2007

Bayesian Neural Networks for Classification

Skyler Richard Saucedo

THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

BAYESIAN NEURAL NETWORKS FOR CLASSIFICATION

By

SKYLER RICHARD SAUCEDO

A Thesis submitted to the
Department of Physics
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Fall Semester, 2007

The members of the Committee approve the Thesis of Skyler Richard Saucedo defended on November 10, 2006.

Harrison B. Prosper
Professor Directing Thesis

Todd Adams
Committee Member

Vasken Hagopian
Committee Member

Approved:

David Van Winkle, Chair
Department of Physics

Joseph Travis, Dean, College of Arts and Sciences

The Office of Graduate Studies has verified and approved the above named committee members.

ACKNOWLEDGEMENTS

I would like to acknowledge my sincere gratitude to Dr. Harrison B. Prosper, who not only was a tremendous help in aiding my understanding of the subject matter, but also in reminding me that indeed, Bob is my uncle.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

We study a model for classification based upon Bayesian statistics. The model, called Bayesian Neural Networks (BNN), is based on a function that is a weighted sum of hyperbolic tangent functions. The goal of the BNN is to approximate posterior class-conditional probabilities. To illustrate this method, we apply it to the task of separating Supersymmetric (SUSY) events from Standard Model proton-proton events at the LHC. Unlike conventional Neural Networks, the BNN model is an average over networks, which is done by integrating over a high dimensional parameter space. Since integrating over the parameter space is analytically impossible, the BNN method uses Hybrid Markov Chain Monte Carlo techniques to sample the desired probability densities while preserving a high acceptance rate. In this thesis we study the correlation properties of a sequence of Neural Networks. The results of this study are of great importantance because they validate the strategy currently used by the D0 collaboration, in the search for single top quarks at Fermilab.

CHAPTER 1

INTRODUCTION

1.1 Classification

Classification, specifically our ability to organize a collection of objects based on recognizable characteristics or patterns, is one of the most important daily processes we perform as sentient beings [1]. Within the scientific community, techniques of pattern recognition encompass a wide range of information processing problems of great practical significance [2], [3], [4], [5] . While humans perform such day-to-day classification procedures effortlessly and as a matter of reflex, these processes become immensely difficult to perform when using computational techniques [6] . Additionally, almost all living beings inherently contain some sort of trained classification system [7], or learned techniques of categorizing patterns (whose methods are outside the scope of this thesis). The focus of this thesis is to understand certain correlation properties of a particular classification model, namely, the model of Bayesian Neural Networks (BNN). The results are relevant to the ongoing search, by the D0 collaboration, for single top quarks at Fermilab. [8]

We can assume that, whichever classification method is used to solve a problem, it can be optimized if the algorithm in question is trained using prior knowledge. However complex the classification procedure, the probability of misclassification, in general, cannot be made arbitrarily small. Moreover, if there is too little training, the probability of correctly identifying a pattern from a given set of data will be low, whereas if there is too much training, we may find a classification procedure that works well on the training data but poorly on other data [9].

We focus on the canonical problem of binary classification, specifically classification into signal and background classes. The goal is to create a model that discriminates a signal event from a background event, in the context of high energy physics (HEP). We begin with a few key elements of classification theory. The model is illustrated using a simple example from HEP. We address the issue of overtraining and how using a Bayesian Neural Network as a classifier mitigates this. We then demonstrate the power of this classifier by applying it to the task of separating supersymmetric (SUSY) from Standard Model proton-proton events at the Large Hadron Collider (LHC). The LHC is a proton-proton particle accelerator under construction at CERN and scheduled to begin operation late 2007.[10]

1.2 Classification Theory

Suppose we characterize the object we wish to classify by a data vector $\vec{x} = (x_1, ..., x_P)$, where P is the number of characteristics, or variables. For a given set of N objects, we wish to seperate these objects into two classes, C_k, where k labels the classes. Since our goal is to classify an object described by $\vec{x}$, the output of our classification function should, ideally, be a probability bounded in the range [0,1]. Additionally, we introduce a set of prior probabilities $P(C_k)$ representing the fraction of N objects of the form $\vec{x}$ of the class C_k . For the case of binary classification, C_k is comprised of the signal class C_S and the background class C_B, such that $P(C_S) + P(C_B) = 1$.

We define the *joint* probability $P(C_k, \vec{x})$ as the probability that an object has characteristics $\vec{x}$ and belongs to class C_k. We can also define the *conditional* probability $P(C_k|\vec{x})$ as the probability that an object described by $\vec{x}$ belongs to class C_k. The prior, joint, and conditional probabilities are related as followed: $P(C_k, \vec{x}) = P(C_k|\vec{x})P(\vec{x})$. Similarly, the joint probability $P(\vec{x}, C_k) = P(\vec{x}|C_k)P(C_k)$ is another way of describing the same union of the object's characteristics and class set C_k. $P(\vec{x}|C_k)$ is the conditional probability that an object has the form $\vec{x}$, given that it is of class C_k. Since the two joint probabilities are equal

to each other, we can write

$$P(C_k|\vec{x}) = \frac{P(\vec{x}|C_k)P(C_k)}{P(\vec{x})}. \tag{1.1}$$

Equation (1.1) is *Bayes' theorem*, initially derived by Reverend Thomas Bayes and published posthumously in 1763. Eq. (1.1) represents the desired output of our classification function. For binary classification, we note that

$$P(\vec{x}) = P(\vec{x}|C_S)P(C_S) + P(\vec{x}|C_B)P(C_B). \tag{1.2}$$

Equation (1.2) holds because $\vec{x}$ must be assigned either to the signal or background classes and therefore,

$$P(C_S|\vec{x}) + P(C_B|\vec{x}) = 1. \tag{1.3}$$

Substituting Eq.(1.2) into Eq.(1.1) yields,

$$P(C_S|\vec{x}) = \frac{P(\vec{x}|C_S)P(C_S)}{P(\vec{x}|C_S)P(C_S) + P(\vec{x}|C_B)P(C_B)}. \tag{1.4}$$

