Lecture Notes for STAT546: Computational Statistics

—Lecture 11: Monte Carlo

Faming Liang

Purdue University

September 25, 2024

Smoothing SAMC for Bayesian model selection

SSAMC is different from SAMC in two aspects. First, the gain factor sequence used in SSAMC is a little more restrictive than that used in SAMC. In SSAMC, the gain factor sequence is required to satisfy the following condition:

(B_1) The sequence $\{\gamma_t\}$ is positive and non-increasing, and satisfies the conditions:

$$(i)\ \overline{\lim_{t\to\infty}} |\gamma_t^{-1} - \gamma_{t+1}^{-1}| < \infty, \quad (ii)\ \sum_{t=1}^{\infty} \gamma_t = \infty, \quad (iii)\ \sum_{t=1}^{\infty} \gamma_t^{\zeta} < \infty, \tag{1}$$

for any $\zeta > 1$.

The trade-off with the condition (A_1) is that a higher order noise term can be included in updating θ_t as prescribed in (6). For example, $\{\gamma_t\}$ can be set as

$$\gamma_t = \frac{t_0}{\max\{t_0, t\}}, \quad t = 1, 2, \ldots, \tag{2}$$

where t_0 is a pre-specified number.

Second, SSAMC allows multiple samples to be generated at each iteration, and employs a smoothed estimate of $p_t^{(i)}$ in updating θ_t, where $p_t^{(i)} = \int_{E_i} f_{\theta_t}(x)dx$ is the limiting probability that a sample is drawn from E_i at iteration t. Let $x_t^{(1)}, \ldots, x_t^{(\kappa)}$ denote the samples generated by a MH kernel with the invariant distribution $f_{\theta_t}(x)$. Since κ is usually a small number, say, 10 to 20, the samples form a sparse frequency vector $\boldsymbol{e}_t = (e_t^{(1)}, \ldots, e_t^{(m)})$ with $e_t^{(i)} = \sum_{l=1}^{\kappa} I(x_t^{(l)} \in E_i)$.

As suggested by many authors, see e.g., Burman (1987) and Hall and Titterington (1987), the frequency estimate can be improved by a smoothing method. Since the partition has been assumed to be smooth, information in nearby subregions can be borrowed to help produce more accurate estimates of $\boldsymbol{p}_t$.

Liang (2009a) smoothed the frequency estimator $\boldsymbol{e}_t/\kappa$ using the Nadaraya-Watson kernel method. Then

$$\widehat{p}_t^{(i)} = \frac{\sum_{j=1}^{m} W\left(\frac{\Lambda(i-j)}{mh_t}\right)\frac{e_t^{(j)}}{\kappa}}{\sum_{j=1}^{m} W\left(\frac{\Lambda(i-j)}{mh_t}\right)}, \tag{3}$$

where $W(z)$ is a kernel function with bandwidth h_t, and Λ is a rough estimate of the range of $\lambda(x)$, $x \in \mathcal{X}$. Here, $W(z)$ is chosen to have a bounded support; that is, there exists a constant C such that $W(z) = 0$ if $|z| > C$. With this condition, it is easy to show $\widehat{p}_t^{(i)} - e_t^{(i)}/\kappa = O(h_t)$. There are many choices for $W(z)$, e.g., an Epanechnikov kernel or a double-truncated Gaussian kernel.

The former is standard, and the latter can be written as

$$W(z) = \begin{cases} \exp(-z^2/2), & \text{if } |z| < C, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The bandwidth h_t is chosen as a power function of γ_t; that is, $h_t = a\gamma_t^b$ for $a > 0$ and $b \in (0, 1]$, where b specifies the decay rate of the smoothing adaptation in SSAMC. For a small value of b, the adaptation can decay very slowly.

In Liang (2009a), $W(z)$ was set to the double-truncated Gaussian kernel with $C = 3$ and

$$h_t = \min\left\{\gamma_t^b,\ \frac{\text{range}\{\lambda(x_t^{(1)}), \ldots, \lambda(x_t^{(\kappa)})\}}{2(1+\log_2(\kappa))}\right\}, \tag{5}$$

where $b = 1/2$, and the second term in $\min\{\cdot,\cdot\}$ is the default bandwidth used in conventional density estimation proceduress.

