Hamiltonian Monte Carlo in Bayesian Neural Networks

Juan Maroñas
jmaronasm@gmail.com

PRHLT Research Center,
Universitat Politècnica de València

April 2019

1 Introduction

In this document I show how can we evaluate the predictive distribution in a classification scenario when using Neural Networks to parameterize the likelihood model in a Bayesian setting. In this case I will be using Hamiltonian Monte Carlo [2, 3] to get samples from the posterior distribution over the model parameters. For an alternative and shorter explanation see [1] chapter 11 section 5. Rather than a technical report this document aims at briefly explain the different parts of the model in order to be able to follow and understand the provided implementation. I assume the reader knows what Markov Chain Monte Carlo (MCMC) algorithms (in particular Hamiltonian Monte Carlo) are, what a Bayesian setting is and how can we benefit from reasoning under it, and why do we need MCMC to draw samples from the posterior in this particular setting.

The key points of Markov Chain Hamiltonian Monte Carlo is take the Hamiltonian function from the mechanical physics and make an analogy with unormalized probability distributions through the canonical distribution of an energy function (something also used in Energy Based Models). Once this is done, we "move" in the mechanical system to draw a new sample from the desired distribution. In principle, this sample would be directly as the Hamiltonian preserves volume, i.e we move in curves of equal probability. However, moving in this mechanical system involves computing solutions of a system of partial derivatives, which needs to be discretize for many models of interest. To correct the errors this integration method can produce, we use a Metropolis-Hasting correction on the proposed sample [1]. The discretization is done using the leapfrog numerical method, see https://en.wikipedia.org/wiki/Leapfrog_integration

[1] There are some additional things to mention here like e.g the negation of the momentum and why this is not done in practice but this goes beyond the intention of these notes.

2 Model

For simplicity, I assume that the prior over the parameters of a neural network w (these includes any parameters: bias, weight, γ and β from Batch Normalization...) follows a factorized normal distribution with mean and variance given by μ_w and σ_w^2:

$$p(w) = \prod_{k=1}^{|W|} \frac{1}{\sqrt{2\pi}\sqrt{\sigma_w^2}} \exp\big(-\frac{1}{2\sigma_w^2}(w_k - \mu_w)^2\big) \tag{1}$$

With partial log derivative w.r.t to weight w_k given by:

$$\frac{\partial \log p(w)}{\partial w_k} = -\frac{1}{\sigma_w^2}(w_k - \mu_w) \tag{2}$$

Note that computing this derivative is straightforward just by noting that the terms of the log product for values of $k' \neq k$ does not contribute to the derivative of w_k, and that many terms are constant w.r.t w.

On the other side, in a classification scenario where t denotes the class target of our feature x, the posterior probability of the weights given a set of observations $\mathcal{O} = \{(x_i, t_i)\}_{i=1}^N$ is given by:

$$p(w|\mathcal{O}) = \frac{p(\mathcal{O}|w) \cdot p(w)}{Z} \tag{3}$$

where Z is a normalization factor which is computed by marginalizing the numerator over w. Our set of observations is assumed to be iid, thus we approximate the likelihood term as:

$$p(\mathcal{O}|w) = \prod_{i=1}^{N} p(t_i|x_i, w) \tag{4}$$

where we assume that the input distribution $p(x_i|w) = 1$ is not modelled. Our final posterior distribution takes the form:

$$p(w|\mathcal{O}) = \frac{\prod_{i=1}^{N} p(t_i|x_i, w) \cdot p(w)}{Z} \tag{5}$$

If we assume that the terms of the likelihood $p(t_i|x_i, w)$ follow a Bernoulli (for the binary classification problem) or a categorical (for the multiclass one), then the derivative of the log likelihood of the posterior can be expressed as:

$$\frac{\partial \log p(w|\mathcal{O})}{\partial w_k} = \frac{\partial \sum_{i=1}^{N} -\mathrm{CE}(x_i, t_i, w)}{\partial w_k} + \frac{\partial \log p(w)}{\partial w_k} \tag{6}$$

Where CE is the cross entropy function given by $-t_i \log f_w(x_i)$, $f_w()$ is the model parameterized by w (in this particular case will be a neural network)[2]. It can be clearly seen that the first term corresponds to the gradients used in the update rule of stochastic gradient guided methods, and that can be easily computed using automatic differentiation packages like Autograd from PyTorch. The second term has already been computed and $-\frac{\partial \log Z}{\partial w_k} = 0$ as it has no dependence on the parameter.

As a consequence, the total gradient is given by:

$$-\frac{\partial \log p(w|\mathcal{O})}{\partial w_k} = \frac{\partial \sum_{i=1}^{N} \text{CE}(x_i, t_i, w)}{\partial w_k} + \frac{1}{\sigma_w^2}(w_k - \mu_w) \tag{7}$$

where the first tem is obtained with back propagation and will depend on the Neural Network topology.

