metropolis and Hybrid Monte Carlo methods

An alternative to use of the Gaussian approximation in evaluating the integrals in equations (4) and (6) is numerical integration by a Monte Carlo method. However, Monte Carlo integration using points selected at random from some simple distribution on R^N is infeasible, due to the high dimensionality of the space, and the great variation in the posterior probability density for $\mathbf{w}$. The "Metropolis algorithm" of Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller (1953) is more feasible, but still very slow for this application. However, it forms the basis for the "Hybrid Monte Carlo" method of Duane, Kennedy, Pendleton, and Roweth (1987), which appears more promising.

Suppose that we wish to evaluate

$$\langle g \rangle \;=\; \int_{R^N} g(\mathbf{q})\, P(\mathbf{q})\, d\mathbf{q} \tag{11}$$

The Metropolis algorithm generates a sequence of vectors, $\mathbf{q}_0, \mathbf{q}_1, \ldots$ that form an ergodic Markov chain with stationary distribution $P(\mathbf{q})$. The integral in equation (11) is then approximated as

$$\langle g \rangle \;\approx\; \frac{1}{M} \sum_{t=I}^{I+M-1} g(\mathbf{q}_t) \tag{12}$$

Here, I initial values are discarded, in hopes of reaching the stationary distribution, after which M values of g are averaged. In the limit as M increases, this approximation converges to the true value of $\langle g \rangle$. Unfortunately, it is hard to tell how long it takes to reach the stationary distribution (and hence how large I should be), or how correlated are the values of $\mathbf{q}_t$ at successive iterations (and hence how large M should be). However, difficulties of this sort are probably unavoidable in the sorts of applications where the Metropolis algorithm is used.

Generation of the Markov chain is described in terms of an "energy" function, defined by $E(\mathbf{q}) = -\log(P(\mathbf{q})) - \log(Z_E)$, with Z_E a positive constant chosen for convenience. The algorithm starts with an arbitrary choice for $\mathbf{q}_0$. At each iteration, t, a candidate, $\widetilde{\mathbf{q}}_{t+1}$, for the next state is randomly picked according to some distribution, $P(\widetilde{\mathbf{q}}_{t+1} \mid \mathbf{q}_t)$. This candidate state is accepted if it has lower energy than the previous state; if it has higher energy, it is accepted with probability $\exp(-\Delta E)$, where $\Delta E = E(\widetilde{\mathbf{q}}_{t+1}) - E(\mathbf{q}_t)$. In other words,

$$\mathbf{q}_{t+1} \;=\; \begin{cases} \widetilde{\mathbf{q}}_{t+1} & \text{if } U < \exp(-\Delta E) \\ \mathbf{q}_t & \text{otherwise} \end{cases} \tag{13}$$

with U a random number chosen uniformly from $[0,1)$.

This Markov chain will have $P(\mathbf{q}) = Z_E^{-1} \exp(-E(\mathbf{q}))$ as its stationary distribution as long as the selection of candidate states does not rule out some movements between states or introduce a bias in how often states are visited. A sufficient condition for the process to be unbiased is that it satisfies "detailed balance", which, for the above acceptance rule, requires that the probability of considering $\mathbf{q}_B$ for the state at iteration $t+1$, given that the state at iteration t is $\mathbf{q}_A$, is the same as the probability of considering $\mathbf{q}_A$ for state $t+1$ when state t is $\mathbf{q}_B$. One strategy is to choose candidate states that differ from the current state at only a single component, which component being selected at random. Alternatively, one can try changing all components simultaneously, perhaps by picking a new state from a multivariate Gaussian centred on the current state.

The Metropolis algorithm could be used to evaluate the integrals in equations (4) and (6), with $\mathbf{w}$ playing the role of $\mathbf{q}$, and with an energy function derived from equations (8), (9), and (1), along with a convenient selection of Z_E:

$$\begin{aligned} E(\mathbf{w}) \;&=\; -\log(P(\mathbf{w} \mid (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))) \;-\; \log(Z_E) \\ &=\; \frac{|\mathbf{w}|^2}{2\omega^2} \;+\; \sum_{i=1}^{n} \frac{|\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{w})|^2}{2\sigma^2} \end{aligned} \tag{14}$$

This is essentially the objective for maximum penalized likelihood estimation (equation (3)).

Although much better than the simple Monte Carlo method, the Metropolis algorithm will still be very slow for complex networks. The weights in a neural network generally have strong interactions, so that generating candidate states by perturbing a single weight does not work well — in order to achieve a reasonable acceptance probability, the size of the changes must be very small, and consequently a large number of iterations are required to reach and explore the stationary distribution. Furthermore, recalculating the output of a network after changing a single weight is often almost as expensive as calculating the output from scratch (though this is not true for networks without hidden units, or with only one hidden layer).

Generating a candidate state by randomly perturbing all weights at once does not solve this problem, since a randomly chosen direction in the high-dimensional weight space is unlikely to be close to that desired. What is needed is an elaboration of the Metropolis algorithm that makes use of the gradient information provided by a backpropagation network to find candidate directions in which changes have a high probability of being accepted. The "Hybrid Monte Carlo" method devised by Duane, Kennedy, Pendleton, and Roweth (1987) for use in quantum chromodynamics calculations does this. It also eliminates much of the random walk aspect of the Metropolis algorithm, further speeding exploration of the parameter space.

Employing a further analogy with physical systems, the Hybrid Monte Carlo technique augments the "position" vector, $\mathbf{q}$, with a "momentum" vector, $\mathbf{p}$, of the same dimensionality. The combination is referred to as "phase space". The "potential energy" function, $E(\mathbf{q})$, used in the Metropolis algorithm is extended to a "Hamiltonian" function, $H(\mathbf{q},\mathbf{p})$, that incorporates both "potential" and "kinetic" energy:

$$H(\mathbf{q},\mathbf{p}) \;=\; E(\mathbf{q}) \,+\, \frac{1}{2}\,|\mathbf{p}|^2 \tag{15}$$

The Hybrid Monte Carlo algorithm generates a Markov chain, $(\mathbf{q}_0,\mathbf{p}_0),(\mathbf{q}_1,\mathbf{p}_1),\ldots$, with the stationary distribution

$$\begin{aligned} P(\mathbf{q},\mathbf{p}) &= Z_H^{-1}\exp(-H(\mathbf{q},\mathbf{p})) \\ &= Z_E^{-1}\exp(-E(\mathbf{q}))\,\cdot\,(2\pi)^{-\frac{N}{2}}\exp(-|\mathbf{p}|^2/2) \\ &= P(\mathbf{q})\,P(\mathbf{p}) \end{aligned} \tag{16}$$

The marginal distribution for $\mathbf{q}$ is thus the same as for the Metropolis algorithm, so the value of $\langle g\rangle$ can again be estimated using equation (12). The momentum components have Gaussian distributions, independent of $\mathbf{q}$, and of each other.

This Markov chain is generated using two types of transitions. The "dynamic" moves are (almost) deterministic, and are used to explore surfaces over which H is (approximately) constant, which we wish to visit with equal probabilities. The "stochastic" moves allow the algorithm to explore states with different values of H, with probabilities proportional to $\exp(-H)$. For this, any valid Metropolis-style move capable of changing H would suffice, but a simpler method, used by Duane, *et al*, is to unconditionally replace the current momentum vector, $\mathbf{p}$, by one drawn from the stationary momentum distribution: $P(\mathbf{p}) = (2\pi)^{-\frac{N}{2}}\exp(-|\mathbf{p}|^2/2)$.

