

# Annealing Contour Monte Carlo for Neural Network Training

Faming Liang

Department of Statistics, Texas A&M University  
College Station, TX 77843, USA

April 21, 2004

## Abstract

In this article we propose a general Monte Carlo optimization algorithm, the so called annealing contour Monte Carlo algorithm. The algorithm is illustrated by a neural network training example and compared with simulated annealing. The new algorithm is superior to simulated annealing in both theory and practice. The numerical results show that the new algorithm has made a drastic improvement over simulated annealing in neural network training in whatever computational times or the quality of the solutions.

## 1 Introduction

The multi-layer perceptrons (MLPs) [1] are perhaps the most well known type of feedforward neural networks. Figure 1 illustrates a MLP with structure 4-3-1 (four input units, three hidden units, and one output unit). In MLP, each unit independently processes the values fed to it by the units in the preceding layer and then presents its output to the units in the next layer for further processing, and the output unit produces an approximate to the target value. Given a group of connection weights  $(\alpha, \beta)$ , the MLP approximator can be written as

$$\hat{f}(x_k|\alpha, \beta) = \varphi'(\alpha_0 + \sum_{i=1}^M \alpha_i \varphi(\beta_{i0} + \sum_{j=1}^p \beta_{ij} x_{kj})), \quad (1)$$

where  $p$  denotes the number of input units,  $M$  denotes the number of hidden units,  $x_k = (x_{k1}, \dots, x_{kp})$  is the  $k^{th}$  input pattern,  $\alpha_i$ 's and  $\beta_{ij}$ 's are the connection weights from the hidden units to the output unit and from the input units to the hidden units, respectively. Here the bias unit is treated as a special unit with a constant input, say, 1. The  $\varphi(\cdot)$  and  $\varphi'(\cdot)$  are the activation functions of the hidden units and the output unit, respectively. In this paper they both are set to the sigmoid function, although they may not be the same necessarily. To force  $\hat{f}$  to converge to the target function, it is usually to minimize the following objective function

$$H(\alpha, \beta) = \sum_{k=1}^N (\hat{f}(x_k|\alpha, \beta) - y_k)^2, \quad (2)$$

where  $y_k$  denotes the target output corresponding to the input pattern  $x_k$ . So the problem of MLP training can be stated to choose the connection weights  $(\alpha, \beta)$  such that the function  $H(\alpha, \beta)$  is minimized. In the following we will call  $H(\alpha, \beta)$  the energy function of the network.

Due to the nonlinearity and high dimensionality of  $H(\alpha, \beta)$ , MLP training has posed as a challenge optimization problem in scientific computing for a long time. Many algorithms have been proposed for the problem in the literature. These algorithms can be divided into two categories, namely, deterministic and stochastic. The deterministic algorithms include backpropagation [2], and some second-order training algorithms [3, 4, 5, 6]. These algorithms suffer from a common difficulty: They tend to converge to local minima, rendering the network learning insufficiently and generalizing poorly. To alleviate this difficulty, a number of algorithms have been proposed to help the network escape from local minima based on some perturbations on the backpropagation process. For example, Ref. [7, 8, 9] propose to perturb the weights by adding noise or based on a stochastic rule, and Ref.[10] proposes to perturb the activation function when backpropagation has converged to local minima. In fact, the effects of these perturbations are limited. They are not effective at enabling the network to escape from local minima and only make the network fail to converge to a global minimum within a reasonable number of iterations. [11, 12].

The most famous stochastic algorithm for neural network training is simulated annealing [13]. Ref.[14, 15] show that simulated annealing can improve the convergence of neural network training. Theoretically, simulated annealing

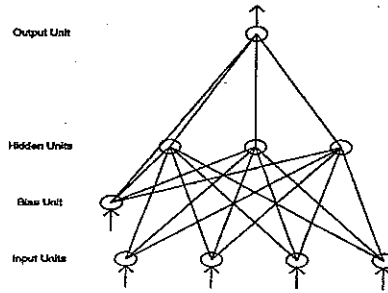


Figure 1: Structure of a MLP with four input units, three hidden units and one output unit. The arrows show the direction of data feeding, where each unit independently processes the values fed to it by the units in the preceding layer and then presents its output to the units in the next layer for further processing.

could converge to the global minimum [16], but the required cooling schedule is so slow that it is impossible to be implemented exactly. This method usually displays slow convergence even for simple learning tasks [17]. The other stochastic algorithms include the genetic algorithm [18], and some MCMC algorithms for Bayesian neural networks [19, 20, 21, 22]. Although the genetic algorithm have produced some results competitive with that of simulated annealing, the theory is not clear if it will converge to the global minimum. Bayesian neural networks are not the focus of this paper.

