ABSTRACT

BEAM, ANDREW LANE. Understanding the Genetic Etiology of Complex Phenotypes using Bayesian Neural Networks. (Under the direction of Jon Doyle.)

Gene-gene interactions, or epistasis, are widely believed to be fundamental to the genetic etiology of many complex phenotypes. However, accounting for these interactions in genetic association studies containing millions of markers is computationally very difficult. Consideration of all possible interactions is intractable, as a typical genetic association study may have billions or trillions of possible interactions. This dissertation describes a flexible Bayesian neural network method designed to address many of these challenges. First, some of the necessary computational infrastructure required by the Bayesian neural network model is developed. This model relies on the Hamiltonian Monte Carlo (HMC) algorithm, which can be very slow for large datasets. By using graphics processing units (GPUs) to evaluate certain portions of the HMC algorithm, the time needed to perform the simulation is reduced by several orders of magnitude. This work demonstrates some of the largest fully Bayesian analyses to date.

The GPU framework is leveraged to enable the use of Bayesian neural networks on large genetic datasets. It is shown that this method is capable of accurately identifying causal genetic loci in a variety of simulated scenarios. In addition, a novel Bayesian test of variable relevance is derived and implemented. It is demonstrated that this test achieves high sensitivity and specificity and allows for precise discrimination between causal and non-causal genetic markers. In comparison to existing methods, this approach shows good power to detect genetic markers associated with disease status. The Bayesian neural network model is extended to analyze cell-based, dose-response genetic association studies. The method is compared to an existing approach where it again performs very well. The methods developed as part of this dissertation show a promising ability to analyze large, complex genetic data in a variety of scenarios.

Understanding the Genetic Etiology of Complex Phenotypes using Bayesian Neural Networks

by
Andrew Lane Beam

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Bioinformatics

Raleigh, North Carolina

2014

APPROVED BY:

Alison Motsinger-Reif	Eric Stone
Steffen Heber	Ronald Endicott

Jon Doyle
Chair of Advisory Committee

DEDICATION

To Mom and Dad for their never ending support and generosity.

To Kristyn for her love and for always laughing at my jokes.

BIOGRAPHY

Andrew was born in Concord, North Carolina. He attended North Carolina State University where he earned three degrees in Computer Science, Computer Engineering, and Electrical Engineering. After a brief stint as a bioinformatics analyst for a biotech company, he returned to N.C. State where he earned a master's degree in statistics while doing research with the Enivronmental Protection Agency. Having enjoyed the work he did in statistical genetics and bioinformatics as part of this degree, he decided to stay and pursue his Ph.D. in the bioinformatics program. After being a life-long North Carolinian and having spent ten years at N.C. State, he will be leaving NC and moving to Boston. Andrew has accepted a position as a post-doctoral fellow with the Center for Biomedical Informatics at Harvard Medical School.

ACKNOWLEDGEMENTS

Many people need to be thanked for allowing me to get to this point. I would like to thank my advisor, Dr. Jon Doyle, for the latitude he gave me to explore various ideas in this work as well as for his guidance and support throughout. Additionally, I would like to thank all of my committee members for all of their help throughout my time here at N.C. State. I would specifically like to thank Dr. Alison Motsinger-Reif. She has been pivotal in my educational trajectory as well as being a constant pillar of support and guidance.

I would also like to thank all of my family and friends. My mother and father have been nothing but supportive through my many academic endeavors, and for that I am eternally grateful. I am also deeply grateful for the love and support offered by my wife, Kristyn. Marriage to a graduate student comes with its own unique set of challenges, especially near the end of the dissertation process when physical presence does not always guarantee mental presence. I would be truely lost without her.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

It is an exciting time to be in genomic science. Completion of the draft of the human genome [Venter et al., 2001, Collins et al., 2003, Lander et al., 2001] over a decade ago brought with it the promise of personally tailored medical care, new insights in biology, and a deeper understanding of the foundations of life [Collins and McKusick, 2001]. To date there has been great success in leveraging this information to improve our understanding of the genetic basis of several diseases such as breast cancer [Wooster et al., 1995] and late-onset Alzheimer's disease [Strittmatter et al., 1993] among many others. Since 2006 there have been roughly 2,000 genetic loci that have been robustly associated with some form of disease or trait [Hindorff et al., 2011]. In the period since these breakthroughs there has been an explosion of genomic information. Public databases have experienced an exponential growth in the size of their contents. GenBank, the database of the National Center for Biotechnology Information, estimates the number of base pairs stored on its servers doubles approximately every 18 months [Benson et al., 2008]. This is roughly the speed at which transistor capacity doubles according to Moore's law, implying a dizzying pace of change for genomic science seen previously in integrated circuit development. Many more organisms have also since had their entire genomes sequenced. According to the Kyoto Encycolpedia of Genes and Genomes [Kanehisa and Goto, 2000], over 3,000 organisms now have completed genome sequences. One project nearing completion which encapsulates

the breadth of organisms being sequenced is the "bacon cheeseburger" genome. This composite genome represents sequencing projects of cows [Zimin et al., 2009], pigs [Archibald et al., 2010], chickens (for the eggs in mayonnaise) [Wallis et al., 2004], tomatoes [Consortium et al., 2012], wheat [Brenchley et al., 2012], lettuce [van Oeveren et al., 2011], onion [Jakše et al., 2008], and cucumber [Huang et al., 2009]. Exciting times, indeed.

What has yet to emerge from this sea of discovery is a complete picture of how genetic variation explains trait variation. Thus far a large portion of variants that have been linked to a trait have large effects that are mostly independent of the genetic context in which they are found. For some traits, such as height or cholesterol levels [Manolio et al., 2009], the familial patterns of trait expression suggest an underlying genetic basis, yet efforts thus far have only been able to attribute some small fraction of this variation to genetics. This is often referred to as the problem of the 'missing heritability' [Manolio et al., 2009, Eichler et al., 2010]. A review in 2011 estimated that early genetic association studies were able to explain less than 10% of the trait variation on average, while larger and more recent studies were usually capable of attributing 20-30% of trait variance to genetic variance [Zuk et al., 2012]. Despite this improvement, large swaths of trait variation remain unexplained, limiting our understanding of how genes influence traits.

Several theories have emerged in an attempt to explain where this missing heritability might be hiding. One theory is the missing heritability is due to rare genetic variants and that studies of common variants are simply missing where the real action is located [Zuk et al., 2014]. Another hypothesis is that traits are determined by the small, additive effects of hundreds to thousands of loci and that current study sample sizes are simply too small to pick these effects up reliably [Ioannidis, 2007]. Yet another theory argues that epigenetic status, or chemical changes to the DNA molecules that do no change the actual sequence, is the cause of the missing heritability [Slatkin, 2009]. Finally, many have argued that gene-gene interactions, or *epistasis*, underlies much of the phantom heritability observed in association studies [Zuk et al., 2012]. It remains to be seen which, if any, of these theories will ultimately be validated.

However, there is growing consensus that epistasis is a critical component of many genetic architectures, so the goal of this thesis will be to investigate and develop methods that can better account for genetic interactions.

1.1 The Role of Epistasis in Genetic Architectures

Epistasis is believed to be central to many complex diseases such as diabetes, asthma, hypertension and multiple sclerosis [Moore, 2003, Cordell, 2002]. While epistasis is believed to be critical in many genetic architectures, the concept is occasionally used without a precise definition [Cordell, 2002]. To be clear, in this thesis we are primarily interested in epistasis as defined as departure from an *additive* genetic model. Consider a simplified scenario with a continuous trait y and a genotype, $G =< G_1, \ldots, G_p >$ which contains the genotype status for p binary markers. An additive genetic model is one in which the trait y is simply the sum of all of the effects due to an individual's status at each marker, $G_1, \ldots, G_p$. If we let the effect of having the genotype $G_i = 1$ be represented as β_i, then this can be written using the following form:

$$y = \beta_0 + \sum_{i=1}^{p} \beta_i \cdot G_i$$

One form of epistasis would be to consider multiplicative interactions between any two markers. Using the same notation as before, this can be written as:

$$y = \beta_0 + \sum_{i=1}^{p} \beta_i \cdot G_i + \sum_{i=1}^{p} \sum_{j=1}^{p} \gamma_{ij} \cdot G_i * G_j$$

However, for a study containing a million markers, the number of interactions one must consider is $\binom{10^6}{2} \approx 500 * 10^9$, which is computationally infeasible with current hardware. Even if we could model that many possible interactions, it is not guaranteed that this new model is an accurate version of the underlying genetic etiology, despite having included all possible multiplicative interactions. Instead of searching for plausible parametric genetic models, the goal of this

thesis is to completely relax the structural assumptions on the relationship between y and G. To do this we develop a nonparametric Bayesian framework based on a neural network model.

1.2 Neural Networks in Genetic Epidemiology

An in-depth exposition of all the properties of neural networks is beyond the scope of this dissertation, though details needed to understand and implement the basic model will be presented as they are needed. However, we wish to provide some background material on neural networks to put the work presented here into proper context. Neural networks have been around in some form for nearly 70 years and were first introduced by McCulloch and Pitts through the notion of *threshold logic* [McCulloch and Pitts, 1943], though it was Rosenblatt's *perceptron* model [Rosenblatt, 1958] that most closely resembles what is now known as a neural network. Since then, the term neural network has grown to encompass a very wide class of approaches for supervised learning (i.e. classification and regression) as well as for unsupervised tasks. Depending upon the specific task and network used, one of several terms including *autoencoder*, *restricted boltzman machine*, *convolutional neural network*, *radial basis network*, or *multilayer perceptron* (MLP) may be used to describe the approach. The last of these, the MLP, will serve as the basis for the work presented in this thesis. Recently, MLPs have seen something of a comeback after falling out of vogue with the machine learning community for nearly two decades. This resurgence has been fueled in large part by the success in training 'deep' networks [Bengio, 2009], which are networks with many hidden layers, on problems in computer vision and speech recognition. This success has largely been to due to faster computers and the use of graphics processing units (GPUs) in addition to a regularization technique known as 'dropout' [Hinton et al., 2012, Krizhevsky et al., 2012], which prevents the network from over-fitting during the training process.

Outside of the computer vision and speech recognition communities, neural networks have enjoyed success in analyzing genetic association studies [Motsinger-Reif and Ritchie, 2008, Motsinger-Reif et al., 2008a]. The primary reason for their success in this field, as well as

in many others, is their ability to automatically discover nonlinear functions of input variables. This makes neural networks a promising approach for studying traits in which there might be epistasis. There is however, much confusion surrounding exactly how a network learns this nonlinear mapping [Olden and Jackson, 2002]. Neural networks are often described as 'black boxes' or as doing something that is fundamentally unintelligible. We present a simple example here to illustrate what a neural network does is not all that mysterious when viewed at the appropriate level, as well as to illustrate how they might be useful in a genetic association study.

Consider a famous example due to [Minsky and Papert, 1987] which demonstrates how an exclusive-or (XOR) function of two binary variables is not learnable by a linear classifier. For our purposes, assume we have a trait whose status is fully determined by two SNPs such that the status = 1 if and only if XOR(SNP 1, SNP2) = 1, and is 0 otherwise. Note that this function defines an epistatic relationship, as determination of trait status depends on knowledge of *both* markers. Figure 1.1 shows a plot of this type of trait.

Now consider a simple neural network containing two hidden units, where each hidden unit performs a nonlinear transformation of the input SNPs. The specifics of this transformation are left vague for now, but will be made concrete in later chapters. After this nonlinear transformation, the situation becomes much simpler. In Figure 1.2 the value of hidden unit 1 is plotted against the value of hidden unit 2.

Note that this network was stopped before convergence was reached for display purposes. If training continued both orange dots would have been located at (0,1). So what has the network done in this instance? What we wish to convey is that it has learned a simpler *representation* of the inputs. Note for instance the existence of a linear decision rule to separate the two classes, namely Status $= 1 - h_1 + h_2$, where h_1 and h_2 are the output values from the hidden units. Adding more hidden units effectively increases the number of dimensions in which the network can learn the representation. So there is nothing all that magical about what the network is doing, it is simply learning a different representation of the inputs in an attempt to minimize some measure of loss.

Figure 1.1: Example of a 2 locus epistatic interaction. Trait status is determined by an XOR operation of SNP 1 and SNP 2. Note there does not exist a way to separate the two classes with a linear decision rule.

1.3 Bayesian Neural Networks

Bayesian neural networks are more recent concept, first introduced in a comprehensive way in [Neal, 1995]. Since their introduction they have received relatively little attention, with the notable exception of winning the Neural Information Processing Systems (NIPS) feature selection challenge, by a rather wide margin, in 2003 [Guyon et al., 2004]. Compared to standard neural networks, they require much more computation, potentially limiting their usefulness on large problems. There are also far fewer software packages available that implement Bayesian neural networks, with the only apparent packages with any sort of documentation or function-

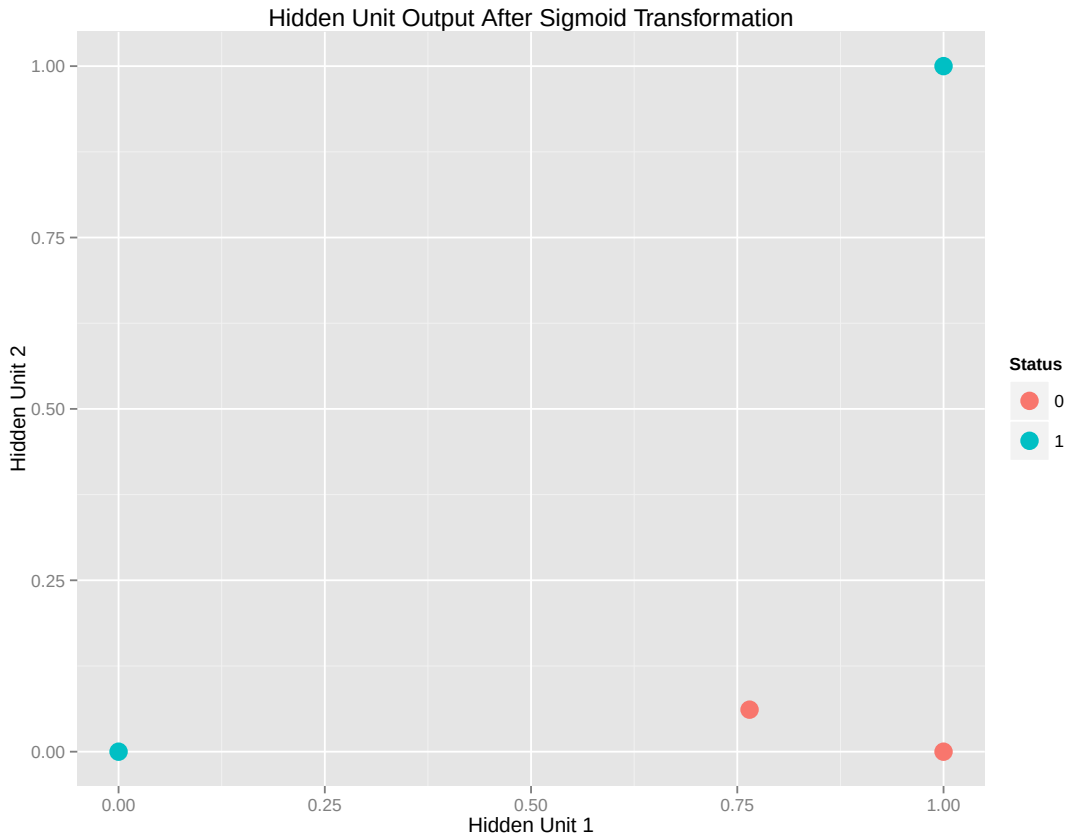

Figure 1.2: The output of both hidden units after the nonlinear logistic transformation, with training halted before convergence. Note that the classes can clearly be separated by a linear decision boundary now.

ality being the command line tool offered as part of [Neal, 1995] and one matlab package that accompanied [Nabney, 2002]. The scarcity of up to date packages has contributed to Bayesian neural networks relative obscurity. In contrast, standard neural networks have hundreds of packages in most languages such as R, Matlab, Python, Java, C/C++, Lisp, and Perl, just to name a few.

Bayesian neural networks are related to several other Bayesian and non-Bayesian methods. These relationships are shown in Figure 1.3. If we consider a Bayesian neural network with an infinite number of hidden units, then this model is equivalent to *Gaussian Process* [Neal, 1995, Rasmussen, 2006], which is another popular and well-studied method in statistics and

machine learning. Conversely, if we consider a neural network without any hidden units at all, then the model collapses to Bayesian versions of logistic and linear regression, depending upon the form of the output layer. Finally, if instead of performing a fully Bayesian analysis on a network with no hidden units, we use the *maximum a postori* parameter estimates, we obtain popular penalized regression models under various priors. If the network prior is Gaussian, the MAP estimate corresponds to a Ridge penalty (also called an L_2 penalty) while if we had used a Laplace/Double exponential prior, the MAP estimate yields the very popular LASSO model (i.e. an L_1 penalty). Figure 1.3 shows these relationships graphically.

1.4 Overview of the Thesis

In this thesis we investigate using Bayesian neural networks for genetic association studies of various forms, in the presence of gene-gene interactions. The main contributions of this work are the development of a GPU infrastructure for the Hamiltonian Monte Carlo algorithm (Chapter 2), a Bayesian test of variable importance (Chapter 3), an extension to existing Bayesian neural network techinques for multi-response data (Chapter 4), and a flexible software package for fitting and visualizing models of this sort (Chaper 5).

Since the work presented in this thesis represents three projects that were submitted to three different journals, there will necessarily be some overlap in presentation of the material. Chapters 2, 3, and 4 all present the fundamentals of Bayesian inference and at least some discussion of Hamiltonian Monte Carlo. Readers comfortable with the material after its initial introduction can safely skip the subsequent presentations in later chapters. Likewise, the formulation for the neural network model and the Bayesian extension occurs in both Chapters 3 and 4.

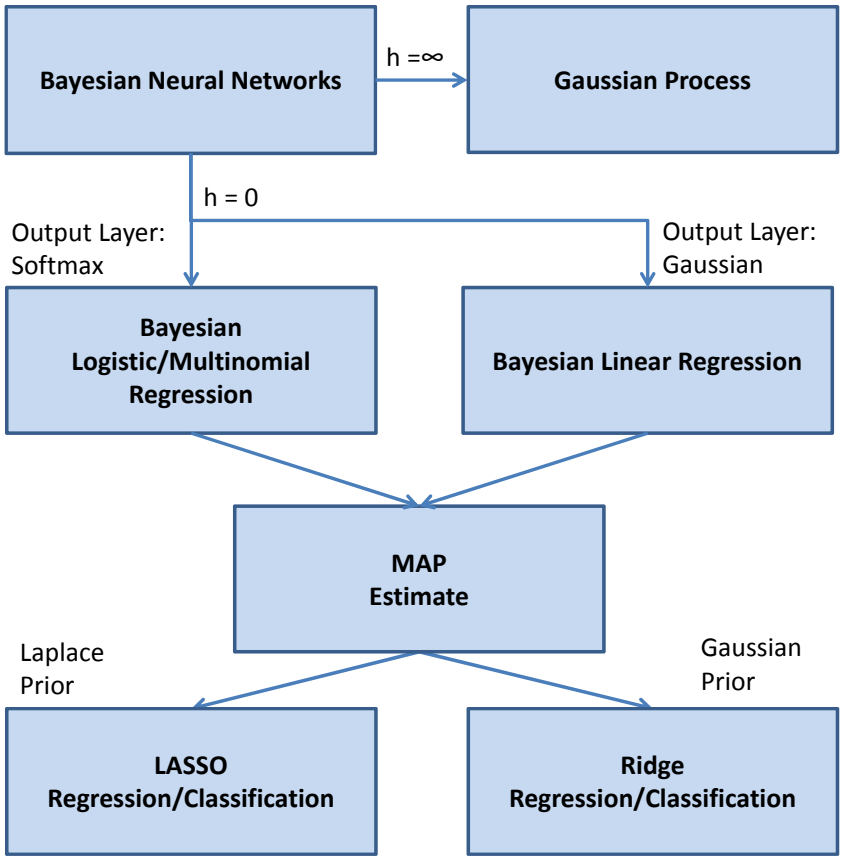

Figure 1.3: Relationship between Bayesian neural networks and several other popular statistical and machine learning methods.

Chapter 2

High-Dimensional Hamiltonian Monte Carlo via Graphics Processing Units

2.1 Introduction

Bayesian methods continue to grow in popularity for a variety of data analyses and have many appealing properties, especially when the primary task is prediction. Bayesian methods offer a coherent and principled framework for model regularization that is often crucial in high-dimensional settings. Many popular frequentist approaches, such as the Lasso [Tibshirani, 1996] and Ridge Regression [Hoerl and Kennard, 1970], have been used extensively in high-dimensional settings are well known as approximating a Bayesian posterior distribution under various priors. Probabilistic inference based on the full posterior distribution using analytical or deterministic numerical methods in large scale problems is almost always intractable, so Markov Chain Monte Carlo (MCMC) techniques are often used to draw samples from the posterior distribution based on only the kernel of the posterior density. These approaches do not require the normalizing constant (i.e. the marginal density of the data) which may be

prohibitively expensive to compute. However for most of the popular MCMC methods, such the the random-walk Metropolis-Hastings (MH) algorithm, the expected computation needed to draw a nearly independent sample from the posterior grows on the order of d^2 [Neal, 2011], where d is the dimensionality of the parameter vector. This 'curse of dimensionality' renders the Metropolis-Hastings (MH) algorithm ineffective for high-dimensional problems.

A variant of the MH algorithm, known as Hamiltonian Monte Carlo (HMC), has gained popularity for generating samples from the posterior kernel. HMC uses information about the gradient of the logarithm of the posterior kernel to obtain samples from a proposal distribution. These informed proposals avoid much of the random-walk behavior that plagues Metropolis-Hastings to provide much faster mixing times, as the number of steps needed is only on the order of $d^{5/4}$ [Neal, 2011]. Though HMC's mixing properties are much better than random-walk MH, HMC comes with a much higher cost to generate samples from the proposal distribution. Using the popular 'leap-frog' algorithm to perform the simulation of Hamiltonian dynamics, each sample requires evaluating the gradient of the log-posterior kernel with respect to each parameter being sampled some number L times, where reasonable values for L can range anywhere from 10 to 10,000 depending on the complexity of the statistical model. This can be a severe computational burden when the dimensionality of the parameter is very high.

Herein we propose simple modifications to alleviate the expensive portions of HMC. We formulate each task in terms of simple matrix or element-wise operations that can be readily carried out on a graphics processing unit (GPU). We also propose the use of a set of persistent GPU objects to avoid the cost involved with transferring data between main and GPU memory. While these changes may seem like only a minor set of modifications, the resulting speedup is shown to be significant. In addition, with simplicity comes generality, so the techniques illustrated here should be applicable to a wide range of Bayesian models based on HMC sampling.

In the remainder of this section we will briefly discuss the basics of HMC and outline a framework for implementation using the Python programming language. In section 2, we explicitly

show the proposed representation for multinomial regression. In section 3 we demonstrate the effectiveness first using a synthetic dataset and next using a real, multi-class classification problem. We compare timing results with a popular penalized regression package in R.

2.1.1 Hamiltonian Monte Caro: A Brief Overview

The primary goals of the Bayesian inference are to obtain probabilistic inference about a parameter of interest and make predictive inference based on a given set of data. The model consists of a sampling density ($f(x|\theta)$) that generates data (X) conditional on the parameter (θ) and a prior density ($\pi(\theta)$). Bayesian inference is based on the posterior density, $p(\theta|x)$ given by the conditional density of θ given $X = x$:

$$p(\theta|x) = \frac{f(x|\theta)\pi(\theta)}{m(x)}$$

where $m(x) = \int f(x|\theta)\pi(\theta)\, d\theta$ denotes the marginal density of data at $X = x$. It is well known, even for most common models, the marginal density $m(x)$ can not be computed analytically or by means of (deterministic) numerical methods especially when the dimension d of θ exceeds 5 (e.g. see Chapter 5 of [Davis and Rabinowitz, 2007]). In such cases, Monte Carlo methods are often used which avoids the use of normalizing constant $m(x)$ by using the ratios of the posterior density evaluated at two different values of θ. In particular, notice that:

$$\frac{p(\theta|x)}{p(\theta'|x)} = \frac{f(x|\theta)}{f(x|\theta')} \cdot \frac{\pi(\theta)}{\pi(\theta')} = \frac{L(\theta|x)}{L(\theta'|x)} \cdot \frac{k(\theta)}{k(\theta')},$$

where $L(\theta|x) = c(x)f(x|\theta)$ and $k(\theta) = c\pi(\theta)$ denote the likelihood function and prior kernel that do not involve any multiplicative components which are functions of data x only. Most of the popular Monte Carlo methods, including the entire suite of MCMC methods, make use of the above property to generate samples from the posterior distribution based on only the posterior kernel $K(\theta|x) = L(\theta|x)k(\theta)$. The MH algorithm is one such approach to draw (dependent) samples from the posterior distribution using the sample path of a Markov chain

whose stationary distribution is the posterior distribution [**?**]. Starting with an arbitrary initial value $\theta^{(0)}$, the MH algorithm proceeds iteratively by generating $\theta^{(t)}$ from the previous value $\theta^{(t-1)}$ using the following two steps:

- Generate a candidate value θ from a (conditional) proposal density $T(\theta|\theta^{(t)})$
- Accept $\theta^{(t)} = \theta$ with probability $\rho(\theta|\theta^{(t-1)})$ or set $\theta^{(t)} = \theta^{(t-1)}$.

The crux of the algorithm depends of the acceptance probability given by:

$$\rho(\theta|\theta') = \min\left(1, \frac{K(\theta|X)T(\theta'|\theta)}{K(\theta'|X)T(\theta|\theta')}\right)$$

It can be shown that the above scheme generates a Markov chain $\{\theta^{(t)};\ t = 0, 1, 2, \ldots\}$ whose stationary distribution is the posterior distribution $p(\theta|x)$ under very mild regularity conditions on the proposal density $T(\theta|\theta')$ [Tierney, 1994]. Hamitonian Monte Carlo (HMC) [Duane et al., 1987] modifies this basic algorithm by simulating Hamiltonian dynamics to propose new states. HMC moves in directions of high posterior density by taking steps guided by the gradient of the log-posterior with respect to θ, similar in spirit to *gradient descent* algorithms used in optimization problems. Here we present briefly the technical aspects of HMC, but for a more comprehensive treatment, please see [Neal, 2011].