SSAMC algorithm

(a) *Sampling*: Simulate samples $x_{t+1}^{(1)}, \dots, x_{t+1}^{(\kappa)}$ using the MH algorithm from the distribution $f_{\theta_t}(x)$ as defined in (**??**). The simulation should be done in an iterative manner; that is, generating $x_{t+1}^{(i+1)}$ with a proposal $q(x_{t+1}^{(i)}, \cdot)$, where $x_{t+1}^{(0)} = x_t^{(\kappa)}$.

(b) *Smoothing*: Calculate $\widehat{\boldsymbol{p}}_t = (\widehat{p}_t^{(1)}, \dots, \widehat{p}_t^{(m)})$ in (3).

(c) *Weight updating*: Set

$$\theta_{t+\frac{1}{2}} = \theta_t + \gamma_{t+1}(\widehat{\boldsymbol{p}}_t - \boldsymbol{\pi}). \tag{6}$$

If $\theta_{t+\frac{1}{2}} \in \Theta$, set $\theta_{t+1} = \theta_{t+\frac{1}{2}}$; otherwise, set $\theta_{t+1} = \theta_{t+\frac{1}{2}} + \boldsymbol{c}^*$, where $\boldsymbol{c}^* = (c^*, \dots, c^*)$ can be any vector which satisfies the condition $\theta_{t+\frac{1}{2}} + \boldsymbol{c}^* \in \Theta$.

Table 1: Estimated posterior probability for the change-point example. (Recompiled from Liang, 2009a)

	SSAMC		SAMC		MSAMC		RJMCMC	
k	P(%)	SD	P(%)	SD	P(%)	SD	P(%)	SD
7	0.101	0.002	0.094	0.003	0.098	0.002	0.091	0.005
8	55.467	0.247	55.393	0.611	55.081	0.351	55.573	0.345
9	33.374	0.166	33.373	0.357	33.380	0.223	33.212	0.205
10	9.298	0.103	9.365	0.279	9.590	0.135	9.354	0.144
11	1.566	0.029	1.579	0.069	1.646	0.030	1.569	0.040
12	0.177	0.004	0.180	0.010	0.187	0.004	0.185	0.010
13	0.016	0.001	0.015	0.001	0.017	0.000	0.017	0.001
14	0.002	0.000	0.001	0.000	0.002	0.000	0.001	0.000

Annealing SAMC

Like conventional MCMC algorithms, SAMC is able to find the global energy minima if the run is long enough. However, due to the broadness of the sample space, the process may be slow even when sampling has been biased to low energy subregions. To accelerate the search process, Liang (2007b) proposed to shrink the sample space over iterations.

Suppose that the subregions $E_1, \ldots, E_m$ have been arranged in ascending order by energy; that is, if $i < j$, then $H(x) < H(y)$ for any $x \in E_i$ and $y \in E_j$. Let $\varpi(u)$ denote the index of the subregion that a sample x with energy u belongs to. For example, if $x \in E_j$, then $\varpi(H(x)) = j$. Let $\mathcal{X}_t$ denote the sample space at iteration t. Annealing SAMC initiates its search in the entire sample space $\mathcal{X}_0 = \bigcup_{i=1}^m E_i$, and then iteratively searches in the set

$$\mathcal{X}_t = \bigcup_{i=1}^{\varpi(u_t^* + \aleph)} E_i, \quad t = 1, 2, \ldots, \tag{7}$$

where u_t^* denotes the best function value obtained by iteration t, and $\aleph > 0$ is a user specified parameter which determines the broadness of the sample space at each iteration. Since the sample space shrinks iteration by iteration, the algorithm is called annealing SAMC. Let Θ_t denote the state space of θ_t. In summary, ASAMC consists of the following steps:

Annealing SAMC algorithm

(a) *Initialization*: Partition the sample space $\mathcal{X}$ into m disjoint subregions $E_1, \ldots, E_m$ according to the objective function $H(x)$; specify a desired sampling distribution $\boldsymbol{\pi}$; initialize x_0 by a sample randomly drawn from the sample space $\mathcal{X}$, $\theta_0 = (\theta_0^{(1)}, \ldots, \theta_0^{(m)}) = (0, 0, \ldots, 0)$, $\aleph$, and $\mathcal{X}_0 = \bigcup_{i=1}^m E_i$; and set the iteration number $t = 0$.

(b) *Sampling*: Draw sample x_{t+1} by a single or few MH moves which admit the following distribution as the invariant distribution,

$$f_{\theta_t}(x) \propto \sum_{i=1}^{\varpi(u_t^* + \aleph)} \frac{\psi(x)}{\exp\{\theta_t^{(i)}\}} I(x \in E_i), \tag{8}$$

where $I(x \in E_i)$ is the indicator function, $\psi(x) = \exp\{-H(x)/\tau\}$, and τ is a user-specified parameter.