The dynamic moves follow Hamilton's equations, which specify the derivatives of $\mathbf{q}$ and $\mathbf{p}$ with respect to a fictitious "time" variable, τ, as follows:

$$\frac{d\mathbf{q}}{d\tau} \;=\; +\frac{\partial H}{\partial \mathbf{p}} \;=\; \mathbf{p} \tag{17}$$

$$\frac{d\mathbf{p}}{d\tau} \;=\; -\frac{\partial H}{\partial \mathbf{q}} \;=\; -\nabla E(\mathbf{q}) \tag{18}$$

A candidate state, $(\tilde{\mathbf{q}}_{t+1}, \tilde{\mathbf{p}}_{t+1})$, is generated by first negating the momentum vector with probability one-half, then following the above dynamics for some predefined period of time, and then again negating the momentum vector with probability one-half. The result is accepted or rejected as in equation (13), except that H is used instead of E:

$$(\mathbf{q}_{t+1}, \mathbf{p}_{t+1}) \;=\; \begin{cases} (\tilde{\mathbf{q}}_{t+1}, \tilde{\mathbf{p}}_{t+1}) & \text{if } U < \exp(-\Delta H) \\ (\mathbf{q}_t, \mathbf{p}_t) & \text{otherwise} \end{cases} \tag{19}$$

with $\Delta H = H(\tilde{\mathbf{q}}_{t+1}, \tilde{\mathbf{p}}_{t+1}) - H(\mathbf{q}_t, \mathbf{p}_t)$, and U a random number chosen uniformly from $[0, 1)$.

In fact, Hamiltonian dynamics leaves H invariant, as does negation of the momentum, so these candidate states would always be accepted if the differential equations were solved exactly. When solved numerically, however, H may change, and the new state may be rejected.

The validity of this procedure is due to two properties of Hamiltonian dynamics. The first is time reversibility — if following the dynamics for some period of time maps $(\mathbf{q}_A, \mathbf{p}_A)$ to $(\mathbf{q}_B, \mathbf{p}_B)$, then it also maps $(\mathbf{q}_B, -\mathbf{p}_B)$ to $(\mathbf{q}_A, -\mathbf{p}_A)$. The second is Liouville's theorem, which states that the volume of a region of phase space does not change as it evolves according to Hamiltonian dynamics. Together, these ensure that the selection of candidate states described above produces detailed balance.

These crucial properties remain true even when equations (17) and (18) are discretized with some non-zero step size, ϵ, provided this is done using the "leapfrog" method:

$$\mathbf{p}(\tau + \tfrac{\epsilon}{2}) \;=\; \mathbf{p}(\tau) \;-\; \tfrac{\epsilon}{2}\,\nabla E(\mathbf{q}(\tau)) \tag{20}$$

$$\mathbf{q}(\tau + \epsilon) \;=\; \mathbf{q}(\tau) \;+\; \epsilon\,\mathbf{p}(\tau + \tfrac{\epsilon}{2}) \tag{21}$$

$$\mathbf{p}(\tau + \epsilon) \;=\; \mathbf{p}(\tau + \tfrac{\epsilon}{2}) \;-\; \tfrac{\epsilon}{2}\,\nabla E(\mathbf{q}(\tau + \epsilon)) \tag{22}$$

The time reversibility of the above is easily verified. The preservation of phase space volume can be confirmed by finding the Jacobian of the transformation from $(\mathbf{q}, \mathbf{p})$ at time τ to $(\mathbf{q}, \mathbf{p})$ at time $\tau + \epsilon$ and verifying that its determinant is one.

Equations (20) to (22) are iterated a number of times, L, in order to generate a candidate state. All but the first and last half-step updates of $\mathbf{p}$ can in fact be merged and implemented as full steps. A value of L significantly greater than one is desirable, since this leads to faster exploration of phase space. The direction of motion is randomly reset between moves, so progress is made at the rate of a random walk. In k moves of a single step each, a distance proportional to $\sqrt{k}$ will be traversed on average. If, instead, k/L moves consisting of L steps each are made, the average distance traversed will be proportional to $L\sqrt{k/L} = \sqrt{kL}$. Of course, if L is too large, the rejection rate might be high, or the stochastic moves will not be done as often as is desirable.

For large values of L, the error in H is almost independent of ϵ up to some critical value, at which point it increases rapidly. This critical value is not known *a priori*, and indeed might vary during the simulation. Accordingly, selecting ϵ at random from some fairly wide distribution seems desirable, in order to ensure that at least some moves are accepted, without pessimistically fixing ϵ at a very small value. Mackenzie (1989) points out that varying the trajectory length by randomly choosing ϵ or L also prevents high correlations from arising when the trajectory time matches the period of some oscillation in the system. In the experiments reported below, L is fixed, but a new value of ϵ is chosen for every iteration, using

$$\epsilon \quad = \quad \epsilon_0 \exp(\nu C) \tag{23}$$

where ϵ_0 and ν are constants specifying the median of the distribution and its spread, and C is a random number drawn from the Cauchy distribution, $P(C) \propto 1/(1+C^2)$.

A beautiful feature of the Hybrid Monte Carlo technique is that the finite discretization of equations (20) to (22) introduces no systematic error whatever into the final results, due to the occasional rejection of moves that increase H. Choices for ϵ and L are thus based solely on their effect on the degree of correlation between samples and on the speed of convergence to the stationary distribution.

Hybrid Monte Carlo with simulated annealing

Since the posterior weight distribution for backpropagation networks typically has many local minima, it seems appropriate to use the technique of "simulated annealing", due to Kirkpatrick, Gelatt, and Vecchi (1983), in the hope of reducing the time required for the algorithm to settle to its stationary distribution, where it is likely to be found in one of the wider and deeper minima. In this method, the rejection probability for candidate transitions (given previously by equation (19)) is relaxed at first, allowing a freer exploration of the state space, and only gradually reduced to its final form.

Simulated annealing is conventionally implemented by introducing a "temperature" parameter which is reduced according to some schedule as the simulation progresses. When the temperature is set to T, the acceptance probability for a candidate transition that increases the total energy by ΔH is $\exp(-\Delta H/T)$, which reduces to the previous value (equation (19)) when, in the end, the temperature is set to one. I use a somewhat different approach, defining the temperature as the average squared magnitude of the momentum components: $T = |\mathbf{p}|^2/N$. The initial state is chosen so that T is high; it gradually declines as a cooling process is simulated, eventually reaching an average value of one when the stationary distribution for $\mathbf{p}$ is reached.

This cooling process is implemented by replacing the stochastic move in the Hybrid Monte Carlo algorithm used by Duane, *et al*, in which a new value for $\mathbf{p}$ is picked from its stationary distribution, by the following update:

$$\mathbf{p}_{t+1} \quad = \quad \alpha|\mathbf{p}_t|\mathbf{u} \; + \; T_0^{\frac{1}{2}}\,(1-\alpha^2)^{\frac{1}{2}}\,\mathbf{n} \tag{24}$$

Here, α is a constant in $[0,1)$, $\mathbf{u}$ is a random vector of unit length with uniformly distributed direction, and $\mathbf{n}$ is a random vector in which each component is picked independently from a Gaussian distribution with unit variance. T_0 is the "heat sink" temperature, and is set to one unless otherwise stated. It is easy to verify that the desired stationary distribution (equation (16)) is invariant under this update, and that it forces the algorithm to explore all parts of phase space.