HMC introduces auxiliary 'momentum' variables that are used to simulate and update the 'position' variables. The position variables represent the parameter vector we are interested in sampling. These two sets of variables are updated according to Hamilton's equations. In practice a discretized version, known as the 'leap-frog' method is often used to update the momentum and position vectors. Given a momentum vector at iteration t, $\eta^{(t)}$, and a parameter vector of interest, $\theta^{(t)}$, the update proceeds by first taking a half-step of size $\epsilon/2$ for $\eta^{(t)}$, a full

step of size ϵ for $\theta^{(t)}$, and final half step for $\eta^{(t)}$:

$$\eta^{(t+\frac{1}{2})} = \eta^{(t)} + \frac{\epsilon}{2} \cdot \nabla \log(K(\theta^{(t)}|X)) \tag{2.1}$$

$$\theta^{(t+1)} = \theta^{(t)} + \epsilon \cdot \eta^{(t+\frac{1}{2})} \tag{2.2}$$

$$\eta^{(t+1)} = \eta^{(t+\frac{1}{2})} + \frac{\epsilon}{2} \cdot \nabla \log(K(\theta^{(t+1)}|X)) \tag{2.3}$$

Accordingly, the acceptance probability has to be modified to incorporate the momentum variables. If we could simulate Hamilton's equations perfectly, we would accept every proposal, but because the discretization introduces error, we use the following the acceptance procedure to ensure the validity of the Markov chain is intact:

$$\min\left(1, \frac{\exp\left(\log(K(\theta^{(t+1)}|X)) - \eta^{(t+1)} \bullet \eta^{(t+1)}\right)}{\exp\left(\log(K(\theta^{(t)}|X)) - \eta^{(t)} \bullet \eta^{(t)}\right)}\right) \tag{2.4}$$

where $\eta^{(t+1)} \bullet \eta^{(t+1)}$ is the dot-product of the vector $\eta^{(t+1)}$ with itself. The procedure outlined in (2.1)-(2.3) is one 'leap-frog' update. It is usually desirable to move as far as possible from the current state, so often L leap-frog updates are done per iteration, where L now becomes a tuning parameter of the algorithm, as is the step size ϵ. HMC thus requires L evaluations of the gradient for the generation of every proposal state, followed by an evaluation of the log-posterior kernel to decide if the proposal should be accepted or rejected. The choice of the stepsize ϵ and the trajectory length L depends on the particular statistical model and judicious choices must be made to acheive good mixing of the markov chain. Preliminary runs and trace plots are often used to select these crucial tuning parameters, while automatic selection remains the central challenge to 'turnkey' HMC methods. Work continues to be done on fully automated HMC with recent efforts [Hoffman and Gelman, 2012] achieving good success. All HMC methods remain computationally expensive relative to their non-gradient based MCMC counterparts and, as we illustrate through examples, this disadvantage can be reduced if we

leverage GPU-based computing.

2.1.2 GPU Programming in Python

The use of general purpose graphics processing units (GPGPUs) for computationally intensive tasks has seen a considerable increase in recent years due to their potential to yield impressive speed-ups for certain classes of problems. GPU computing represents a substantially different paradigm than traditional CPU-based processing which typically executes instructions serially using a single (or as is becoming more common several) processor(s). In contrast, GPUs are composed of 100s to 1000s of processing cores that are able to perform computations on blocks of data in a highly parallel fashion. However they do not, in general, perform serial tasks well, so GPUs are maximally useful when computation can be decomposed in terms of a self-contained function (often called a 'kernel') that can be executed in parallel on sub-blocks of data (we omit using this definition of a kernel further to avoid confusion with the statistical concept of a kernel used in this paper). Frameworks to aid in program design of this single instruction, multiple data (SIMD) paradigm have seen development from hardware vendors and open-source projects. Currently, two of the most popular frameworks are Nvida's proprietary Compute Unified Device Architecture (CUDA) [Nvidia, 2008] which can only be used with Nvidia hardware and OpenCL [Khronos, 2008] which seeks to enabled open-source, platform independent GPU computing. Both frameworks provide C-like compilers that allow users to write functions to be executed on a GPU in addition to providing utilities to aid in program development.

Python [Sanner et al., 1999] is an interpreted programming language available for all major operating systems. It has a growing set of statistical libraries and features a syntax that will be familiar to R users. Python has a increasingly mature set of GPU interfaces that facilitate easy parallel programming, a feature which R presently lacks. The PyCuda library [Klockner et al., 2012] is built on top of the CUDA and platform provides many abstractions that make GPU programming much easier. It features support for GPU array objects that mirror Pythonic

arrays and are similar in nature to matrix objects in R.

The GPU array abstraction allows for operations on and between GPU arrays without the need to explicitly manage the underlying hardware. Once a GPU array has been created, it can be used much like a normal array, except that the computation happens using the GPU as opposed to the central processing unit (CPU). Moreover, it uses Python's built in memory management to automatically free dereferenced objects, freeing the programmer from having to explicitly free unused memory. Should a problem require a custom piece of GPU code, PyCuda features a 'just-in-time' compiler that can compile source modules at runtime, which provides a great deal of flexibility. Additionally, the *cuda* package in the SciPy library [Jones et al., 2001] is built on top of PyCuda and offers many linear algebra functions. One of the draw backs of typical GPU programming is that execution code and data must be transferred from main memory to the GPU's local memory. This transfer incurs a latency penalty, which may result in a GPU-based program taking longer to execute than a CPU implementation if the latency is large relative to the execution time. Using PyCuda and the cuda-SciPy libraries, we can easily create persistent GPU based objects and repeatedly update these objects which remain in GPU memory to avoid transfer latencies. We note that while we use Python for demonstration purposes, the approaches described here should be easily ported to any CUDA based programming environment.

2.2 Methods

2.2.1 Bayesian Multinomial Regression

Let $x_i =< x_{i1}, \ldots, x_{ip} >^T$ be a vector of predictors and $y_i =< y_{i1}, \ldots, y_{iK} >^T$ be a vector of mutally exclusive indicators of class membership, where each $y_{ik} \in \{0, 1\}$. We wish to classify new observations into 1-of-K classes. The relationship between x_i and y_i is achieved through

the generalized logit, often called the *softmax* function:

$$Pr(y_{ik} = 1|x_i, \beta_k) = \psi(x_i, \beta_k) = \frac{exp(x_i^T \beta_k)}{\sum_{j=1}^{K} exp(x_i^T \beta_j)} \tag{2.5}$$

where β_k is the the $p \times 1$ vector of regression coefficients associated with the k-th class. Assuming each y_i follows a multinomial distribution, the log-likelihood for all n observations, $l(B|X, Y)$, is:

$$l(B|X, Y) = \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log(\psi(x_i, \beta_k)) \tag{2.6}$$

where $B_{p \times k} = [\beta_1, \ldots, \beta_K]$ is the matrix containing the vectors of regression coefficients for all K classes. We set $\beta_K = 0$ to ensure identifiability of the parameters (notice that as the multinomial probabilities add up to one, we need a constraint to identify the β's uniquely). Following the recommendations in [Gelman et al., 2008], we place a non-conjugate, weakly informative standard Cauchy distribution on each element of B. In this notation β_{jk} indexes the single regression coefficient of B associated with variable j for class k, while β_k refers to the $p \times 1$ vector of all regression coefficients for class k. The corresponding log-posterior kernel is:

$$\log(K(B|X, Y)) = \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log(\psi(x_i, \beta_k)) + \sum_{k=1}^{K} \sum_{j=1}^{p} -\log(1 + \beta_{jk}^2) \tag{2.7}$$

where $X_{n \times p} = [x_1, \ldots, x_n]^T$ and $Y_{n \times k} = [y_1, \ldots, y_n]^T$. We can now rewrite the first term of the log-likelihood as $Y \star \log(\psi(XB))$, which can be computed using only linear or element-wise operations. In detail, XB is carried out via matrix-matrix multiplication, followed by a row-wise softmax transformation $\psi(\cdot)$, and finally an element-wise $\log(\cdot)$ transformation. An element-wise multiplication between Y and the resulting $n \times k$ matrix, $\log(\psi(XB))$, yields the log-likelihood values for all n observations and k classes. Instead of looping over elements of this matrix to sum the total log-likelihood value, we can pre-multiply by $1_{1 \times n}^T$ to sum over all observations and post-multiply by $1_{k \times 1}$ to sum over all classes. The contribution from the prior

in (2.7) can also be computed in similar fashion. Let $\Gamma_{p\times k}$ be the matrix containing the prior's contribution, where $\Gamma[j,k] = -\log(1+\beta_{jk}^2)$. The equivalent form of (2.7) that only uses linear or element-wise operations is shown below.

$$\log(K(B|X,Y)) = 1_n^T \left(Y * \log(\psi(XB))\right) 1_k + 1_p^T \Gamma 1_k \tag{2.8}$$

Using similar reasoning, we can rewrite the evaluation of the gradient in terms of matrix and element-wise operations. The result is shown below, please see the appendix for the full derivation in addition to discussion on maintaining identifiability during the gradient updates.

$$\left[\frac{\partial \log(K(B|X,Y))}{\partial B}\right]_{p\times k} = X^T \left(Y - \psi(XB)\right) + \Gamma \tag{2.9}$$

For this statistical model, we propose the use persistent PyCuda GPU arrays for X, Y, B, η, and $\left[\frac{\partial \log(K(B|X,Y))}{\partial \beta}\right]$, allowing us to perform the entire HMC simulation in GPU memory. without having to move objects from main memory into the GPU's local memory.

2.3 Results

2.3.1 Computational Efficiency Analysis

We begin by first presenting some timing results for both the CPU and GPU implementations that will serve as a baseline for comparison. All simulations were written in Python and executed on a computing cluster node with 2 Xeon E5-2670 processors, 256GB of RAM, and a NVIDIA Tesla K20 GPU with 2500 CUDA cores and 5GB of RAM using NVIDIA's CUDA 5.0 SDK. The CPU version was implemented using the numpy library and the GPU version was implemented using the previously mentioned PyCuda and CUDA scikit for SciPy. It should be noted that the numpy library for linear algebra in Python is a high-level wrapper for the BLAS library [Lawson et al., 1979] and will, in general, be comparable in speed to a pure C

implementation for linear algebra tasks. The CPU and GPU implementations used identical representations, with the only differences being to differences required by numpy and PyCuda. However, any differences in execution speed should be largely attributable to speeds offered by the hardware the code was executed on (i.e. CPU vs GPU). For all computations, 32-bit floating point numbers were used in both GPU and CPU implementations.

For several values of sample size (N), number of classes (K), and dimension (p) we performed a single gradient evaluation and a separate, single leap-frog update five times and recorded the the average time taken. Each value of X and B was drawn from a $N(0, 10^{-3})$. Y generated according to (2.5), given the values of X and B. We swept each value shown in Table 2.1, for a total of 224 parameter combinations. The results of the simulations are summarized in Figures 2.1 and 2.2.

Table 2.1: Parameter values used in timing evaluations.

Parameter	**Values Swept**
N	$100, 1000, 5000, 1000$
K	$2, 3, 4, 5, 10, 15, 20$
P	$10, 50, 100, 500, 1000, 5000, 10000, 20000$

Figures 2.1 and 2.2 show that the computational core of the HMC algorithm (the gradient evaluation and leap-frog update) benefit greatly from the use the GPU. For moderately large problems ($N > 1000$ and $p \geq 5000$) the GPU is 100-150 times faster than the corresponding CPU version. Only for certain instances of the smallest problem size ($N = 100$) was the GPU slower than the CPU. However, even for small sample sizes many cases were still several times faster on the GPU. Though larger problems benefit proportionally more from the parallelism offered by the GPU, problems of more modest size may still benefit from this approach.

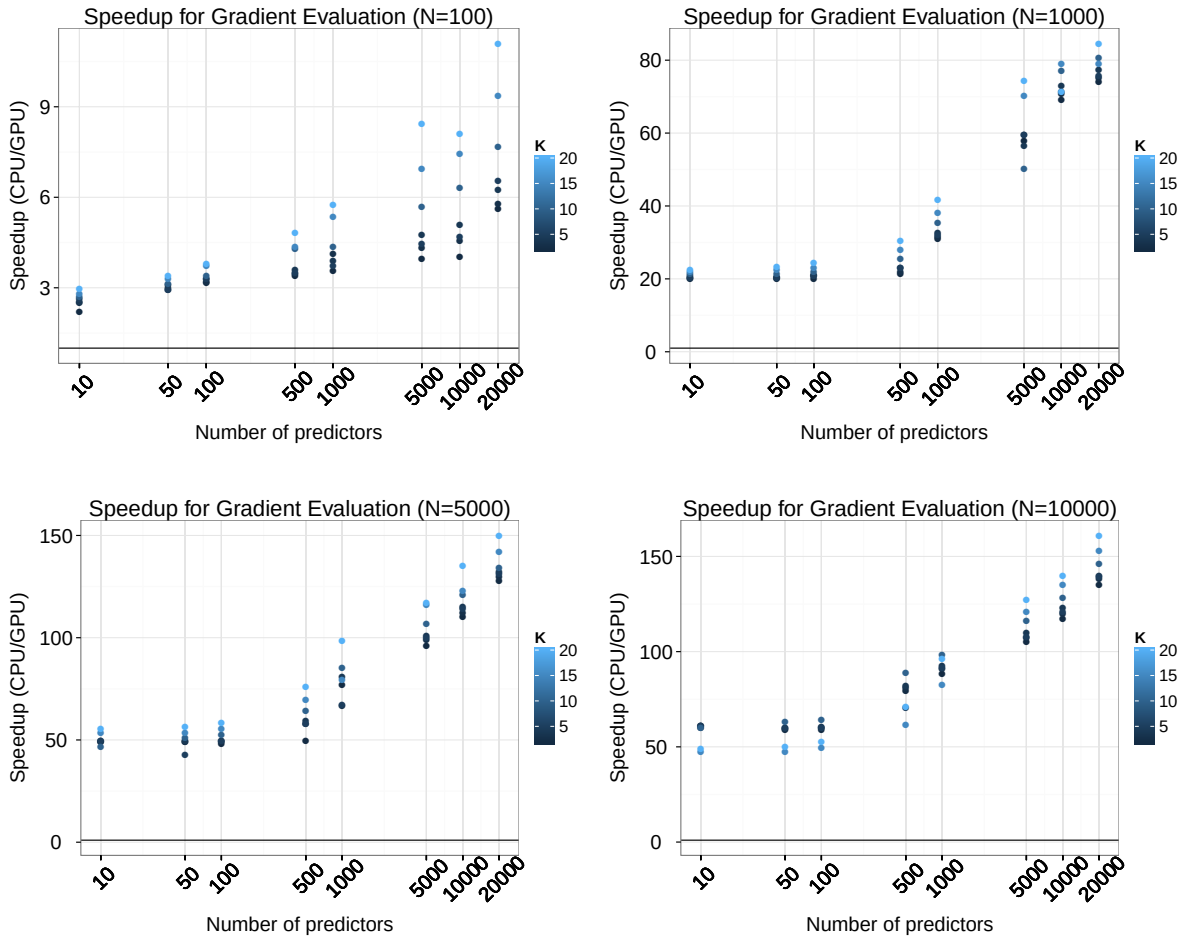

Figure 2.1: Average timing results for one gradient evaluation for various combinations of sample size, parameter dimension and number of classes. Each dot represents one such combination. Each panel represents a different sample size, the x-axis is the number of predictors, and the color of the dot represents the number of classes (lighter colors correspond to larger values).The y-axis is the time taken for the CPU divided by the time taken for the GPU. The black horizontal line is at 1 - values below this line are faster for the CPU and values above are faster on the GPU.

2.3.2 Handwritten Digit Recognition

In this section, we apply the formulation of Section 2 to a real dataset. The MNIST handwritten digit dataset [LeCun et al., 1995] is a popular benchmark in computer vision and machine learning. Each of the 70,0000 non-sparse observations represents the pixel intensity from a

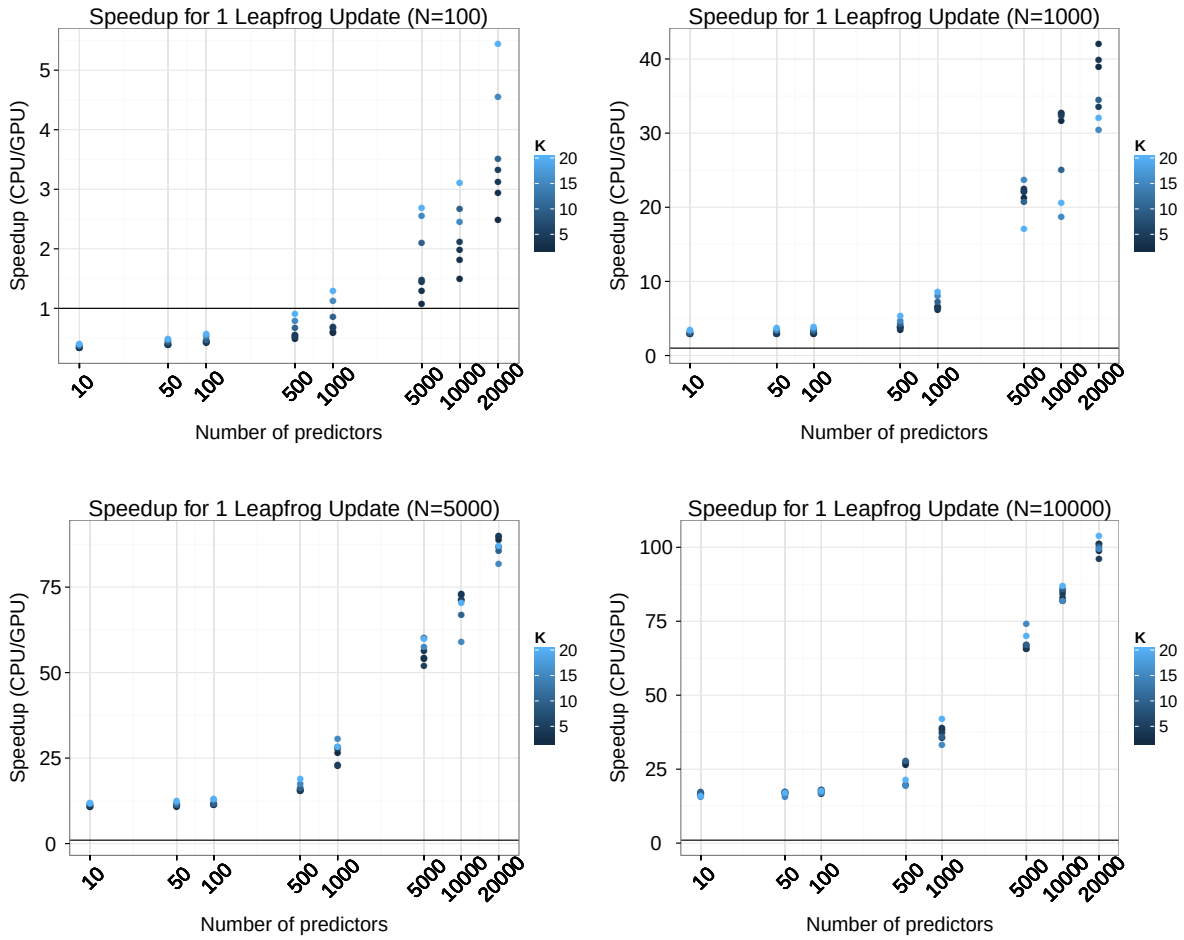

Figure 2.2: Average time required for one 1 leap-frog update for various combinations of sample size, parameter dimension, and number of classes. Each dot represents one such combination. Each panel represents a different sample size, the x-axis is the number of predictors, and the color of the dot represents the number of classes (lighter colors correspond to larger values).The y-axis is the time taken for the CPU divided by the time taken for the GPU. The black horizontal line is at 1 - values below this line are faster for the CPU and values above are faster on the GPU.

28x28 image of a digit between 0 and 9. The images have been preprocessed, centered, and normalized such that they all are of comparable intensities. Using this dataset, we performed multinomial regression to classify each observation into one of the 10 possible classes. We used a 'flattened' representation such that the 28x28 square becomes one vector with 784 features.

We used the first 60,000 observations to build the model and withheld the final 10,000 as a test set. Thus our data is a predictor matrix that includes an intercept, $X_{60,000x785}$ and a response matrix, $Y_{60,0000x10}$ and a coefficient matrix β_{785x10}. We used the model outlined in Section 2 and performed HMC with $L = 100$ and $\epsilon = 10^{-4}$ during the burn in phase and $\epsilon = 7 * 10^{-5}$ during the sampling phase. The values of L and ϵ were selected to achieve an acceptance rate between 0.65 and 0.75 for the sampling phase in accordance with the suggestions in [Neal, 2011]. In order to get the chain moving during the burnin phase, we adopt an *annealed* acceptance strategy. Let α be the usual acceptance probability in (2.4), we modify the procedure such that at iteration t we accept a proposal instead with probability $\alpha^{T_{(t)}}$, where $T_{(t)}$ is set according to an *annealing schedule*, $T_{(t)} = \max(1, r * T_{(t-1)})$, for some $r < 1$. This procedure helps the simulation get started and lessens the chance of getting stuck in a local minima early in the simulation. We do not anneal during the sampling phase, so the stationary distribution of the Markov chain remains the one defined by (2.7). We allowed the chain to burn in for 100 iterations with an initial $T_{(0)} = 10^3$ and $r = 0.9$ before beginning the sampling phase. For predicting class membership, we predicted the class that had the highest average softmax value in the posterior samples.

For comparison, we also fit a penalized multinomial regression model under an L_1 penalty, i.e. the Lasso penalty. This model is well known to be equivalent to the *maximum a posteri* (MAP) estimate of a Bayesian posterior under a Laplace prior. We used the *glmnet* [Friedman et al., 2010] package in R, which is uses one of the most efficient methods available for this type of model. In [Friedman et al., 2010] the authors compared *glmnet* to a popular approximate Bayesian software package BBR [Genkin et al., 2007] that also produces a MAP estimate for each parameter. At that time BBR was one of the fastest approaches available and Friedman et. al found that glmnet was significantly faster on most datasets they analyzed. This indicates that *glmnet* should serve as a good baseline for timing performance evaluation. Note also that the approach offered here does not provide a MAP estimate, but instead provides samples from the *full* posterior distribution.

In our comparison the shrinkage parameter (λ) in *glmnet* was selected via 5-fold cross-validation. Using the parallel processing option in *glmnet*, we were able to fit each fold simultaneously on separate CPU cores, making the time required for 5-fold CV equivalent to fitting a single model. The number of λ values tested during the cross-validation has a large impact on the time needed to build the model, so we first used the default of 100 values for λ and then fit using larger grids of 500 and 1000. The results are summarized in the Table 2.2:

Table 2.2: Results for MNIST dataset. The Bayesian-HMC method always burned in for 100 iterations and used $L = 100$ and $\epsilon = 7 * 10^{-5}$ for HMC sampling after the burn in phase.

	Bayesian-HMC		**glmnet**		
Tuning Parameter	100 samples	500 samples	$\#\lambda = 100$	$\#\lambda = 500$	$\#\lambda = 1000$
Train Accuracy	0.928	0.932	0.922	0.932	0.932
Test Accuracy	0.92	0.921	0.920	0.926	0.926
Time (hr:min:sec)	0:3:54	0:11:42	6:43:59	2:25:12 [1]	3:41:3[1]

2.4 Discussion

We have outlined a general method for performing HMC-based Bayesian analyses using GPUs. Our results show that GPUs can greatly speed up the expensive parts of HMC, often reducing the time needed for the core components of HMC by more the 100-fold for large problems. This is a considerable amount time saved and not only reduces tedium experienced by the analyst waiting for a simulation to finish, but also opens Bayesian analyses to a larger class of problems. This framework should easily extend to more complex models, including hierarchical models, as the gradient calculations can be propagated from the highest levels down to the lowest levels using matrix multiplication. There many additional libraries available in Python not explored here that can aid in running these types of models on a GPU. In particular, Theano [Bergstra et al., 2010] and PyAutoDiff can perform automatic differentiation and gradient evaluation

[1] A larger grid of values resulting in a shorter execution time than a smaller grid was confirmed as normal behavior. Personal correspondence with *glmnet* package maintainer.

using GPUs. This type of framework would free a user from having to calculate the gradient by hand and program the corresponding GPU-friendly representation. HMC-based sampling with automatic differentiation is currently available in the stand alone software package known as Stan [Stan Development Team, 2013], which is growing quickly in popularity. Stan uses a much more sophisticated sampling scheme than the one presented here, namely it uses the 'No-U-Turn' (NUTS) sampler that adaptively tunes the step size ϵ and the trajectory length L. We explored the use of Stan for comparison in this study, because it compiles a BUGS like model specification into fast C++ code. This is an exciting and ongoing project, but unfortunately at the time of this writing, some of the computational infrastructure needed by the models in this study were not fully optimized [2], so we omitted Stan from the comparisons shown here.

In comparison to a popular penalized regression approach on the large MNIST dataset, we were able to complete a fully Bayesian analysis in a fraction of the time, while achieving nearly identical accuracy results. Conceivably we could have used this speedup to explore more complex models and prior structures to possibly achieve greater accuracy, but since timing was of primary interest and the connection between penalized regression and Bayesian models, we did not explore more sophisticated models. These results are not comprehensive evaluation of timings between these two methods, but is only meant to highlight that it is possible to do a fully Bayesian analysis in relatively short order on a large problem.

In conclusion, it seems that in the future all types of analyses must evolve to cope with the burden imposed by ever increasing amounts data. The framework presented here has the potential to enable the use of fully Bayesian approaches on these ever larger datasets. Moreover, on board GPU RAM will continue to increase according to Moore's law, allowing for ever larger simulations to be performed completely in GPU memory. However, before concluding we pause for a moment of truth-in-advertising. The type of speedup offered here is not limited to Bayesian approaches, but can be used in general by any method where likelihood and gradient evaluations can be expressed in terms of linear algebra operations such as gradient descent, Gauss-Newton,

[2] Personal correspondence with Stan development team

and many others. HMC is only one such method that is highly amenable to GPU speedup. However, the Bayesian approach is very attractive due to the coherent model regularization as well as the quantification of parameter uncertainty offered by the posterior distribution. Corresponding estimates of parameter uncertainty are currently very difficult for the penalized regression methods, so we find the Bayesian framework very appealing in high-dimensional settings. We hope the results shown in this study will encourage further development in the use of GPUs for high-dimensional statistical problems.