In this paper, we provide a new Monte Carlo optimization algorithm, the so-called annealing contour Monte Carlo algorithm, for neural network training. We show that the new algorithm is superior to simulated annealing in both theory and practice. The superiority of the new algorithm is illustrated by the parity problem. The numerical results show that the new algorithm has made a drastic improvement over simulated annealing in whatever computational times or quality of the solutions.

## 2 Annealing Contour Monte Carlo

In this section we first describe the general CMC algorithm [23], and then present its space annealing version for optimization problems. Suppose that we want to make an inference for the following Boltzmann distribution,

$$f(x) = \frac{1}{Z} \exp\{-H(x)/\tau\}, \quad x \in \mathcal{X},$$

where  $\tau$  is called the temperature,  $\mathcal{X}$  is called the sample space,  $H(x)$  is called the energy function, and  $Z$  is the normalizing constant of the distribution. CMC works in the style of importance sampling, that is, it may work directly on any non-negative function  $\psi(x)$  defined on the sample space  $\mathcal{X}$ , instead on  $f(x)$  necessarily, although we often set  $\psi(x) = \exp\{-H(x)/\tau\}$  for convenience. For function  $\psi(x)$ , we usually require that  $\int_{\mathcal{X}} \psi(x) dx < \infty$ . Let  $T(\cdot \rightarrow \cdot)$  denote a proposal distribution which can be designed as in the conventional Metropolis-Hastings algorithm [24, 25]. Suppose that the sample space has been partitioned into  $m$  non-overlapped subregions according to some criterion chosen by the user. For example, in all simulations of this paper we partition the sample space according to the energy function, and index the subregions by  $E_1, \dots, E_m$  in ascending order of energy, that is, for any  $x \in E_i$  and  $y \in E_j$ , if  $i < j$  then  $H(x) < H(y)$ . Let  $g(E_i)$  denote the weight associated with the subregion  $E_i$ . The  $g(E_i)$  is determined by our parameter setting and sample space partition as described below. The CMC simulation proceeds in several stages.

Let  $\hat{g}^{(s,k)}(E_i)$  denote the estimate of  $g(E_i)$  at the  $k^{\text{th}}$  iteration of the  $s^{\text{th}}$  stage of the simulation. In the first stage ( $s = 1$ ), the simulation starts with the initial estimates  $\hat{g}^{(0,0)}(E_1) = \dots = \hat{g}^{(0,0)}(E_m) = 1$ , and a random sample  $x_0$  ( $k = 0$ ), and then iterates as follows.

- (a) Propose a new configuration  $x^*$  in the neighborhood of  $x_k$  according to a pre-specified distribution  $T(\cdot \rightarrow \cdot)$ .
- (b) Accept  $x^*$  with probability

$$\min\left\{\frac{\hat{g}^{(s,k)}(E_{I_{x^*}}) \psi(x^*) T(x_k \rightarrow x^*)}{\hat{g}^{(s,k)}(E_{I_{x_k}}) \psi(x_k) T(x^* \rightarrow x_k)}, 1\right\}, \quad (3)$$

where  $I_z$  denotes the index of the subregion where  $z$  belongs to. If it is accepted, set  $x_{k+1} = x^*$ . Otherwise, set  $x_{k+1} = x_k$ .

(c) Update the weights  $\hat{g}(E_i)$ 's. Set  $\hat{g}^{(s,k+1)}(E_{I_{x_{k+1}+i}}) = \hat{g}^{(s,k)}(E_{I_{x_{k+1}+i}}) + \delta_s \rho^i \hat{g}^{(s,k)}(E_{I_{x_{k+1}}})$  for  $i = 0, \dots, m - I_{x_{k+1}}$ .