If this update is done with an α only slightly less than one, the temperature will decline only slowly as the simulation proceeds. For $T \gg 1$, the amount of energy, ΔH, lost in each application of equation (24) will be proportional to T, and hence the change in "entropy", $\Delta S = \Delta H/T$, will be constant. This rate of cooling is advocated by Otten and van Ginneken

(1984), though they implement it in a different fashion.

These stochastic cooling moves are alternated with dynamical moves, in which a candidate state, $(\widetilde{\mathbf{q}}_{t+1}, \widetilde{\mathbf{p}}_{t+1})$ is found by iterating equations (20) to (22) L times, starting with $\mathbf{p}_t$ and $\mathbf{q}_t$. (It is not necessary to explicitly allow for negation of $\mathbf{p}$ before and after this, since the momentum direction is randomized by equation (24) in any case.) The resulting state is accepted or rejected based on the change in H. This decision is not made using equation (19), however, since this would take no account of the current temperature.

Instead, the following approach is used. The N-dimensional position and momentum vectors, $\mathbf{q}$ and $\mathbf{p}$, are constructed so as to contain N_d "dummy" components and N_s "significant" components, with $N = N_d + N_s$, $\mathbf{q} = \langle \mathbf{q}_d, \mathbf{q}_s \rangle$, and $\mathbf{p} = \langle \mathbf{p}_d, \mathbf{p}_s \rangle$. The potential energy function, $E(\mathbf{q})$, depends only on $\mathbf{q}_s$, so the derivative of E with respect to $\mathbf{q}_d$ is zero, and, in fact, there is no need to even store these dummy coordinate values. It will also turn out that only the total magnitude of the dummy momentum components need be stored during the computation, not their individual values. These dummy momentum components contribute to the currently measured temperature, and interact with the other momentum components during the stochastic moves (equation (24)). They also form the basis for the decision to accept or reject the candidate state in the dynamical moves.

This decision is made by considering a transfer of energy to or from the dummy momentum components in order to make up for the change resulting from discretization errors in following the dynamics. The accept/reject decision is not based on the change in H (which is zero, after this compensation), but instead on the change that would result in the volume of phase space accessible to the dummy momentum components with the given energy. An exception is when the transfer would require taking more energy from the dummy components than they possess, in which case the move is rejected.

The kinetic energy in the dummy momentum components is $K_d = \frac{1}{2} r^2$, where $r = |\mathbf{p}_d|$. The volume of R^{N_d} in which K_d is within dK_d of this value is proportional to $r^{N_d - 1} dr \propto K_d^{N_d/2-1} dK_d$. An energy change of ΔH due to the discrete dynamics will be compensated for by a change in kinetic energy in the dummy components from K_d to $K_d - \Delta H$, which implies that the accessible volume of phase space changes by the factor

$$\rho \;=\; (1 - \Delta H / K_d)^{N_d/2-1} \tag{25}$$

Using a rule analogous to those based on energy changes (equations (13) and (19)), dynamical moves for which ρ is greater than one are always accepted. Moves for which ρ is less than one are accepted with probability ρ.

This method produces the same stationary distribution as would be obtained (eventually) without annealing. That the approach to the stationary distribution resembles that of conventional simulated annealing can be seen by noting that $T_d = r^2/N_d = 2K_d/N_d$, the temperature measured using only the dummy momentum components, is approximately equal to, T, the temperature measured using all components, as long as N_d is fairly large. The acceptance probability for a dynamical move increasing H by $\Delta H \ll K_d$ is thus

$$\begin{aligned} \rho \;=\; (1 - \Delta H/K_d)^{N_d/2-1} &= (1 - \Delta H/K_d)^{-1} \cdot (1 - (\Delta H/T_d)/(N_d/2))^{N_d/2} \\ &\approx (1 - (\Delta H/T_d)/(N_d/2))^{N_d/2} \\ &\approx \exp(-\Delta H/T) \end{aligned} \tag{26}$$

Hence, moves that increase H by an amount that is small compared to K_d are accepted with approximately the same probability as in conventional simulated annealing, given the definition being used for temperature.

If the dummy momentum components are randomly drawn from their stationary distribution before each dynamical move, as is the case when α in equation (24) is zero, the probability of accepting a move that increases H by ΔH is the same as given by equation (19), as can be seen by integrating ρ from equation (25) over the resulting distribution for K_d.

For the experiments described below, I set N_d equal to N_s.

This method of annealing can also be used with the plain Metropolis algorithm. Significant and dummy momentum components are maintained as above. Updates using equation (24) are alternated with possible moves from $\mathbf{q}_t$ to a state $\tilde{\mathbf{q}}_{t+1}$ picked as follows:

$$\tilde{\mathbf{q}}_{t+1} \;=\; \mathbf{q}_t \;+\; \epsilon\mathbf{p}_t \tag{27}$$

As above, these moves are accepted or rejected based on the change in phase space volume accessible to the dummy momentum components when the change in E that results is compensated for by a change in K_d. The value of ϵ can be randomly selected using equation (23), as with the Hybrid Monte Carlo method.

Bayesian backpropagation using Hybrid Monte Carlo

In this section I will formulate the Bayesian training problem for backpropagation networks in a form appropriate to the method described above. This requires the selection of a network parameterization with which simulated annealing works well, and the choice of techniques for automatically finding weight scale factors and the output noise level.

If simulated annealing is used with the network weights, $\mathbf{w}$, as position coordinates, and with the energy function of equation (14), the early, high-temperature portion of the simulation will explore values of $\mathbf{w}$ with little regard to either their prior probability or their likelihood given the data. This seems undesirable. Instead, when the temperature is high, we would like the simulation to still favour values for $\mathbf{w}$ that we believe *a priori* to be more likely, while paying less attention to the data. This can be accomplished by transforming to a network parameterization in which the desired prior is a uniform distribution. The prior will then not contribute to the energy function, and hence will not be diluted at high temperature. Essentially, we map the prior in the statistical model to the entropy of the analogous physical system, rather than to its energy.

This approach could be used to implement the Gaussian prior of equation (9), but it is somewhat more convenient to use the similarly-shaped logistic distribution, which is easily obtained by transformation from a uniform distribution on $[0,1]$. Each weight, w_i, is parameterized as follows:

$$w_i \;=\; \frac{s_{g_i}}{1-s_{g_i}} \cdot \log\left(\frac{a_i}{1-a_i}\right) \tag{28}$$

Here, g_i identifies a group of weights that w_i belongs to, with s_{g_i} being a scaling parameter shared by all weights in that group; $s_{g_i}/(1-s_{g_i})$ is roughly analogous to ω in equation (9). The s_g and a_i parameters have uniform prior distributions in the interval $[0,1]$. For purposes of the Hybrid Monte Carlo algorithm, these intervals are extended to $(-\infty,+\infty)$ by replication, with every other replica being reversed — i.e. if q_i is the position coordinate corresponding to a_i, then

$$a_i \;=\; \begin{cases} q_i \;-\; \lfloor q_i \rfloor & \text{if } \lfloor q_i \rfloor \text{ is even} \\ 1 \;-\; (q_i \;-\; \lfloor q_i \rfloor) & \text{if } \lfloor q_i \rfloor \text{ is odd} \end{cases} \tag{29}$$

The reversals ensure that the effect of w_i varies continuously with q_i, at least for weights on connections into sigmoidal units, which saturate as their total input goes to infinity. The derivatives of E with respect to these underlying position coordinates, which are needed to implement the dynamics via equations (20) to (22), can easily be obtained from the derivatives of E with respect to the w_i, which are provided by the basic backpropagation algorithm.