2.5 Appendix

Code from this paper is available at `https://github.com/beamandrew/HMC_GPU`

2.5.1 Gradient derivations

We need to compute the gradient of the log-kernel with respect to each β_{jk}. Application of the chain rule to (2.7) and differentiating with respect to β_{jk} yields:

$$\frac{\partial \log(K(B|x_i, y_i))}{\partial \beta_{jk}} = \frac{\partial \partial \log(K(B|x_i, y_i))}{\partial \log(\psi(x_i, \beta_k))} \frac{\partial \log(\psi(x_i, \beta_k))}{\partial \beta_{jk}} \left[\sum_{c=1}^{K} y_{ic} * \log(\psi(x_i \beta_c)) \right] + \frac{\partial}{\partial \beta_{jk}} \left[-\log(1 + \beta_{jk}^2) \right] \tag{2.10}$$

$$\frac{\partial l \log(K(B|x_i, y_i))}{\partial \beta_{jk}} = y_{ik} \frac{\partial \log(\phi(x_i, \beta_k))}{\partial \beta_{jk}} \left[\log(\psi(x_i, \beta_k)) \right] + \frac{2 * \beta_{jk}}{1 + \beta_{jk}^2} \tag{2.11}$$

$$\frac{\partial \log(K(B|x_i, y_i))}{\partial \beta_{jk}} = y_{ik} - \psi(x_i, \beta_k) + \frac{2 * \beta_{jk}}{1 + \beta_{jk}^2} \tag{2.12}$$

where (2.12) follows from the fact that $\frac{\partial \log(\psi(x_i, \beta_j))}{\partial \beta_{jk}} \log(\psi(x_i, \beta_k) = 1 - \log(\psi(x_i, \beta_k))$ if $y_{ik} = 1$ and $\frac{\partial \log(\psi(x_i, \beta_k))}{\partial \beta_{jk}} \log(\psi(x_i, \beta_k) = -\log(\psi(x_i, \beta_k))$ if $y_{ik} = 0$. We need sum over all n observations to obtain the full gradient for β_{jk}, but instead of summing over all cases in a loop, we pre-multiply the vector of differences, $Y_{nxk} - \psi(X_{nxp} B_{pxk})$, by X_{pxn}^T, yielding the matrix of partial

derivatives:

$$\left[\frac{\partial \log(K(\beta|X,Y))}{\partial \beta}\right]_{pxk} = X^T\left(Y - \psi(X\beta)\right) + \Gamma \tag{2.13}$$

where the Γ matrix is now the $p \times k$ matrix with $\Gamma[j,k] = \frac{2*\beta_{jk}}{1+\beta_{jk}^2}$. Note that the equation (2.13) does not take into account the identifiability constraint, $\beta_K = 0$. In order to maintain identifiability, we must first set $\beta_K = 0$ and ensure that the gradient for β_K is uniformly 0 (as well as the momentum component for β_K used in the HMC procedure). The way in which we do this is context dependent. If we are operating in a programming environment that allows for sub-array access, also known as *slicing*, we can simply update the sub-array of the gradient matrix containing only the unconstrained coefficients. R and Python both support this kind of memory access, however there may be some penalty involved in accessing the underlying sub-array. To evaluate the best-case scenario for CPU-based code, we maintain a *separate* $(p-1)\times k$ matrix for the gradient and leap-frog update timings in Section 3.1. This would correspond to the ability to access the sub-array for the unconstrained coefficients with no penalty. This approach allows us to put a lower-bound on the speed-up offered by the GPU.

However, PyCuda does not currently support slicing on GPU arrays. We could write custom CUDA C-code to only update the unconstrained coefficients, but this would greatly increase the complexity of the provided code, and users who are not familiar with GPU programming would likely get little from it. Instead, we create a $p \times k$ *mask* matrix M, where $M_{jk} = 1$ if $k \neq K$ and 0 otherwise. We element-wise multiply this mask to the result in (2.13) to zero out all of the elements associated with β_K. Doing this maintains identifiability during each gradient update, but comes at cost of $O(pk)$ each time, since we have to element-wise multiply by M everytime we update the gradient. However, as the results section show, this additional cost is relatively small compared to speedup offered by the GPU framework. We present this method to maintain identifiability because routinely we are interested the interpretation of each regression coefficient. However, if the task is *only* prediction, this concern can be safely ignored.

For completeness, we have included the results of evaluating the expressions in (2.7) and (**??**) when identifiability is not a concern. These results are summarized Figures 2.3 and 2.4.

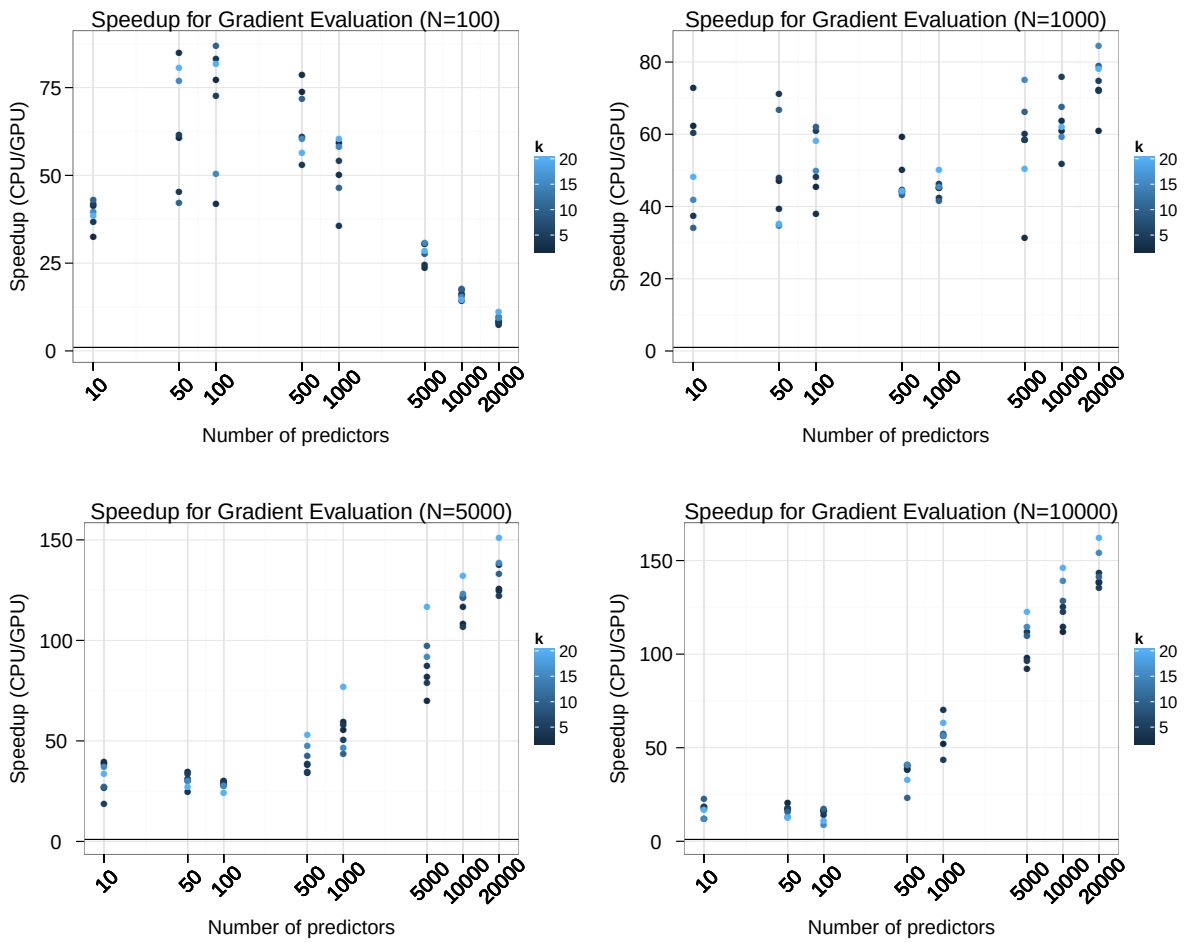

Figure 2.3: Timing results for gradient evaluation when identifiability is not enforced.

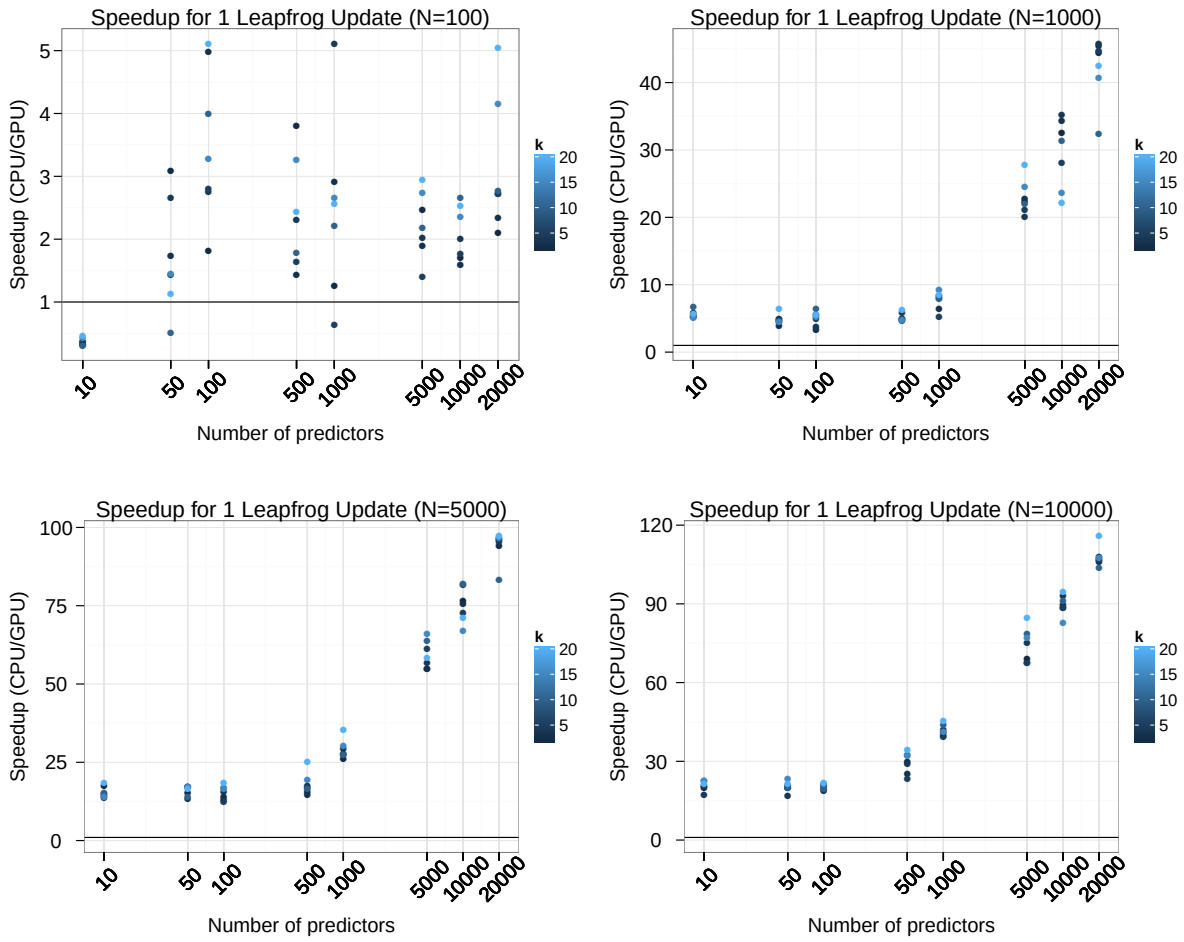

Figure 2.4: Timing results for 1 leap-frog update when identifiability is not enforced.

Chapter 3

Bayesian Neural Networks for Genetic Association Studies of Complex Disease

3.1 Background

The ability to rapidly collect and genotype large numbers of genetic variants has outpaced the ability to interpret such data, leaving the genetic etiology for many diseases incomplete. The presence of gene-gene interactions, or epistasis, is believed to be a critical piece of this missing heritability [Manolio et al., 2009]. This has in turn spurred development on advanced computational approaches to account for these interactions, with varying degrees of success [Motsinger-Reif et al., 2008b, Motsinger-Reif et al., 2008a, Koo et al., 2013]. The main computational challenge comes from the vast number of markers that are present in a typical association study. This problem is exacerbated when interactions between two or more markers must be considered. For example, given an experiment that genotypes 1,000 markers, examining all possible interactions between two of the markers involves consideration of nearly half a million combinations. This situation becomes exponentially worse as higher order interactions are

considered. Modern genome-wide association studies (GWASs) routinely consider 1-2 million single nucleotide polymorphisms (SNPs), which would require examining half a trillion potential interactions. As whole genome sequencing (WGS) methods become commonplace, methods that cannot cope with large data sets will be of little utility. Data on this scale will require approaches that can find interactions without having to enumerate all possible combinations. As genotypic technology advances, datasets now routinely include millions of SNPs.

Several distinct types of methods have emerged that attempt to address this challenge. Perhaps one of the most popular approaches from the last decade has been Multifactor Dimensionality Reduction (MDR) [Moore et al., 2006, Hahn et al., 2003], and extensions of the method. MDR is a combinatorial search that considers all possible interactions of a given order and selects the best model via cross validation. Because MDR is an exhaustive search, it suffers from the previously discussed scalability issue, though recent work using graphics processing units has attempted to lessen this deficit [Greene et al., 2010]. MDR is reliant upon a permutation testing strategy to assess statistical significance for each marker, so the computational burden becomes prohibitive for large datasets. Permutation testing computes a p-value for-a statistic of interest (such as an accuracy measure from MDR) by randomly permuting the class labels and calculating the statistic on the permuted dataset. This procedure is repeated many times to compute a null distribution for the statistic of interest. The relative percentage of instances in the permuted null distribution that are less than or equal to the actual statistic from the unpermuted data is taken as the desired one-sided p-value. Unfortunately, this can be extremely expensive for large datasets when many hypotheses are simultaneously tested, leading to a large multiple testing scenario. To get the required resolution for a Bonferroni corrected p-value of 0.05 when considering a set of 1,000 SNPs, one must perform 20,000 permutations. This makes permutation testings infeasible for even moderately sized datasets.

Another popular approach is Bayesian Epistasis Association Mapping (BEAM) [Zhang and Liu, 2007]. BEAM partitions markers into groups representing individual (i.e. marginal) genetic effects, interactions, and a third group resenting background markers that are uninvolved with

the trait. BEAM employs a stochastic Markov Chain Monte Carlo (MCMC) search technique to probabilistically assign markers to each group and uses a novel B-statistic based on the MCMC simulation to assign statistical significance to each marker. This allows BEAM to assign statistical significance without the need to perform a costly permutation test. This method has been demonstrated successfully on data sets with half a million markers. However, the recommended amount of MCMC iterations needed is quadratic in the number of SNPs considered [Zhang and Liu, 2007], possibly limiting its effectiveness for larger datasets.

Many popular machine learning algorithms have also been adopted for use in analyzing association studies. Notable examples are decision trees (both bagged, i.e. random forests, [Lunetta et al., 2004, Diaz-Uriarte and de Andres, 2006] and boosted [Li et al., 2011] support vector machines (SVM) [Guyon et al., 2002], Bayesian networks [Jiang et al., 2011], and neural networks [Motsinger-Reif et al., 2008b]. In particular, tree-based methods such as random forests and boosted decision trees have been found to perform well in several previous association studies [Lunetta et al., 2004, Li et al., 2011, Diaz-Uriarte and de Andres, 2006]. Machine learning approaches are appealing because they assume very little *a priori* about the relationship between genotype and phenotype, with most methods being flexible enough to model complex relationships accurately. However, this generality is something of a double-edged sword as many machine learning algorithms function as black boxes, providing investigators with little information on which variables may be most important. Typically it is the goal of an association study to determine which variables are most important, so a black box may be of little use. Some approaches have easy adaptations that allow them to provide such measures. Both types of tree based methods (bagged and boosted) can provide measures of relative variable importance [Breiman, 2001, Friedman, 2001], but these indicators lack measures of uncertainty, so they are unable to determine how likely a variable's importance measure is to occur by chance without resorting to permutation testing.

In this study, we propose the use of Bayesian neural networks (BNNs) for association studies to directly address some the issues with current epistasis modeling. While BNNs have been

previously developed and applied for other tasks [Lisboa et al., 2003, Baesens et al., 2002, Neal, 1995, Neal, 1992] they have yet to see significant usage in bioinformatics and computational biology. Like most complex Bayesian models, BNNs require stochastic sampling techniques that draw samples from the posterior distribution, because direct or deterministic calculation of the posterior distribution is often intractable. These posterior samples are then used to make inferences about the parameters of the model or used to make predictions for new data. Standard MCMC methods that employ a random walk such as the Metropolis-Hastings (RW-MH) algorithm [Metropolis et al., 1953, Hastings, 1970] (which is the algorithm that forms the core of BEAM [Zhang and Liu, 2007]) explores the posterior distribution very slowly when the number of predictors is large. If d is the number of parameters in a model, the number of iterations needed to obtain a nearly independent sample is $O(d^2)$ [Neal, 2011] for RW-MH. This makes the RW-MH algorithm unsuitable for neural network models in high-dimensions, so the Hamiltonian Monte Carlo (HMC) algorithm is instead used to generate samples from the posterior. HMC has more favorable scaling properties, as the number of iterations needed is only $O(d^{5/4})$ [Neal, 2011]. HMC achieves this favorable scaling by using information about the gradient of the log-posterior distribution to guide the simulation to regions of high posterior probability. Readers familiar with standard neural network models will notice an inherent similarity between Bayesian neural networks sampled using HMC and traditional feed-forward neural networks that are trained using the well known back-propagation algorithm [Rumelhart et al., 1988], as both take steps in the steepest direction using gradient based information. Though HMC will in general explore the posterior distribution in a more efficient manner than RW-MH, the evaluation of the gradient can very expensive for large data sets. Recent work has shown that this drawback can be lessened through the use of parallel computing techniques [Beam et al., 2014a].

The BNN framework outlined here has several features designed to address many of the challenges inherent in analyzing large datasets from genetic association studies. These advantages are outlined below.

- Quantification of variable influence with uncertainty measures. This allows variable influence to be assessed relative to a null or background model using a novel Bayesian testing framework. This avoids reliance on a permutation testing strategy.

- Automatic modeling of arbitrarily complex genetic relationships. Interactions are accounted for without having to examine all possible combinations. This is achieved from the underlying neural network model.

- An efficient sampling algorithm. HMC scales much better than other MCMC methods, such as the RW-MH algorithm, in high-dimensions.

- Computational expediency through the use of GPUs. The time needed to build the model is greatly reduced using the massive parallel processing offered by GPUs.

We offer evidence for these claims using several simulated scenarios and a demonstration on a real GWAS dataset. In addition, we compare the proposed approach to several popular methods so that relative performance can be assessed.

3.2 Methods

Neural networks are a set of popular methods in machine learning that have enjoyed a flurry of renewed activity spurred on by advances in training so-called deep networks [Hinton et al., 2012, Bengio, 2009]. The term neural network can refer to a very large class of modeling techniques, but to be clear we use the term here to refer to multilayer feed-forward perceptions (MLPs). In the most basic sense, neural nets represent a class of non-parametric methods for regression and classification. They are non-parametric in the sense that they are capable of modeling any smooth function on a compact domain to an arbitrary degree of precision without the need to specify the exact relationship between input and output. This is often succinctly stated as neural nets are *universal function approximators* [Hornik et al., 1989]. This property makes them appealing for many tasks, including modeling the relationship between genotype and

phenotype, because a sufficiently complex network will be capable of automatically representing the underlying function.

Some draw backs of classical neural nets are natural consequences of their strengths. Due to their flexibility, neural nets are highly prone to over-fitting to data used to train them. Over-fitting occurs when the network starts to represent the training data exactly, including noise that may be present, which reduces its ability to generalize to new data. Many methods exist to address this issue, but popular methods such as weight decay are well known to be approximations to a fully Bayesian procedure [Neal, 1995, Williams, 1995]. Another issue with standard neural nets is they are often regarded as black boxes in that they do not provide much information beyond a predicted value for each input. Little intuition or knowledge can be gleaned as to which inputs are most important in determining the response, so nothing coherent can be said as to what drives the networks predictions. Discussions of the advantages and disadvantages of neural nets for gene mapping have been reviewed in [Motsinger-Reif and Ritchie, 2008]. First we describe the base neural network model, and then describe how this can be incorporated into a Bayesian framework.

The network is defined in terms of a directed, acyclic graph (DAG) where inputs are feed into a layer of hidden units. The output of the hidden units are then fed in turn to the output layer which transforms a linear combination of the hidden unit outputs into a probability of class membership. Specifically consider a network with p input variables, h hidden units, and 2 output units to be used to predict whether an observation belongs to class 1 or class 2. Let $x_i =< x_{i1}, , x_{ip} >^T$ be the input vector of p variables and $y_i =< y_{i1}, y_{i2} >$ be the response vector, where $y_{i1} = 1$ if observation i belongs to class 1 and 0 if not, with y_{i2} is defined in the same way for class 2. Hidden unit k first takes a linear combination of each input vector followed by a nonlinear transformation, using the following form:

$$h_k(x_i) = \phi\left(b_k + \sum_{j=1}^{p} w_{kj} * x_{ij}\right) \tag{3.1}$$

where $\phi(\cdot)$ is a nonlinear function. For the purposes of this study, consider the logistic transformation, given as:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Several other activation functions such as the hyperbolic tangent, linear rectifier, and the radial basis/Gaussian functions are often used in practice. Each output unit takes a linear combination of each h_k followed by another nonlinear transformation. Let $f_1(x_i)$ be the output unit that is associated with class 1:

$$f_1(x_i) = \psi \left(B_1 + \sum_{k=1}^{h} W_{k1} * h_k(x_i) \right) \tag{3.2}$$

Note we have used upper case letters to denote parameters in the output layer and lowercase letters to indicate parameters belonging to the hidden layer. The $\psi(\cdot)$ function is the softmax transformation of element z_1 from the vector $z =< z_1, \ldots, z_n >$:

$$\psi(z_1) = \frac{\exp(z_1)}{\sum_{i=1}^{n} \exp(z_i)}$$

In this representation, $f_1(x_i)$ represents the estimated conditional probability that y_i belongs to class 1, given the input vector x_i. A similar definition is made for output unit 2, $f_2(x_i)$. Note that for the case of only 2 classes, $f_2(x_i) = 1 - f_1(x_i)$ because the softmax transformation forces the outputs to sum to 1.

Having described the formulation for standard neural networks we next describe how this can be extended using the Bayesian formulation. Bayesian methods define a probability distribution over possible parameter values, and thus over possible neural networks. To simplify notation, let $\theta = \{B, W, \beta, w\}$ represent all of the network weights and biases shown in equations 3.1, 3.2. The posterior distribution for θ given the data x_i and y_i, is given according to Bayes rule:

$$p(\theta|x_i, y_i) = \frac{L(\theta|x_i, y_i) \cdot \pi(\theta)}{m(x)} \tag{3.3}$$

where $m(x) = \int L(\theta|x_i, y_i) \cdot \pi(\theta) d\theta$ is the marinal density of the data. $L(\theta|, x_i, y_i)$ is the *likelihood* of θ given then data and $\pi(\theta)$ is the prior distribution for the network parameters. However, in practice we only need to be able to evaluate the numerator of (4.5) up to a constant because we will be relying on MCMC sampling techniques that draw from the correct posterior without having to evaluate $m(x)$, which may be intractable in high dimensions.

Often it is better to work with the log-likelihood $l(\theta|x_i, y_i) = \log(L(\theta|x_i, y_i))$, because the raw likelihood can suffer from numerical overflow or underflow for large problems. In this study we assume the log-likelihood for a neural network with 2 output units is binomial:

$$l(\theta|x_i, y_i) = y_{i1} \cdot \log(f_1(x_i)) + y_{i2} \cdot \log(1 - f_1(x_i)) \tag{3.4}$$

Next every parameter in the model must be given a prior distribution. The prior distribution codifies beliefs about the values each parameter is likely to take before seeing the data. This type of formulation is extremely useful in high-dimensional settings such as genomics, because it enables statements such as 'most of the variables are likely to be unrelated to this response' to be quantified and incorporated into the prior. In this study, we adopt a specific prior structure known as the *Automatic Relevance Determination* (ARD) prior. This prior was originally introduced in some of the foundational work on Bayesian neural nets [Neal, 1995, Neal, 1998] and later used in SVMs for cancer classification [Li et al., 2002].

The ARD prior groups network weights in the hidden layer together in a meaningful and interpretable way. All of the weights in the hidden layer that are associated with the same input variable are considered part of the same group. Each weight in a group is given a normal prior distribution with mean zero and a common variance parameter. This shared group-level variance parameter controls how large the weights in a group are allowed to become and performs shrinkage by pooling information from several hidden units, which helps to prevent

overfitting [Neal, 1998]. Each of the group-level variance parameters is itself given a prior distribution, typically an Inverse-Gamma distribution with some shape parameter α_0 and some scale parameter β_0. These parameters often referred to as hyper parameters, and can themselves be subject to another level of prior distributions, but for the purposes of this study, we will leave them fixed as user specified values. Specifically, for a network with h hidden units, the weights in the hidden layer for input variable j will have the following prior specification:

$$w_{j1}, \ldots, w_{jp} \sim N(0, \sigma_j^2)$$

$$\sigma_j^2 \sim IG(\alpha_0, \beta_0)$$

This structure allows the network to automatically determine which of the inputs are most relevant. If variable j is not very useful in determining whether an observation is a case or a control, then the posterior distribution for σ_j^2 will be concentrated around small values. Likewise, if variable j is useful in determining the response status, most of the posterior mass for σ_j^2 will be centered on larger values.

3.2.1 Hamiltonian Monte Carlo (HMC) for Neural Networks

Here we briefly give an overview of Hamiltonian Monte Carlo (HMC) for neural networks, but please see [Neal, 1995, Neal, 1992, Neal, 2011] for a thorough treatment. HMC is one of many Markov Chain Monte Carlo (MCMC) methods used to draw samples from probability distributions that may not have analytic closed forms. HMC is well suited for high-dimensional models, such as neural nets, because it uses information about the gradient of the log-posterior to guide the sampler to regions of high posterior probability. For neural networks, we adopt the two-phase sampling scheme of Neal used in [Neal, 1995]. In the first phase, we update the values of the variance parameters using a Gibbs update, conditional on the current values of the networks weights. In the next phase, we leave the variance parameters fixed and update the network weights using HMC. We repeat this procedure of Gibbs-coupled HMC updates until

we have acquired the desired number of posterior samples.

The higher level variance parameters, including those in the ARD prior, have simple closed forms because the Inverse-Gamma distribution is a conditionally conjugate prior distribution for the variance parameter of a Normal distribution. To obtain a new value for a variance parameter, conditional on the values of the weights a parameter controls, one makes a draw from the following Inverse-Gamma distribution:

$$\sigma^2_{new} \sim IG\left(\alpha_0 + \frac{n_w}{2}, \beta_0 + \sum_{i=1}^{n_w} \frac{w_i^2}{2}\right) \tag{3.5}$$

where each w_i is a weight controlled by this variance parameter, n_w is the number of weights in the group, and α_0, β_0 are the shape and scale parameters respectively of the prior distribution.

