Advanced Machine Learning

Markov chain Monte Carlo

Use and study of Metropolis-Hasting and Hamiltonian Monte Carlo sampling algorithms

Authors:
Eric Bataller
Luuk Hesselink
Victoria Libucha
Gabriel Raya

February, 2020

I. INTRODUCTION

Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. Monte Carlo methods are mainly used in two problem types: numerical integration, and generating draws from a probability distribution.

Monte Carlo methods can be theoretically used to solve any problem having probabilistic interpretation. By the law of large numbers, integrals described by the expected value of some random variable can be approximated by taking the empirical mean of independent samples of the variable. As a classical example of Monte Carlo sampling methods we find rejection sampling. In a nutshell, rejection sampling is based on the observation that to sample a random variable x that follows a probability density $P(x) = P(x)^*/Z$, one can perform a random sampling of a simpler proposal density Q(x) which we can evaluate (within a multiplicative factor Z_Q). And assuming that we know the value of a constant c such that $cQ(x)^* > P(x)^*$ for all x, we can turn those x samples that follow Q(x) into samples from the P(x) distribution by assigning a generated uniformly distributed random variable u from the interval [0,$cQ(x)^*$] to every x sample and accepting it or rejecting it as part of the set of samples $\{x^{(r)}\}$ that follow the P(x) distribution depending on whether $u < P(x)^*$ or $u > P(x)^*$, respectively.

Markov chain Monte Carlo (MCMC) are computer-driven sampling methods. The idea in MCMC sampling is to sample, not directly from P(x), but from a different distribution such that, in the limit of a large number of samples, effectively the samples will be from P(x).It allows one to characterize a distribution without knowing all of the distribution's mathematical properties by randomly sampling values out of the distribution.To achieve this we forward sample from a Markov transition whose stationary distribution is equal to P(x). Therefore we combine Monte Carlo techniques with a Markov chain scheme. The Markov chain property of MCMC is the idea that the random samples are generated by special sequential process. Each random sample is used as a stepping stone to generate the next random sample (hence, the *chain*). In contrast to rejection sampling, where the accepted points $\{x^{(r)}\}$ are independent samples from the desired distribution, Markov chain Monte Carlo methods involve Markov process in which a sequence of states $\{x^{(t}\}$ is generated, each sample $x^{(t)}$ having a probability distribution that depends on the previous value $x^{(t-1)}$. Since successive samples are dependent, the Markov chain may have to be run for a considerable time in order to generate samples that are effectively independent samples from P.

II. MARKOV CHAINS

Consider the conditional distribution $Q(x^{t+1}; x^t)$. If we are given an initial sample x^1, then we can recursively generate samples x^1,x^2,...,x^T. After a long time T >> 1, (and provided the Markov chain is 'irreducible', meaning that we can eventually get from any state to any other state and 'aperiodic', meaning we don't periodically revisit any state) the samples are from the unique stationary distribution $Q_\infty(x)$ which is defined as (for a continuous variable)

$$Q_\infty(x') = \int_x Q(x'; x) Q_\infty(x), \tag{1}$$

The condition for a discrete variable is analogous on replacing integration with summation. The idea in MCMC is, for a given distribution $P(x)$, to find a transition $Q(x'; x)$ which has $P(x)$ as its stationary distribution. If we can do so, then we can draw samples from the Markov chain by forward sampling and take these as samples from $P(x)$ as the chain converges towards its stationary distribution.

Note that for every distribution $P(x)$ there will be more than one transition $Q(x'; x)$ with $P(x)$ as its stationary distribution. This is why there are many different MCMC sampling methods, each with different characteristics and varying suitability for the particular distribution at hand. A special property of the chain is that, while each new sample depends on the one before it, new samples do not depend on any samples before the previous one (this is the "Markov" property).

III. METROPOLIS-HAISTING ALGORITHM

Rejection sampling works well only if the proposal density $Q(x)$ is similar to $P(x)$. In large and complex problems it is difficult to create a single density $Q(x)$ that has this property. Specially in high-dimensional problems. In contrast to rejection sampling, here it is not necessary that $Q(x'; x)$ look at all similar to $P(x)$ in order for the algorithm to be practically useful. The idea behind Metropolis-Hasting method is the following: Suppose we are at a state point x^t.