Note that by chosing appropriate values for the a_i, any given set of weight values can be obtained regardless of the values of the scale parameters, s_g. This does not mean that these parameters are redundant, however — they act to bias the network toward having weights with similar magnitudes within each group. Each s_g will tend to a value such that $s_g/(1-s_g)$ is roughly the average magnitude of the weights in group g, since this allows the parameters, a_i, underlying the weights in that group to have values nearer the centre of $[0,1]$, where they may vary over a significant region (with significant probability) without much effect on the values of the w_i, and hence not much effect on the fit to the data. In turn, these values for the s_g discourage weights in their group from having magnitudes larger than the average, since that would require the corresponding a_i to have a value in a small region near 0 or 1, of correspondingly small probability.

Estimating the output noise level, σ, from the data could be done in a fashion similar to that described for the weight scaling factors. However, I have chosen instead to integrate out this parameter analytically, as do Buntine and Weigend (1991). Note that once σ is considered unknown, the conditional distributions for the $\mathbf{y}_i$ given the $\mathbf{x}_i$ are not independent (even with $\mathbf{w}$ fixed), and hence we must use the expression for the posterior given by equation (7), rather than that of equation (8).

It is most convenient to work in terms of $\tau = \sigma^{-2}$. Using equation (1), the total probability of the training outputs for given $\mathbf{w}$ and τ is

$$P(\mathbf{y}_1,\ldots,\mathbf{y}_n \mid \mathbf{x}_1,\ldots,\mathbf{x}_n,\,\mathbf{w},\,\tau) \;=\; \prod_{i=1}^{n} (2\pi)^{-\frac{D}{2}}\,\tau^{\frac{D}{2}}\,\exp(-\tau\,|\mathbf{y}_i - f(\mathbf{x}_i,\mathbf{w})|^2/2) \tag{30}$$

$$\;=\; (2\pi)^{-m/2}\,\tau^{m/2}\,\exp(-\tau s/2) \tag{31}$$

where $m = nD$ is the total number of output values, and $s = \sum_{i=1}^{n} |\mathbf{y}_i - f(\mathbf{x}_i,\mathbf{w})|^2$ is the total squared error for these outputs.

It is convenient to use a Gamma prior for τ, as its form is conjugate to the above:

$$P(\tau) \;=\; \frac{(s_0/2)^{m_0/2}}{\Gamma(m_0/2)}\,\tau^{m_0/2-1}\exp(-\tau s_0/2) \tag{32}$$

with $m_0, s_0 > 0$. Integrating out τ then gives

$$P(\mathbf{y}_1,\ldots,\mathbf{y}_n \mid \mathbf{x}_1,\ldots,\mathbf{x}_n,\,\mathbf{w}) \;=\; \int_0^\infty P(\tau)\,P(\mathbf{y}_1,\ldots,\mathbf{y}_n \mid \mathbf{x}_1,\ldots,\mathbf{x}_n,\,\mathbf{w},\,\tau)\,d\tau$$

$$= \frac{(s_0/2)^{m_0/2}}{\Gamma(m_0/2)\,(2\pi)^{m/2}} \int_0^\infty \tau^{(m_0+m)/2-1} \exp(-\tau(s_0+s)/2)\,d\tau$$

$$= \frac{(s_0/2)^{m_0/2}}{\Gamma(m_0/2)\,(2\pi)^{m/2}} \frac{\Gamma((m_0+m)/2)}{((s_0+s)/2)^{(m_0+m)/2}}$$

$$= \frac{\Gamma((m_0+m)/2)\,s_0^{m_0/2}}{\Gamma(m_0/2)\,\pi^{m/2}} \; (s_0+s)^{-(m_0+m)/2} \tag{33}$$

As an energy function, we can thus use the following (remembering that the parameterization underlying $\mathbf{w}$ is such that the prior is uniform):

$$E(\mathbf{w}) \;=\; \frac{m_0+nD}{2} \left[\log\left(s_0 + \sum_{i=1}^n |\mathbf{y}_i - f(\mathbf{x}_i,\mathbf{w})|^2\right) \;-\; \log\left(s_0\right)\right] \tag{34}$$

The subtraction of $\log(s_0)$ above is done merely to adjust the minimum possible energy to zero for convenience. In the experiments reported below, I use $m_0 = s_0 = 0.1$ in order to produce a very vague prior for τ.

The predictive distribution for a test output given the corresponding test input and the training data is found using equation (5) (equation (6) is not applicable when σ is unknown). Using equation (33) twice, we get

$$P(\mathbf{y}_{n+1} \mid \mathbf{x}_{n+1}, (\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_n,\mathbf{y}_n), \mathbf{w}) \;=\; \frac{P(\mathbf{y}_1,\ldots,\mathbf{y}_n, \mathbf{y}_{n+1} \mid \mathbf{x}_1,\ldots,\mathbf{x}_n, \mathbf{x}_{n+1}, \mathbf{w})}{P(\mathbf{y}_1,\ldots,\mathbf{y}_n, \mid \mathbf{x}_1,\ldots,\mathbf{x}_n, \mathbf{w})}$$

$$= \frac{\Gamma((m_0+m+D)/2)\,(s_0+s)^{(m_0+m)/2}}{\Gamma((m_0+m)/2)\,\pi^{D/2}} \; (s_0+s+s_{n+1})^{-(m_0+m+D)/2} \tag{35}$$

where s_0, m_0, s, and m are as above, and $s_{n+1} = |\mathbf{y}_{n+1} - f(\mathbf{x}_{n+1},\mathbf{w})|^2$. Averaging the above probability density function over the values of $\mathbf{w}$ taken from the simulation approximates the predictive distribution for $\mathbf{y}_{n+1}$ given by equation (5).

Results on a test problem

I have evaluated the feasibility of using the Hybrid Monte Carlo method for Bayesian network training by applying it to the "robot arm" test problem of MacKay (1991, 1992). The plain Metropolis algorithm was also tested, and for both methods the effect of annealing was evaluated as well.

The problem is to learn a mapping from two real-valued inputs representing joint angles for a robot arm to two real-valued outputs that predict the resulting arm position, defined by

$$\widehat{y}_1 \;=\; 2.0\cos(x_1) \;+\; 1.3\cos(x_1+x_2) \tag{36}$$

$$\widehat{y}_2 \;=\; 2.0\sin(x_1) \;+\; 1.3\sin(x_1+x_2) \tag{37}$$

Gaussian noise of standard deviation 0.05 is added to $(\widehat{y}_1, \widehat{y}_2)$ to give the observed position, (y_1, y_2).

