Bayesian neural network estimation of next-to-leading-order cross sections

by

René Alexander Ask

THESIS

for the degree of

MASTER OF SCIENCE

Faculty of Mathematics and Natural Sciences
University of Oslo

Autumn 2021

Bayesian neural network estimation of next-to-leading-order cross sections

René Alexander Ask

Bayesian neural network estimation of next-to-leading-order cross sections

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

This is my abstract.

Acknowledgments

Acknowledgments yo

Contents

Introduction

Motivation, context and problem.

Outline of the Thesis

Give outline of thesis

Chapter 1

Hamiltonian Monte Carlo

In this chapter, I'll explain the details of the Hamiltonian Monte Carlo (HMC) method. It's a Monte Carlo (MC) sampling technique that merges Gibbs sampling and a modified version of Metropolis sampling with Hamiltonian dynamics. It avoids the random walk behaviour of the Metropolis algorithm and generates successive samples with smaller correlation. In this chapter, the first thing I'll discuss is Markov chains. From that, follows an outline of Gibbs- and Metropolis sampling. After that, I'll deal with Hamiltonian dynamics before I bring it all together to form the Hamiltonian Monte Carlo algorithm for sampling. This forms the basis for the practical Bayesian learning of the neural networks studied in this thesis.

1.1 Markov Chain Monte Carlo

Monte Carlo Markov chain (MCMC) is a scheme to sample points θ proportional to a distribution $\pi(\theta)$. It generates a new point θ_i given a point θ_{i-1}. By Markov chain, we mean a sequence of points $\theta_1, \theta_2, \ldots$ that are possibly dependent, but are each occuring in the sequence in proportion to $\pi(\theta)$. Note that $\pi(\theta)$ here is not an exact probability distribution because it need not be normalized to unity. However, suppose $P(\theta)$ is the underlying probability distribution, then $\pi(\theta) \propto P(\theta)$. Typically, in Bayesian applications, we have a prior $P(\theta)$ and a likelihood $P(D|\theta)$. In this case $\pi(\theta) = P(D|\theta)P(\theta)$ and $\pi(\theta) \propto P(\theta|D)$, that is, it's proportional to the posterior distribution. A few important properties of the Markov chain, originally introduced by Metropolis et. al and built upon by Hastings [1]:

1. **Ergodicity**. Each point θ_i is chosen from a distribution that only depends on the previous point in the sequence, θ_{i-1}. For this, we introduce a transition probability $T(\theta_i|\theta_{i-1})$. Conceptually, means that any point θ can be reached given a long enough sequence of samples.

2. **Detailed balance**. The transition probability is chosen to obey

   $$\pi(\theta)T(\theta'|\theta) = \pi(\theta')T(\theta|\theta'),$$

   which ensures that the Markov chain is ergodic. Mathematically, we can express this condition as

   $$\pi(\theta') = \int \pi(\theta)T(\theta'|\theta)\mathrm{d}\theta.$$

3. We allow the transition $\theta \to \theta$, hence the transition probability $T(\theta|\theta)$ may be non-zero.

4. The transition probablities integrate to unity, thus

   $$\int T(\theta'|\theta)\mathrm{d}\theta = 1.$$

5. Finally, the transition probabilities are required to be time-independent.

1.2 Gibbs sampling

Gibbs sampling [2] is a sampling technique for multi-dimensional parameters $\gamma \in \mathbb{R}^d$, for $d > 1$. Suppose $\gamma^{(t)}$ is the parameters at iteration t. Then the parameters $\gamma^{(t+1)}$ at iteration $t+1$ are generated from $\gamma^{(t)}$ by the algorithm

Algorithm 1 Gibbs sampling

Sample $\gamma_1^{(t+1)} \sim P(\gamma_1|\gamma_2^{(t)}, ..., \gamma_d^{(t)})$

Sample $\gamma_2^{(t+1)} \sim P(\gamma_2|\gamma_1^{(t+1)}, ..., \gamma_d^{(t)})$

$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$

Sample $\gamma_d^{(t+1)} \sim P(\gamma_d|\gamma_1^{(t+1)}, ..., \gamma_{d-1}^{(t+1)})$

1.3 The Metropolis-Hastings algorithm

The Metropolis algorithm [1] is a sampling algorithm based on random walks in parameter space. Albeit efficient for some problems, it's not a suitable sampling technique in the context of neural networks. However, a rudimentary understanding of the algorithm will be useful before we embark upon the HMC sampling algorithm.