(c) *Working weight updating*: Update the log-weight θ_t as follows:

$$\theta_{t+\frac{1}{2}} = \theta_t^{(i)} + \gamma_{t+1}\big[I(x_{t+1} \in E_i) - \pi_i\big], \quad i = 1, \ldots, \varpi(u_t^* + \aleph),$$

where the gain factor sequence $\{\gamma_t\}$ is subject to the condition (A_1). If $\theta_{t+\frac{1}{2}} \in \Theta$, set $\theta_{t+1} = \theta_{t+\frac{1}{2}}$; otherwise, set $\theta_{t+1} = \theta_{t+\frac{1}{2}} + \boldsymbol{c}^*$, where $\boldsymbol{c}^* = (c^*, \ldots, c^*)$ and c^* is chosen such that $\theta_{t+\frac{1}{2}} + \boldsymbol{c}^* \in \Theta$.

(d) *Termination*: Check the termination condition, e.g., a fixed number of iterations has been reached. Otherwise, set $t \to t + 1$ and go to step (b).

Learning neural networks for a two-spiral problem

Over the past several decades, feed-forward neural networks, otherwise known as multiple-layer perceptrons (MLPs), have achieved increased popularity among scientists, engineers, and other professionals as tools for knowledge representation. Given a group of connection weights $\boldsymbol{x} = (\alpha, \beta, \gamma)$, the MLP approximator can be written as

$$\hat{f}(\boldsymbol{z}_k|\boldsymbol{x}) = \varphi_o\left(\alpha_0 + \sum_{j=1}^{p}\gamma_j z_{kj} + \sum_{i=1}^{M}\alpha_i\varphi_h\big(\beta_{i0} + \sum_{j=1}^{p}\beta_{ij}z_{kj}\big)\right), \tag{9}$$

where M is the number of hidden units, p is the number of input units, $\boldsymbol{z}_k = (z_{k1}, \ldots, z_{kp})$ is the kth input pattern, and α_i, β_{ij} and γ_j are the weights on the connections from the ith hidden unit to the output unit, from the jth input unit to the ith hidden unit, and from the jth input unit to the output unit, respectively.

The connections from input units to the output unit are also called the shortcut connections. In (9), the bias unit is treated as a special input unit with a constant input, say 1. The functions $\varphi_h(\cdot)$ and $\varphi_o(\cdot)$ are called the activation functions of the hidden units and the output unit, respectively. Popular choices of $\varphi_h(\cdot)$ include the sigmoid function and the hyperbolic tangent function. The former is defined as $\varphi_h(z) = 1/(1 + e^{-z})$ and the latter $\varphi_h(z) = \tanh(z)$. The choice of $\varphi_o(\cdot)$ is problem dependent. For regression problems, $\varphi_o(\cdot)$ is usually set to the linear function $\varphi_o(z) = z$; and for classification problems, $\varphi_o(\cdot)$ is usually set to the sigmoid function.

The problem of MLP training is to minimize the objective function

$$H(\boldsymbol{x}) = \sum_{k=1}^{N} \left(y_k - \widehat{f}(\boldsymbol{z}_k|\boldsymbol{x}) \right)^2 + \lambda \left[\sum_{i=0}^{M} \alpha_i^2 + \sum_{i=1}^{M} \sum_{j=0}^{p} \beta_{ij}^2 + + \sum_{j=1}^{p} \gamma_j^2 \right], \tag{10}$$

by choosing appropriate connection weights, where y_k denotes the target output corresponding to the input pattern $\boldsymbol{z}_k$, the second term is the regularization term, and λ is the regularization parameter. The regularization term is often chosen as the sum of squares of the connection weights, which stabilizes the generalization performance of the MLP. Henceforth, $H(\boldsymbol{x})$ will be called the energy function of the MLP.