The algorithm iterates till a stable histogram has been produced in the space of subregions. A histogram is said stable if its shape will not change much with further more samples (a statistic is defined below to measure the stability of the histogram). Once the histogram is stable, we will reset the histogram, reduce the modification factor  $\delta_s$  to a smaller value, set  $s \leftarrow s + 1$ , and proceed to the next stage simulation with the initial estimates  $\hat{g}^{(s,0)}(E_i) = \hat{g}^{(s-1, K_{s-1})}(E_i)$ , where  $K_{s-1}$  is the total number of iterations performed at stage  $s - 1$ . The parameter  $\rho \geq 0$  is a user set parameter. The modification factor  $\delta_1$  is usually set to a large number, for example, 1 or 2, which allows us to reach all subregions very quickly even for a large system. In the followed stages, it will be reduced monotone in a function like  $\delta_{s+1} = \gamma \delta_s$  with  $\gamma < 1$ , or  $\delta_{s+1} = \sqrt{1 + \delta_s} - 1$ . The algorithm will run till  $\delta_s$  has been reduced to a very small value, for example, less than  $10^{-8}$ .

About the convergence of the algorithm, we have the following theorem, of which proof can be found in Ref. [23].

**Theorem 2.1** As  $\delta_s \rightarrow 0$  and  $k \rightarrow \infty$ , for any non-negative function  $\psi(x)$  with  $\int_{\mathcal{X}} \psi(x) dx < \infty$ , we have

$$\hat{g}^{(s,k)}(E_i) \longrightarrow g(E_i) = c \sum_{j=1}^i \rho^{i-j} \int_{E_j} \psi(x) dx = c \left[ \int_{E_i} \psi(x) dx + \rho g(E_{i-1}) \right], \quad \text{almost surely} \quad (4)$$

for  $i = 1, \dots, m$ , where  $c$  is a constant which can be determined by an additional constraint on  $g(E_i)$ 's, for example, one of  $g(E_i)$ 's is equal to a known number.

This theorem implies that the ratios of  $g(E_i)$ 's can be estimated correctly as  $\delta_s \rightarrow 0$  and  $k \rightarrow \infty$ . The convergence in (4) can be intuitively argued as follows. The self-adjusting ability of the CMC move will make it be able to overcome any barriers of the energy landscape, and jump randomly between subregions with (observed) visiting frequency to each subregion being proportional to

$$\frac{\int_{E_i} \psi(x) dx}{g(E_i)}, \quad \text{for } i = 1, \dots, m. \quad (5)$$

By the uniqueness of the stationary distribution of the Markov chain [26], we know as  $\delta_s = 0$ , (4) is the only solution for  $\hat{g}(E_i)$  to converge to such that the simulation will result in the visiting frequencies (5). The self-adjusting ability of the CMC move is a big advantage of the algorithm. With this ability it can overcome any barriers of the energy landscape. While simulated annealing is lack of this ability. Once it is trapped in a local energy minimum, it will be possibly trapped there for ever, as the moving ability of the system decreases as the temperature decreases. In this sense, CMC is superior to simulated annealing in theory.

The convergence can be checked on-line in a single run. For example, we can define the following statistic

$$S_k = \frac{1}{m} \sum_{i=1}^m \left| \frac{\hat{P}_{i,(k+1)b}}{\hat{P}_{i,kb}} - 1 \right|,$$

to measure the stability of the histogram, where  $b$  is the batch size, and  $\hat{P}_{i,kb}$  is the normalized visiting frequency to subregion  $E_i$  calculated at the  $(kb)^{th}$  iteration of a stage. Note we define  $\frac{0}{0} = 1$  in  $S_k$  to accommodate the empty  $E_i$ 's. The  $E_i$ 's are set according to our knowledge to the energy function in prior to the run, so probably we may over-set the maximum or under-set the minimum values of  $H(x)$ . This will cause some empty subregions.

Following from (4) and (5), we know if  $\rho = 0$ , the resulting visiting frequency to each subregion will be

$$P_{\rho=0}(E_i) \propto \text{constant}, \quad i = 1, \dots, m.$$

Hence, CMC will result in a free random walk in the space of subregions. But within the same subregion,  $\psi(x)I(x \in E_i)$  will be simulated from by the Metropolis-Hastings algorithm, where  $I(\cdot)$  is the indicator function. So CMC can be regarded as a generalization of the Wang-Landau algorithm [27] to continuous systems. From the viewpoint of importance sampling, CMC is sampling from the trial density

$$\pi(x) = \sum_{i=1}^m \frac{\psi(x)}{g(E_i)} I(x \in E_i),$$

as  $\delta_s \rightarrow 0$ . If the sample space is partitioned according to the energy function, it is easy to image that  $\pi(x)$  is actually a function defined on a contour plot, a different weight  $g_i$  associates with a different energy level. In this sense, we call the algorithm the contour Monte Carlo algorithm.