The transition probability in the Metropolis algorithm is chosen to be

$$T(\theta'|\theta) = q(\theta'|\theta)A(\theta, \theta'), \tag{1.1}$$

where $q(\theta'|\theta)$ is called the proposal distribution and $A(\theta, \theta')$ is the acceptance probability given by

$$A(\theta, \theta') = \min\left(1, \frac{\pi(\theta')q(\theta|\theta')}{\pi(\theta)q(\theta'|\theta)}\right). \tag{1.2}$$

The point θ' is accepted with probability $A(\theta, \theta')$.

Algorithm 2 Metropolis-Hastings

Sample $\theta' \sim q(\theta'|\theta)$

$p = \min\left(1, \frac{\pi(\theta')q(\theta|\theta')}{\pi(\theta)q(\theta'|\theta)}\right)$

if $p \geq u$ **then**
  $\theta \to \theta'$ ▷ Accept transition
else
  $\theta \to \theta$ ▷ Reject transition
end if

1.4 Hamiltonian dynamics

Hamiltonian dynamics [3] plays a central part in the HMC algorithm. For completeness, I'll first give a brief survey of Lagrangian mechanics from which we derive the Hamiltonian. The Hamiltonian then lays the foundation for the Hamiltonian dynamics.

1.4.1 Lagrangian Mechanics

Assume a set of *generalized coordinates* $q = (q_1, ..., q_n)$. Generally, the Lagrangian can be written as

$$L(q, \dot{q}, t) = K(q, \dot{q}, t) - V(q, \dot{q}, t), \tag{1.3}$$

where K is the kinetic energy and V is the potential energy of the system. I'll restrict the treatment to the case where there's no explicit dependence on time t. The solutions $q(t)$ can be found by solving the Euler-Lagrange equations given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = 0. \tag{1.4}$$

1.4.2 Hamiltonian Mechanics

The Hamiltonian is constructed by the Legendre transformation,

$$H(q, p, t) = \sum_i p_i \dot{q}_i(p_i) - L(q, \dot{q}(p), t), \tag{1.5}$$

where

$$p_i = \frac{\partial L}{\partial \dot{q}_i} \tag{1.6}$$

The equations of motion, known as *Hamilton's* equations, are given by

$$\frac{\mathrm{d}p_i}{\mathrm{d}t} = \frac{\partial H}{\partial p_i}, \qquad \frac{\mathrm{d}p_i}{\mathrm{d}t} = -\frac{\partial H}{\partial q_i}. \tag{1.7}$$

For the purpose of utilizing this framework in the context of HMC, it's assumed that the form of the Lagrangian is

$$L(q, \dot{q}) = K(\dot{q}) - V(q), \tag{1.8}$$

where

$$K(\dot{q}) = \sum_i \frac{1}{2} m_i \dot{q}_i^2. \tag{1.9}$$

The generalized momentum is thus

$$p_i = \frac{\partial K}{\partial \dot{q}_i} = m\dot{q}_i, \tag{1.10}$$

from it follows that

$$H(q, p) = K(p) + V(q) = \sum_i \frac{p_i^2}{2m_i} + V(q). \tag{1.11}$$

1.4.3 Leapfrog integration

To run one step of HMC, we need to simulate a Hamiltonian system of the form discussed in the former section. The common choice of algorithm to integrate the equations is *leapfrog* integration [4]. This integrator is *symplectic*, which means it conserves local volumes in phase space. This effectively translates to an approximately conserved value of $H(q, p)$ throughout a simulation, with slight oscillations about a mean value.

Assume we approximate the true coordinates and momenta by $(\hat{q}, \hat{p})$. A single leapfrog integration step can then be written as in algorithm 3. h represents the stepsize used in the algorithm.