Table 2: Comparison of ASAMC, SAMC, SA and BFGS for the two-spiral example. (Liang, 2007b)

Algorithm	Mean	SD	Min	Max	Prop	Iteration($\times 10^6$)	Time
ASAMC	0.620	0.191	0.187	3.23	15	7.07	94m
SAMC	2.727	0.208	1.092	4.089	0	10.0	132m
SA-1	17.485	0.706	9.02	22.06	0	10.0	123m
SA-2	6.433	0.450	3.03	11.02	0	10.0	123m
BFGS	15.50	0.899	10.00	24.00	0	—	3s

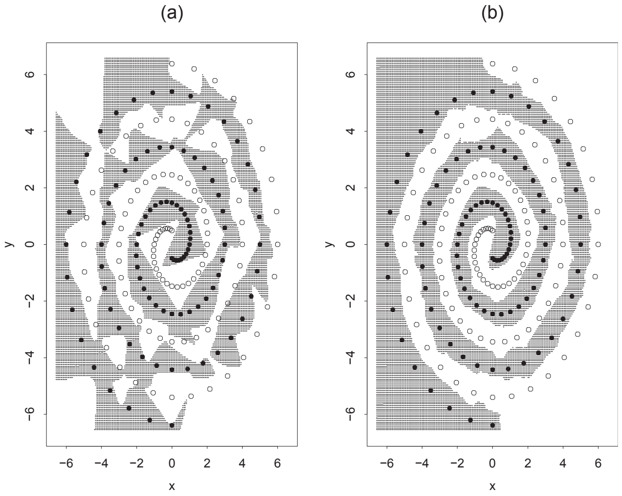

Figure 1: Classification maps learned by ASAMC with a MLP of 30 hidden units. The black and white points show the training data for two different spirals. (a) Classification map learned in one run. (b) Classification map averaged over 20 runs. This figure demonstrates the success of ASAMC in minimization of complex functions. (Liang, 2007b)

Annealing Evolutionary SAMC

Like the genetic algorithm, AESAMC works on a population of samples. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ denote the population, where n is the population size, and $x_i = (x_{i1}, \ldots, x_{id})$ is a d-vector and is called an individual or chromosome in terms of genetic algorithms. Clearly, the minimum of $H(x)$ can be obtained by minimizing the function $\boldsymbol{H}(\boldsymbol{x}) = \sum_{i=1}^{n} H(x_i)$. An unnormalized Boltzmann density can be defined for the population as follows,

$$\psi(\boldsymbol{x}) = \exp\{-\boldsymbol{H}(\boldsymbol{x})/\tau\}, \quad \boldsymbol{x} \in \mathcal{X}^n, \tag{11}$$

where $\mathcal{X}^n = \mathcal{X} \times \cdots \times \mathcal{X}$ is a product sample space. The sample space can be partitioned according to the function $\boldsymbol{H}(\boldsymbol{x})$ into m subregions: $\boldsymbol{E}_1 = \{x : \boldsymbol{H}(\boldsymbol{x}) \leq u_1\}$, $\boldsymbol{E}_2 = \{x : u_1 < \boldsymbol{H}(\boldsymbol{x}) \leq u_2\}$, $\ldots$, $\boldsymbol{E}_{m-1} = \{x : u_{m-2} < \boldsymbol{H}(\boldsymbol{x}) \leq u_{m-1}\}$, and $\boldsymbol{E}_m = \{x : \boldsymbol{H}(\boldsymbol{x}) > u_{m-1}\}$, where $u_1 < u_2 < \ldots < u_{m-1}$ are $m-1$ known real numbers. Note that the sample space is not necessarily partitioned according to the function $\boldsymbol{H}(\boldsymbol{x})$. For example, it can also be partitioned according to $\lambda(\boldsymbol{x}) = \min\{H(x_1), \ldots, H(x_n)\}$.

The population can then evolve under the framework of ASAMC with an appropriate proposal distribution for the MH moves. At iteration t, the MH moves admit the following distribution as the invariant distribution,

$$\boldsymbol{f}_{\theta_t}(\boldsymbol{x}) \propto \sum_{i=1}^{\varpi(\boldsymbol{u}_t^*+\aleph)} \frac{\psi(\boldsymbol{x})}{\exp\{\theta_t^{(i)}\}} I(\boldsymbol{x} \in \boldsymbol{E}_i), \quad \boldsymbol{x} \in \mathcal{X}_t^n, \tag{12}$$

where $\boldsymbol{u}_t^*$ denotes the best value of $\boldsymbol{H}(\boldsymbol{x})$ obtained by iteration t.