Then, using some prescription, we generate a candidate new point y using a probability density $\tilde{Q}(y;x)$ (probability density in y given a center x). We then compare the target densities P(x) and P(y):

$$a = \frac{P(y)}{P(x)} \tag{2}$$

If $a > 1$ then we accept the candidate point $y = x^{t+1}$; this is reasonable since the candidate point is more probable than the old one. However, if $a < 1$ we may accept $y = x^{t+1}$, with probability a. If not, then we stick to the old point, and have $x^{t+1} = x^t$. The fact that we allow for a reduction in probability is what allows us to wander all over the space; otherwise we would simply be working our way towards a point of (locally) maximal probability. This method will work provided that we have that:

$$\tilde{Q}(y;x) = \tilde{Q}(x;y) \tag{3}$$

This means that the change of state from x^t to x^{t+1} is equally probable as the opposite step. This will ensure we are fulfilling the *detailed balance* and imposes a symmetry property over the proposal density $\tilde{Q}(x',x)$. The transition function $Q(x';x)$ that we will use therefore in order to have $P(x)$ as the stationary distribution of the Markov chain could be written as:

$$Q(x';x) = \int_y \tilde{Q}(y;x) \times \left[\theta\left(\frac{P(y)}{P(x)} > 1\right)\delta(x'-y) + \theta\left(\frac{P(y)}{P(x)} < 1\right)\left\{\frac{P(y)}{P(x)}\delta(x'-y) + \left(1 - \frac{P(y)}{P(x)}\right)\delta(x'-x)\right\}\right] \tag{4}$$

where $\theta(condition)$ is equal to 1 when the condition between brackets is fulfilled and 0 otherwise, and δ is the Dirac function. This transition function fully comprehend the algorithm process. And by integrating it in the limit of $t \to \infty$ steps where $Q_\infty(x) = P(x)$ we can easily demonstrate the convergence of the Markov chain:

$$P(x') = \int_x Q(x';x)P(x), \tag{5}$$

Notice that to be able to solve this integral we have to make use of the detailed balance property. For the rest we are completely free in the choice of $\tilde{Q}$: we may change it at will at any moment. The choice of $\tilde{Q}$ does influence the performance of the algorithm, though. If we take 'small' steps the probability of accepting a candidate point is probably high, but it will take a longer time to cover the space. If the steps are 'large' then we move over the space quickly, but we may expect that not many of such proposed steps are accepted. A good rule of thumb seems to be that about half of the candidates should be accepted.

For this kind of methods with local proposal distributions (local in the sense that are unlikely to propose a candidate far from the current sample), if the target distribution has isolated islands of high density, then the chance that we would move from one island to the other is very small. Conversely, if we attempt to make the proposal less local by using one with a high variance the chance then of landing at random on a high density island is remote. Auxiliary variable methods use additional dimensions to aid exploration (suppressing random walks behaviors) and in certain cases to provide a bridge between isolated high density islands. In the next section we introduce one of these methods.

IV. HAMILTONIAN MONTE CARLO ALGORITHM

Hamiltonian Monte Carlo is a method for continuous variables that aims to make non-local jumps in the sample space and, in so doing, to jump potentially from one mode to another. We define the distribution from which we wish to sample as:

$$P(x) = \frac{1}{Z_x} e^{H_x(x)}, \tag{6}$$

for some given 'Hamiltonian' $H_x(x)$. We then define another, 'easy' auxiliary distribution from which we can readily generate samples,

$$P(y) = \frac{1}{Z_y} e^{H_y(y)}, \tag{7}$$

so that the joint distribution is given by

$$P(x,y) = P(x)P(y) = \frac{1}{Z}e^{H_x(x)+H_y(y)} = \frac{1}{Z}e^{H(x,y)}, \tag{8}$$

In the standard form of the algorithm, a multi-dimensional Gaussian in chosen for the auxiliary distribution with dim(y) = dim(x), so that

$$H_y(y) = -\frac{1}{2}y^T y, \tag{9}$$

The HMC algorithm first draws from p(y) and subsequently from p(x,y). For a Gaussian p(y), sampling from this is straightforward. In the next 'dynamic' step, a sample is drawn from p(x, y) using a Metropolis MCMC sampler. The idea is to go from one point of the space x,y to a new point x',y' that is a non-trivial distance from x,y and which will be accepted with a high probability. The candidate (x',y') will have a good chance to be accepted if H(x',y') is close to H(x,y) – this can be achieved by following a contour of equal 'energy' H. Hence, making the ratio a from Eq.(2) close to 1 (if the simulation was perfect a would be exactly 1, but given the discretization of the state space the simulation is imperfect, and therefore some of the dynamical proposals will be rejected). The occasional rejections ensure that, asymptotically, we obtain samples (x^t, y^t) from the required joint density.

