GPU Implementation of Bayesian Neural Network Construction for Data-Intensive Applications

Michelle Perry¹, Harrison B Prosper², Anke Meyer-Baese¹
¹Department of Scientific Computing, Florida State University, Tallahassee, FL 32306, USA
²Department of Physics, Florida State University, Tallahassee, FL 32306, USA
E-mail: mep03e@my.fsu.edu

Abstract. We describe a graphical processing unit (GPU) implementation of the Hybrid Markov Chain Monte Carlo (HMC) method for training Bayesian Neural Networks (BNN). Our implementation uses NVIDIA’s parallel computing architecture, CUDA. We briefly review BNNs and the HMC method and we describe our implementations and give preliminary results.

1. Introduction
The search for new physics has entered an era in which more general models, such as the 19-parameter phenomenological minimal supersymmetric standard model (pMSSM) [1], have been used to interpret data at the Large Hadron Collider. Unfortunately, due to the complexity of the calculations, the predictions of these models are available only at a discrete set of parameter points. It would be useful, however, to have a computationally routine way to construct smooth mappings of the model parameters to the model predictions. We propose to construct the mappings using Bayesian neural networks (BNN). The main limitation to the widespread use of BNNs is the time required to construct them. We describe ongoing work to implement the construction of BNNs using the Hybrid Markov Chain Monte Carlo (HMCMC) method using Graphical Processing Units (GPUs). Although the motivation for this work is the search for supersymmetry guided by models such as the pMSSM, our implementation of BNN training is quite general and would be useful for any application that requires the construction of smooth multidimensional functions.

2. Bayesian Neural Networks
Neural networks (NNs) are non-linear “universal approximators” that can model any smooth mapping of the form \( f : \mathbb{R}^n \rightarrow \mathbb{R}^m \), with \( m < n \) [2]. In this work, we focus on approximations to functions with \( m = 1 \), based on the single hidden layer NN model,

\[
f(x, \omega) = a + \sum_{j=1}^{H} b_j \tanh(c_j + \sum_{i=1}^{I} d_{ji} x_i),
\]

where \( \omega \) are the neural network parameters \( \{a, b, c, d\} \), \( H \) is the number of hidden nodes in the network, and \( I \) is the number of inputs. Equation (1) has many symmetries, such that there are \( 2^H H! \) equivalent solutions in the parameter space. Consequently, as a function of \( \omega \), Eq. (1) is a
very complicated, high dimensional function, so advanced methods of traversing the parameter space are necessary. In a traditional NN, the goal is to find a single set of parameters, $\omega^*$, such that $f(x, \omega^*)$ approximates the desired mapping. For BNN [3],[4], the training of the network is treated as an inference problem that yields a probability density $P(\omega|T)$ over the network parameter space, where $T$ denotes the training data, $\{(t, x)\}_k = 1, \ldots, N$ and $t_k$ is the known “target” value associated with the input vector $x_k = (x_1, \ldots, x_I)_k$. The desired mapping is approximated by the average

$$BNN(x) = \int f(x, \omega) P(\omega|T) d\omega, \approx \frac{1}{K} \sum_{j=1}^{K} f(x, \omega_j),$$

where the set $\{\omega_j\}$ is sampled from the posterior density

$$P(\omega|T) \propto \exp\left[-\frac{1}{2\sigma^2} \sum_{k=1}^{N} (t_k - f(x_k, \omega))^2\right] \cdot \text{prior}(\omega).$$

The prior density prior$(\omega)$ is typically chosen to favor small values of the network parameters $\omega$, which yields smoother approximations. Figure 1 shows a simple 2-D example of such a calculation in which a 2-input BNN is used to approximate the function $t(x, y) = \sin(x) \cos(y)$, given a training set comprising $N = 1000$ patterns $(t, x, y)_k$. The posterior density was sampled using the Hybrid Monte Carlo method.

3. Hybrid Monte Carlo

Given the complexity of the parameter space of a neural network, standard Markov Chain methods that rely heavily on random walks through the parameter space can be very inefficient. The Hybrid Monte Carlo (HMC) method was developed for lattice gauge theory calculations [5] as a feasible way to cope with complicated parameter spaces. It was later adapted for sampling the posterior density $P(\omega|T)$ [6] associated with BNN. The method discretizes Hamilton’s equations, yet produces no discretization error due to the exact compensation for this error from the random acceptance of poor points in the space, even with large step sizes.

The Hybrid (Markov Chain) Monte Carlo algorithm treats the parameter space as a Hamiltonian system, $H = U(\omega) + K(p)$, with “potential energy”

$$U(\omega) = -\ln P(\omega|T)$$

and “kinetic energy”

$$K(p) = \frac{1}{2} \sum_{i=1}^{N} p_i^2,$$

where each network parameter $\omega_i$ is associated with a “momentum” parameter $p_i$. One HMC iteration consists of the generation of a random momentum vector and the propagation of the system from the current point in the parameter space through the space in $L$ steps, using a discrete form of Hamiltonian’s equations:

$$\frac{dp}{dt} = -\frac{\partial H}{\partial \omega} = -\nabla_\omega U$$

and

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = p.$$  

In this work, we use the following implementation of the HMC algorithm, which we write as pseudocode. The pseudocode represents one HMC iteration with step size $\text{eps}$. 