As in MacKay's work, this map is approximated using a network with a single intermediate layer of hidden units, whose values are denoted by h_j. The network function $\widehat{\mathbf{y}} = f(\mathbf{x},\mathbf{w})$ is given by

$$\hat{y}_k = w^o_k + \sum_j w^{ho}_{jk} h_j \tag{38}$$

$$h_j = \tanh\left(w^h_j + \sum_i w^{ih}_{ij} x_i\right) \tag{39}$$

The weight vector $\mathbf{w}$ is here split into sub-vectors giving the bias weights for the output units, $\mathbf{w}^o$, the weights on connections from hidden to output units, $\mathbf{w}^{ho}$, the bias weights for the hidden units, $\mathbf{w}^h$, and the weights on connections from input to hidden units, $\mathbf{w}^{ih}$. The weights are parameterized as in equation (28), with three scale parameters — s_1 applying to weights in $\mathbf{w}^{ih}$, s_2 to weights in $\mathbf{w}^h$, and s_3 to weights in $\mathbf{w}^o$ and $\mathbf{w}^{ho}$. This corresponds to MacKay's use of three weight regularizers. Note, however, that the logistic prior I use for weights differs slightly from MacKay's Gaussian prior.

I used the same training and test sets as MacKay. Both data sets contain 200 input-output pairs, with x_1 picked uniformly from the ranges $[-1.932, -0.453]$ and $[+0.453, +1.932]$ and x_2 picked uniformly from the range $[0.534, 3.142]$. A grid of cases including points outside these ranges was also used, in order to test how the uncertainty of the predictions increases for inputs outside the region the training set was drawn from. For the first set of experiments, described next, only the first 100 training items were used.

The first objective was to compare the Hybrid Monte Carlo method with the plain Metropolis algorithm (using the moves of equation (27)), and to determine, for each method, whether the use of simulated annealing provides any benefit. A network architecture with eight hidden units, trained on a set of 100 data items, was used for these tests.

Based on informal experiments, a value of $L = 1000$ for the number of steps in the Hybrid Monte Carlo method was selected. Good results were sometimes obtained with 200 iterations, so this was selected as the duration for the later experiments. The plain Metropolis algorithm was run for 200,000 iterations, which results in the same number of function evaluations (and hence roughly the same computation time, approximately twenty minutes on a machine rated at about 25 MIPS).

Each run began with network weights chosen at random from the prior distribution (which is uniform in the underlying parameterization). For the Hybrid Monte Carlo runs with annealing, a starting temperature of 15 was established by running for 10 iterations with α in equation (24) set to zero and T_0 set to 15. The simulation was then gradually cooled by running for 130 iterations with α set to 0.95 and T_0 set to one. Finally the temperature was dropped to one (if it wasn't there already) by running for 60 iterations with α set to zero and T_0 set to one. For the runs without annealing, α was zero and T_0 was one for all 200 iterations. The runs of the plain Metropolis algorithm were for 1000 times as many iterations in each phase, with α set to 0.99995 instead of 0.95 during the annealing phase (producing the equivalent cooling effect).

The success of each run was evaluated by the average squared error on the 200 item test set, using the integrated prediction given by equation (4). For the Hybrid Monte Carlo method, this integral was estimated as the corresponding average over the last 50 iterations of the simulation. For the plain Metropolis algorithm, every 1000th iteration from the last 50,000 was used. Note that the expected prediction error using the true model is 0.0050.

The form of distribution for the step size, ϵ, given by equation (23) appears to be reasonable for all four techniques. The best values for the constants ϵ_0 and ν are not apparent *a priori*, however, and might well be different in each case. Runs with a range of values for these constants were therefore performed.

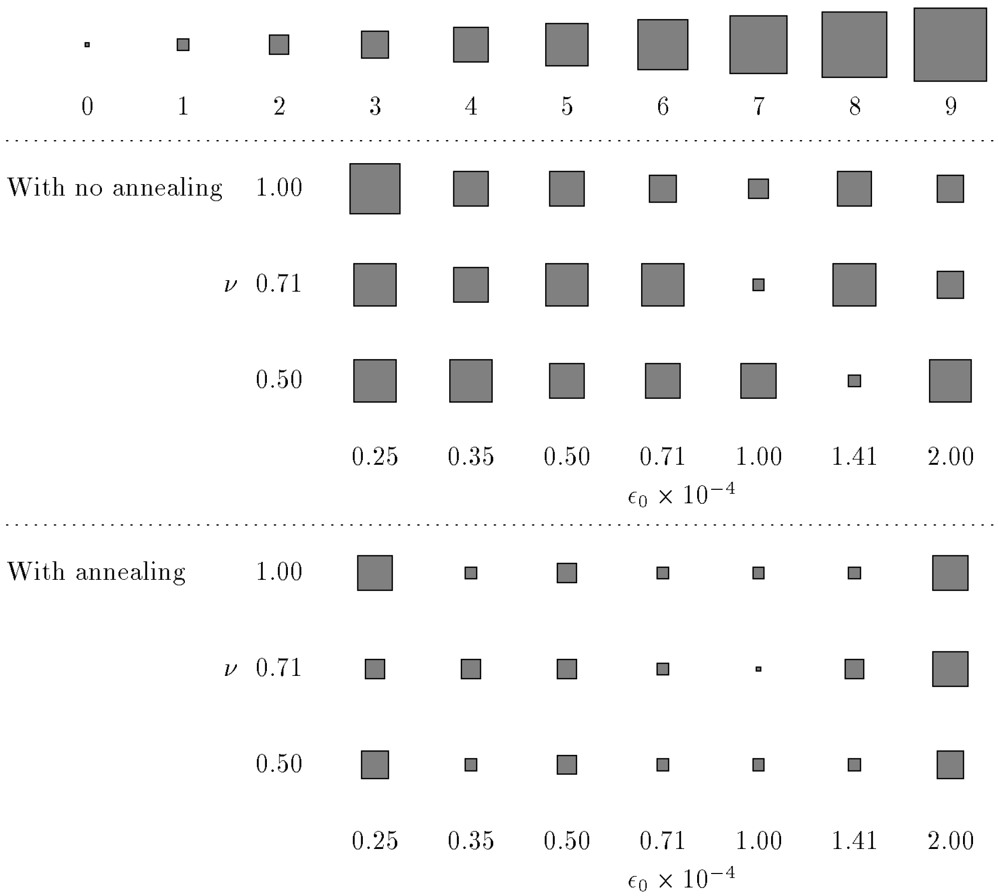

Figure 1: Performance of the Hybrid Monte Carlo method for various parameters of the step size distribution, with and without annealing. The size of each square gives the number of runs out of nine that failed.

For the Hybrid Monte Carlo method, with and without annealing, values for ϵ_0 from 0.25×10^{-4} to 2.00×10^{-4} in geometric steps differing by a factor of $\sqrt{2}$ were tried, in combination with values for ν of 0.50, 0.71, and 1.00. In each case, nine runs with different random number seeds were performed (the same nine seeds for each combination).

The results are summarized in Figure 1. For each combination of ϵ_0 and ν, with and without use of annealing, the number of runs out of nine that failed to achieve a prediction error under 0.0075 is shown. The values $\epsilon_0 = 1.00 \times 10^{-4}$ and $\nu = 0.71$ appear to be best, both with and without annealing. The smallest prediction error obtained using these values was 0.006565 without annealing, and 0.006502 with annealing. Use of annealing clearly improves the probability of success, particularly when the step size distribution is not quite optimal, though good results are sometimes obtained without annealing as well.