Table 3: Average optimality gap values over the 40 test functions. (Liang, 2009d)

Algorithm	5000[a]	10000[a]	20000[a]	50000[a]
Genocop III[b]	636.37	399.52	320.84	313.34
Scatter Search[b]	4.96	3.60	3.52	3.46
C-GRASP[c]	6.20	4.73	3.92	3.02
DTS[d]	4.22	1.80	1.70	1.29
ASAMC[e]($\aleph = 10$)	4.11(0.11)	2.55(0.08)	1.76(0.08)	1.05(0.06)
ASAMC[e]($\aleph = 5$)	3.42(0.09)	2.36(0.07)	1.51(0.06)	0.94(0.03)
ASAMC[e]($\aleph = 1$)	3.03(0.11)	2.02(0.09)	1.39(0.07)	0.95(0.06)
SA[e]($t_{high} = 20$)	3.58(0.11)	2.59(0.08)	1.91(0.06)	1.16(0.04)
SA[e]($t_{high} = 5$)	3.05(0.11)	1.80(0.09)	1.17(0.06)	0.71(0.03)
SA[e]($t_{high} = 2$)	2.99(0.12)	1.89(0.09)	1.12(0.06)	0.69(0.03)
SA[e]($t_{high} = 1$)	2.36(0.11)	1.55(0.08)	1.06(0.06)	0.67(0.03)
SA[e]($t_{high} = 0.5$)	2.45(0.11)	1.39(0.07)	1.06(0.07)	0.75(0.06)
AESAMC[f]($\aleph = 10$)	1.59(0.08)	0.82(0.02)	0.68(0.01)	0.50(0.01)
AESAMC[f]($\aleph = 1$)	1.93(0.27)	0.79(0.01)	0.66(0.02)	0.49(0.01)

Trajectory Averaging: Toward optimal convergence rate

Consider the stochastic approximation algorithm:

$$\theta_{t+1} = \theta_t + \gamma_{t+1} H(\theta_t, X_{t+1}) + \gamma_{t+1}^{1+\tau} \eta(X_{t+1}), \tag{13}$$

where, as previously, γ_{t+1} denotes the gain factor, $\tau > 0$, $\eta(\cdot)$ is a bounded function, and X_{t+1} denotes a stochastic disturbance simulated from $f_{\theta_t}(x)$, $x \in \mathcal{X} \subset \mathbb{R}^d$, using a MCMC sampler. This algorithm can be rewritten as an algorithm for search of zeros of a function $h(\theta)$,

$$\theta_{t+1} = \theta_t + \gamma_{t+1}[h(\theta_t) + \epsilon_{t+1}], \tag{14}$$

where $h(\theta_t) = \int_{\mathcal{X}} H(\theta_t, x) f_{\theta_t}(x) dx$ corresponds to the mean effect of $H(\theta_t, X_{t+1})$, and $\epsilon_{t+1} = H(\theta_t, X_{t+1}) - h(\theta_t) + \gamma_{t+1}^{\tau} \eta(X_{t+1})$ is the observation noise.

It is well known that the optimal convergence rate of (14) can be achieved with $\gamma_t = -F^{-1}/t$, where $F = \partial h(\theta_*)/\partial\theta$, and θ_* denotes the zero point of $h(\theta)$. In this case, (14) is reduced to Newton's algorithm. Unfortunately, it is often impossible to use this algorithm, as the matrix F is generally unknown. Then, in a sequence of papers, Ruppert (1988), Polyak (1990), and Polyak and Juditsky (1992) showed that the trajectory averaging estimator is asymptotically efficient; that is,

$$\bar{\theta}_n = \sum_{t=1}^{n} \theta_t / n \tag{15}$$

can converge in distribution to a normal random variable with mean θ_* and covariance matrix Σ, where Σ is the smallest possible covariance matrix in an appropriate sense.

The trajectory averaging estimator allows the gain factor sequence $\{\gamma_t\}$ to be relatively large, decreasing slower than $O(1/t)$. As discussed by Polyak and Juditsky (1992), trajectory averaging is based on a paradoxical principle, a slow algorithm having less than optimal convergence rate must be averaged.

Liang (2009e) showed that the trajectory averaging estimator can also be applied to general stochastic approximation MCMC (SAMCMC) algorithms. In Liang (2009e), the bias term $\eta(x)$ in (13) was assumed to be 0, while some other conditions previously imposed on SAMC have been relaxed. For example, the solution space Θ has been relaxed from a compact set to $\mathbb{R}^d$ based on the varying truncation technique introduced by Chen (1993), and the uniform boundedness condition $\sup_{x \in \mathcal{X}} V(x) < \infty$ has been weakened to $\sup_{x \in \mathcal{X}_0} V(x) < \infty$ for a subset $\mathcal{X}_0 \subset \mathcal{X}$.