HMC then proceeds by performing L number of leap-frog updates for the weights, given the values of the variance parameters. The algorithm introduces a fictious momentum variable for every parameter in the network that will be updated by simulating Hamiltonian dynamics on the surface of the log-posterior. Since HMC was orginally created in statistical physics it is often presented in terms of energy potentials which is equivalent to an exponentiation of the negative log-posterior, but we will describe the algorithm directly in terms of the log-posterior, which is more natural for our purposes. The full algorithm for sampling the posterior of all parameters (network weights and variance parameters) is shown below.

A few details in the HMC algorithm as shown need further explanation. First the momentum variables (m) are refreshed after every sequence of L leap-frog updates, shown in the algorithm as `InializeMomentum(`α`)`. In the most simple formulation each momentum component is a independent draw from a Normal distribution, with mean 0 and standard deviation of 1. However, this can lead to wasted computation because the sampler may start out in bad direction by chance, requiring more leap-frop updates until the sampler is heading in a useful direction resulting in random-walk like behavior. To combat this, we use the persistent momentum refreshes [Nabney, 2002, Neal, 1995] which initializes the momentum using a

HMC Algorithm for Neural Networks

Input: **X**: matrix of predictors **Y**: matrix of class membership
P: log-posterior density to be sampled
{**ε**, **L**, $\mathbf{N_s}$, **α**}: HMC Parameters
Output: $\mathbf{N_s}$ Posterior Samples of Model Parameters

BEGIN ALGORITHM:
θ_0 = InitializeNetworkParameters()
σ_0^2 = InitializeVarianceParameters()
$m_{prev} \sim N(0,1)$
FOR i in 1 to N_s DO:
σ_i^2 = GibbsUpdate(θ_{i-1})
$\gamma_0 = \theta_{i-1}$
m_0 = InitializeMomentum(α, m_{prev})
FOR t in 1 to L DO:

$$m_* = m_{t-1} + \frac{\epsilon}{2}\nabla_\gamma P\left(\gamma_{t-1}\middle|\mathrm{X},\mathrm{Y};\sigma_i^2\right)$$

$$\gamma_t = \gamma_{t-1} + \epsilon * m_*$$

$$m_t = m_* + \frac{\epsilon}{2}\nabla_\gamma P\left(\gamma_t\middle|X,Y;\sigma_i^2\right)$$

END
$m_{prev} = m_t$
IF Accept(γ_t): $\theta_i = \gamma_t$
ELSE: $\theta_i = \theta_{i-1}$
END

Figure 3.1: Algorithm for HMC-based posterior sampling for the neural network model.

weighted combination of the final momentum value of the previous leap-frog update and a draw of standard normal random variable. Using the notation of the algorithm, this is shown below:

$$m_0 = \alpha * m_{prev} + \sqrt{1-\alpha^2} * \zeta$$

where $\zeta \sim N(0,1)$. If the proposal is rejected the momentum must be negated. This must be done to ensure that the canonical distribution is left intact [Nabney, 2002]. This formulation

reduces the number of leap-frog updates (L) needed to reach a distant point by suppressing random-walk behavior while leaving the correct target distribution of the Markov chain intact. [Nabney, 2002, Neal, 1995, Neal, 2011]. The `Accept`(γ_t) function returns true if the new proposal, γ_t, is accepted according to a modified Metropolis-Hastings acceptance probability. `Accept`(γ_t) returns true with the following probability:

$$\bar{\alpha} = \min\left(1, \frac{P(\gamma_t|X, Y; \sigma_i^2) - \frac{1}{2}m_t^T m_t}{P(\gamma_{t-1}|X, Y; \sigma_i^2) - \frac{1}{2}m_{t-1}^T m_{t-1}}\right)$$

However, the posterior distribution is often very 'bumpy' with many posterior modes [Neal, 1995]. This property may be exacerbated in high dimensions, so becoming stuck in one mode for extended periods of time is a great concern. To alleviate this we modify the acceptance probability, $\bar{\alpha}$, to correspond to a *flattened* version of the posterior whose acceptance probability is given as $\alpha^* = \bar{\alpha} \cdot T$, for $T > 1$, which is equivalent to sampling from $P(\theta|X, Y; \sigma_t^2)^{\frac{1}{T}}$. While its true that we are no longer sampling from the exact posterior $P(\theta|X, Y; \sigma_t^2)$, under mild regularity conditions the posterior modes of the correct distribution remain intact [Andrieu et al., 2003]. Since none of the parameters have biological interpretations modifying the posterior in this way of little concern, if the full procedure is capable of maintaining a favorable discriminatory ability. We find the trade-off between ease of sampling across a wide-range possible scenarios and exactness of the posterior to be acceptable.

3.2.2 HMC Using Graphics Processing Units (GPUs)

Previous work has shown how the gradient and log-posterior evaluations needed by HMC can be sped-up by as much as 150x for large problems using Graphics Processing Units (GPUs) [Beam et al., 2014a]. We adopt that framework here and express the gradient calculations as matrix-vector operations or element-wise operations. Similarly, evaluation of the log-posterior can be expressed in terms of linear operators and element-wise operations. Using GPUs for these operations is well known in the neural network literature [Bergstra et al., 2010, Lopes and

Ribeiro, 2009, Oh and Jung, 2004] as the gradient of the log-posterior corresponds roughly to the well-known back-propagation algorithm and evaluation of the log-posterior corresponds to the feed-forward operation in standard neural networks. However, to our knowledge this study represents the first GPU-enabled implementation of Bayesian neural networks. Without GPU computing, it is likely that the computational burden imposed by large datasets would be too great for the Bayesian neural network framework to be feasible.

All of methods discussed in this study are implemented in the Python programming language. All GPU operations were conducted using the Nvidia CUDA-GPU [Nvidia, 2008] programming environment and accessed from Python using the PyCuda library [Klockner et al., 2012]. Source code containing the Bayesian neural network package is available at `https://github.com/beamandrew/BNN`.

3.2.3 Bayesian Test of Significance for ARD Parameters

Given the ARD prior a natural question to ask is how large do values of σ_j^2 need to be for input j to be considered relevant compared to a variable that is completely unrelated. This question can be framed in terms of a Bayesian hypothesis test. In this framework we will assume that under the null hypothesis a variable is completely irrelevant in determining the status of the response. If this were a simple linear model, this would be equivalent to saying the regression coefficient for this variable has a posterior mean of zero. In the neural network model the ARD parameters that determine how relevant each input is are strictly positive, so we need a baseline or null model for the ARD parameters in order to determine if we can reject this null hypothesis of irrelevance. In order to construct and test this hypothesis, we make a simplifying assumption that weights for unrelated variables have a normal distribution with mean 0 and variance σ_{null}^2 i.e. $w_{kj} \sim N(0, \sigma_{null}^2)$. Due to the complex statistical model, the true posterior distributions for the weights under the null may not be exactly normal, but this approximation will be useful in simplifying the calculations. Additionally since, the prior for each weight is normal, this approximation will most likely not be too far from true posterior form under the

stated null hypothesis.

Since σ^2_{null} represents the null ARD parameter associated with a variable of no effect, we wish to test whether a variable of interest is significantly greater than this null value. We use the phrases null and significance here because of their familiar statistical connotations, but they should not be confused with the p-value based frequentist hypothesis testing procedure, as we are operating within a fully Bayesian framework. Our goal becomes testing whether the mean, μ_j of the posterior distribution for the ARD parameter, σ^2_j, is greater than the mean of the null, μ_{null}, for the null ARD parameter σ^2_{null}. Specifically we wish to test the following null hypothesis:

$$H_0 : \mu_{null} = \mu_j$$

against the one-sided alternative:

$$H_a : \mu_{null} < \mu_j$$

To test this, we need to know the closed form of μ_{null}. Making use of the iterative two-stage sampling scheme, we will derive this form by induction. We will also make use of several well-known facts of random variables. Firstly, if a random variable X has an inverse-gamma distribution, i.e. $X \sim IG(\alpha, \beta)$, then the mean or expected value of X, $E[X]$, is given by $\frac{\beta}{\alpha - 1}$. Next, if the sequence of random variables $X_1 \dots X_n$ are each independently and identically distributed as $N(0, \sigma^2)$, then $\sum_{i=1}^{n} \left(\frac{X_i}{\sigma}\right)^2 =< \frac{X_1}{\sigma}, \dots, \frac{X_n}{\sigma} >^T < \frac{X_1}{\sigma}, \dots, \frac{X_n}{\sigma} > \sim \chi^2_n$, i.e. a chi-squared random variable with n degrees of freedom. This sum has an expected value of n, from the definition of a chi-squared random variable. This implies the conditional expected value $E[< X_1, , X_n >^T < X_1, , X_n > |\sigma] = \sigma^2 * n$. Using these basic facts we will show that under the null, the two-stage sampling scheme leaves expected value of the ARD parameter invariant, i.e. $\mu_{null} = \mu_{prior}$.

For a network with h hidden units, let $w_j =< w_{1j}, \ldots, w_{hj} >$ be a vector containing all of the weights associated with input j, where each component of w_j is initially distributed according to the prior, $N(0, \sigma_j^2)$ and $\sigma_j^2 \sim IG(\alpha_0, \beta_0)$. The mean, for σ_j at the start of the simulation is $\mu_0 = \frac{\beta_0}{\alpha_0 - 1}$. We begin the simulation at iteration i=1 and perform a Gibbs update of σ_j^2. The Gibbs update for the shape parameter, $\alpha_1 = \alpha_0 + n_w/2$, is iteration independent and will remain fixed for the entirety of the simulation. However, the Gibbs update for the scale parameter, $\beta_1 = \beta_0 + (w_j^T w_j)/2$, depends upon the current values of the weights, and thus will take on a random value at each iteration. However, we can compute the expected value for β_1 as:

$$\begin{aligned} E[\beta_1] &= E\left[\beta_0 + \frac{w_j^T w_j}{2}\right] \\ &= \beta_0 + \frac{1}{2} E[w_j^T w_j] \\ &= \beta_0 + \frac{1}{2}(h \cdot E[\sigma_0^2]) \\ &= \beta_0 + \frac{h}{2} \cdot \frac{\beta_0}{\alpha_0 - 1} \\ &= \beta_0 + \frac{h}{2} \cdot \mu_0 \end{aligned}$$

Thus, the expected value of β_1 after the first Gibbs update is $\beta_0 + \frac{h}{2} \cdot \mu_0$. Note that this expectation is independent of simulation iteration, so this result will hold for all $\beta_1, \beta_2, \cdots, \beta$.

Next, we use this fact to compute the expected value of the ARD parameter, $E[\sigma_1^2]$:

$$
\begin{aligned}
E[\sigma_1^2] &= E\left[\frac{\beta_1}{\alpha_0 + h/2 - 1}\right] \\
&= \frac{E[\beta_1]}{\alpha_0 + h/2 - 1} \\
&= \frac{\beta_0 + h/2 \cdot \mu_0}{\alpha_0 + h/2 - 1} \\
&= \frac{\beta_0}{\alpha_0 - 1 + h/2} + \frac{h/2 \cdot \mu_0}{\alpha_0 - 1 + h/2} \\
&= \frac{\frac{1}{\alpha_0 - 1}}{\frac{1}{\alpha_0 - 1}} \cdot \frac{\beta_0}{\alpha_0 - 1 + h/2} + \frac{h/2 \cdot \mu_0}{\alpha_0 - 1 + h/2} \\
&= \frac{\mu_0}{1 + h/2 \cdot \frac{1}{\alpha_0 - 1}} + \frac{h/2 \cdot \mu_0}{\alpha_0 - 1 + h/2} \\
&= \mu_0 \left(\frac{1}{1 + h/2 \cdot \frac{1}{\alpha_0 - 1}} + \frac{h/2}{\alpha_0 - 1 + h/2}\right) \\
&= \mu_0 \left(\frac{\alpha_0 - 1}{\alpha_0 - 1 + h/2} + \frac{h/2}{\alpha_0 - 1 + h/2}\right) \\
&= \mu_0 \left(\frac{\alpha_0 - 1 + h/2}{\alpha_0 - 1 + h/2}\right) \\
&= \mu_0
\end{aligned}
$$

Thus, the Gibbs update of the ARD parameter does not change the expected value under the null, since we defined $E[\sigma_0^2] = \mu_0$. This establishes the base case, and now we show the induction step. Given $E[\beta_{t+1}] = E[\beta_t] = \beta_0 + \frac{h}{2} \cdot \mu_0$ and $E[\sigma_t] = \mu_0$ then:

$$
\begin{aligned}
E[\sigma_{t+1}] &= E\left[\frac{\beta_{t+1}}{\alpha_0 + h/2 - 1}\right] \\
&= \frac{E[\beta_{t+1}]}{\alpha_0 + h/2 - 1} \\
&= \frac{\beta_0 + h/2 \cdot \mu_0}{\alpha_0 + h/2 - 1} \\
&= \mu_0
\end{aligned}
$$

where the simplification between lines 3 and 4 proceeds as before. This concludes the proof.

3.3 Results

3.3.1 Existing Methods Used for Comparison

We selected several methods to serve as baselines for evaluation of the BNNs performance. As previously mentioned BEAM and MDR are widely used methods and so were included in our evaluation. We used a custom compiled 64-bit version of BEAM using the source provided on the website [Zhang, 2014] of the authors of [Zhang and Liu, 2007]. The java-based MDR package was downloaded from the MDR source-forge repository (http://sourceforge.net/projects/mdr/) and called from within a Python script. To evaluate the effectiveness of tree-based methods, we used an approach nearly identical to that in [Li et al., 2011], which was based on boosted decision trees. The boosted decision tree model provides measures of relative influence for each variable that indicate how important a given variable is, relative to the others in the model. To fit the boosted tree model we used the `gbm` package in R. Finally, we also included the standard 2 degrees-of-freedom chi-square test of marginal effects.

As discussed, some approaches such as MDR and GBM require a permutation testing strategy to assess statistical significance. This makes assessing their performance on large datasets difficult, due to the amount time required to perform the permutation test. During our pilot investigations on a dataset containing 1,000 SNPs, each individual run of MDR was found to

take roughly 1 minute to complete. The time needed to complete the required 20,000 permutations would be roughly 2 weeks. If we wish to evaluate a methods effectiveness on hundreds or thousands of such datasets, this run time becomes prohibitive. As such, we divided our primary analysis into two sections. In the first section, we evaluated methods that *do not* rely on permutation testing on datasets containing 1,000 SNPs each. However, since we wish to compare the results of the BNN to that of MDR and GBM, we performed a second set of analyses on smaller datasets that only contained 50 SNPs each, for which permutation testing is feasible. This two-pronged strategy allowed us to evaluate a wide range of popular approaches in a reasonable amount of time, while serving to underscore the need for methods that do not rely on permutation testing.

3.3.2 Parametric Models of Multi-Locus Relationships

In this section we performed an analysis of three biallelic models of genotypic relationships. These models have been used previously [Zhang and Liu, 2007, Li et al., 2011] and are meant to reflect theoretical and empirical evidence for genetic relationships involving multiple loci [Li and Reich, 2000]. Tables 3.1, 3.2, and 3.3 contain the risk relative of disease for each genotype combination, where a capital and lower case letters represent the major and minor alleles, respectively.

Table 3.1: Additive Risk Model

Genotype	AA	Aa	aa
BB	η	$\eta(1+\theta)$	$\eta(1+2\theta)$
Bb	$\eta(1+\theta)$	$\eta(1+2\theta)$	$\eta(1+3\theta)$
bb	$\eta(1+2\theta)$	$\eta(1+3\theta)$	$\eta(1+4\theta)$

Table 3.2: Threshold Risk Model

Genotype	AA	Aa	aa
BB	η	η	η
Bb	η	$\eta(1+\theta)$	$\eta(1+\theta)$
bb	η	$\eta(1+\theta)$	$\eta(1+\theta)$

Table 3.3: Epistatic Risk Model

Genotype	AA	Aa	aa
BB	η	η	$\eta(1+4\theta)$
Bb	η	$\eta(1+2\theta)$	η
bb	$\eta(1+4\theta)$	η	η

The symbols η and θ in the tables represent the baseline risk and effect size, respectively. We simulated genotypes for the disease SNPs for a range of minor allele frequencies (MAFs) and simulated the disease status for 1,000 cases and 1,000 controls using the risks given in Tables 3.1, 3.2, and 3.3. We embedded the causal SNPs in a background of 998 non-causal SNPs, for a total of 1,000 SNPs to be considered. For each combination of effect size, $\theta \in \{0.5, 1.0, 1.5, 2.0\}$, MAF$\in \{0.1, 0.2, 0.3, 0.4, 0.5\}$, and model type (Additive, Threshold and Epistasis) we generated 100 datasets. This yielded a total of 6,000 datasets for evaluation. All datasets in this section were created using the R statistical programming language [Team et al., 2005].

We ran BNN, BEAM, and the χ^2 test on each dataset and recorded whether or not both disease SNPs were declared as significant by each method. We took the fraction of datasets where both disease SNPs were correctly identified as an estimate of statistical power. For BEAM and the χ^2 test, we used the canonical Bonferroni corrected significance threshold of

$p < 0.05$. We used the recommended parameter settings for BEAM [Zhang and Liu, 2007] and performed $1 * 10^6$ sampling iterations for each dataset. For the BNN approach, we used a network with 1 hidden layer and 5 logistic units and a softmax output layer with 2 units. The network parameters in the hidden layer are given ARD priors, while the network parameters in the output are given a common Gaussian prior. The hyper parameters for the Inverse-Gamma prior for the ARD parameters were $\alpha_0 = 5$,$\beta_0 = 2$ while the hyper parameters for the Gaussian priors were $\alpha_0 = 0.1$,$\beta_0 = 0.1$. The parameters for the HMC algorithm were $\epsilon = 5 * 10^{-2}$, $L = 15$, $\alpha = 0.75$, and $T = 5 * 10^3$. The cutoff value for the novel Bayesian ARD testing framework was 0.6. We discarded the first 25 samples as burn-in and kept 100 samples to be used for inference. Processing of each dataset by the BNN took approximately 3 minutes. The results are shown below in Figures 3.2, 3.3 and 3.4.

BNNs were found to be uniformly more powerful than both BEAM and the χ^2 test for the additive model. BNNs show excellent power, even for small effect sizes and achieve 100% power for second smallest effect size across all tested MAFs. In contrast, BEAM showed relatively little power for the smallest effect size and never achieves 100% for all MAFs, even at the highest level of effect size. The threshold model tells a similar story. For all but 3 combinations of MAF and effect size, the BNN model is again uniformly more powerful than both BEAM and the 2 test. The picture from the epistatic model is slightly more mixed. BEAM appeared to do a better job at the smallest effect size, while performing equally well as BNNs on the remaining three effect size levels. All three methods had almost no power to detect the causal SNPs for a MAF of 0.5. These results suggest that BNN is uniformly more powerful the χ^2 test for these genetic models, and may be more powerful than BEAM in most instances.

3.3.3 Simulated Epistatic Relationships without Marginal Effects

In this section, we evaluated the performance of all the methods examined in the previous section (BNN, BEAM, and the χ^2) as well as GBM and MDR. Since MDR and GBM rely on permutation testing, we reduced the size of the dataset to accommodate this strategy. To generate

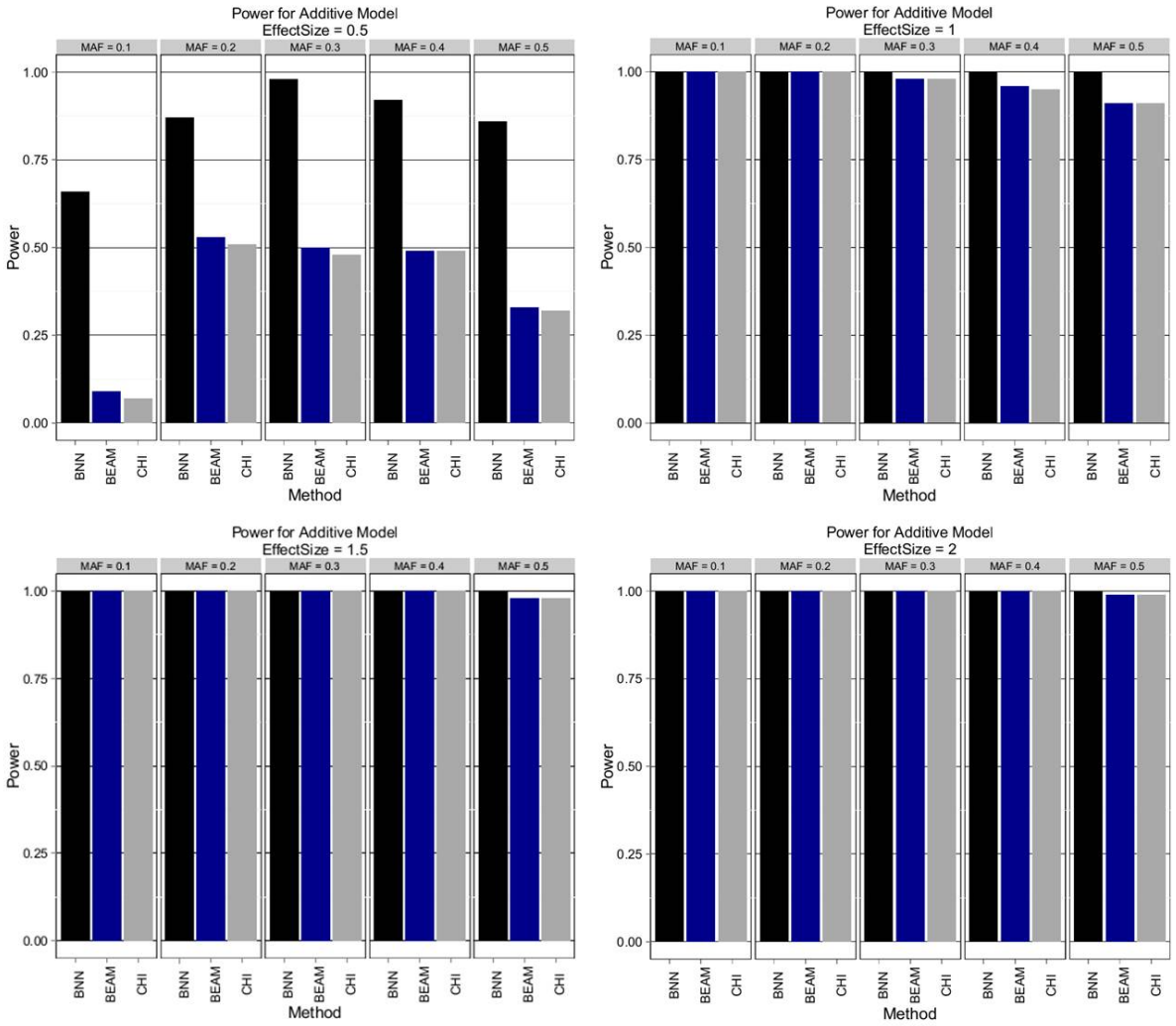

Figure 3.2: Additive Model. Estimated power to detect both disease SNPs using Bayesian neural networks (BNN), BEAM, and χ^2 test (CHI) with 2 d.f. Effect sizes of $\{0.5, 1.0, 1.5, 2.0\}$ are shown in order from left to right, top to bottom. Within each pane results are stratified by minor allele frequency (MAF).

test datasets, we used the GAMETES software package [Urbanowicz et al., 2012]. This package allows users to specify the proportion of variance for case/control status that is due to genetic variants (i.e. broad-sense heritability) as well as how many loci are involved in determining trait status. These relationships are generated such that there are minimal marginal effects, resulting in relationships that are nearly purely epistatic. Relationships without marginal effects are in

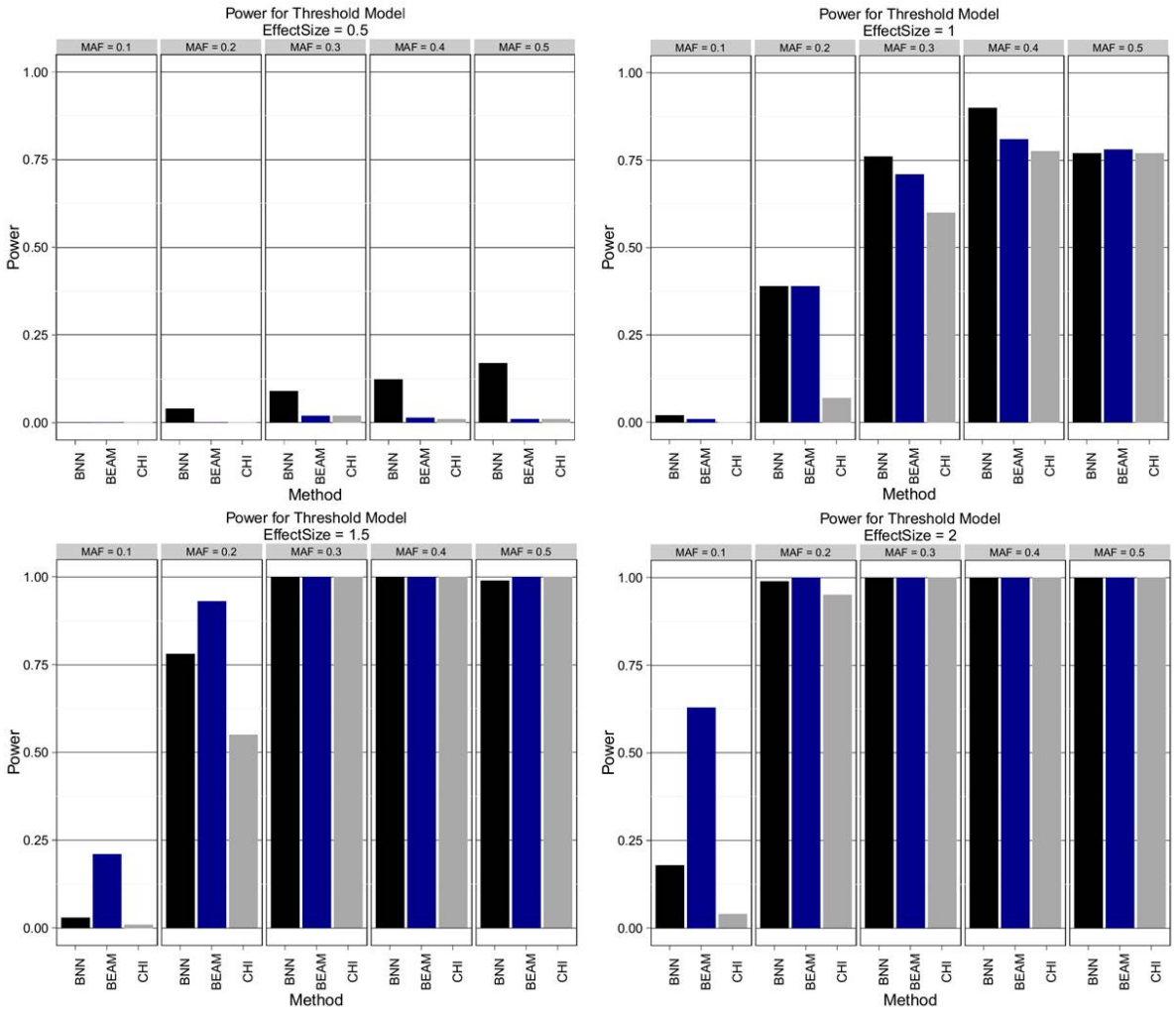

Figure 3.3: Threshold Model. Estimated power to detect both disease SNPs using Bayesian neural networks (BNN), BEAM, and χ^2 test (CHI) with 2 d.f. Effect sizes of $\{0.5, 1.0, 1.5, 2.0\}$ are shown in order from left to right, top to bottom. Within each pane results are stratified by minor allele frequency (MAF).