The effect of annealing on the progress of the simulations is further illustrated in Figure 2, which shows that runs without annealing often remained stuck for many iterations. Examination of the situation at these times shows this is typically due not to prolonged residence in a local minimum

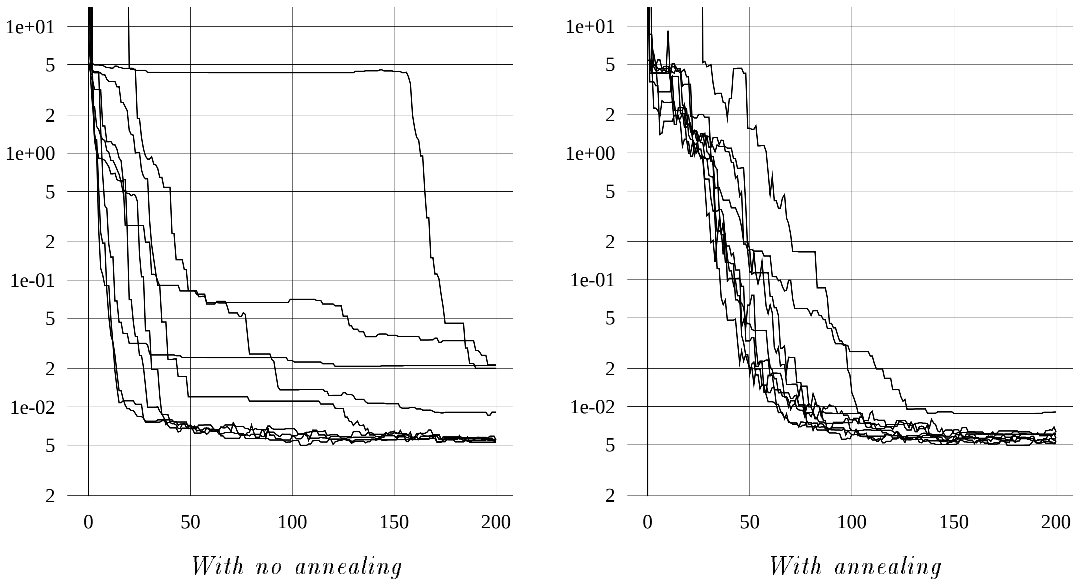

Figure 2: Progress of runs with and without annealing. The average squared error on the training set is plotted for each iteration of the simulation, for each of the nine runs done with $\epsilon_0 = 0.71 \times 10^{-4}$ and $\nu = 0.50$.

of E, but rather to a very high rejection rate (over 95%) arising from the presence of one or more large, tightly-constrained weights. It is possible that with some other parameterization of the weights this problem would be less severe. For the runs that did not suffer from this problem, whether annealed or not, the rejection rates were about 70% at the end of the simulation, and somewhat lower in the earlier stages.

Performance of the plain Metropolis method, with and without annealing, was assessed for values of ϵ_0 ranging from 1×10^{-4} to 32×10^{-4} in geometric steps differing by a factor of 2, in combination with values for ν of 0.50, 0.71, and 1.00. Five runs with different random number seeds were performed for each such combination.

The results were much worse than those using Hybrid Monte Carlo, with the simulations clearly not converging in the allotted time. The best average squared error on the test set achieved by any run was 0.016 (over twice the threshold for "failure" in Figure 1). Best results were obtained in the annealed runs with $\epsilon_0 = 4 \times 10^{-4}$ and $\nu = 0.71$. Annealing was generally somewhat beneficial for all parameter settings.

Further experiments were performed in order to compare the results of using the Hybrid Monte Carlo method with the results MacKay obtained using the Gaussian approximation. These experiments used the full training set of 200 cases and a network with nineteen hidden units.

Five runs of the Hybrid Monte Carlo method with annealing were done, with $L = 1000$, $\epsilon_0 = 0.25 \times 10^{-4}$ and $\nu = 0.9$. The first 20 iterations of each run were done with $\alpha = 0$ and $T_0 = 40$. This was followed by 680 iterations with $\alpha = 0.99$ and $T_0 = 1$, at which point the temperature was forced to one (if not there already) by running for 150 iterations with $\alpha = 0$ and $T_0 = 1$. The networks from the last 100 iterations were used for prediction. These training

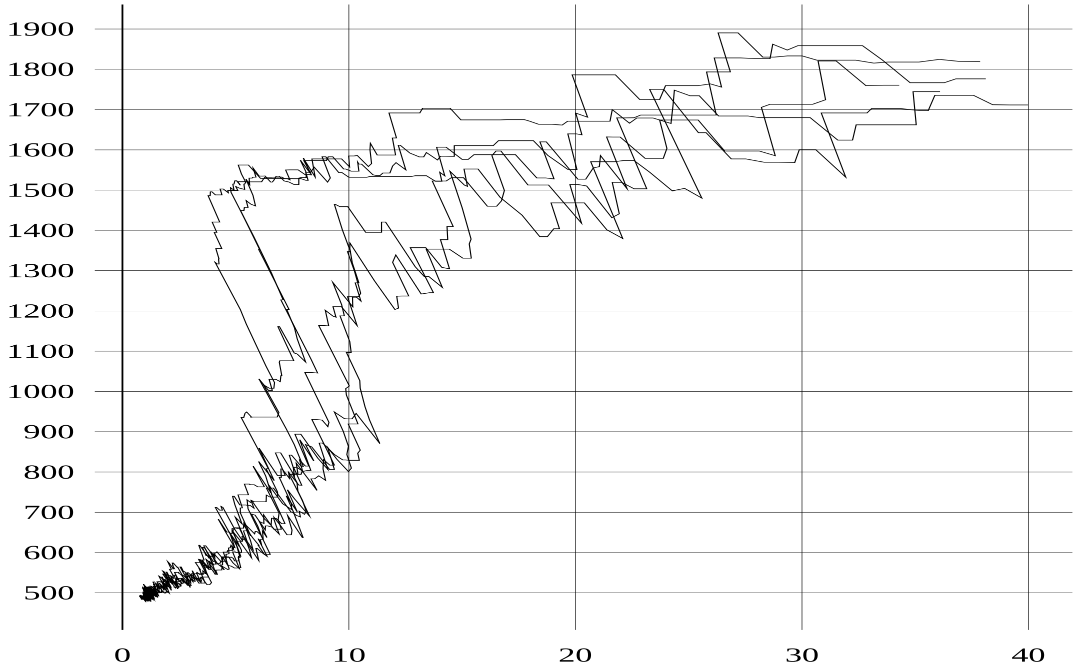

Figure 3: Annealing of the five simulations with the full training set. The horizontal axis gives the temperature, measured from the momentum components. The vertical axis gives the potential energy, related to the squared error on the training set by equation (34).

parameters were arrived at after a few preliminary runs, and are not necessarily optimal. Each run required approximately seven hours on a machine rated at about 25 MIPS.

Figure 3 plots the progress of potential energy versus measured temperature for these runs during the annealing period (iterations 20 through 700). The initial iterations produced states with temperatures near $T_0 = 40$. By the end of the annealing period, the temperatures had declined to near the target final value of $T = 1$, and the energy was also apparently near equilibrium. Note the sharp decline in energy at around $T = 10$ seen in two of the runs, which is reminiscent of a phase transition in a physical system. The other three runs failed to make this transition before the temperature had declined below this level. It is a feature of the method of annealing used that, as seen here, such late transitions result in a rise in temperature toward the level at which the transition should have occurred. This rollback seems desirable, as it returns the annealing process to its proper track.