3 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) is a Markov Chain Monte Carlo used to sample from a desired distribution, in this case $p(w|\mathcal{O})$. HMC uses an alternative random variable, which is call the momentum random variable denoted by p with density $p(p)$. The good point is that the joint probability defined in HMC factorizes, and thus this auxiliary variable does not influence the samples from the desired posterior. In this case, my model is fully specified by the next Hamiltonian function:

$$H(w, p) = U(w) + K(p) \tag{8}$$

where $U(w) = -\log \prod_{i=1}^{N} p(t_i|x_i, w) \cdot p(w)$ and $K(p) = \frac{1}{2} p^T p$. The canonical joint probability distribution associated to the Hamiltonian system is given by:

$$p(w, p) \propto \exp(-H(w, p)) = \exp(-U(w) - K(p)) \tag{9}$$

Under this definition it can be easily check that $p(w, p) = p(w) \cdot p(p)$, that $p(w|\mathcal{O}) \propto \exp(-U(w))$ and that $p(p) = \mathcal{N}(p|0, I)$. The last ingredient is the derivative of the kinetic energy, which is given by:

$$\frac{\partial K(p)}{\partial p} = p \tag{10}$$

Note that generating a sample from this probability distribution, and then ignoring the momentum variable p will give us a sample that we can propose as a candidate state for our MCMC algorithm. Ideally (in the case we could simulate the dynamics exactly), these samples are always accepted because the

[2]Note that this is a particularization for multiclass problem. If the problem is binary and your model just outputs one value, then CE $= -t \log(f_w(x)) - (1 - t) \log(1 - f_w(x))$ is the binary cross entropy

Hamiltonian is conserved (hence the acceptance ratio is always 1). In practice a Metropolis accept-reject step is needed to correct simulation errors.

The final algorithm (applied concurrently to each weight in the network) is given by:[3]

Input: ϵ,T,L # Leapfrog step size, number of MCMC and Leapfrog steps
Output: $w^* \sim p(w|\mathcal{O})$
Initialize: $w_0 \sim N(0, I)$ # initial value of your markov chain
for $t = 1$ *until* T **do**
  $p_0 \sim \mathcal{N}(0, I)$ *# resample momentum variable*
  $m_L = m_0$
  $w_L = w_0$
  for $l = 1$ *until* L **do**
    $p' = p_L - \frac{\epsilon}{2} \frac{\partial p(w|\mathcal{O})}{\partial w}\Big|_{w=w_L}$
    $w_1 = w_0 + \epsilon \cdot p'$
    $p_1 = p' - \frac{\epsilon}{2} \frac{\partial p(w|\mathcal{O})}{\partial w}\Big|_{w=w_1}$
    $p_L = p_1$
    $w_L = w_1$
  end
  # Metropolis Hasting Correction
  $u \sim \text{Uniform}[0, 1]$
  $\alpha = min(1, \exp(-H(w_L, p_L) + H(w_0, p_0))$
  if $u < \alpha$, **then** $w_0 = w_L$;
  if $t = T$, **then return** $w^* = w_0$;
end

Note that the presented pdf is just a particularization of this algorithm for a standard prior $p(w)$ that factorizes across w. If the prior is coupled, then the derivate above does not hold, i.e let $p(w) = \mathcal{N}(w|\mu_w, \Sigma_w)$ be a joint distribution with no diagonal covariance Σ:

$$\frac{\partial \log p(w)}{w} = \frac{1}{p(w)} \left[-p(w)\Sigma_w^{-1}(w - \mu_w)\right] = -\Sigma^{-1}(w - \mu_w) \tag{11}$$

Note that for the case in which Σ_w is a diagonal matrix, the result matches the one provided above: The derivative w.r.t w_k is given just by deriving $\sum_k -\sigma_w^{-1}(w_k - \mu_w)$, whose result is just w_k when deriving w.r.t w_k due to the derivation of a sum. But for the multivariate case one needs to solve a system of linear equations to obtain the derivative (and the dimension of this

[3]Note that the exponential of the kinetic function is the same as the standard normal, we only require the factor $(\sqrt{2\pi})^{-1}$ make the kinetic function a probability distribution. For $U(w)$ the normalization factor is given by Z, which is the numerator integrated over w. MCMC algorithms can sample from unormalized distributions [1], as they cancel in the Metropolis ratio – thus we do not need to compute these values

problem grows with cubically the number of parameters, as we need to compute the cholesky factor).

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] Radford M. Neal. Bayesian learning for neural networks. 1996.

[3] Radford M. Neal. Mcmc using hamiltonian dynamics. 2012.