Algorithm 3 Leapfrog integration (single step)

1. $\hat{p}_i(t + h/2) = \hat{p}_i(t) - \frac{h}{2}\, \partial V(\hat{q}(t))/\partial q_i$
2. $\hat{q}_i(t + h) = \hat{q}_i(t) + \frac{h}{m_i}\hat{p}_i(t + h/2)$
3. $\hat{p}_i(t + h) = \hat{p}_i(t + h/2) - \frac{h}{2}\, \partial V(\hat{q}(t + h))/\partial q_i$

1.5 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo [5] is largely developed and expanded upon by R. Neal. The probability distribution to sample from is expressed in terms the Canonical distribution

$$P(q) \propto \exp(-V(q)), \tag{1.12}$$

where q represents the parameters of the model. $V(q)$ can then always be expressed as

$$V(q) = -\log P(q) - \log Z, \tag{1.13}$$

for some Z. To utilize the framework of Hamiltonian dynamics explained in section 1.4, we introduce momentum variables p_i such that we can expressed the total Hamiltonian as

$$H(q, p) = K(p) + V(q), \tag{1.14}$$

with a corresponding canonical distribution over phase-space

$$P(q, p) \propto \exp\left(-H(q, p)\right). \tag{1.15}$$

The algorithm is summarized in

Algorithm 4 Hamiltonian Monte Carlo

Randomly choose $\lambda = \pm 1$
Randomly sample p with Gibbs sampling
Start from current state (q, p)
for $l = 1, ..., L$ **do**
    $(q', p') \leftarrow \text{Leapfrog}(q, p)$
end for
$p = \min\left(1, \exp\left[-\left(H(q', p') - H(q, p)\right)\right]\right)$
if $p \geq u$ **then**
    $(q, p) \leftarrow (q', p')$
else
    $(q, p) \leftarrow (q, p)$
end if

Chapter 2

Bayesian Learning for Neural Networks

2.1 Neural Networks

In this chapter, I'll introduce the mathematical formalism underpinning neural networks. For convenience, well adopt the terminology used by Tensorflow[6] to help make the transition from mathematics to their machine learning framework easier. A neural network is most generally defined as a non-linear function $f : \mathbb{R}^p \to \mathbb{R}^d$. This non-linear function is built-up as follows:

- An input layer where a real observable $x \in \mathbb{R}^p$ is fed.
- A set of hidden layers.
- An output layer which consists of the output of the function.

2.1.1 Basic mathematical structure

Denote a layer of the network as ℓ. Assume there are L layers in total, such that $\ell \in \{1, 2, ..., L\}$. Denote the *activation* at layer ℓ as a_j^ℓ and its *bias* as b_j^ℓ. Moreover, denote its *logits* as z_j^ℓ. The following recursive equation defines the general structure of a neural network

$$a_j^\ell = \sigma_\ell \left(\sum_k W_{jk}^\ell a_j^{\ell-1} + b_j^\ell \right) \equiv \sigma_\ell(z_j^\ell), \tag{2.1}$$

where σ_ℓ denotes some possibly non-linear function associated with layer ℓ, W_{jk}^ℓ denotes the *kernel* of layer ℓ connecting the activation from layer $\ell - 1$ to layer ℓ, and b_j^ℓ is again the bias of layer ℓ. Typically σ_ℓ is called an *activation* function. Together, the kernel and the bias are called the *weights* of layer ℓ. The weights of the network serve as the adjustable or *learnable* parameters of the model.

2.1.2 Backpropagation

The standard approach to train a neural network is minimization of some cost function by employing the *backpropagation* algorithm[7]. The algorithm boils down to four equations defining a recursive algorithm that approximates the gradient with respect to the parameters of the model. Consider E as the so-called *loss* functions (which in the case of Bayesian neural network is the potential energy function $V(q)$). The first of the four equations quantifes the error,

$$\Delta_j^L = \frac{\partial E}{\partial z_j^L}. \tag{2.2}$$

The second equation allows us to compute the error at layer ℓ given we know the error at layer $\ell + 1$,

$$\Delta_j^\ell = \left(\sum_k \Delta_k^{\ell+1} W_{kj}^{\ell+1} \right) \sigma_\ell'(z_j^\ell). \tag{2.3}$$

The final two equations relate these errors to the gradient of the cost function with respect to the model parameters. For the kernels, we have

$$\frac{\partial E}{\partial W_{jk}^\ell} = \frac{\partial E}{\partial z_j^\ell} \frac{\partial z_j^\ell}{\partial W_{jk}^\ell} = \Delta_j^\ell a_k^{\ell-1}. \tag{2.4}$$

For the biases, the gradients are

$$\frac{\partial E}{\partial b_j^\ell} = \frac{\partial E}{\partial z_j^\ell} \frac{\partial z_j^\ell}{\partial b_j^\ell} = \Delta_j^\ell. \tag{2.5}$$

With these four equations, we can fit the neural network using minimization techniques such as stochastic gradient descent or more complex methods such as ADAM (pages 13-19 in [8]). Although not the focus of this thesis, we might use these methods in conjunction with HMC to speed up convergence to the stationary distribution.