Figure 4 shows the variation in squared error on the training and test sets for the networks sampled in the last 100 iterations of one of the runs, along with the squared error on the test set using the averaged predictions from all networks sampled up to the given iteration (within the last 100), which approximates the optimal prediction of equation (4). There is clearly a benefit from averaging the predictions, rather than just picking a network at random from the posterior distribution.

The average squared error on the test set, using predictions averaged over the last 100 iterations,

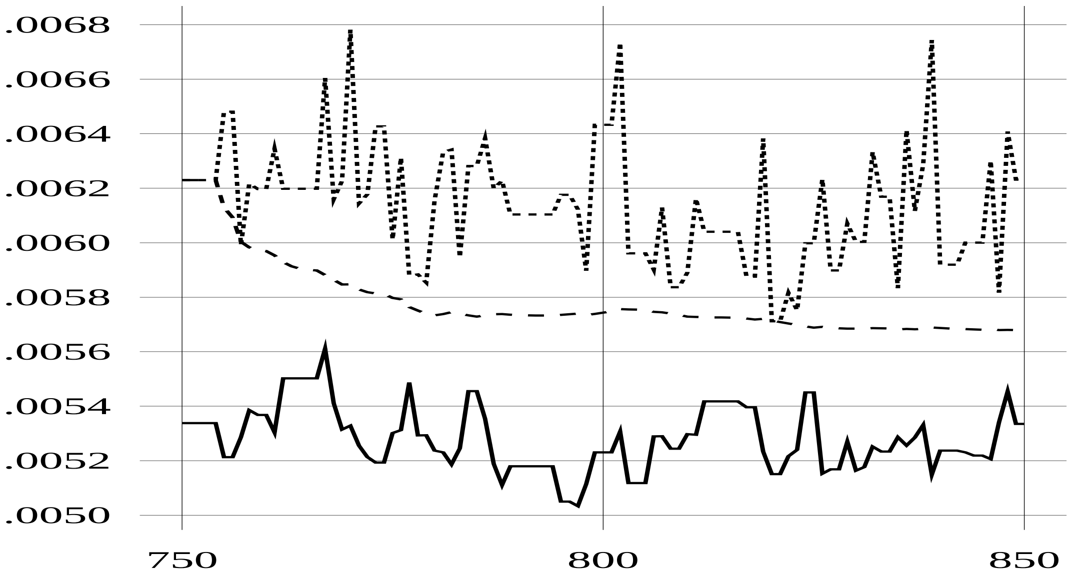

Figure 4: Average squared error on the training and test sets for networks sampled in the last 100 iterations one of the runs. The solid line shows the training error using the network weights at each iteration; the dotted line shows the corresponding error on the test set. The dashed line shows the error on the test set obtained with the averaged predictions of all networks sampled up to that point (within the last 100).

was 0.00547, 0.00568, 0.00571, 0.00571, and 0.00574 for the five runs. In terms of the "test error" measure used by MacKay, which is normalized so that the true model has an expected error of 400 (the number of outputs in the test set), these figures are 438, 454, 457, 457, and 459. The three best runs shown in MacKay (1991) (Figure 3.12) had test errors of 446, 451, and 452. The run he selects as best on the basis of the training data had a test error of 459. The two methods appear to have produced essentially equivalent results. Note that this is a problem for which the Gaussian approximation method was already known to do quite well.

The varying degrees of certainty with which predictions are made are illustrated in Figure 5, which parallels Figure 3.1 of MacKay (1991), which is also for a network with nineteen hidden units. As expected, and as seen in MacKay's results, the uncertainty increases for test cases outside the region from which training data was drawn.

Discussion

In this paper, I have shown how the Hybrid Monte Carlo method can be applied to the Bayesian training of backpropagation networks, and demonstrated the method's feasibility on a simple test problem. The empirical results obtained show that for this application the Hybrid Monte Carlo method is superior to a straightforward application of the Metropolis algorithm. Annealing improved the probability of quickly reaching a good approximation to the stationary distribution, though some runs without annealing also produced good results. The results obtained on the test problem were at least as good as those obtained by MacKay using the Gaussian approximation method.

There is considerable scope for improving on the implementation of the Hybrid Monte Carlo technique that I have used. Integration methods satisfying time-reversibility and Liouville's theorem that are accurate to a higher order than the leapfrog method are possible (see Creutz and Gocksch (1989)), and might improve performance (though they have been disappointing in the quantum chromodynamics application). The kinetic energy term in the Hamiltonian can be generalized to the form $\mathbf{p}^T\mathbf{M}\mathbf{p}$, as is done by Bennett (1975). A good choice for $\mathbf{M}$ will eliminate differences of scale in different directions, thereby permitting a choice of ϵ that allows efficient exploration in all directions. Non-quadratic forms for the kinetic energy could also be considered.

In the present implementation, the requirements for the validity of the method are maintained even during the early iterations (such as during annealing), prior to the time during which states are actually sampled. This is not necessarily desirable — a faster approach to the stationary distribution might be achieved by "cheating" a bit. In particular, the values of L and of ϵ (or of ϵ_0 and ν) might be adapted on the basis of the rejection rate observed. If this adaptation is done slowly, the departure from valid behaviour should be slight. Once sampling starts, the values of these quantities would be kept fixed. A value of $\mathbf{M}$ for the generalized kinetic energy mentioned above could also be found adaptively.

In addition to such elaborations of the Hybrid Monte Carlo method, the way in which the Bayesian training problem is formulated might also be improved. In particular, the scale factors that are currently represented explicitly might be integrated out analytically (as the noise level currently is). This is not done at present because complications then arise in transforming to a parameterization where the prior is uniform.

It will be interesting to compare the performance on real data sets of the Hybrid Monte Carlo method with that of the Gaussian approximation approach. It may well be that each method is superior for a particular class of problems. A combined approach may also be useful, in which the Hybrid Monte Carlo method is used to check whether the posterior is close to being Gaussian in a particular application, and if so, the Gaussian approximation method is used for most of the computations.

One advantage of the Gaussian approximation method is the ease with which one can calculate the probability of the training data under the network model being used (equation (10)). These probabilities can then be used to choose between networks with different architectures (e.g. different numbers of hidden units). Obtaining this information via a Monte Carlo simulation is not so straightforward. The problem is essentially the same as that of calculating "free energy" differences in simulations of physical systems (see Valleau and Torrie (1977) for a review).

All generally applicable solutions to this problem involve some form of "multi-stage sampling", in which the vast gap between the prior and posterior weight distributions is bridged in several steps. The idea can be illustrated by the following decomposition of the desired probability:

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \mathbf{x}_1, \ldots, \mathbf{x}_n) \;=\; \prod_{i=1}^{n} P(\mathbf{y}_i \mid \mathbf{x}_i,\, (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_{i-1}, \mathbf{y}_{i-1})) \tag{40}$$

The conditional probabilities in the above product are the predictive probabilities for each training case, given the preceding cases as training data. It should be possible to evaluate these factors with reasonable efficiency and accuracy by running a simulation for each value of i, and applying equation (35). In contrast, an estimate of the probability of the entire training set obtained by running a single simulation in which the weights are unconstrained by any training data would be hopelessly inaccurate if run for anything less than an astronomical length of time.