some sense 'harder' than those with marginal effects, because the causal SNPs contribute to trait status only through their interaction. Preliminary analysis on the reduced SNP datasets indicated that if the same models were used as in the previous section, most methods would have nearly 100% power for all simulated scenarios, which would provide little useful feedback for discerning which approaches were working best. This was the primary motivation for using

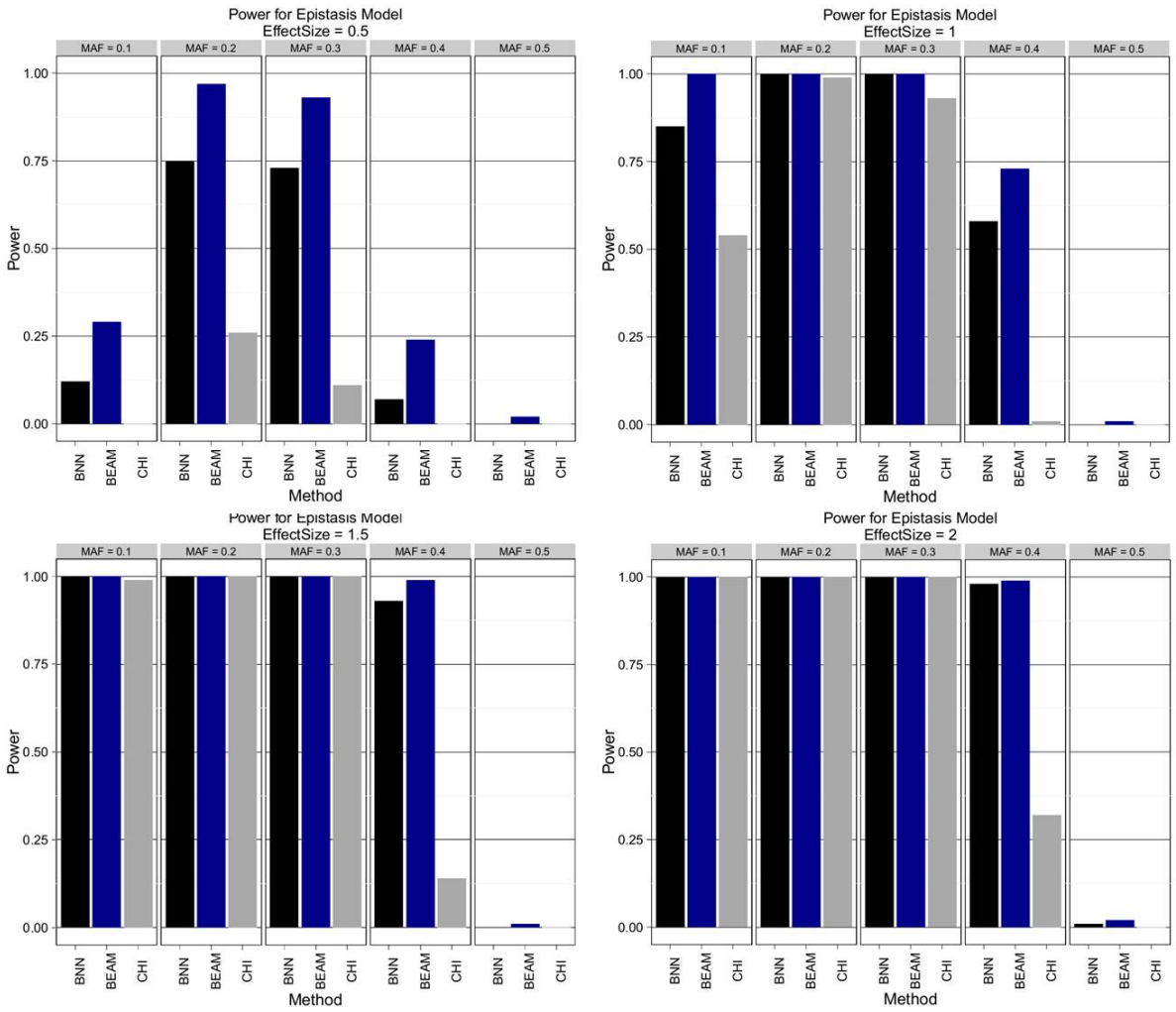

Figure 3.4: Epistatic Model. Estimated power to detect both disease SNPs using Bayesian neural networks (BNN), BEAM, and χ^2 test (CHI) with 2 d.f. Effect sizes of $\{0.5, 1.0, 1.5, 2.0\}$ are shown in order from left to right, top to bottom. Within each pane results are stratified by minor allele frequency (MAF).

the 'harder', purely epistatic relationships instead of the parametric models we used previously.

Using GAMETES, we analyzed two levels of heritability (5% and 10%) across a range of MAFs (0.05, 0.1, 0.2, 0.3, 0.4, 0.5). Power was measured as in the previous section using 100 instances for each heritability/MAF combination for a total of 1200 data sets used in evaluation. The results are shown below in Figures 3.5 and 3.6.

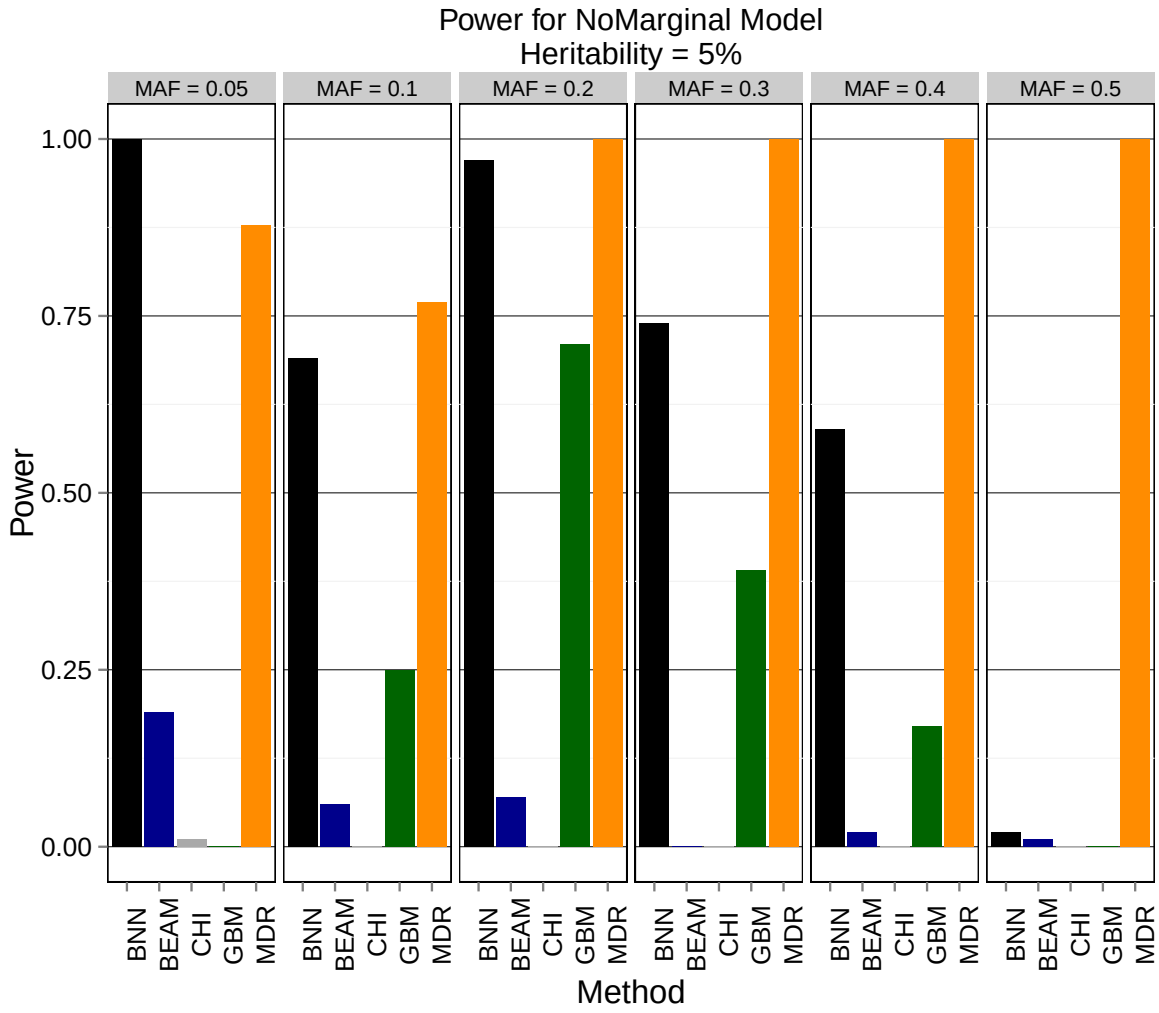

Figure 3.5: Purely Epistatic Model with 5% heritability. Estimated power to detect both disease SNPs of Bayesian neural networks (BNN), BEAM, 2 test (CHI) with 2 d.f., gradient boosted trees (GBM), and MDR. The results are stratified by minor allele frequency (MAF).

BNN outperform all methods from the previous section (BEAM and χ^2 test) by a very wide margin. This suggests that BEAM may be less robust to detect causal SNPs in the absence of marginal effects than previously thought, as it never achieves 25% power in any of the scenarios tested. Again, we find these results encouraging as they indicate that BNNs are indeed powerful relative to existing approaches. Additionally, BNN outperformed the GBM method in all but

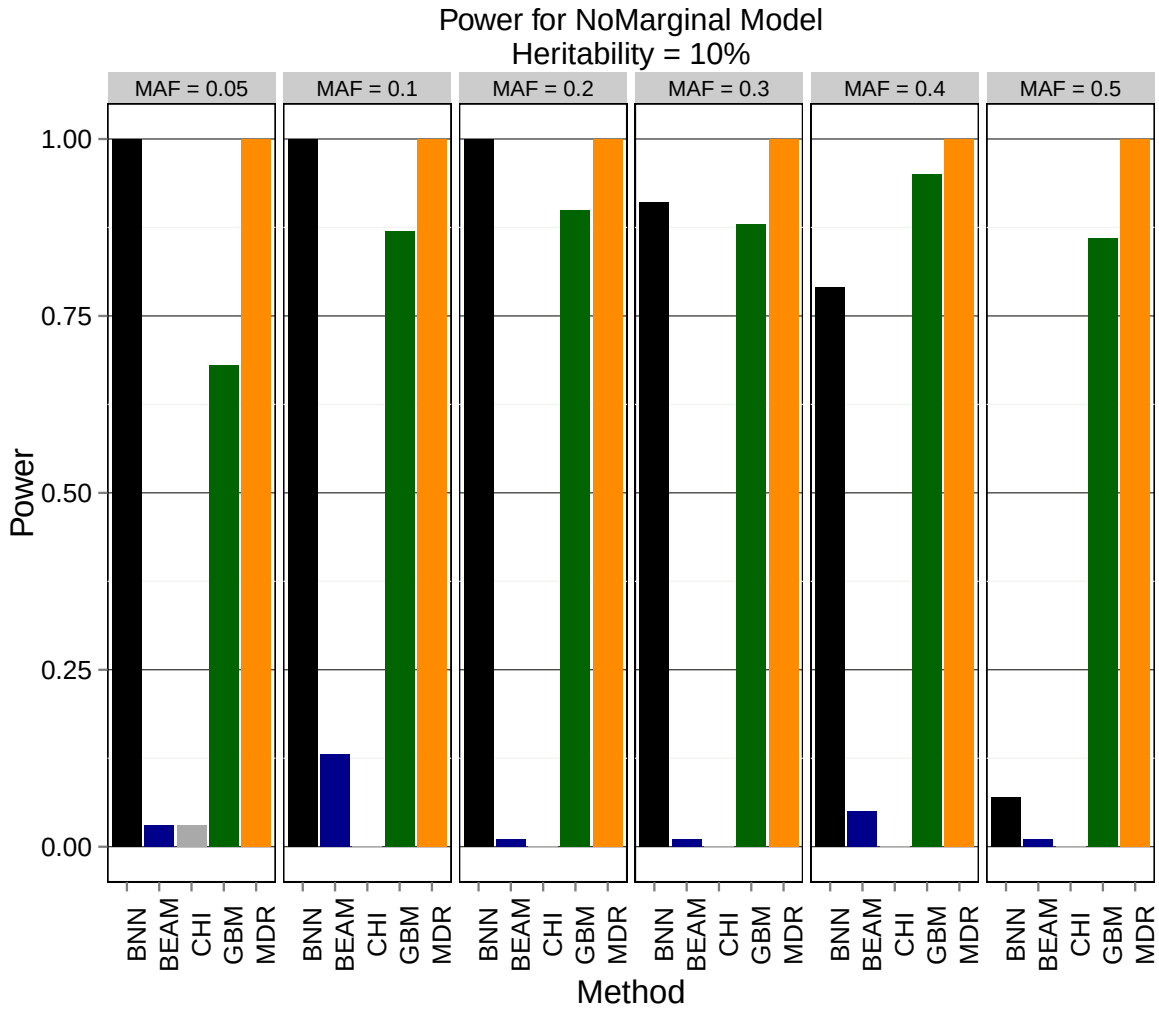

Figure 3.6: Purely Epistatic Model with 10% heritability. Estimated power to detect both disease SNPs of Bayesian neural networks (BNN), BEAM, 2 test (CHI) with 2 d.f., gradient boosted trees (GBM), and MDR. The results are stratified by minor allele frequency (MAF).

2 scenarios, indicating that BNN maybe be more adept at detecting purely epistatic signals across a broad array of MAFs and effect sizes. MDR performs well across every parameter combination tested, but as mentioned previously it is incapable of performing this analysis on a GWAS scale due to the exhaustive search technique and the need to perform permutation testing to assess statistical significance. To conclude this section, we note that BNN was the

only method that did well across a variety of genetic models, number of SNPs, and MAFs while being capable of scaling to GWAS-sized data. This provides evidence that BNN framework is deserving of further investigation as an analysis technique for association studies.

3.3.4 Sensitivity and Specificity Analysis of the ARD Test

The cutoff value used for the ARD test has an obvious impact on the methods performance. In the extreme case, a cutoff of 0 would result in nothing being significant while a cutoff value of 1 would result in everything being declared as such. The cutoff value controls the tradeoff between sensitivity (i.e. the true positive rate) and specificity (i.e. the true negative rate, which is equivalent to 1 the false positive rate). Evaluation of the false positive rate for the cutoff value of 0.6 used in the previous experiments indicates that the BNN method properly controls the amount of false positives. We observed an average false positive rate (FPR) of roughly 0.005 and 0.06 for the parametric models and the purely epistatic models, respectively as shown in Figure 3.7.

To examine the trade off between the true positive rate (TPR) and FPR as the cutoff value is changed, we modulated the cutoff from 0 to 1 in increments of 0.01 and recorded the true positive and false positive rate for each data set in the two previous sections. In Figure 3.8, we averaged the TPR and FPR over effect size and MAF to produce a receiver-operator characteristic (ROC) curve for each of the 4 genetic models. The legend displays the area under the curve (AUC) for each model.

These results show that BNN-ARD test for variable importance is able to achieve a high true positive rate, while maintaining a low false positive rate, which is an indication the method is performing as well and as expected.

3.3.5 Analysis of Tuberculosis Data

To evaluate the performance of Bayesian neural networks on a real dataset, we analyzed a GWAS designed to find genetic markers associated with tuberculosis (TB) disease progression.

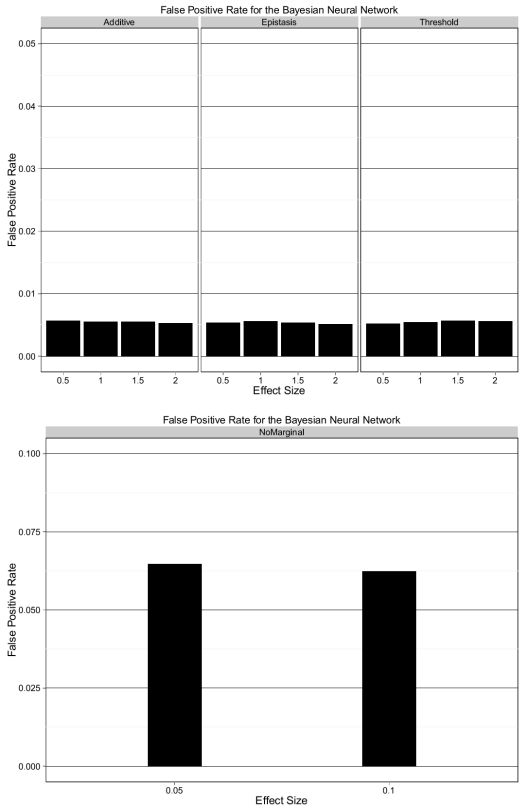

Figure 3.7: False Positive Rates (FPR) for each model/effect size combination, averaged over MAF.

The dataset describe in [Oki et al., 2011], contains information on roughly 60,000 SNPS and 105 subjects. For our study, each subject was classified as currently infected with any form of tuberculosis (i.e. extrapulmonary or pulmonary) or having a latent form of TB confirmed through a positive tuberculin skin test (purified protein derivative positive). Quality control was performed and SNPs with missing values were excluded, as were SNPs that were found to be out of Hardy-Weinberg equilibrium at the 0.05 level. After QC, there were 16,925 SNPs available for analysis and 104 subjects. Based on evidence of subpopulations in this data [Oki et al., 2011], subjects were assigned to one of three clusters created using the top two principal components

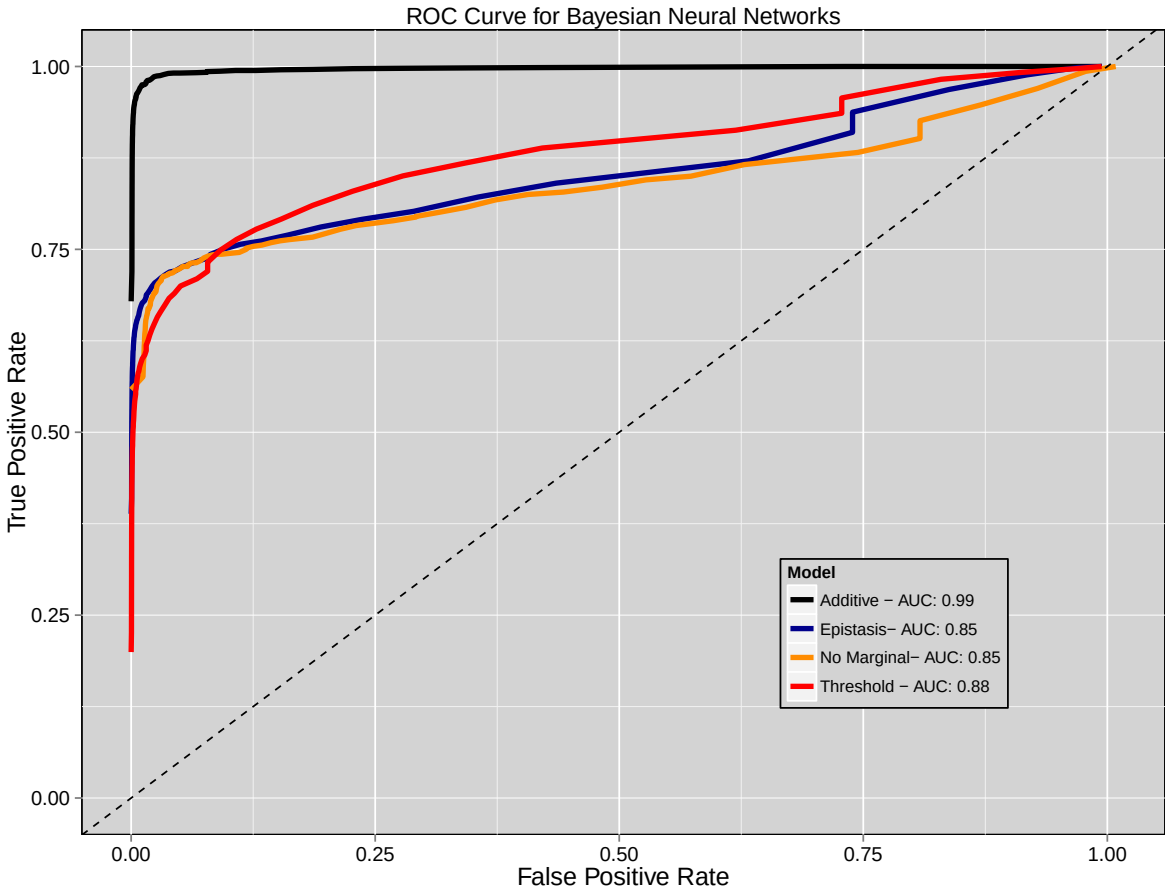

Figure 3.8: Receiver-Operator Characteristic (ROC) curve for BNNs. Each line represents the ROC curve for a different genetic model, averaged over effect size and MAF. The area under the curve (AUC) for each model is shown in the legend.

and cluster membership was included as a covariate in the model. Sampling of the Bayesian neural network was conducted as outlined in the previous section, with ARD hyper-parameters of $\alpha_0 = 3$, $\beta_0 = 1$. We performed 100 burn-in iterations followed by 1,000 sampling iterations which took approximately 20 hours. The top five SNPs based on posterior ARD probabilities are shown below in Table 3.4. The SNP reported as the 2nd most significant in [Oki et al., 2011] (rs10490266) appeared in our analysis as the 31st most significant SNP. Only one of the SNPs in Table 3.4 is currently known to be located within a gene (rs1378124 - MATN2) according to dbSNP. Every SNP reported in Table 3.4 is located on a the same chromosome and within 10-50

Table 3.4: Top 5 SNPs based on posterior ARD probabilities. Note these probabilities are presented in terms of involvement (larger indicates a SNP is more likely to be involved).

SNP	CHR	$Pr(\mu_j > \mu_{null})$
rs966414	2	0.524
rs1378124	8	0.515
rs9327930	5	0.509
rs4721214	7	0.502
rs4721214	9	0.498

MB of loci previously reported as having a statistically significant association with pulmonary tuberculosis susceptibility [Png et al., 2012] in an Indonesian population. The loci reported in [Png et al., 2012] were unfortunately either not part of the original SNP library or removed during the QC process in this study. Due to the small sample size of this dataset, it is hard to say conclusively which of the SNPs reported here and in [Oki et al., 2011] are most likely to replicate in a larger study. However, we present this analysis to demonstrate that the BNN framework is capable of analyzing data sets containing a high number of SNPs in a relatively short amount time.

3.4 Conclusions

In this study we have proposed the use of Bayesian neural networks for association studies. This approach was shown to be powerful across a broad spectrum of different genetic architectures, effect sizes, and MAFs. Of the approaches that do not rely on permutation testing, BNN was uniformly more powerful than the standard 2 test and almost uniformly more powerful than the powerful than the popular BEAM method in the scenarios considered. BNN again showed a near uniformly better performance than the GBM method. MDR was very competitive with BNN in our evaluations, however MDR is incapable of scaling to larger datasets due to both

its exhaustive search technique and reliance on permutation testing. In conclusion, we have demonstrated that BNNs are a powerful technique for association studies while having the capability of scaling to large GWAS sized datasets.

Availability of Code

Source code implementing the GPU-based Bayesian neural network framework outlined in this paper is available at `https://github.com/beamandrew/BNN`.

Chapter 4

Bayesian Nonparamteric Methods for Cell-Based Dose-Response Data

4.1 Background

Understanding the genetic factors that underly differential drug response in a population continues to be a primary goal in pharmecogenomics and drug development [Ritchie, 2012]. Association studies based on the use of *in vitro* cell lines, such a lymphoblastoid cell lines (LCLs), are becoming an attractive alternative to traditional, human based clinical trials [Welsh et al., 2009, Wheeler and Dolan, 2012, Brown et al., 2014]. Cell lines offer increased sample sizes relative to traditional studies, resulting in higher statistical power to detect genetic variants that may be involved in driving drug response. However, statistical best practice for the unique type of data offered by these studies has yet to be conclusively established.

Recent work has shown that considering the full set dose-responses instead instead of summary statistics can greatly increase the power to detect causal genetic variants [Brown et al., 2012a, Brown et al., 2011, Brown et al., 2014, Brown et al., 2012c, Beam and Motsinger-Reif, 2013]. This approach is based on a statistical method known as *multivariate analysis of variance* or MANOVA, which is an extension to the well known analysis of variance (ANOVA)

framework. Recent studies using a MANOVA based framework have reveal genetic loci associated with differential responses in anti-cancer agents [Brown et al., 2014, Brown et al., 2012a]. However, little study to date has been done to investigate the potential effect of gene-gene interactions, or *epistasis*, which is thought to be a critical piece in the genetic architecture of many complex phenotypes [Moore, 2003, Moore, 2005, Carlborg and Haley, 2004]. Though epistasis is thought to be a common phenomenon it can be a source of confusion due to lack of agreed upon definition [Cordell, 2002]. Specifically in this study, we are interested in epistasis defined as deviation from an *additive* genetic model.

To investigate how genetic interactions may affect the power of MANOVA we performed a simulation study across several possible plausible models of SNP-driven dose-response involving multiple loci. In addition to investigating the MANOVA approach, we develop a novel Bayesian nonparametric model that is capable of automatically detecting genetic interactions in multi-response data, even in the presence of gene-gene interactions. We have previously developed a Bayesian neural network testing framework [Beam et al., 2014b] for case-control studies that is capable of identifying causal loci in the presence of genetic interactions in a computationally tractable way. In this study, we extend this framework to model dose-response data that contain multiple continuous responses with miminal distributional assumptions. Through comparisons of several possible genetic models, we hope to gain insight as to how deviation from an additive model may affect the statistical power of each method.