If  $\rho > 0$ , CMC will also result in a random walk in the space of subregions, but with more weight toward low indexed regions. If we code the subregions appropriately such that the subregion we want to sample from most intensively is coded as  $E_1$ , and then  $E_2, E_3$ , and so on, a choice  $\rho > 0$  will result in an efficient run. This typically happens in Monte Carlo optimization runs, where we set  $\rho > 0$  to bias the random walk to the low energy region.

Since now we are only interested in minimizing  $H(x)$ , we propose the following space annealing version for CMC to accelerate the optimization process. Suppose the sample space has been partitioned according to the energy function into  $m$  subregions,  $E_1, \dots, E_m$ , where  $E_i$ 's are arranged in ascending order by energy. Let  $I_z$  denote the index of the subregion where a sample  $x$  with energy  $H(x) = z$  belongs to. Let  $M^{(s,k)}$  denote the number of subregions we search from at the  $k^{th}$  iteration of the  $s^{th}$  stage of the simulation, that is, we search from  $\bigcup_{i=1}^{M^{(s,k)}} E_i$  at the  $k^{th}$  iteration of the  $s^{th}$  stage of the simulation. The  $M^{(s,k)}$  starts with  $M^{(1,1)} = m$  and then evolves as  $M^{(s,k)} = I_{H_{\min} + \Delta}$ , where  $H_{\min}$  is the minimum energy value sampled so far in the run, and  $\Delta$  is a user set parameter which controls the search space of each iteration. Since  $H_{\min}$  decreases monotone, the search space shrinks iteration by iteration. In this sense we call the modified CMC algorithm annealing CMC. Of course, the performance of annealing CMC depends on the value of  $\Delta$ . If  $\Delta$  is too large, the algorithm may take a long time to locate the global minimum. If  $\Delta$  is too small, the algorithm may miss the global minimum for ever if our proposal distribution is not very spread.

### 3 Numerical Results

In this section we compared annealing CMC and simulated annealing through one benchmark example, the parity problem. The training set of the  $N$ -parity problem [2] consists of  $2^N$  training pairs, where each of the input patterns consists of  $N$  binary numbers (0 or 1), and the output is 1 if the input pattern consists of an odd number of 1s and 0 otherwise. This problem is a very demanding classification task for MLPs because the target output changes whenever a single bit in the input vector changes [28]. Ref. [29] reported that backpropagation solved the 8-parity problem using a 3-layer MLP with  $2N$  hidden units. Ref. [10] reported that backpropagation and their modified backpropagation solved the 4, 5, 6, and 7-parity problems using 3-layer MLPs with  $2N$  hidden units.

To show the superiority of the new algorithm in optimization, we trained MLPs with  $2N - 2$  hidden units for the  $N$ -parity problems with  $N = 4, \dots, 8$ . Note here we are only interested in the performance comparison of the above two training algorithms, the network used may not be of minimum structure. In annealing CMC, we partition the sample space into subregions  $E_1, \dots, E_{321}$  with an equal energy bandwidth of 0.2, that is, we set  $E_1 = \{x \in \mathcal{X} : H(x) \leq 0.2\}$ ,  $E_2 = \{x \in \mathcal{X} : 0.2 < H(x) \leq 0.4\}$ ,  $\dots$ , and  $E_{321} = \{x \in \mathcal{X} : H(x) > 64\}$ . Note that many of the subregions are empty for the problems with  $N < 8$ . For example, when  $N = 4$ ,  $E_{81}, \dots, E_{321}$  are all empty. In simulation, we set  $\psi(\cdot) = \exp\{-H(\alpha, \beta)\}$ ,  $\Delta = 5$ ,  $\rho = 1.5$ ,  $n_1 = 10^5$ ,  $n_{s+1} = 1.1n_s$ ,  $\delta_1 = e - 1$ ,  $\delta_{s+1} = \sqrt{\delta_s + 1} - 1$ , and  $\delta_{end} = 0.05$ . The simulation starts with  $\delta = \delta_1$  and stops when  $\delta_s < \delta_{end}$  or a solution with  $H(x) < 0.1$  has been located. Each iteration makes one step of local moves (described below). We have two types of local moves which happen equally likely at each iteration. Let  $x_k = (\alpha, \beta)$  denote the current configuration of the MLP. In the type I move, a component of  $x_k$  is picked up at random to undergo a modification by a Gaussian random variable  $\epsilon \sim N(0, \sigma^2)$ . In the type II move, a spherical proposal distribution is used: A direction is generated uniformly, and then the radius is drawn from  $N(0, \sigma^2)$ . The  $\sigma$  is calibrated such that the overall acceptance rate of the local moves is about 0.5 at each stage. We set  $\sigma = 0.5/\sqrt{\log(1 + \delta)}$  for all annealing CMC simulations of this example. Note the variance of the proposal distribution increases as the simulation proceeds. This is due to the property of CMC that it will result in a random walk in the space of subregions. Hence, although the search step size is enlarged as simulation proceeds, the acceptance rate of the local moves will not be affected much. For each  $N$ , annealing CMC was run for 20 times independently. The computational results were summarized in Table 1.