Rather than gradually increasing the size of the training set in this fashion, one can instead approach the final state in a number of other ways. One form of the "thermodynamic integration" method, for example, involves gradually reducing the temperature from a high value, where the probability of the data can be simply calculated, to the final desired value of one. If simulated annealing is being used, the data required for this method are available in any case, though for accurate results one must take care that the annealing is sufficiently slow that the simulation stays close to the stationary distribution at all stages.

The Hybrid Monte Carlo method should be generally applicable to any Bayesian statistical problem with continuous parameters for which derivatives of the posterior probability can be calculated. Many problems can be solved by far faster methods, of course. Even when Monte Carlo methods are required, the variant of the Metropolis algorithm known as the "Gibbs Sampler", described by Gelfand and Smith (1990) may be preferable when the conditional distributions for each parameter are easily sampled from. When, as for neural networks, this is not the case, the Hybrid Monte Carlo method may be the only feasible way of sampling from the posterior without making assumptions that may be difficult to justify.

Acknowledgements

I thank David MacKay for a number of helpful comments, and for providing the robot arm data. This work was supported by the Natural Sciences and Engineering Research Council of Canada, and by the Ontario Information Technology Research Centre.

References

Bennett, C. H. (1975) "Mass tensor molecular dynamics", *Journal of Computational Physics*, vol. 19, pp. 267-279.

Buntine, W. L. and Weigend, A. S. (1991) "Bayesian back-propagation", *Complex Systems*, vol. 5, pp. 603-643.

Creutz, M. and Gocksch, A. (1989) "Higher-order hybrid Monte Carlo algorithms", *Physical Review Letters*, vol. 63, pp. 9-12.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987) "Hybrid Monte Carlo", *Physics Letters B*, vol. 195, pp. 216-222.

Gelfand, A. E. and Smith, A. F. M. (1990) "Sampling-based approaches to calculating marginal densities", *Journal of the American Statistical Association*, vol. 85, pp. 398-409.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983) "Optimization by simulated annealing", *Science*, vol. 220, pp. 671-680.

MacKay, D. J. C. (1991) *Bayesian Methods for Adaptive Models*, Ph.D thesis, California Institute of Technology.

MacKay, D. J. C. (1992) "A practical Bayesian framework for backprop networks", to appear in *Neural Computation.*

Mackenzie, P. B. (1989) "An improved hybrid Monte Carlo method", *Physics Letters B*, vol. 226, pp. 369-371.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953) "Equation of state calculations by fast computing machines", *Journal of Chemical Physics*, vol. 21, pp. 1087-1092.

Otten, R. H. J. M. and van Ginneken, L. P. P. P. (1984) "Floorplan design using simulated annealing", *IEEE International Conference on Computer-Aided Design (ICCAD-84).*

Valleau, J. P. and Torrie, G. M. (1977) "A guide to Monte Carlo for statistical mechanics: 2. Byways", in B. J. Berne (editor) *Statistical Mechanics, Part A: Equilibrium Techniques (Modern Theoretical Chemistry, Volume 5)*, New York: Plenum Press.

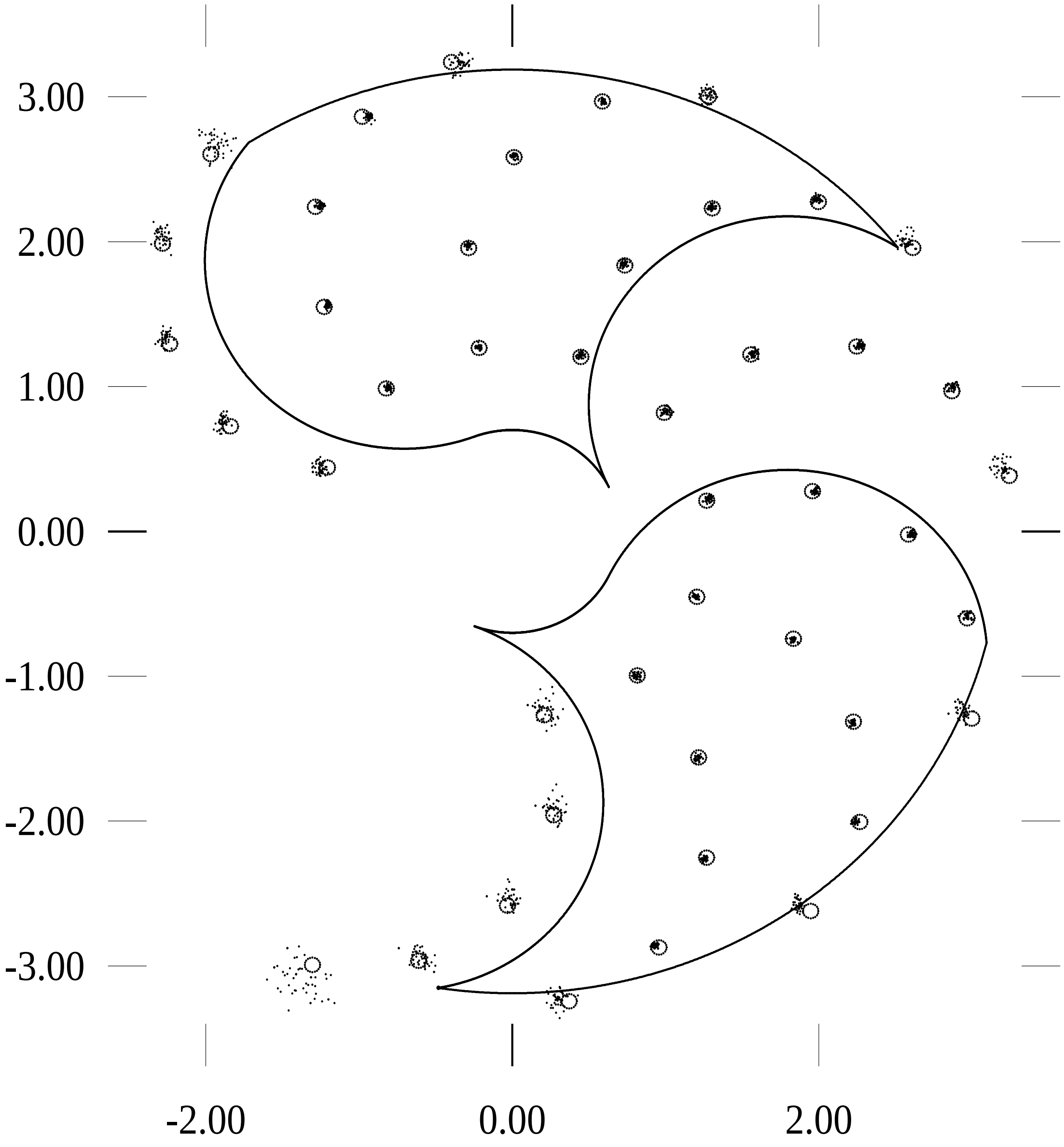

Figure 5: Uncertainty of predictions. This is the output space of the network. The two outlined areas are the regions from which the training data was drawn. The circles indicate the true (noise-free) outputs for a grid of test cases in the input space. The dots in the vicinity are the outputs of every second network from the last 100 iterations of one of the simulation runs. (These predictions do not include the uncertainty due to the estimated amount of inherent noise.)