The update/step in the state space would go as follows: We wish to make an update $x' = x + \Delta x$, $y' = y + \Delta y$ for small Δx and Δy such that the Hamiltonian H(x,y) is conserved,

$$H(x', y') \approx H(x, y), \tag{10}$$

We can satisfy this (up to first order) by considering the Taylor expansion

$$H(x', y') = H(x + \Delta x, y + \Delta y) \approx H(x, y) + \Delta x^T \nabla_x H(x, y) + \Delta y^T \nabla_y H(x, y) + O(|\ \Delta x\ |^2) + O(|\ \Delta y\ |^2), \tag{11}$$

Conservation, up to first order, therefore requires

$$\Delta x^T \nabla_x H(x, y) + \Delta y^T \nabla_y H(x, y) = 0 \tag{12}$$

This is a single scalar requirement, and there are therefore many different solutions for Δx and Δy that satisfy this single condition. It is customary to use Hamiltonian dynamics, which corresponds to the setting:

$$\Delta x = \epsilon \nabla_y H(x, y), \quad \Delta y = -\epsilon \nabla_x H(x, y) \tag{13}$$

where ϵ is a small value to ensure that the Taylor expansion is accurate. Hence

$$x(t+1) = x(t) + \epsilon \nabla_y H(x, y), \quad y(t+1) = y(t) - \epsilon \nabla_x H(x, y) \tag{14}$$

For the HMC method, $\nabla_x H(x, y) = \nabla_x H(x)$ and $\nabla_y H(x, y) = \nabla_y H(y)$. For the Gaussian case, $\nabla_y H(y) = -y$.

There are specific ways to implement the Hamiltonian dynamics called Leapfrog discretisation that are more accurate than the simple time-discretisation. In the pseudocode of the algorithm (Appendix B) the reader can see an example of such a discretisation.

In order to make a symmetric proposal distribution, at the start of the dynamic step, we choose $\epsilon = +\epsilon_0$ or $\epsilon = -\epsilon_0$ uniformly. We then follow the Hamiltonian dynamics for many time steps (usually of the order of several hundred) to reach a candidate point (x',y'). If the Hamiltonian dynamics is numerically accurate, H(x',y') will have roughly the same value as H(x, y). We then do a Metropolis step, and accept the point (x',y') if H(x',y') > H(x, y) and otherwise accept it with probability exp(H(x',y')-H(x,y)). If rejected, we take the initial point x, y as the sample.

In HMC we use not just the potential $H_x(x)$ to define candidate samples, but the gradient of $H_x(x)$ as well. An intuitive explanation for the success of the algorithm is that it is less myopic than straightforward Metropolis since the gradient enables the algorithm to feel its way to other regions of high probability by contouring around paths in the augmented space. One can also view the auxiliary variables as momentum variables – it is as if the sample has now a momentum which can carry it through the low-density x-regions. Provided this momentum is high enough, we can escape local regions of significant probability.

V. BAYESIAN INFERENCE FOR PERCEPTRON LEARNING WITH MCMC

MCMC is particularly useful in Bayesian inference because of the focus on posterior distributions which are often difficult to work with via analytic examination. In these cases, MCMC allows the user to approximate aspects of posterior distributions that cannot be directly calculated (e.g., random samples from the posterior, posterior means, etc.). Bayesian inference uses the information provided by observed data about a (set of) parameter(s), formally the likelihood, to update a prior state of beliefs about a (set of) parameter(s) to become a posterior state of beliefs about a (set of) parameter(s). Formally, Bayes' rule is defined as

$$P(w \mid D) \propto P(D \mid w)P(w) \tag{15}$$

where w indicates a (set of) parameter(s) of interest and D indicates the data, $P(w \mid D)$ indicates the posterior or the probability of w given the data, $P(D \mid w)$ indicates the likelihood or the probability of the data given w, and $P(w)$ indicates the prior or the a–priori probability of w. The symbol $\propto$ means "is proportional to". The important point for this exposition is that the way the data are used to update the prior belief is by examining the likelihood of the data given a certain (set of) value(s) of the parameter(s) of interest. Ideally, one would like to assess this likelihood for every single combination of parameter values. When an analytical expression for this likelihood is available, it can be combined with the prior to derive the posterior analytically. Often times in practice, one does not have access to such an analytical expression. In Bayesian inference, this problem is most often solved via MCMC: drawing a sequence of samples from the posterior, and examining their mean, range, and so on.