2
HMC_iteration(q0) {
    q = q0;
    p = norm_dist();
    p = p - 0.5*eps*delU(q); // half step in momentum
    for(int i=0; i<L; i++) { // do L steps
        q = q + eps * p; // step in q
        // step in p, except at last step
        if(i != L-1)
            p = p - eps * delU(q);
    }
    p = p - 0.5*eps*delU(q); // half step in momentum
    // calculate Hamiltonian for old and new points
    H0 = U(q0) + K(p0);
    H = U(q) + K(q);
    // decide which point to keep
    if(rand < exp(H - H0)){
        return q; // accept
    } else {
        return q0; // reject
    }
}

4. The BNN Algorithm in Practice
The BNN algorithm has proven to be very effective. Even with only 400 iterations, the BNN provides an accurate model of the function \( t(x,y) = \sin x \cos y \), as shown rather strikingly in Figure 1. This is, of course, a very simple example. However, BNNs have been used successfully in high profile research in high energy physics such as the discovery of single top quark production [7] and the discovery of the Higgs boson [8]. But there is a serious problem that limits their widespread use, namely the time required to train them when the number of inputs and the amount of training data becomes large.

In the HMC algorithm, the calculation of the potential energy \( U \) is the most time consuming. As the amount of training data, number of inputs, and number of hidden nodes increase, the evaluation time of \( U \) increases. For applications in high energy physics, such as those testing multi-parameter physics models, a faster implementation of BNNs is desirable. Each term in the sum in Eq. (3) is independent and data parallel, and with training sets of over \( 10^6 \) events, these applications are ideal for GPUs.

5. GPU Implementation
For data-intensive applications, our approach is to run the HMC algorithm on a CPU but evaluate the sum in Eq. (3) on the GPU.

The GPU is a processor developed for graphics rendering, which devotes more processors to data processing than does a CPU, but fewer to data flow and caching. Therefore, GPUs are ideal for compute intensive, highly parallel, applications such as the construction of BNNs. In this work, we use the CUDA C Programming language for NVIDIA GPUs. Each term in the sum in Eq. (3) is computed in one thread. The dynamic nature of CUDA allows the threads to be distributed to available CUDA cores as efficiently as possible. The more cores that are available on the GPU — a typical GPU has 256-1024, the greater the degree of parallelization. A parallel reduction of the results from each thread provides additional speedup in computing the sum. Our goal is to reduce the time required to train multivariate BNNs by two orders of magnitude.
Figure 1. The left is a BNN function trained with targets of the form \( t = \sin(x)\cos(y) \) and a discrete set of points \((x,y)\) sampled from a uniform distribution. \( N = 1000, H = 5, \) and \( I = 2, \) and 400 HMC iterations. The right is a plot of \( \sin(x)\cos(y) \).

for applications with > 20 inputs and \( 10^6 \) training events. Preliminary results indicate speedup of 10–50× the serial implementation (Speedups obtained on a GeForce GTX 480 GPU versus an Intel Xeon CPU using a test example with input and hidden node ranges expected in pMSSM calculations). We plan to implement further optimizations. The calculation of smooth prediction functions for the phenomenological minimal supersymmetric standard model is an application currently being explored using a GPU BNN application.

6. Prospects and Summary
The GPU implementation of the training of BNN promises to greatly expand the use of these highly flexible non-linear functions to approximate multi-dimensional real-valued functions, such as smooth maps from the parameter space of multi-parameter physics models, such as the pMSSM and its variants, to their predictions. Our preliminary implementations indicate a speed-up of at least an order of magnitude. We believe that with careful optimization another order of magnitude speed-up is feasible.

References
[1] 2013 Phenomenological mssm interpretation of the cms 2011 5fb-1 results Tech. rep. CERN Geneva
[2] Hornik K 1991 *Neural Networks* **4** 251 – 257
[3] Neal R M 1996 *Bayesian Learning for Neural Networks*, Vol. 118 of Lecture Notes in Statistics (Springer-Verlag)
[4] 2001 *Neural Networks* **14** 257 – 274
[5] Duane S, Kennedy A, Pendleton B J and Roweth D 1987 *Phys. Lett. B* **195** 216 – 222
[6] Neal R M 2010 *Handbook of Markov Chain Monte Carlo* chap MCMC Using Hamiltonian Dynamics, pp 113–162
[7] Abazov V *et al.* (D0 Collaboration) 2009 *Phys. Rev. Lett.* **103** 092001
[8] Chatrchyan S *et al.* (CMS Collaboration) 2013 to be submitted to *Phys. Rev. D.*