4.2 Methods

4.2.1 Multivariate Analysis of Variance (MANOVA)

First, we provide a brief review of the MANOVA framework for dose-response studies. In a dose-response study, a chemical is administered to a cell culture across several concentrations and a quantitative response (e.g mRNA expression or total ATP) is measured at each concentration. Until recently the prevailing methodology in dose-response association mapping has been to

fit a parametric logistic function, known as the *hill-slope* model, that relates the activty or response to concentration (c_i) using the following form:

$$f(c_i) = Max - \frac{Max - Min}{1 + (\frac{c_i}{IC_{50}})^{-w}} \tag{4.1}$$

where Max, Min are the upper and lower asymptotes, and w is the hill-slope. The IC_{50} parameter is the concentration at which the response is 50% of maximum and is usually given special importance because it a measure of chemical *potency*. The IC_{50} is then treated as quantitative trait and quantitative trait locus (QTL) mapping techniques attempt to identify loci that appear to be associated with this trait. In general, this approach results in an drastic loss of power relative to potential alternatives [Brown et al., 2012b, Beam and Motsinger-Reif, 2013]. Recent work has shown that multivariate analysis of variance (MANOVA) has high power across a wide variety of possible dose-response relationships, making it an attractive option for genome-wide association mapping.

Let $x_i =< x_{i1}, \ldots, x_{ip} >^T$ be a vector containing the genotypes of all p markers for individual i, where each $x_{ij} \in \{0, 1, 2\}$ contains the number of instances of the minor allele. Let $y_i =< y_{i1}, \ldots, y_{ik} >$ be the vector containing the responses for subject i at each of the k concentrations, where each $y_{ik} \in \mathbb{R}$. MANOVA models the expected value of y_{ik}, $E[y_{ik}]$ as a linear function of genotype status, x_i:

$$E[y_{ik}] = \beta_0 + \beta_k^T x_i \tag{4.2}$$

where β_k is the $p \times 1$ vector of regression coefficients associated with concentration k. The least-squares estimator for $B_{p\times k} =< \beta_1, \ldots, \beta_p >$ is given by $(X^T X)^{-1} X^T Y$. where $X_{n\times p} =< x_i, \ldots, x_n >^T$ and $Y_{n\times k} =< y_1, \ldots, y_n >$. For the case when $n < p$ or when X is not full rank, as is often the case for GWASs, smaller sub-models may be fit and analyzed. For example, if 1 million markers are genotyped for only $1,000$ individuals, each marker may be fit separately

along with any possible confounding covariates such as age or gender.

This model has several advantages including interpretability and a simple, closed-form solution that is amenable to fast calculation, enabling it's use on large datasets. MANOVA is also based on a well understood theoretical foundation that allows for null-hypothesis significance testing. However, the linearity assumption may be somewhat restrictive, and if a trait is influenced or determined by interactions between markers, this model may only partially capture the true relationship. One approach to lessen this restriction would be to explicitly include all interactions as terms in the model, and build the linear model on this expanded set of covariates. However, this quickly becomes infeasible for even 2^{nd} order interactions. In a study containing one million markers, there are approximately $5 * 10^9$ possible 2^{nd} order interactions.

A different approach to account for interactions would be to use a class of models that relax the linearity assumption and impose very little structural constraints on the relationship between x_i and y_{ik}. Neural networks are one such approach and have a rich history of success in the machine learning and genetic epidemiology communities [Motsinger-Reif and Ritchie, 2008, Motsinger-Reif et al., 2008a]. Neural nets come with several theoretical guarantees that make them appealing for modeling potentially nonlinear functions. Perhaps the most germane property is given by the *universal function approximation* theorem [Hornik et al., 1989]. This theorem ensures that a neural network is capable in principle of modeling generic functions between input and output, regardless of the function's complexity. Subsumed in the class of functions neural networks can represent is the linear model used by MANOVA. Thus if the true model is indeed linear, the relationship learned by the neural network would automatically collapse to represent the linear relationship between input and output. However if the true relationship is more complex, it will include any nonlinearities without having to specify them *a priori*. In addition to the flexibility offered by the base neural network model, a previously developed Bayesian testing framework allows for nonparametric testing of SNP significance. In the next section we present and develop a Bayesian neural network framework for dose-response studies.

4.2.2 Bayesian Neural Networks

Bayesian neural networks represent an extension to the familiar neural network framework. Recent work in [Beam et al., 2014b] has shown they can successfully model genetic interactions in case-control studies. Here we present an augmented design capable of modeling multi-response data, such as the kind observed in dose-response studies.

Neural networks construct a hierarchical representation where inputs are transformed into (potentially) nonlinear features that can undergo further nonlinear transformations. For our purposes we consider networks with only one such transformation. Networks with this architecture are often described as having one *hidden layer*, because the graphical model describing the network has one such transformation occurring at the same hierarchical level between input and output. Figure 4.1 shows a graphical depiction of a neural network model for dose-response data.

The first layer of Figure 4.1 shows the input SNPs represented as the number of minor alleles present at each of the p markers. The inputs are feed into several *hidden units*, where each hidden unit ($h_j(x_i)$) performs the following nonlinear *logistic* transformation:

$$h_j(x_i) = \frac{1}{1 + \exp(-(a_j + x_i^T w_j))} \tag{4.3}$$

where w_j is a $p \times 1$ vector of weights associated with hidden unit j, and a_j is the unit's bias. The outputs from the hidden units are then fed into each output unit ($f_k(x_i)$). Each output unit takes a linear combination of the outputs from the hidden units, using the following form:

$$f_k(x_i) = \alpha_k + \sum_{j=1}^{H} h_j(x_i) * \beta_{kj} \tag{4.4}$$

where α_k is the bias and β_{kj} is the weight given to the output from hidden unit j, for output unit k. Equations (4.3) and (4.4) form the base neural network model. To incorporate this into a Bayesian framework, we must define several additional components. Bayesian methods define

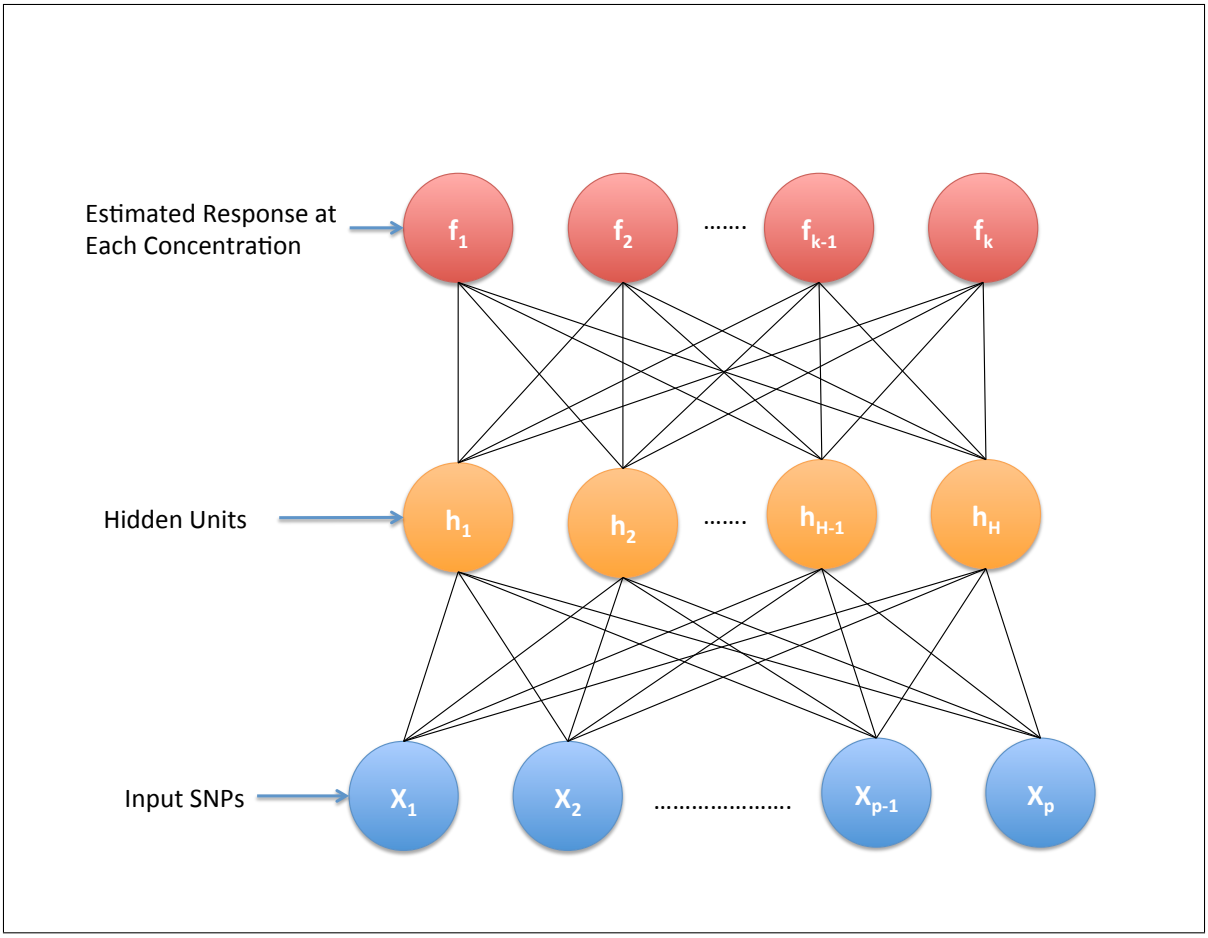

Figure 4.1: Graphical depiction of a neural network with several output units. The blue nodes on the bottom represent SNPs in MAF coding, the orange nodes represent the hidden unit functions in equation (4.3), and the red nodes at the top represent the estimated response for each of the concentrations measured. This architecture allows the network to model the response at each concentration as a nonlinear combination of the input SNPs.

a posterior distribution over possible parameter values that combines a prior distribution that is updated based on the data. Letting θ represent all of the network parameters in equations (4.3), (4.4), Bayes' rule defines the posterior distribution of θ, given the data x_i, y_i:

$$p(\theta|x_i, y_i) = \frac{L(\theta|x_i, y_i)p(\theta)}{\int L(\theta|x_i, y_i)p(\theta)d\theta} \tag{4.5}$$

where $p(\theta)$ is the prior distribution for the network parameters and $L(\theta|x_i, y_i)$ is the likelihood

function. The network's output for each observation ($f_{ik}(x_i)$) is given an independent Gaussian likelihood with mean $f_{ik}(x_i)$ and unit variance, shown below:

$$L(\theta|x_i, y_i) = \exp\left(-\frac{(y_{ik} - f_{ik}(x_i))^2}{2}\right) \tag{4.6}$$

The remaining piece of equation 4.5 is the specification of the prior distribution for the network parameters ($p(\theta)$). There are several possible choices for the prior but the one adopted here is the *Automatic Relevance Determination* (ARD) prior [Neal, 1998, Wipf and Nagarajan, 2007]. Briefly, the ARD prior groups weights in the hidden layer together in a meaningful way. Weights that are connected to the same SNP across hidden units are given a shared parameter that controls how large the values they take on are allowed to become. Specifically, the weights in each hidden unit that are associated with SNP j are given the following hierarchical prior:

$$\beta_{kj} \sim N(0, \sigma_j^2) \tag{4.7}$$

$$\sigma_j^2 \sim IG(\alpha_0, \beta_0) \tag{4.8}$$

where $N(\cdot, \cdot)$ and $IG(\cdot, \cdot)$ represent Normal and Inverse-Gamma densities, respectively. This prior allows the network to automatically determine which inputs are the most relevant. If SNP j seems to be related to the response, then the posterior distribution for σ_j^2 will be concentrated around larger values. We have previously developed a Bayesian framework that allows for testing of variable importance relative to a null model [Beam et al., 2014b]. This framework provides a posterior probability of each SNP's involvement in the response and is capable of being applied to continuous, dose-response data in addition to the case-control scenario for which it was originally developed.

The denominator of equation (4.5) is a integral that, in high-dimensions, is intractable. Thus stochastic integration techniques, such as Markov Chain Monte Carlo (MCMC) are often used. MCMC algorithms are able to draw samples from the posterior distribution $p(\theta|x_i, y_i)$ and only require the ability to evaluate $L(\theta|x_i, y_i)p(\theta)$ for specific values of θ. However, popular

algorithms such as the well known random-walk Metropolis-Hastings (RW-MH) [Metropolis et al., 1953, Hastings, 1970] explore the posterior distribution too slowly to be useful in a high-dimensional setting. Recently an algorithm known as Hamiltonian Monte Carlo (HMC) has been shown to explore high-dimensional posterior distributions much more efficiently by leveraging information about the posterior's gradient, which guides the simulation to regions of high probability. This efficiency does not come free, as evaluation of the gradient must be done 10s to 1000s of times *per iteration*, possibility limiting HMC's usefulness when the number of parameters is large. To enable the use of HMC on complex models in high-dimensions, we have developed a GPU-based sampling scheme that allows neural network models to be used on large datasets [Beam et al., 2014a]. A graphical overview of the entire Bayesian neural network procedure is provided in Figure 4.2.

This framework comes with several advantages relative to the MANOVA approach. As discussed previously, the relaxation of the linear assumption allows for a broader class of possible genetic architectures to be detected. Additionally, the BNN approach provides a nonparameteric procedure for variable selection while MANOVA relies on normality of errors for its testing procedure to be valid. For instance, p-values from a MANOVA analysis may not be valid for scenarios in which the number of individuals for one genotype is rare (e.g. very few homozygotes for the minor allele). Additionally, normalization procedures must often be performed in order to ensure the error distributions conform as closely as possible to the assumed distribution. In contrast, due the lack of strict assumptions required by the Bayesian neural network framework, there is no reason in principle why a BNN would be affected by any of these issues.

4.3 Results and Discussion

We investigated three plausible relationships between genotype and drug-response. For all simulations a Bayesian neural network with 10 hidden units was used. The HMC simulation was performed for 375 iterations, with the first 25 discarded as burn-in. For the HMC sampler a step-size value of 0.02, a momentum-persistence value of 0.75, and a temperature value of 1000

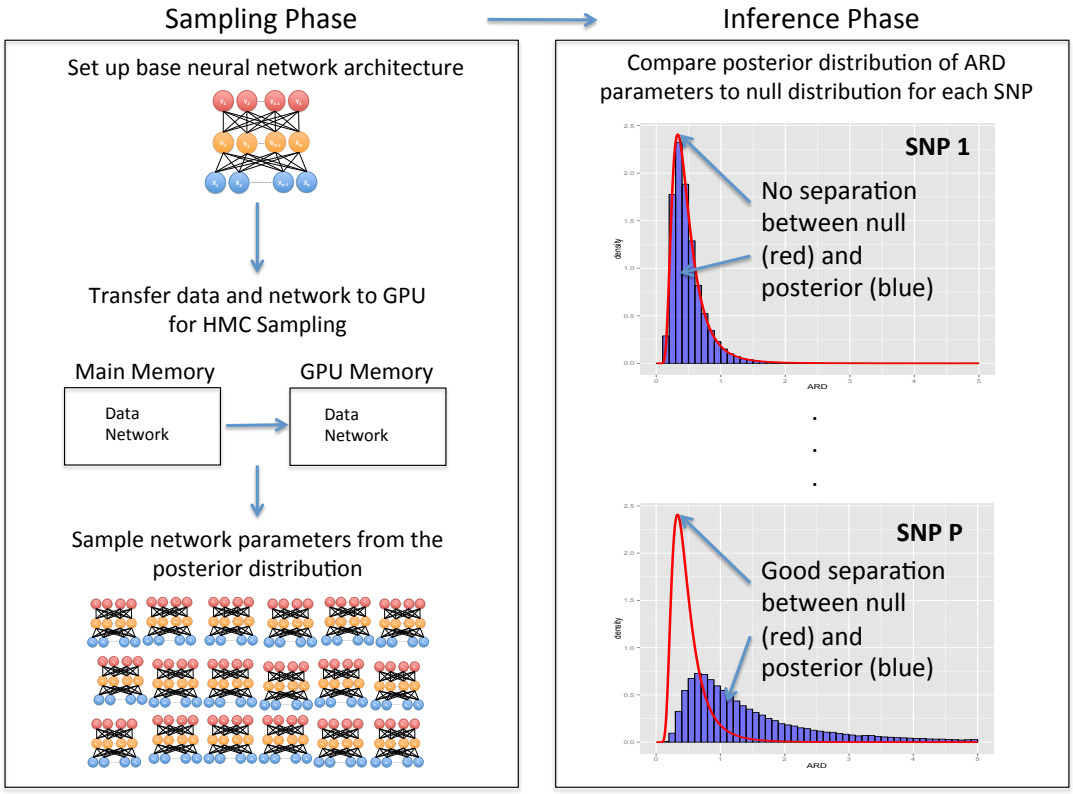

Figure 4.2: Overview of the Bayesian neural network method for dose-response studies. First the network architecture is established and transferred along with the data to GPU memory. The HMC simulation is performed on the GPU and the samples are then transferred back to main memory. The posterior for each SNP's ARD parameter is compared to the null distribution. Bayesian posterior probabilities are computed to assess how likely each SNP is to be involved in determining drug-response. In the right panel SNP 1 shows little evidence of being involved with this trait while SNP P has strong evidence of involvement.

were used. The ARD hyper-parameter values were set to $\alpha_0 = 3$ and $\beta_0 = 1$ while the output layer used $\alpha_0 = 0.1$ and $\beta_0 = 0.1$, and we used an ARD cut-off value of 0.4. Each response was normalized to have mean 0, unit variance. All BNN code was written in Python and is avilable at `https://github.com/beamandrew/BNN`. Analyzing each dataset took the BNN approximately 4 minutes.

We used a Bonferonni cut-off value of $p < 0.05$ for the MANOVA procedure. MANOVA was conducted using the `manova()` function in R.

4.3.1 Additive Model

As a baseline for the Bayesian neural network model, we first performed a simulation study using a simple additive model involving 2 loci. We generated a mean dose-response for each concentration, $\mu =< \mu_1, \ldots, \mu_6 >$, according to the hill-slope model in equation (4.1):

$$\mu_k = 1 - \frac{1}{1 + x_k^{-1.5}} \tag{4.9}$$

for $k \in \{1, \ldots 6\}$ and $x = 10^{-4} * \{0.03125, 0.0625, 0.10, 0.25, 0.5, 1.25\}$, resulting in a mean dose-response curve for 6 concentrations. Deviations from the mean induced by SNP status were generated according to a linear model plus a heteroskedastic noise term, yielding an observation i at each concentration, y_{ik}:

$$y_{ik} = \mu_k * \left(1 + \frac{\theta}{2} * S_1 - \frac{\theta}{2} * S_2 + \epsilon\right) \tag{4.10}$$

where $\epsilon \sim N(0, 0.1)$ and $S_1, S_2 \in \{0, 1, 2\}$ represent number of minor alleles at the two causal loci, and $\theta \in [0, 1]$ represents the effect size of each SNP. In this model being homozygous for the minor allele causes a θ% change relative to the baseline mean, μ_k. For example, $\theta = 0.02$ would correspond to a 2% change for a subject that is homozygous for the minor allele at S_1 while being homozygous for the major allele at S_2:

$$\begin{aligned} y_k &= \mu_k * \left(1 + \frac{0.02}{2} * 2 - \frac{0.02}{2} * 0 + \epsilon\right) \\ &= \mu_k + \mu_k * 0.02 + \epsilon_k \end{aligned}$$

Also note that one locus (S_1) is associated with increased drug response while the other locus (S_2) confers a decreased average response. To evaluate the sensitivity of both MANOVA and

BNN to effect size (θ) and minor allele frequency (MAF), we simulated data sets for $\theta = \{0.01.0.02, 0.03, 0.04, 0.05\}$ and MAF $= \{0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$. For each combination of θ and MAF, we generated 100 data sets resulting in 3500 data sets used in evaluation. Each dataset contained $2,000$ observations and 998 background SNPs. Background SNPs were generated according to a random MAF, ranging uniformly from 0.01 to 0.5. For each parameter combination, the number of times out of 100 each method correctly identified both causal SNPs was taken as an estimate of statistical power. The results are shown in Figure 4.3.

Both BNNs and MANOVA display good power across a variety of parameter combinations, with MANOVA having a slight edge for a scenarios. This is perhaps unsurprising as MANOVA is testing the linear hypothesis directly, while the BNN is testing a much more general hypothesis.

4.3.2 Additive Model with Interactions

Next, we considered an additive model with an interaction term, shown below.

$$y_k = \mu_k * \left(1 + \frac{\theta}{2} * S_1 - \frac{\theta}{2} * S_2 + \frac{\theta}{2} * S_1 * S_2 + \epsilon\right) \tag{4.11}$$

In this model there is deviation from additivity induced by the the interaction term $S_1 * S_2$. We again swept over the same range of values for θ and MAF as done previously. The results are shown in Figure 4.4.

Again, both methods appear to have good power across a spectrum of parameter values. However, MANOVA does not perform as well as for larger values of MAF. The BNN also experiences a loss of power for the highest level of MAF. For most parameter combinations, the BNN outperformed MANOVA.

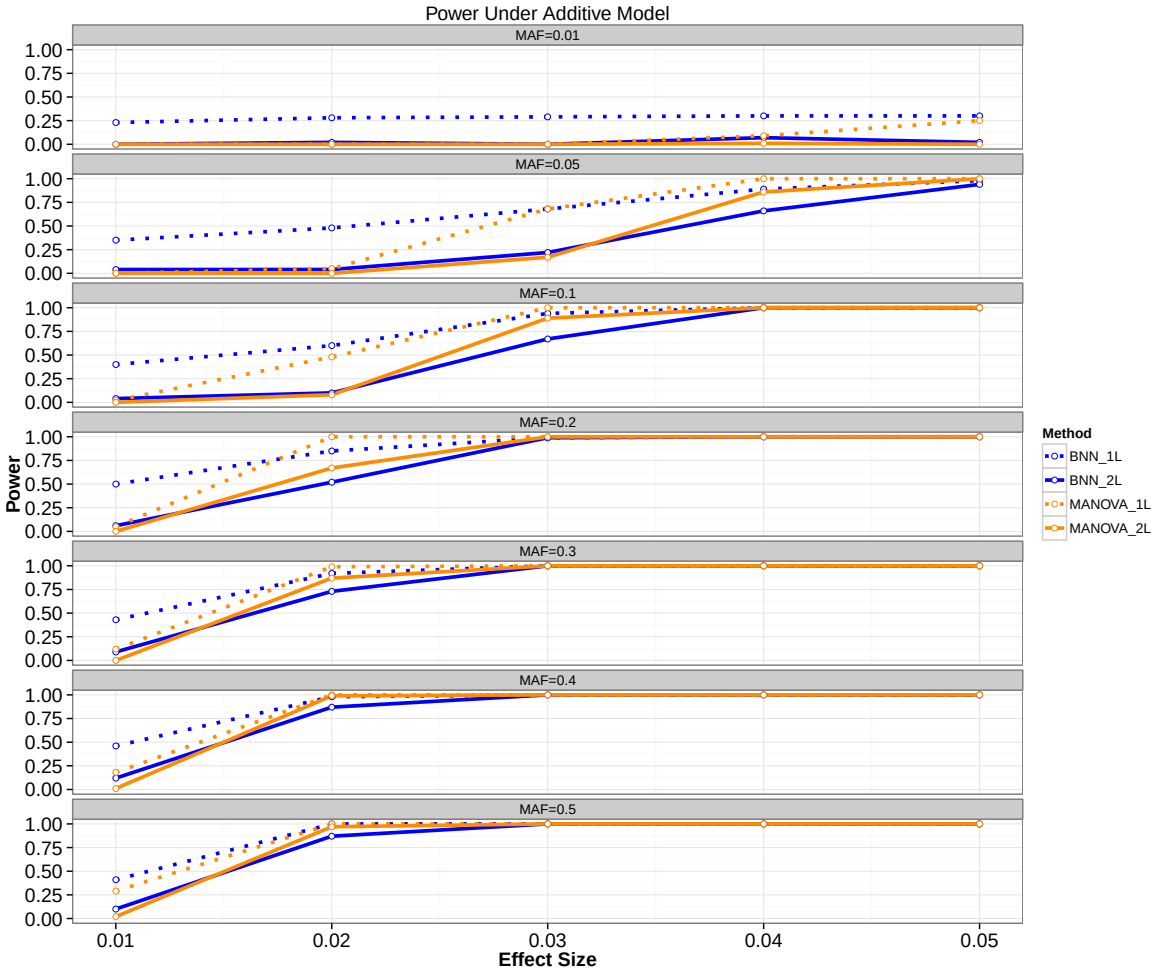

Figure 4.3: Power results for the additive model. The power for the Bayesian neural network (BNN) model is shown in blue while MANOVA is shown in orange. Solid lines indicate the power to detect both loci, while the dashed lines indicate power to detect at least one of the causal loci.