In simulated annealing, we let the temperature decrease geometrically with the highest temperature  $t_{high} = 0.5$  and the temperature decreasing factor  $q = 0.95$ . We set  $n_1 = 500$ , and  $n_i = 1.1n_{i-1}$ , where  $n_i$  denotes the number of iterations at the  $i^{th}$  temperature level. The simulation stops when  $t < 0.025$  or a solution with  $H(x) < 0.1$  has been located. The local moves used are the same with that used in annealing CMC except for the variance of the proposal distribution. We set  $\sigma = 1$  for all simulations of simulated annealing for this example. With this setting, the acceptance rate of the local moves at the high temperature levels is about 0.9 for each value of  $N = 4, \dots, 8$ , and then decreases gradually as temperature decreases. This suggests that our implementation for simulated annealing is effective and efficient. For each value of  $N$ , simulated annealing was run for 20 times independently. The computational results were summarized in Table 1. We have also tried some other settings of  $t_{high}$  and  $\sigma$ , for example,  $t_{high} = 0.5, 1, 2$  and  $\sigma = 1, 2$ , the results are all similar to or worse than that reported in Table 1.

Table 1 shows that annealing CMC has made a drastic improvement over simulated annealing in MLP training, in whatever CPU times or the quality of the solutions. The quality of the solutions can be measured by the "Mean",

Algorithm	N	Mean	SE	Minimum	Maximum	Proportion	Time(s)
ACMC	4	0.076	0.007	0.000	0.100	20	2.75
	5	0.135	0.039	0.029	0.750	18	9.25
	6	0.111	0.030	0.010	0.667	19	35.65
	7	0.284	0.082	0.032	1.334	14	155.1
	8	0.459	0.141	0.024	2.336	13	419.80
SA	4	0.223	0.062	0.024	0.750	15	17.00
	5	0.562	0.154	0.038	2.445	11	66.98
	6	0.514	0.084	0.052	1.000	8	188.89
	7	1.880	0.704	0.056	12.977	8	469.05
	8	4.020	0.954	0.094	19.930	2	1404.45

Table 1: Comparison of annealing CMC (ACMC) and simulated annealing (SA) for parity problems. "Mean" is the sample mean, and "SE" is the standard deviation of the sample mean. They are calculated based on 20 runs. "Minimum" and "Maximum" are the best and worst results, respectively, obtained in the 20 runs. The "Proportion" is the number of solutions (out of 20) with  $H(\alpha, \beta) < 0.1$ . The "Time" is the averaged CPU time (seconds) of each run on a 400MHZ computer.

"Minimum", "Maximum", or "Proportion". In any of these criteria, annealing CMC is consistently better than simulated annealing for each  $N$ . We have also tried other examples, for example, the two-spiral problem [30]. The results also show that annealing CMC has made a significant improvement over simulated annealing in MLP training. Due to the page limit, the details of the computation are omitted.

At last, we would like to mention one point: Just as simulated annealing, annealing CMC is a general optimization algorithm for both discrete and continuous problems. Its application area is as wide as that of simulated annealing.