2.1.3 Cost function for regression

In this thesis, we're concerned with regression tasks. The activation function of the final layer σ_L is then just the identity function. The typical cost function chosen to solve regression tasks is the L_2-norm, which for a single output can be written as

$$E(y, \hat{y}) = \frac{1}{2} \|y - \hat{y}\|_2^2, \tag{2.6}$$

where $\hat{y}$ denotes the model output and y the ground-truth. Now, the model output in this case is $\hat{y}_j = a_j^L = z_j^L$. Therefore,

$$\Delta_j^L = \frac{\partial E}{\partial z_j^L} = a_j^L - y_j. \tag{2.7}$$

The backpropagation algorithm for a single datapoint can now be written down. The forward pass is found in algorithm 5.

Algorithm 5 Backpropagation: forward pass

$a_j^0 = x_j$ for $j = 1, ..., p$ ▷ Initialize input
for $\ell = 1, 2, .., L-1$ **do**
  for $j = 1, 2, .., n$ **do**
    $a_j^\ell \leftarrow \sigma\left(\sum_k W_{jk}^\ell a_k^{\ell-1} + b_j^\ell\right)$
  end for
end for
for $j = 1, 2, .., m$ **do** ▷ m outputs
  $a_j^L \leftarrow \sigma\left(\sum_k W_{jk}^L a_k^{L-1} + b_j^L\right)$
end for

The backward pass of the algorithm is stated in algorithm 6.

Algorithm 6 Backpropagation: backward pass

for $j = 1, 2, .., m$ **do** ▷ m outputs
  $\Delta_j^L \leftarrow a_j^L - y_j$
  $\partial E / \partial b_j^L \leftarrow \Delta_j^L$
  $\partial E \Big/ \partial W_{jk}^L \leftarrow \Delta_j^L a_k^{L-1}$
end for
for $\ell = L-1, ..., 1$ **do**
  for $j = 1, ..., n$ **do**
    $\Delta_j^\ell \leftarrow \left(\sum_k \Delta_k^{\ell+1} W_{kj}^{\ell+1} \right) \sigma'(z_j^\ell)$
    $\partial E / \partial b_j^\ell \leftarrow \Delta_j^\ell$
    $\partial E \Big/ \partial W_{jk}^\ell \leftarrow \Delta_j^\ell a_k^{\ell-1}$
    update b_j^ℓ and W_{jk}^ℓ.
  end for
end for

2.2 Interpretations of probability

2.3 Bayesian inference

Given a series of observations $D = \{x^{(1)}, ..., x^{(n)}\}$, one defines a probabilistic model $P(x^{(i)}, \theta)$ for a set of parameters θ. The foundation for Bayesian inference is Bayes' theorem, which can be formulated as

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \tag{2.8}$$

Some useful terminology is in order. $P(\theta)$ is called the *prior* distribution and embodies our prior knowledge of the distribution of θ before any new observations are considered. $P(D|\theta)$ is called the *likelihood* function and provides information about θ learned from observing the data D. The likelihood function is often written

$$L(\theta) \equiv L(\theta|D) = P(D|\theta). \tag{2.9}$$

One should remember that this is specific to the data D, even though it's sometimes written as $L(\theta)$. The *posterior* distribution $P(\theta|D)$ models our belief about the distribution of θ after observer D. More succinctly, we can write Bayes' theorem as

$$P(\theta|D) \propto L(\theta|D)P(\theta), \tag{2.10}$$

because its rarely of interest, or tractable, to compute $P(D)$, known as the *evidence*.