We can ask ourselves if we could infer the parameters w of the posterior of a Perceptron during a learning process. For the specific case of a neural network we know that we train it by minimizing an objective function

$$M(w) = G(w) + \alpha E_w(w) \tag{16}$$

made up of an error function

$$G(w) = -\sum_n \left[t^{(n)} lny(x^{(n)}; w) + (1 - t^{(n)}) ln(1 - y(x^{(n)}; w)) \right] \tag{17}$$

and a regularizer

$$E_W(w) = \frac{1}{\sum_i} w_i^2 \tag{18}$$

We interpret the output y(x;w) of the neuron literally as defining (when its parameters w are specified) the probability that an input x belongs to class t = 1, rather than the alternative t = 0. Thus $y(x; w) = P(t = 1 \mid x; w)$. Then each value of w defines a different hypothesis about the probability of class 1 relative to class 0 as a function of x. We define the observed data D to be the targets $\{t\}$ - the inputs $\{x\}$ are assumed to be given, and not to be modelled. To infer w given the data, we require a likelihood function and a prior probability over w. The likelihood function measures how well the parameters w predict the observed data; it is the probability assigned to the observed t values by the model with parameters set to w. Now the two equations $P(t = 1 \mid x; w) = y$ and $P(t = 0 \mid x; w) = 1 - y$ can be written as the single equation

$$P(D \mid w) = exp[-G(w)] \tag{19}$$

Similarly the regularizer can be interpreted in terms of a log prior probability distribution over the parameters:

$$P(w \mid \alpha) = \frac{1}{Z_W(\alpha)} exp[-\alpha E_W(w)] \tag{20}$$

If E_W is quadratic as defined above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = \frac{1}{\alpha}$, and $\frac{1}{Z_W(\alpha)}$ is equal to $(\frac{\alpha}{2\pi})^{\frac{K}{2}}$, where K is the number of parameters in the vector w. In the case of a single neuron, the number of connections.

The objective function M(w) then corresponds to the inference of the parameters w, given the data:

$$P(w \mid D, \alpha) = \frac{P(D \mid w)P(w \mid \alpha)}{P(D \mid \alpha} = \frac{1}{Z_M} exp(-M(w)) \tag{21}$$

So the w found by (locally) minimizing M(w) can be interpreted as the (locally) most probable parameter vector, w^*. Therefore, the posterior can be drawn by making use of the MH or the HMC algorithms, for example.

A. Results and discussion

We are asked to infer the posterior $P(w \mid D, \alpha)$ of a neuron with three weights (w0,w1, and w2) for the given data x and t by using the Hamiltonian Monte Carlo and the Metropolis-Hasting algorithms. The results are shown in Fig.(1).