## References

- [1] D. Rumelhart and J. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge: MIT Press.
- [2] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by back-propagating errors. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Vol. 1, pp.318-362). Cambridge, MA: MIT Press.
- [3] Watrous, R.L. (1987). Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. In *Proceedings of the International Conference in Neural Networks* (pp.619-627). New York: IEEE Press.
- [4] Kramer, A.H. & Sangiovanni-Vincentelli, A. (1988). Efficient parallel learning algorithms for neural networks. In D.S. Touretzky (Ed.), *Advance in Neural Information Processing Systems, 1* (pp.75-89). San Mateo, CA: Morgan Kaufmann.
- [5] Kollias, S. & Anastassiou, D. (1989). An adaptive least squares algorithm for efficient training of artificial neural networks. *IEEE Transactions on Circuits and Systems*, **36**, 1092-1101.
- [6] Battiti, R. & Masulli, F. (1990). BFGS optimization for faster and automated supervised learning. In *Proceedings of the International Neural Networks Conference* (pp.757-760). Paris, France. Dordrecht: Kluwer.
- [7] Von Lehmen, A., Paek, E.G., Liao, P.F., Marrakchi, A., & Patel, J.S. (1988). Factors influencing learning by backpropagation. In *Proceedings of IEEE International Conference on Neural Networks* (pp.335-341). New York: IEEE Press.
- [8] Abunawass, A.M., & Owen, C.B. (1993). A statistical analysis of the effect of noise injection during neural network training. *SPIE Proceedings, 1996*, 362-371.
- [9] Hanson, S.J. (1991). Behavioral diversity, search and stochastic connectionist systems. In M. Commons, S. Grossberg, & J. Staddon (Eds.), *Neural Network Models of Conditioning and Action* (pp.295-345). Mahwah, NJ: Erlbaum.
- [10] Tang, Z., Wang, X., Tamura, H., & Ishii, M. (2003). An algorithm of supervised learning for multilayer neural networks. *Neural Computation*, **15**, 1125-1142.

- [11] Ingman, D., & Merlis, Y. (1991). Local minimization escape using thermodynamic properties of neural networks. *Neural Networks*, 4, 395-404.
- [12] Wang, C. & Principe, J.C. (1999). Training neural networks with additive noise in the desired signal. *IEEE Trans. Neural Networks*, 10, 1511-1517.
- [13] Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimization by simulated annealing. *Science*, 220, 671-680.
- [14] Amato, S., Apolloni, B., Caporali, G., Madiesani, U., & Zanaboni, A. (1991). Simulated annealing approach in backpropagation. *Neurocomputing*, 3, 207-220.
- [15] Owen, C.B., & Abunawass, A.M. (1993). Applications of simulated annealing to the backpropagation model improves convergence, *SPIE Proceedings*, 1966, 269-276.
- [16] Geman, S. & Geman, D. (1984). Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6, 721-741.
- [17] Hassoun, M. (1995). *Fundamentals of Artificial Neural Networks*, Cambridge, MA: MIT Press.
- [18] van Rooij, A.J.F., Jain, L.C., and Johnson, R.P. (1996). *Neural Network Training Using Genetic Algorithms*. Singapore: World Scientific.
- [19] Neal, R.M. (1996). *Bayesian Learning for Neural Networks*. New York: Springer.
- [20] Müller, P. and Insua, D.R. (1998). Issues in Bayesian analysis of neural network models. *Neural Computation*, 10, 749-770.
- [21] de Freitas, N., Niranjan, M., Gee, A.H., and Doucet, A. (2000). Sequential Monte Carlo methods to train neural network models. *Neural Computation*, 12, 955-993.
- [22] Liang, F. (2003). An effective Bayesian neural network classifier with a comparison study to support vector machine. *Neural Computation*, 15, 1959-1989.
- [23] Liang, F. (2003). A contour based Monte Carlo algorithm. *Technical Report*, Department of Statistics, Texas A&M University.
- [24] 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*, 21, 1087-1091.
- [25] Hastings, W.K. (1970). Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika*, 57, 97-109.
- [26] Tierney, L. (1994). Markov chains for exploring posterior distributions (with discussion). *Ann. Statist.*, 22, 1701-1786.
- [27] Wang, F. and Landau, D.P. (2001). Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical Review Letters*, 86, 2050-2053.
- [28] Hjelmas, E. & Munro, P.W. (1999). A comment on the parity problem (Report 7). Gjøvik, Norway: Gjøvik College.
- [29] Tesauro, G. & Janssens, B. (1988). Scaling relations in backpropagation learning. *Complex System*, 2, 39-44.
- [30] Lang, K.J. and Witbrock, M.J. (1988). Learning to tell two spirals apart. In D. Touretzky, G. Hinton, and T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models* (pp.52-59). San Mateo: Morgan Kaufmann.