4.3.3 Purely Interactive Model

Finally, we investigated a model in which the SNPs affect the response *only* through an interaction, shown below. The results are shown in Figure 4.5.

$$y_k = \mu_k * \left(1 + \frac{\theta}{2} * S_1 * S_2 + \epsilon\right) \tag{4.12}$$

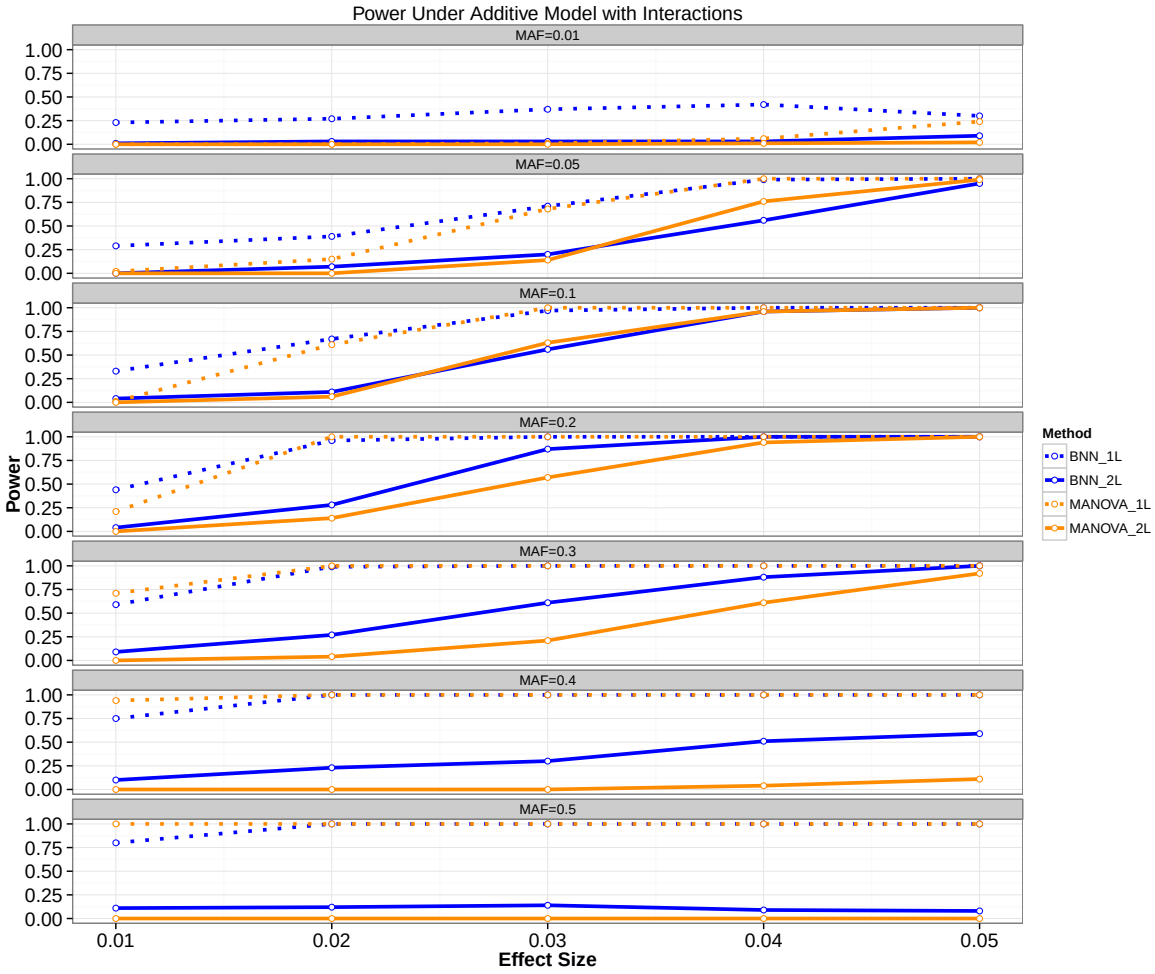

Figure 4.4: Power results for the additive model with interactions. The power for the Bayesian neural network (BNN) model is shown in blue while MANOVA is shown in orange. Solid lines indicate the power to detect both loci, while the dashed lines indicate power to detect at least one of the causal loci.

Perhaps surprisingly, MANOVA displays good power to detect the causal loci for several parameter combinations, despite the true model ostensibly lacking any marginal effects. Both methods struggled for smaller values of MAF, but this is expected due the relatively rare nature of positive responses for small levels of MAF. Across all scenarios test for this purely interactive model, both methods achieve near identical performance.

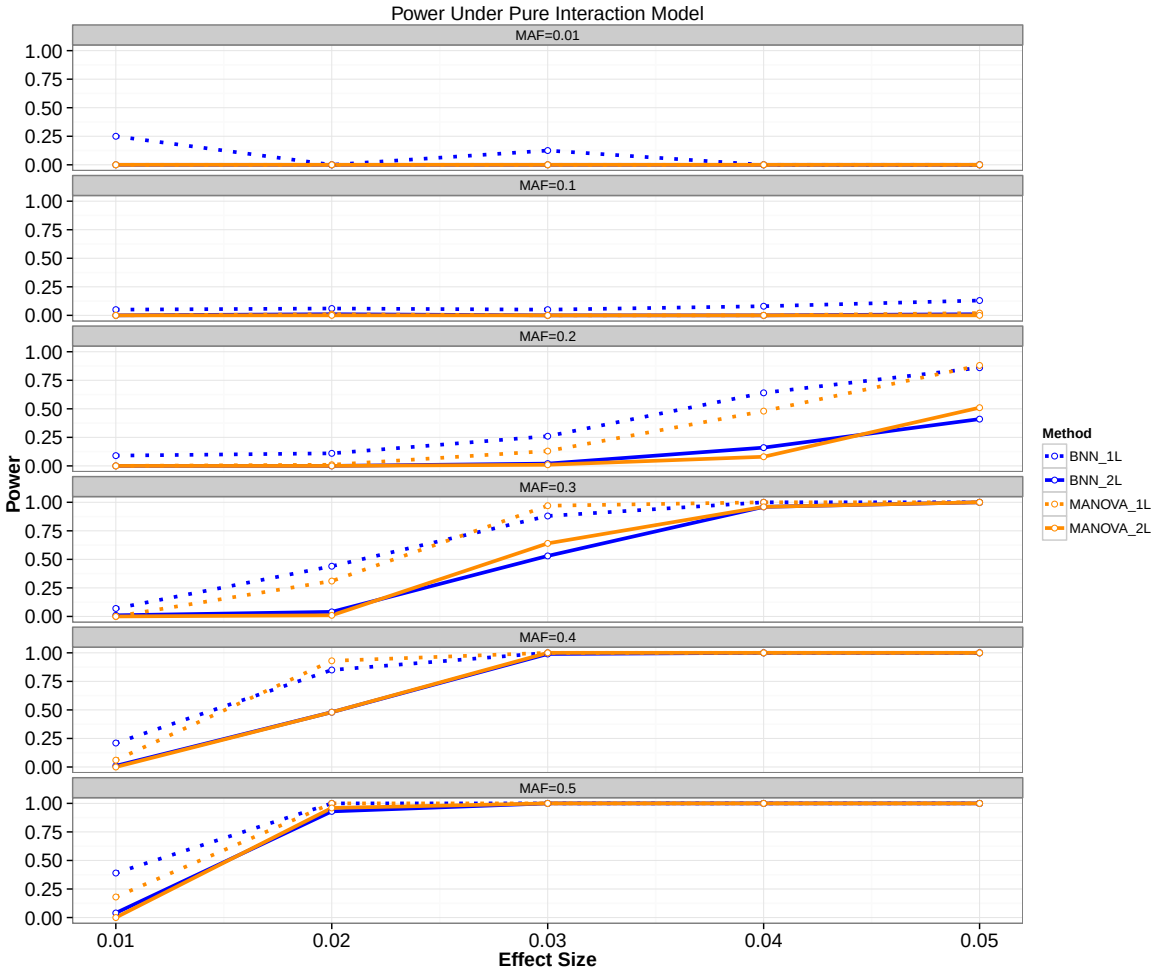

Figure 4.5: Power results for the purely interactive mode. The power for the Bayesian neural network (BNN) model is shown in blue while MANOVA is shown in orange. Solid lines indicate the power to detect both loci, while the dashed lines indicate power to detect at least one of the causal loci.

4.3.4 Further Analysis of Simulated Models

The results of the preceding section motivated a more in-depth analysis of the reasons behind MANOVA's apparent ability to detect models that only contain interactions and the reduced ability by both methods to detect the model containing main effects and an interaction term. The apparent loss of power by both methods for higher values of MAF in Figure 4.4 may be an

artifact of the specific simulation parameters used. Notice for example, that both MANOVA and BNNs have good power to detect one locus for all simulated scenarios, as evidenced by the dotted line. This suggests that the loss of power under this model is due to the decreased ability to detect the second locus (S_2), as examination of the raw simulation results indicates this locus was rarely identified as significant. To examine this hypothesis, we will assume a more general population genetics framework and then reexamine the results of the simulated models in the previous section as specific instances of this broader viewpoint.

A marginal effect is the expected value of y_{ik} given the status at one locus, averaged over all possible values for all other loci. Since the MANOVA procedure only detects marginal effects, a deeper understanding of how they may manifest in these putative models of genetic influence may be valuable. What we intend to show is that the discrete nature of the minor allele coding can introduce artifacts such as main effects in models where none are explicitly present and how marginal effects are attenuated by an interaction term in models containing both. First, we will examine the marginal effect of having a specified genotype at one locus (e.g. $S_2 = 1$) independently of the status at the other locus, and determine generally how this marginal effect changes under different simulated models as a function of effect size and MAF.

For the moment assume we an infinite population of individuals and we are interested in the relationship between two loci S_1 and S_2 and a quantitative trait, y. Let p_{xy} be the proportion of individuals containing x copies of the minor allele for locus S_1 and y copies of the minor allele for locus S_2. As done previously in equation (4.11), assume that the expected value of y, $E[y]$ is given by an additive relationship:

$$E[y] = \beta_1 * S_1 + \beta_2 * S_2 \tag{4.13}$$

where β_1, β_2 are the effects of having the S_1, S_2 genotypes, respectively. Consider the marginal effect of having the genotype $S_2 = 1$. The conditional expectation of y given $S_2 = 1$, $E(y|S_2 = 1)$, can be computed using the law of total probability, with a weighted sum over all possible

values for S_1:

$$E[y|S_2 = 1] = \sum_{i=0}^{2} E(y|S_2 = 1, S_1 = i) * p_{i1} \tag{4.14}$$

Substituting equation (4.13) into (4.14) and simplifying:

$$\begin{aligned} E[y|S_2 = 1] =& E[y|S_2 = 1, S_1 = 0] \cdot p_{01} + E[y|S_2 = 1, S_1 = 1] \cdot p_{11} + \\ & E[y|S_2 = 1, S_2 = 2] \cdot p_{21} \\ & = \beta_2 \cdot p_{01} + (\beta_1 + \beta_2) \cdot p_{11} + (2\beta_1 + \beta_2) \cdot p_{21} \end{aligned}$$

Rearranging and simplifying the last line yields:

$$E[y|S_2 = 1] = \beta_1 \cdot (p_{11} + 2p_{21}) + \beta_2 \cdot (p_{01} + p_{11} + p_{21}) \tag{4.15}$$

Let the conditional expectation shown in equation 4.15 be referred to as the marginal effect of $S_2 = 1$ under the additive model, or μ_A. Now consider that instead of following a simple additive model, the expected value of y is given by an additive model plus an interaction term, shown below:

$$E[y] = \beta_1 * S_1 + \beta_2 * S_2 + \beta_3 * (S_1 \cdot S_2) \tag{4.16}$$

Again, using a similar approach as before and summing over all possibilities for S_1 the conditional expected value is:

$$E[y|S_2 = 1] = \beta_1(p_{11} + p_{21}) + \beta_2(p_{01} + p_{11} + p_{21}) + \beta_3(p_{11} + 2p_{21}) \tag{4.17}$$

Let this expected value be referred to as μ_{AI}. Equations (4.15),(4.17) give the marginal effect of having the genotype $S_1 = 1$ under an additive model and an additive model with an interaction for arbitrary effect sizes and for arbitrary genotype frequencies. Note that if $\beta_1, \beta_2, \beta_3 > 0$,

then $\mu_{AI} > \mu_A$ or the marginal effect under the model with interactions would be larger, thus MANOVA would have higher power to detect this SNP as causal. However, using the values used in the simulation ($\beta_1 = \beta_3 = \frac{\theta}{2}, \beta_2 = -\frac{\theta}{2}$) the equations reduce to the following expressions:

$$\mu_A = -\frac{\theta}{2}(p_{01} - p_{21})$$
$$\mu_{AI} = -\frac{\theta}{2}(p_{01} - p_{11} - 2p_{21})$$

Here it is clear that $|\mu_{AI}| < |\mu_A|$ since $p_{xy} > 0 \; \forall x, y$. This is due to the negative S_2 effect, which induces a *smaller* marginal effect for this locus under the additive model with an interaction. This explains why in the simulations the power to detect this locus was reduced when compared to the purely additive model. The reason MANOVA has reduced power when MAF is high because the interaction term appears more often and since it has the opposite sign as the S_2 locus, it effectively cancels out the response, making it appear as if S_2 has a smaller marginal effect. Table 4.1 shows the values for μ_{AI} and μ_A for each possible level of S_2 for the effect sizes used in the simulations (i.e. $\beta_1 = \beta_3 = \frac{\theta}{2}, \beta_2 = -\frac{\theta}{2}$):

Table 4.1: Relationship beteween μ_{AI} and μ_A for each possible level of S_2.

S_2 Status	μ_{AI}	μ_A	$\lvert\mu_{AI}\rvert$? $\lvert\mu_A\rvert$
0	$\frac{\theta}{2}(p_{10} + 2p_{20})$	$\frac{\theta}{2}(p_{10} + 2p_{20})$	$=$
1	$-\frac{\theta}{2}(p_{01} - p_{11} - 2p_{21})$	$-\frac{\theta}{2}(p_{01} - p_{21})$	$<$
2	$-\frac{\theta}{2}(2p_{02} - p_{12} + 2p_{22})$	$-\frac{\theta}{2}(2p_{02} + p_{12})$	$<$

Note that all of the marginal effects for S_2 are smaller than or equal to the additive models marginal effect size. This explains why there was a decrease in power to detect S_2 by MANOVA in the simulations. Now we turn to the model without any explicit marginal effects and examine

if there are indeed any marginal effects present. Recall the purely interactive model is $E[y] = S_1 * S_2 * \beta_3$. Using the same analysis procedure as before we can calculate the marginal effect under an interactive model, μ_I for each level of S_2. This is shown in Table 4.2:

Table 4.2: Relationship beteween μ_I and μ_A for each possible level of S_2. A ? in the third column indicates that no strict inequality can be determined.

$\boldsymbol{S_2}$ **Status**	$\boldsymbol{\mu_I}$	$\boldsymbol{\mu_A}$	$\boldsymbol{\lvert\mu_I\rvert}$? $\boldsymbol{\lvert\mu_A\rvert}$
0	0	$\frac{\theta}{2}(p_{10} + 2p_{20})$	$<$
1	$\frac{\theta}{2}(p_{11} + 2p_{21})$	$-\frac{\theta}{2}(p_{01} - p_{21})$	?
2	$\frac{\theta}{2}(p_{12} + 4p_{22})$	$-\frac{\theta}{2}(2p_{02} + p_{12})$	?

Here there is no clear, strict inequality, but assessments can be made for specific values of p_{01}, p_{11}, p_{21}. Note that in general $|\mu_I| > 0$ in all instances so there will *always* be some amount of marginal effect present for this model. For many values of MAF, $|\mu_I|$ will be a non-trivial amount relative to $|\mu_A|$ resulting in high power for a model can only detect additive effects, such as MANOVA. Thus the reason MANOVA in Figure 4.5 has good power for high values of MAF is because there actually are marginal effects present, even though they were not explicitly included in the model.

The analysis in this section is meant as a small complement to the vast literature on quantitative trait loci (QTL) and the role of epistasis in settings other than the dose-response framework. Several comprehensive investigations have been made using model organisms such as *Drosophila melanogaster* [Mackay et al., 2009, Huang et al., 2012] for which there is considerable evidence of the role of epistasis in quantitative traits. How the results of these simple, 2 loci models might translate to larger epistatic networks in other contexts involving many more loci is yet unclear. However, the simulated models in this study suggest that epistatic interactions in a dose-response framework may manifest as marginal effects for each loci involved in

the interaction, at least for some configurations of the minor allele frequency.

4.4 Conclusions

In this study we have examined how gene-gene interactions can affect the ability to detect causal loci in cell-based, dose-response association studies. We have presented a novel nonparametric procedure in the form of a Bayesian neural network and compared its performance to the MANOVA framework. Using a simulation study of plausible genetic models and a population genetics based analysis, we have shown that MANOVA may be able to detect causal loci even in the presence of genetic interactions. Additionally, we have shown that the BNN is also very capable of detecting causal loci across a range of possible genetic models. Each approach comes with tradeoffs - the MANVO approach is conceptually more simple and computationally less expensive while the BNN approach is more flexible and built upon fewer assumptions at the expense of being more computationally demanding.

Chapter 5

Software for Bayesian Neural Networks in Python

5.1 Overview

Software implementing the approaches outlined in thesis is available for the Python programming language. The code base has grown from a few simple functions into a fully formed and very flexible software package. Though it would difficult to document all of the features in this section, we provide an overview that aims to highlight the most important components. To date, this appears to be the only GPU-enabled software available for building Bayesian neural networks and one of the only available packages for using Bayesian neural networks at all. The software is available for download at the following github repository, `https://github.com/beamandrew/BNN`, where it should remain for the foreseeable future. This chapter is presented not only for its use as software documentation but also in an effort to make concrete several of the main ideas of this dissertation, in the hope of being didactic by doing.The provided code should allow users to recreate any of the results found in this dissertation, in addition to analyzing new data sets of a similar nature.

The source, which is approximately 2,000 lines of code, is organized using an *Object Oriented*

paradigm. There a few types of objects that encapsulate the major components of the network, specifically *Layer*, *Prior*, *Network*, and *Sampler*. This design was aimed at allowing users to specify networks with arbitrary architectures, depth, activation functions, prior types, and sampler designs. In brief, the network is made up a collection of layers. Each *layer* can compute its output (i.e. the output for all the units it is responsible for) given the input from the previous layer. Likewise, it can compute its back-propagation signal (i.e. the error gradient) given the back-propagation signal of the layer above it. A *network* object manages this hierarchy and communicates with the *sampler* to run the simulation. All of the layer objects are left completely general, so different activation functions can be used simply by creating a layer of the desired type. These components are all built on top of the PyCuda library, and make use of the GPU in ways that are hidden from the user. Below an overview of the major components is given.

- **Layer** - Represents one layer, i.e. a collection of hidden units at the same level, in the network. This object maintains the weights for each hidden unit and is responsible for computing the specified activation function. Also responsible for computing the gradient back-propagation signal to be passed to a preceeding layer, if one exists. It also maintains a reference to the prior on the parameters it contains.

- **Prior** - Represents a prior distribution over weights in the same layer. Computes its own contribution to the log-posterior and gradient. Also responsible for performing the Gibbs update on the hyper-parameters it controls.

- **Network** - Contains and manages all of the layers and prior structures. Is responsible for global level commands such as `feedforward()` and `updateAllGradients()`, in addition to calculating the total log-posterior value.

- **Sampler** - Object responsible performing the HMC simulation, collecting and maintaining the posterior samplers. Includes several variations on the main HMC algorithm as well as utilities useful for analyzing the results of the simulation.

5.2 Usage Example

We conclude this section with a usage example that may be helpful in working with the software, as well as demonstrating how various components operate. The are a few step that must taken before anything can be done, regardless of the type of data being analyzed. The software relies on various external modules that must be loaded before it can be used. Additionally, an environment variable named `BNNPATH` must set and point to the source directory. These steps are shown below.

```
import os
import sys
bnn_path = os.environ['BNNPATH']
sys.path.append(bnn_path)
import numpy as np
```

Now we are ready to import the various components of the network and sampler. After these steps are complete we can begin to use the software to analyze the data of interest.

```
from network import BNN
from Sampler import HMC_sampler
from Layer import *
```

Next, we will analyze a case-control dataset, much like the data used in Chapter 3, but this time only using 50 SNPs so that it will be easier visualize certain aspects of the simulation. In this example, the last two SNPs are causal with respect to trait status, while the remaining 48 are unrelated. We will demonstrate usage with a single hidden layer that uses an ARD prior connected to a softmax layer that has a layer-wise Gaussian prior. Assume we already have read in a numpy array for the X matrix that contains our set of SNPs represented as the number of minor alleles present (i.e. 0, 1, or 2 copies of the minor allele). Likewise, we have read in the Y matrix representing case-control status in 'one-hot-coding'. This means Y has two columns, where the first column is a 1 if Y is a control and 0 if not, while column 2 is the

opposite. It is recommended to center and scale the minor allele count representation so that each column has mean 0 and standard deviation 1. This is recommended practice whenever using neural networks.

Now we can define the network architecture in terms of layers. We will first create a hidden layer that uses a Sigmoid (i.e. a logistic) activation function with 5 hidden units:

```
N = len(X) #Number of observations
n_preds = X.shape[1] #Number of predictors
hidden_layer = Sigmoid_Layer(n_units=5,n_incoming=n_preds,N=N)
```

Note that the layer takes arguments for the number of *incoming* connections (`n_incoming`) and the number of hidden units (`n_units`), which implicitly defines the number of *outgoing* connections. Next, we need to specify the ARD prior over parameters in this layer. The ARD prior needs specification of the shape (α_0) and scale (β_0) of the Inverse Gamma prior distribution used. In this implementation, these values must be fixed, but possible extensions would be to allow these parameters have their own priors. Here we use $\alpha_0 = 5$ and $\beta_0 = 2$ as used in Chapter 3, but the user may supply whatever values they think are reasonable. The Bayesian testing mechanism provided in the software automatically accounts for the user supplied values. Note that you should supply the shape and scale parameters as floats to avoid and numerical issues when these parameters are updated during the simulation.

```
hidden_layer_prior = ARD_Prior(shape=5.0,scale=2.0,layer=hidden_layer)
hidden_layer.setPrior(hidden_layer_prior)
```

Notice that the ARD_Prior takes as an argument a reference to the layer object (`hidden_layer`) in this case) for which will be responsible. The second line causes the hidden_layer object to register the `hidden_layer_prior` object as the prior over its parameters. Next, we will create the softmax output layer and connect it to the hidden layer.

```
n_classes = Y.shape[1] #Number of classes - K=2 is binary classification
output_layer = Softmax_Layer(n_classes=n_classes,n_incoming=10,N=N)
```

```
output_layer_prior = Gaussian_Layer_Prior(shape=0.1,scale=0.1,layer=output_layer)
output_layer.setPrior(output_layer_prior)
```

Instead of the output layer, we could have specified more hidden layers if we wished to increase the complexity of the network. The design allows users to specify networks of arbitrary complexity through this mechanism. The next step is to create a list that contains all of the layers, from bottom to top. This list will be used later to tell the network to which level of the DAG a layer belongs.

```
# Add layers to list from bottom to top
layers = list()
layers.append(hidden_layer)
layers.append(output_layer)
```

Finally, we are ready to create the network object that will be responsible for organizing and operating the layers we have created.

```
net = BNN(X=X,Y=Y,layers=layers)
```

If this is the first time you have used the software or you have changed the architecture at all, it will be useful to 'warm-up' the network by performing a feed-forward operation and a gradient update. This will cause PyCuda to compile all of the GPU code that will be needed later. This can be achieved using the following commands:

```
net.feed_forward()
net.updateAllGradients()
```

The network parameters are initialized using draws from Gaussian distributions with mean 0. You can control the standard deviation of this initialization through an optional parameter `init_sd` provided to each layer. Additionally, users can initialize the network to the *maximum a posteriori* value of either the log-likelihood or log-posterior using gradient descent before sampling begins. This is shown below.

```
net.initialize(iters=2000,verbose=True,step_size=1e-3,include_prior=True)
```

As always, when using gradient descent the step size must be small enough to ensure the algorithm does not diverge. Setting the `include_prior` argument to `False` causes the function to perform *maximum likelihood* optimization instead of searching for the MAP estimate.
We are now ready to create the `HMC_sampler` object. For this demonstration we create a sampler that performs HMC-based sampling for the neural network object created previously (`net`) using $L = 15$ leap-frog updates per iteration with a step-size of $\epsilon = 0.01$.

```
hmc = HMC_sampler(net,L=15,eps=0.01)
```

A user can additionally specify if they would like to scale the step-size using the current estimate of the hyper-parameter as in [Neal, 1995] by using the `scale=True` argument. This class has a few utility functions to help users monitor and diagnose their simulations. For instance, if we would like to track the ARD ranking and posterior probability of the Bayesian ARD test during the simulation of variables X_1 and X_2, we instantiate the sampler using the following construct:

```
track_vars = list()
track_vars.append(1)
track_vars.append(2)
hmc = HMC_sampler(net,L=15,eps=0.01,debug=True,track_vars=track_vars)
```

The `debug=True` instructs the sampler to monitor the variables specified in `track_vars` using a one-based index (e.g. 1 specifies X_1). This should be used with some caution as this process will result in a loss of performance and increased execution time. We are now ready to begin the simulation. An annealed simulation such as the one in Chapter 3 with 25 burnin iterations, 200 sampling iterations, a cooling rate of 0.9, an initial system temperature of 1000, and a persistent momentum factor of 0.75, could be invoked using the following command:

```
hmc.simple_annealing_sim(n_keep=200,n_burnin=100,eta=0.9,T0=1000.0,
```

```
persist=0.75,verbose=True)
```

Additionally, we can specify a `var_refresh` argument that tells the sampler how frequently the hyper parameters should be updated via Gibbs sampling. For the default setting of `var_refresh = 1.0`, the hyper parameters are updated before every HMC update, but this can be modified so that they are updated less frequently. Once the simulation is complete, we will likely want to inspect the results. There are several entry points for posterior analysis, but we will start by getting a global view of which variables appear to be the most important. Using the following command will result in a list being printed to the terminal where variables have been ranked according their posterior ARD values.

```
hmc.getARDPosteriorMeanSummary(useMedian=False)
```

By default, the average value of the posterior samples for the ARD mean will be used, but setting the `useMedian` argument to `True` will cause them to be ranked by the median posterior value. We can also inspect the variables that we requested to be monitored.

```
hmc.plot_debug()
```

This will produce a series of plots that show how each tracked variable's ranking and posterior test value change over the course of the simulation. This can be useful for diagnosing if a particular variable's ranking or posterior test probability has stabilized. Other useful functions include visualizing the posterior distrubtion for various predictors ARD parameters. Using `hmc.plotARD(var)` command will allow create a trace plot and historgram for a variable of interest. Below are the commands to plot the histograms and trace plots for the two causal SNPs (49 and 50) and one unrelated SNP.

```
hmc.plotARD(49)
hmc.plotARD(50)
hmc.plotARD(1)
```

The results of these commands are shown in Figure 5.1. It's easy to see that the causal SNPs take on larger values while the unrelated SNP takes on smaller ones, an indication that the ARD prior is operating as intended.