The objective of Bayesian inference is to compute the *predictive* distribution, which can be expressed as the integral over all parameters of the posterior distribution weighted by the likelihood. Mathematically,

$$P(x^{(n+1)}|D) = \int P(x^{(n+1)}|\theta)P(\theta|D)\mathrm{d}\theta = \int L(\theta|x^{(n+1)})P(\theta|D)\mathrm{d}\theta. \tag{2.11}$$

2.4 Bayesian framework for neural networks

We can specialize the equations used in Bayesian inference for neural networks in the context of regression. The predictive distribution $P(y|x)$ seeks to model a function $f : \mathbb{R}^p \to \mathbb{R}$ that for a given x produces an output y. In the infinite data limit, the distribution should be a Dirac Delta function. For finite datasets, however, we instead seek a distribution of outputs given the input features.

Consider a set of observations $D = \{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}$, where $x^{(i)} \in \mathbb{R}^p$ are the input features and $y^{(i)} \in \mathbb{R}$ are the *targets*. The equation for the predictive distribution changes to

$$P(y^{(n+1)}|x^{(n+1)}, D) = \int L(\theta, y^{(n+1)}|x^{(n+1)})P(\theta|D)\mathrm{d}\theta. \tag{2.12}$$

Assuming that the the observations in D are drawn independently, the likelihood function can be expressed as

$$L(\theta|D) = \prod_{i=1}^{n} L(\theta, y^{(i)}|x^{(i)}) = \prod_{i=1}^{n} P(y^{(i)}|x^{(i)}, \theta). \tag{2.13}$$

In the context of regression, the conditional distribution is chosen to be

$$P(y|x, \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(f(x) - y)^2}{2\sigma^2}\right), \tag{2.14}$$

where $f(x)$, the neural network output, is the mean of y and σ is the noise of y.

In practice, however, we instead sample a set of network paramters $\{\theta_1, ..., \theta_n\}$ from which can produce a set of predictions $\{\hat{y}_1, \dots \hat{y}_n\}$ from the neural network model

$$\hat{y}_i = f(x, \theta_i). \tag{2.15}$$

From there we can compute the sample mean

$$\hat{y} = \frac{1}{n} \sum_i f(x, \theta_i), \tag{2.16}$$

which is an approximation to the *maximum likelihood estimate* (MLE). Furthermore, an estimate of the error is provided by the sample variance

$$\mathrm{Var}(y) = \frac{1}{n-1} \sum_i (\hat{y} - y_i)^2, \tag{2.17}$$

2.5 Bayesian learning using HMC

To implement learn from the data D using HMC, we need to define a potential energy function $V(q)$ and a kinetic energy function $K(p)$. To this end, we need to specify

1. A prior for the network parameters, i.e the kernels W_{ij}^{ℓ} and the biases b_j^{ℓ},
2. A prior for the corresponding momenta p,
3. A likelihood function given an input (x, y),

from which we can define the potential energy as follows:

$$V(W, b) = -\sum_{\ell} \log P(W_{ij}^{\ell}) - \sum_{\ell} \log P(b_j^{\ell}) - \sum_i \log P(y^{(i)}|x^{(i)}, W, b), \tag{2.18}$$

where W denotes all the kernels and b denotes all the biases of the model.

Conclusion

Conclusion here.

Appendices

Appendix A

A.1 Appendix 1 title

Some appendix stuff.

Bibliography

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, *Equation of State Calculations by Fast Computing Machines*, *The Journal of Chemical Physics* **21** (1953) 1087–1092, [https://doi.org/10.1063/1.1699114].

[2] S. Geman and D. Geman, *Stochastic relaxation, Gibbs Distributions, and the Bayesian Restoration of Images*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6** (1984) 721–741.

[3] J. S. Helbert Goldstein, Charles Poole, *Classical Mechanics, 3rd ed.*, ch. 2,8. Addison Wesley, 2000.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning*, ch. 11, p. 551. Springer New York, 2006.

[5] R. Neal, *Handbook of Markov Chain Monte Carlo*, ch. 5. Chapman and Hall/CRC, May, 2011. http://dx.doi.org/10.1201/b10905. 10.1201/b10905.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015.

[7] D. E. Rumelhart, G. E. Hinton and R. J. Williams, *Learning representations by back-propagating errors*, *Nature* **323** (1986) 533–536.

[8] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher et al., *A high-bias, low-variance introduction to machine learning for physicists*, *Physics Reports* **810** (May, 2019) .