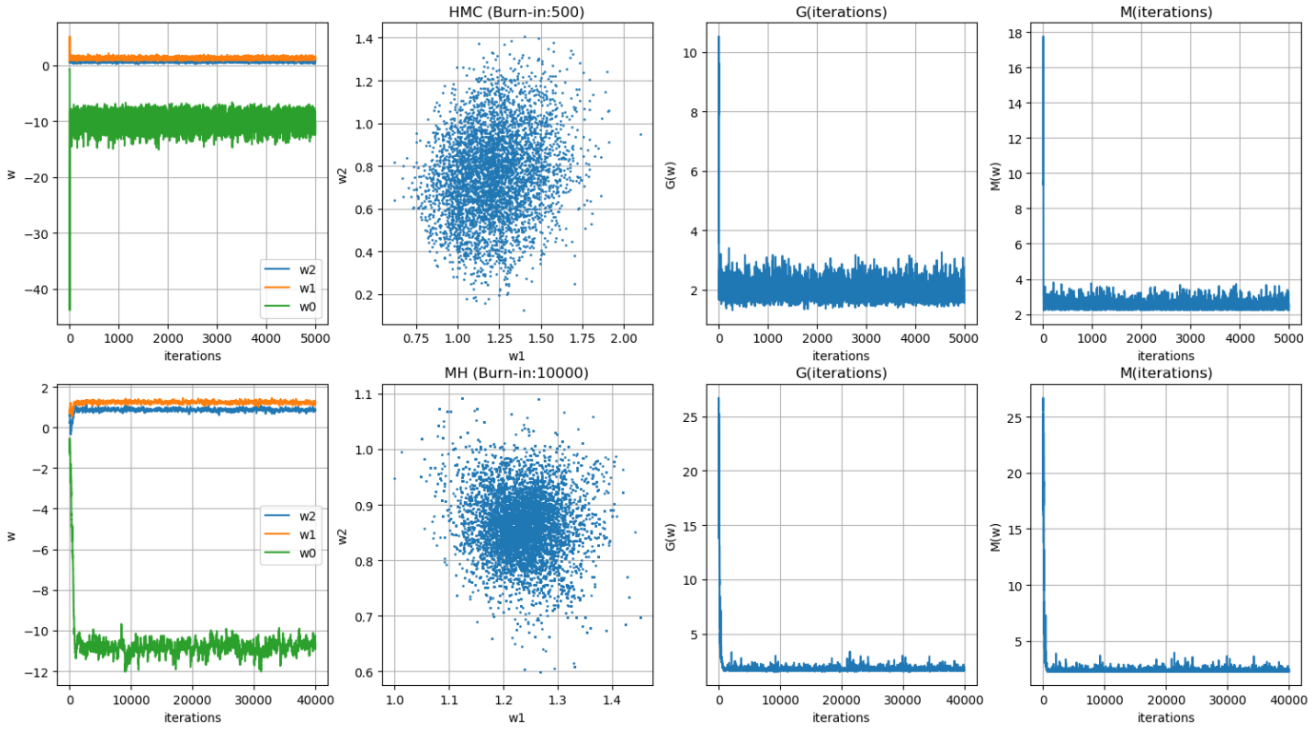

FIG. 1: Results of the HMC (upper row) and MH (lower row) algorithms. The first column correspond to the evolution of the weights as a function of number of iterations. The second column shows the evolution of weights w1 and w2 in the weight space. Notice the different Burn-in numbers in the titles of the plots. The third and forth columns correspond to the evolution of the error function G(w) and the objective function M(w), respectively, as a function of number of iterations.

After plotting the posterior several times, we have seen that it takes around 100 iterations for the HMC sampler to reach the equilibrium. Since this is not deterministic and sometimes it will take a little longer, we preferred to set the Burn-in to 500 just to make sure. At the end of the day the computation is pretty fast and augmenting the number of iterations a little bit doesn't make a significant difference. The autocorrelation of the state of the Hamiltonian Monte Carlo simulation falls much more rapidly with simulation time than that of the Metropolis-Hasting method. In the case of the MH sampler, the algorithm takes much more iterations to reach the equilibrium, around 2000 iterations. Again, just to make sure, we set the Burn-in to 10000 iterations. Notice that a iteration for the HMC algorithm takes more time than for MH algorithm. Hence, the real time of simulation will be similar in total. The acceptance ratio of our HMC algorithm is around 99% .Each single proposal makes use of multiple gradient evaluations along a dynamical trajectory in w; p space, where p are the extra 'momentum' variables of the Hamiltonian Monte Carlo methods. The number of steps 'Tau' was set at random to a number between 100 and 200 for each trajectory. The

step size ϵ was kept fixed and equal to 0.055; it is also recommended that one randomize the step size in practical applications, however. The acceptance ratio of our MH algorithm is around 17%. The proposal distribution used is a multivariate Gaussian with a symmetrical covariance matrix valued (0.01, 0.001, 0.001) in the diagonal and 0 in the rest of the elements. This is, an axial-oriented 3D Gaussian with a wider variance for the w0 axis. The elements of the matrix have been kept relatively small in order to reduce random walk behaviour.

VI. APPENDIX

A. Implementation of the Metropolis-Haisting algorithm

MH Algorithm pseudocode:

- Choose a starting point x
- for t = 2 to T do
- Draw a candidate sample x^{cand} from the proposal $\tilde{Q}(x'; x^{t-1})$
- Let $a = \frac{\tilde{Q}(x^{t-1};x^{cand})P(x^{cand})}{\tilde{Q}(x^{cand};x^{t-1})P(x^{t-1})}$
- if $a \geq 1$ then $x^t = x^{cand}$
- else
- draw a random value u uniformly from the unit interval [0,1].
- if $u < a$ then $x^t = x^{cand}$
- else
- $x^t = x^{t-1}$
- end if
- end if
- end for

B. Implementation of the Hamiltonian Monte Carlo algorithm

HMC Algorithm pseudocode:

- Hx = Hx(x) #This is the objective function (initialized using initial x)
- g = gradHx(x) #set gradient too
- for t = 1 to T do
- y = randn(size(x))
- H = y.T * y / 2 + Hx
- xnew = x ; gnew = g
- for tau = 1 to Tau do #make Tau 'leapfrog' steps
- y = y - epsilon * gnew / 2
- xnew = xnew + epsilon * y
- gnew = gradHx (xnew)
- y = y - epsilon * gnew / 2
- end for
- Hxnew = Hx(xnew)
- Hnew = y.T * y / 2 + Hxnew
- dH = Hnew - H
- if (dH < 0) then g = gnew ; x = xnew ; Hx = Hxnew ;
- elseif (rand() < exp(-dH)) then g = gnew ; x = xnew ; Hx = Hxnew ;

- end if
- end for