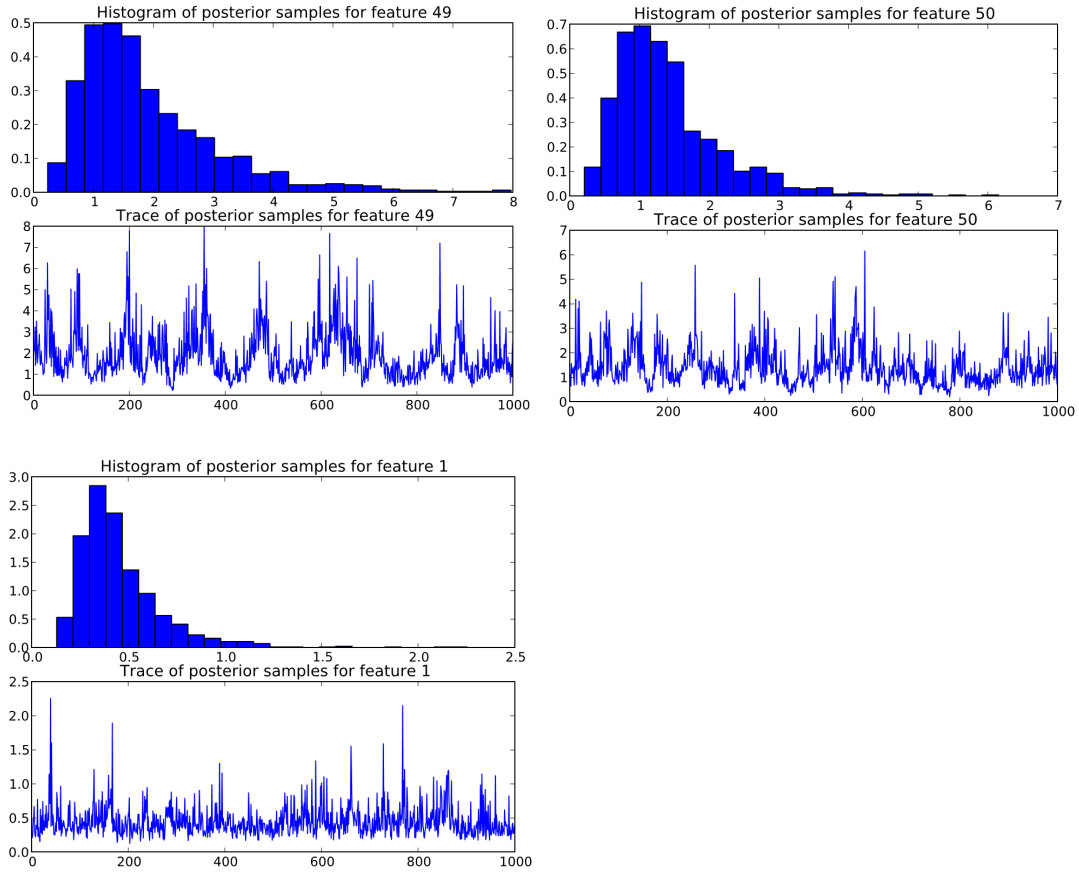

Figure 5.1: Posterior visualizations for the ARD parameters associated with SNPs 49 (upper left), 50 (upper right), and 1 (bottom). The blue bars are a histogram of the ARD parameter posterior samples for each SNP. The line plots are sample values (y-axis) plotted against simulation iteration (x-axis). Notice that the causal SNPs (49 and 50) take on larger values while the unrelated SNP looks more like draws from the prior distribution (i.e. $IG(5, 2)$).

We might also be interested in visualizing the network itself. For instance, it may be of interest to visualize how large the weights in the hidden layer are, since they are fundamental to the ARD testing process. We can instruct the network to produce plots of the values for the

weight at the end of the simulation and to plot the posterior mean for each weight, using the following commands.

```
net.plotCurrentWeights(self,layerID,absval=True)
plotPosteriorWeights(self,layerID,absval=True)
```

These plots show that SNPs 49 and 50 have large weight values across all 5 hidden units, indicating they are useful for determining trait status, while the other SNPs mostly have small weights. The large weight values in turn create large ARD values, as we observed in Figure 5.1. Finally we note that the bottom pane Figure 5.2 has fewer non-white cells for the unrelated SNPs, relative to the single values at the end of the simulation in the top pane. The top pane would be the result if we used a MAP estimate, and would be much noisier than the results we obtain from using the full posterior distribution.

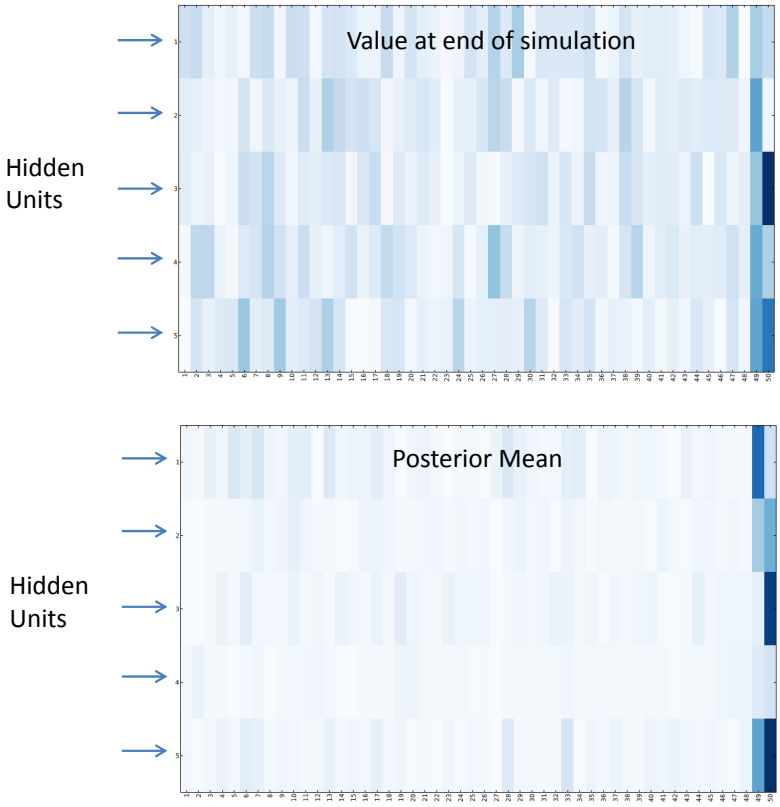

Figure 5.2: Visualization of values for the weights in the hidden layer, where darker colors indicate weights with larger magnitudes. Each horizontal band, indicated by the blue arrows, indicates the weights in each hidden unit across all 50 SNPs. Notice the dark vertical bands for SNPs 49 and 50 on the right side, indicating these SNPs have larger weights associated with them. The top pane shows the values of the of the weights at the end of the simulation, while the bottom pane shows the posterior average for each weight. Notice how the posterior average is a cleaner and sparser representation.

Chapter 6

Conclusion

We conclude with some remarks to summarize what has been learned in the course of this work. The goal of this thesis has been to develop methods with minimal assumptions on the type of relationship between genotype and phenotype. This was motivated by the growing evidence that epistasis is a critical piece of many complex genetic architectures. However, the challenges posed by potential gene-gene interactions in datasets containing millions of markers can be serve in some instances, and pathologically difficult in others. A survey of the literature indicates a dearth of methods that can automatically model interactions among markers in a computationally tractable manner.

To address this need, we investigated the use of a nonparameteric method in the form of a Bayesian neural network. Chapters 3 and 4 demonstrated that this method was very effective at discovering causal genetic markers across a wide range of simulated genetic models. The Bayesian ARD testing framework developed in this thesis was found to be a highly discriminative and practical way of selecting causal markers from a large set of potential variants. Though this approach comes with many useful properties, such as the ability to identify causal genetic markers using minimal assumptions, it carries with it a high computational burden. To ease this and make building models of this type feasible in high-dimensions, Chapter 2 developed infrastructure based on GPUs that was shown to expedite the process by several orders of

magnitude. Without the speedup offered by GPUs, it is unlikely that these models would be of any practical use.

In conclusion, the work presented here offers a promising line of inquiry for methodological research in genetic association studies. Though the genomic revolution has already taken place, we are still very much in the infancy of understanding how, and the extent to which, genetic variation gives rise to phenotypic variation. Much work remains to be done to fill in the missing pieces left blank by our current understanding and methodologies. The methods presented in this thesis offer a new way to investigate this fundamental question that is agnostic to the precise functional relationship between genotype and phenotype. Thus, should epistasis turn out to be relatively rare in the genetic architecture of complex disease, the framework investigated in this thesis would still be of utility, since the Bayesian neural network is capable of automatically capturing additive genetic effects.

As Alan Turing said in his seminal paper *Computing Machinery and Intelligence*, 'We can only see a short distance ahead, but we can see plenty there that needs to be done.' So too, it is with genomic science. Though the challenges are great, the promise of this nascent scientific discipline is equally so, and it is our hope that this work has contributed in some small way to progress in this exciting field.

REFERENCES

[Andrieu et al., 2003] Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine learning*, 50(1-2):5–43.

[Archibald et al., 2010] Archibald, A. L., Bolund, L., Churcher, C., Fredholm, M., Groenen, M. A., Harlizius, B., Lee, K.-T., Milan, D., Rogers, J., Rothschild, M. F., et al. (2010). Pig genome sequence-analysis and publication strategy. *BMC genomics*, 11(1):438.

[Baesens et al., 2002] Baesens, B., Viaene, S., den Poel, D. V., Vanthienen, J., and Dedene, G. (2002). Bayesian neural network learning for repeat purchase modelling in direct marketing. *European Journal of Operational Research*, 138(1):191–211.

[Beam and Motsinger-Reif, 2013] Beam, A. and Motsinger-Reif, A. (2013). Beyond IC50s: Towards robust statistical methods for in vitro association studies. *J Pharmacogenom Pharmacoproteomics*, 2(120):2153–0645.

[Beam et al., 2014a] Beam, A. L., Ghosh, S. K., and Doyle, J. (2014a). Fast Hamiltonian Monte Carlo using GPU computing. *ArXiv e-prints*. 1402.4089; Provided by the SAO/NASA Astrophysics Data System.

[Beam et al., 2014b] Beam, A. L., Motsinger-Reif, A., and Doyle, J. (2014b). Bayesian neural networks for genetic association studies of complex disease. *arXiv preprint arXiv:1404.3989*.

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127.

[Benson et al., 2008] Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., and Wheeler, D. L. (2008). Genbank. *Nucleic acids research*, 36(suppl 1):D25–D30.

[Bergstra et al., 2010] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A CPU and GPU

math compiler in Python. In *Proceedings of the Python for Scientific Computing Conference, SciPy*.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Brenchley et al., 2012] Brenchley, R., Spannagl, M., Pfeifer, M., Barker, G. L., DAmore, R., Allen, A. M., McKenzie, N., Kramer, M., Kerhornou, A., Bolser, D., et al. (2012). Analysis of the bread wheat genome using whole-genome shotgun sequencing. *Nature*, 491(7426):705–710.

[Brown et al., 2011] Brown, C., Havener, T. M., Everitt, L., McLeod, H., and Motsinger-Reif, A. A. (2011). A comparison of association methods for cytotoxicity mapping in pharmacogenomics. *Frontiers in genetics*, 2.

[Brown et al., 2012a] Brown, C. C., Havener, T. M., Medina, M. W., Auman, J. T., Mangravite, L. M., Krauss, R. M., McLeod, H. L., and Motsinger-Reif, A. A. (2012a). A genome-wide association analysis of temozolomide response using lymphoblastoid cell lines reveals a clinically relevant association with mgmt. *Pharmacogenetics and genomics*, 22(11):796.

[Brown et al., 2014] Brown, C. C., Havener, T. M., Medina, M. W., Jack, J. R., Krauss, R. M., McLeod, H. L., and Motsinger-Reif, A. A. (2014). Genome-wide association and pharmacological profiling of 29 anticancer agents using lymphoblastoid cell lines. *Pharmacogenomics*, 15(2):137–146.

[Brown et al., 2012b] Brown, C. C., Havener, T. M., Medina, M. W., Krauss, R. M., McLeod, H. L., and Motsinger-Reif, A. A. (2012b). Multivariate methods and software for association mapping in dose-response genome-wide association studies. *BioData mining*, 5(1):1–15.

[Brown et al., 2012c] Brown, C. C., Havener, T. M., Medina, M. W., Krauss, R. M., McLeod, H. L., Motsinger-Reif, A. A., et al. (2012c). Multivariate methods and software for association mapping in dose-response genome-wide association studies. *BioData mining*, 5(1).

[Carlborg and Haley, 2004] Carlborg, Ö. and Haley, C. S. (2004). Epistasis: too often neglected in complex trait studies? *Nature Reviews Genetics*, 5(8):618–625.

[Collins and McKusick, 2001] Collins, F. S. and McKusick, V. A. (2001). Implications of the human genome project for medical science. *Jama*, 285(5):540–544.

[Collins et al., 2003] Collins, F. S., Morgan, M., and Patrinos, A. (2003). The human genome project: lessons from large-scale biology. *Science*, 300(5617):286–290.

[Consortium et al., 2012] Consortium, T. G. et al. (2012). The tomato genome sequence provides insights into fleshy fruit evolution. *Nature*, 485(7400):635–641.

[Cordell, 2002] Cordell, H. J. (2002). Epistasis: what it means, what it doesn't mean, and statistical methods to detect it in humans. *Human molecular genetics*, 11(20):2463–2468.

[Davis and Rabinowitz, 2007] Davis, P. J. and Rabinowitz, P. (2007). *Methods of numerical integration*. Courier Dover Publications.

[Diaz-Uriarte and de Andres, 2006] Diaz-Uriarte, R. and de Andres, S. A. (2006). Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7:3.

[Duane et al., 1987] Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid monte carlo. *Physics letters B*, 195(2):216–222.

[Eichler et al., 2010] Eichler, E. E., Flint, J., Gibson, G., Kong, A., Leal, S. M., Moore, J. H., and Nadeau, J. H. (2010). Missing heritability and strategies for finding the underlying causes of complex disease. *Nature Reviews Genetics*, 11(6):446–450.

[Friedman et al., 2010] Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1.

[Friedman, 2001] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Ann.Statist*, 29(5):1189–1232.

[Gelman et al., 2008] Gelman, A., Jakulin, A., Pittau, M. G., and Su, Y.-S. (2008). A weakly informative default prior distribution for logistic and other regression models. *The Annals of Applied Statistics*, pages 1360–1383.

[Genkin et al., 2007] Genkin, A., Lewis, D. D., and Madigan, D. (2007). Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304.

[Greene et al., 2010] Greene, C. S., Sinnott-Armstrong, N. A., Himmelstein, D. S., Park, P. J., Moore, J. H., and Harris, B. T. (2010). Multifactor dimensionality reduction for graphics processing units enables genome-wide testing of epistasis in sporadic ALS. *Bioinformatics*, 26(5):694–695.

[Guyon et al., 2004] Guyon, I., Gunn, S. R., Ben-Hur, A., and Dror, G. (2004). Result analysis of the NIPS 2003 feature selection challenge. In *NIPS*, volume 4, pages 545–552.

[Guyon et al., 2002] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422.

[Hahn et al., 2003] Hahn, L. W., Ritchie, M. D., and Moore, J. H. (2003). Multifactor dimensionality reduction software for detecting gene–gene and gene–environment interactions. *Bioinformatics*, 19(3):376–382.

[Hastings, 1970] Hastings, W. K. (1970). Monte carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109.

[Hindorff et al., 2011] Hindorff, L. A., Junkins, H. A., Mehta, J., Manolio, T., et al. (2011). A catalog of published genome-wide association studies. *National Human Genome Research Institute*.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[Hoerl and Kennard, 1970] Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

[Hoffman and Gelman, 2012] Hoffman, M. D. and Gelman, A. (2012). The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian monte carlo. *Journal of Machine Learning Research*, pages 1–30.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[Huang et al., 2009] Huang, S., Li, R., Zhang, Z., Li, L., Gu, X., Fan, W., Lucas, W. J., Wang, X., Xie, B., Ni, P., et al. (2009). The genome of the cucumber, cucumis sativus l. *Nature genetics*, 41(12):1275–1281.

[Huang et al., 2012] Huang, W., Richards, S., Carbone, M. A., Zhu, D., Anholt, R. R., Ayroles, J. F., Duncan, L., Jordan, K. W., Lawrence, F., Magwire, M. M., et al. (2012). Epistasis dominates the genetic architecture of drosophila quantitative traits. *Proceedings of the National Academy of Sciences*, 109(39):15553–15559.

[Ioannidis, 2007] Ioannidis, J. P. (2007). Non-replication and inconsistency in the genome-wide association setting. *Human heredity*, 64(4):203–213.

[Jakše et al., 2008] Jakše, J., Meyer, J. D., Suzuki, G., McCallum, J., Cheung, F., Town, C. D., and Havey, M. J. (2008). Pilot sequencing of onion genomic DNA reveals fragments of transposable elements, low gene densities, and significant gene enrichment after methyl filtration. *Molecular Genetics and Genomics*, 280(4):287–292.

[Jiang et al., 2011] Jiang, X., Neapolitan, R. E., Barmada, M. M., and Visweswaran, S. (2011). Learning genetic epistasis using bayesian network scoring criteria. *BMC bioinformatics*, 12:89–2105–12–89.

[Jones et al., 2001] Jones, E., Oliphant, T., and Peterson, P. (2001). Scipy: Open source scientific tools for python. *http://www. scipy. org/*.

[Kanehisa and Goto, 2000] Kanehisa, M. and Goto, S. (2000). KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30.

[Khronos, 2008] Khronos (2008). The opencl specification. *A. Munshi, Ed.*

[Klockner et al., 2012] Klockner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., and Fasih, A. (2012). PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. *Parallel Computing*, 38(3):157–174.

[Koo et al., 2013] Koo, C. L., Liew, M. J., Mohamad, M. S., and Salleh, A. H. M. (2013). A review for detecting gene-gene interactions using machine learning methods in genetic epidemiology. *BioMed research international*, 2013.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4.

[Lander et al., 2001] Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., et al. (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921.

[Lawson et al., 1979] Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. (1979). Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):308–323.

[LeCun et al., 1995] LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60.

[Li et al., 2011] Li, J., Horstman, B., and Chen, Y. (2011). Detecting epistatic effects in association studies at a genomic level based on an ensemble approach. *Bioinformatics*, 27(13):222–229.

[Li and Reich, 2000] Li, W. and Reich, J. (2000). A complete enumeration and classification of two-locus disease models. *Human heredity*, 50(6):334–349.

[Li et al., 2002] Li, Y., Campbell, C., and Tipping, M. (2002). Bayesian automatic relevance determination algorithms for classifying gene expression data. *Bioinformatics*, 18(10):1332–1339.

[Lisboa et al., 2003] Lisboa, P. J., Wong, H., Harris, P., and Swindell, R. (2003). A bayesian neural network approach for modelling censored data with an application to prognosis after surgery for breast cancer. *Artificial Intelligence in Medicine*, 28(1):1–25.

[Lopes and Ribeiro, 2009] Lopes, N. and Ribeiro, B. (2009). *GPU implementation of the multiple back-propagation algorithm*, pages 449–456. Springer.

[Lunetta et al., 2004] Lunetta, K. L., Hayward, L. B., Segal, J., and Eerdewegh, P. V. (2004). Screening large-scale association study data: exploiting interactions using random forests. *BMC genetics*, 5(1):32.

[Mackay et al., 2009] Mackay, T. F., Stone, E. A., and Ayroles, J. F. (2009). The genetics of quantitative traits: challenges and prospects. *Nature Reviews Genetics*, 10(8):565–577.

[Manolio et al., 2009] Manolio, T. A., Collins, F. S., Cox, N. J., Goldstein, D. B., Hindorff, L. A., Hunter, D. J., McCarthy, M. I., Ramos, E. M., Cardon, L. R., Chakravarti, A., et al. (2009). Finding the missing heritability of complex diseases. *Nature*, 461(7265):747–753.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21:1087.

[Minsky and Papert, 1987] Minsky, M. L. and Papert, S. A. (1987). *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*. MIT press Boston, MA:.

[Moore, 2003] Moore, J. H. (2003). The ubiquitous nature of epistasis in determining susceptibility to common human diseases. *Human heredity*, 56(1-3):73–82.

[Moore, 2005] Moore, J. H. (2005). A global view of epistasis. *Nature genetics*, 37(1).

[Moore et al., 2006] Moore, J. H., Gilbert, J. C., Tsai, C.-T., Chiang, F.-T., Holden, T., Barney, N., and White, B. C. (2006). A flexible computational framework for detecting, characterizing, and interpreting statistical patterns of epistasis in genetic studies of hu disease susceptibility. *Journal of theoretical biology*, 241(2):252–261.

[Motsinger-Reif et al., 2008a] Motsinger-Reif, A. A., Dudek, S. M., Hahn, L. W., and Ritchie, M. D. (2008a). Comparison of approaches for machine-learning optimization of neural networks for detecting gene-gene interactions in genetic epidemiology. *Genetic epidemiology*, 32(4):325–340.

[Motsinger-Reif et al., 2008b] Motsinger-Reif, A. A., Reif, D. M., Fanelli, T. J., and Ritchie, M. D. (2008b). A comparison of analytical methods for genetic association studies. *Genetic epidemiology*, 32(8):767–778.

[Motsinger-Reif and Ritchie, 2008] Motsinger-Reif, A. A. and Ritchie, M. D. (2008). Neural networks for genetic epidemiology: past, present, and future. *BioData mining*, 1(3).

[Nabney, 2002] Nabney, I. (2002). *NETLAB: algorithms for pattern recognition*. Springer.

[Neal, 2011] Neal, R. (2011). MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162.

[Neal, 1992] Neal, R. M. (1992). Bayesian training of backpropagation networks by the hybrid monte carlo method. Technical report, Citeseer.

[Neal, 1995] Neal, R. M. (1995). Bayesian learning for neural networks.

[Neal, 1998] Neal, R. M. (1998). Assessing relevance determination methods using delve. *Nato ASI Series F Computer and Systems Sciences*, 168:97–132.

[Nvidia, 2008] Nvidia, C. (2008). Programming guide.

[Oh and Jung, 2004] Oh, K.-S. and Jung, K. (2004). GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314.

[Oki et al., 2011] Oki, N. O., Motsinger-Reif, A. A., Antas, P. R., Levy, S., Holland, S. M., and Sterling, T. R. (2011). Novel human genetic variants associated with extrapulmonary tuberculosis: a pilot genome wide association study. *BMC research notes*, 4(1):28.

[Olden and Jackson, 2002] Olden, J. D. and Jackson, D. A. (2002). Illuminating the black box: a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154(1):135–150.

[Png et al., 2012] Png, E., Alisjahbana, B., Sahiratmadja, E., Marzuki, S., Nelwan, R., Balabanova, Y., Nikolayevskyy, V., Drobniewski, F., Nejentsev, S., Adnan, I., et al. (2012). A genome wide association study of pulmonary tuberculosis susceptibility in indonesians. *BMC medical genetics*, 13(1):5.

[Rasmussen, 2006] Rasmussen, C. E. (2006). Gaussian processes for machine learning.

[Ritchie, 2012] Ritchie, M. D. (2012). The success of pharmacogenomics in moving genetic association studies from bench to bedside: study design and implementation of precision medicine in the post-gwas era. *Human genetics*, 131(10):1615–1626.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). *Learning representations by back-propagating errors*. MIT Press, Cambridge, MA, USA.

[Sanner et al., 1999] Sanner, M. F. et al. (1999). Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1):57–61.

[Slatkin, 2009] Slatkin, M. (2009). Epigenetic inheritance and the missing heritability problem. *Genetics*, 182(3):845–850.

[Stan Development Team, 2013] Stan Development Team (2013). Stan: A C++ library for probability and sampling, version 2.1.

[Strittmatter et al., 1993] Strittmatter, W. J., Saunders, A. M., Schmechel, D., Pericak-Vance, M., Enghild, J., Salvesen, G. S., and Roses, A. D. (1993). Apolipoprotein e: high-avidity binding to beta-amyloid and increased frequency of type 4 allele in late-onset familial Alzheimer disease. *Proceedings of the National Academy of Sciences*, 90(5):1977–1981.

[Team et al., 2005] Team, R. C. et al. (2005). R: A language and environment for statistical computing. *R foundation for Statistical Computing*.

[Tibshirani, 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

[Tierney, 1994] Tierney, L. (1994). Markov chains for exploring posterior distributions. *the Annals of Statistics*, pages 1701–1728.

[Urbanowicz et al., 2012] Urbanowicz, R. J., Kiralis, J., Sinnott-Armstrong, N. A., Heberling, T., Fisher, J. M., and Moore, J. H. (2012). Gametes: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures. *BioData mining*, 5(1):1–14.

[van Oeveren et al., 2011] van Oeveren, J., de Ruiter, M., Jesse, T., van der Poel, H., Tang, J., Yalcin, F., Janssen, A., Volpin, H., Stormo, K. E., Bogden, R., et al. (2011). Sequence-based physical mapping of complex genomes by whole genome profiling. *Genome research*, 21(4):618–625.

[Venter et al., 2001] Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., et al. (2001). The sequence of the human genome. *science*, 291(5507):1304–1351.

[Wallis et al., 2004] Wallis, J. W., Aerts, J., Groenen, M. A., Crooijmans, R. P., Layman, D., Graves, T. A., Scheer, D. E., Kremitzki, C., Fedele, M. J., Mudd, N. K., et al. (2004). A physical map of the chicken genome. *Nature*, 432(7018):761–764.

[Welsh et al., 2009] Welsh, M., Mangravite, L., Medina, M. W., Tantisira, K., Zhang, W., Huang, R. S., McLeod, H., and Dolan, M. E. (2009). Pharmacogenomic discovery using cell-based models. *Pharmacological reviews*, 61(4):413–429.

[Wheeler and Dolan, 2012] Wheeler, H. E. and Dolan, M. E. (2012). Lymphoblastoid cell lines in pharmacogenomic discovery and clinical translation. *Pharmacogenomics*, 13(1):55–70.

[Williams, 1995] Williams, P. M. (1995). Bayesian regularization and pruning using a laplace prior. *Neural computation*, 7(1):117–143.

[Wipf and Nagarajan, 2007] Wipf, D. P. and Nagarajan, S. S. (2007). A new view of automatic relevance determination. In *Advances in Neural Information Processing Systems*, pages 1625–1632.

[Wooster et al., 1995] Wooster, R., Bignell, G., Lancaster, J., Swift, S., Seal, S., Mangion, J., Collins, N., Gregory, S., Gumbs, C., Micklem, G., et al. (1995). Identification of the breast cancer susceptibility gene BRCA2. *Nature*, 378(6559):789–792.

[Zhang, 2014] Zhang, Y. (2014). Academic website for Yu Zhang. *http://sites.stat.psu.edu/ yuzhang/*.

[Zhang and Liu, 2007] Zhang, Y. and Liu, J. S. (2007). Bayesian inference of epistatic interactions in case-control studies. *Nature genetics*, 39(9):1167–1173.

[Zimin et al., 2009] Zimin, A. V., Delcher, A. L., Florea, L., Kelley, D. R., Schatz, M. C., Puiu, D., Hanrahan, F., Pertea, G., Van Tassell, C. P., Sonstegard, T. S., et al. (2009). A whole-genome assembly of the domestic cow, bos taurus. *Genome Biol*, 10(4):R42.

[Zuk et al., 2012] Zuk, O., Hechter, E., Sunyaev, S. R., and Lander, E. S. (2012). The mystery of missing heritability: Genetic interactions create phantom heritability. *Proceedings of the National Academy of Sciences*, 109(4):1193–1198.

[Zuk et al., 2014] Zuk, O., Schaffner, S. F., Samocha, K., Do, R., Hechter, E., Kathiresan, S., Daly, M. J., Neale, B. M., Sunyaev, S. R., and Lander, E. S. (2014). Searching for missing heritability: Designing rare variant association studies. *Proceedings of the National Academy of Sciences*, 111(4):E455–E464.