Dividing both numerator and denominator by the expression $P(\vec{x}|C_S)P(C_S)$, we can re-express Eq.(1.4) in terms of the *logistic* function

$$P(C_S|\vec{x}) = \frac{1}{1+e^{-f}}, \tag{1.5}$$

where,

$$f = \ln \frac{P(\vec{x}|C_B)P(C_B)}{P(\vec{x}|C_S)P(C_S)}. \tag{1.6}$$

The function f is the natural logarithm of the *Bayes discriminant*. Equation (1.5) allow us to describe the output of the Bayes' discriminant as an isomorphic mapping from the natural log ratio of class-conditional and prior probabilities, to a number representing the posterior probability threshold within [0,1]. Take note that the most efficient algorithm, that is, the one with the smallest misclassification rate, is said to have reached the Bayes limit if it can accurately model Eq.(1.6). No classifier, however sophisticated, can do better than the Bayes limit [11].

The goal is to construct a function $n\,(\vec{x}, \vec{\omega})$ with parameters $\vec{\omega}$, that approximates Eq.(1.4). Figure (1.1) is an illustration of how the training dynamics of a classifier work

for a signal object represented by the label 1 and a background object by the label 0. The figure plots the distribution of the value of the function $n(\vec{x}, \vec{\omega})$ as a function of training epoch. For early training epochs the classification function has learned little from the training data, so it discriminates with an accuracy a little better than a random guess. As training proceeds, the function improves in its ability to discriminate between the two classes, and we observe a bifurcation of the training data. Eventually, if one trains long enough, the function will become so tightly fit to the training data that we obtain delta spikes at zero and one. Our overtrained classifier will then perform very poorly on data not within the training sample.

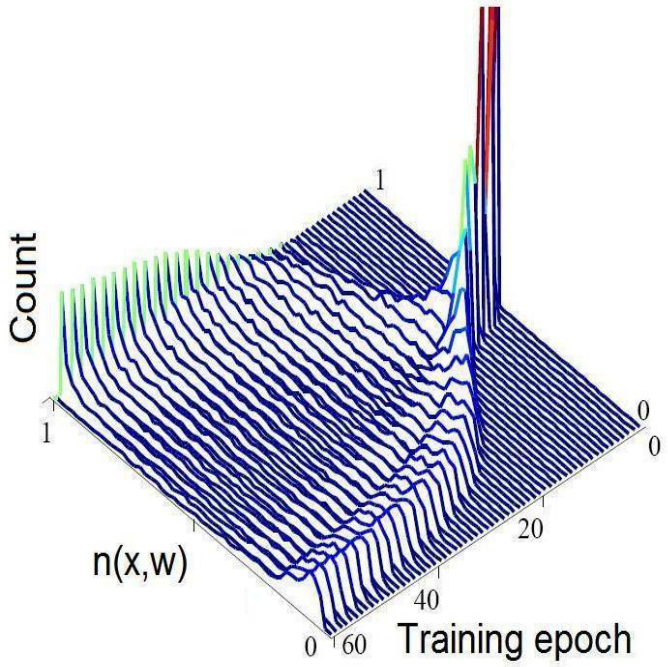

Figure 1.1: Training dynamics of a classifier. With successive training epochs, the classification function improves its ability to discriminate a signal event from a background event.

1.3 Minimizing Misclassification

An optimal classifier $n(\vec{x}, \vec{\omega})$ is one that minimizes the rate of misclassification. The function $n(\vec{x}, \vec{\omega})$ contains parameters, sometimes called weights, that are to be found by finding the best match between $n(\vec{x}, \vec{\omega})$ and $P(C_S|\vec{x})$. To that end, define the *empirical risk function*:

$$E(\vec{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \{t_i - n_i(\vec{x}, \vec{\omega})\}^2, \tag{1.7}$$

which is to be minimized with respect to $\vec{\omega}$. In the case of binary classification, the *target* $t = 1$ for signal while $t = 0$ for *background*. In order to find the minimum, we consider the empirical risk function to be an approximation to

$$E(\vec{\omega}) = \int \int (t - n(\vec{x}, \vec{\omega}))^2 p(t, \vec{x}) dt d\vec{x}. \tag{1.8}$$

One can show [12] that $E(\vec{\omega})$ is minimized when,

$$n(\vec{x}, \vec{\omega}) = \int t p(t|\vec{x}) dt. \tag{1.9}$$

For binary classification, we can write

$$p(t|\vec{x}) = \left\{ \begin{array}{l} \delta(t-1) P(C_S|\vec{x}) \\ \delta(t-0) P(C_B|\vec{x}) \end{array} \right\}, \tag{1.10}$$

which yields the desired result:

$$n(\vec{x}, \vec{\omega}) = P(C_s|\vec{x}). \tag{1.11}$$

This result holds, at least approximately, if the function $n(\vec{x}, \vec{\omega})$ is sufficiently flexible and if we have sufficient training data.

1.4 Neural Networks

Our goal is to fit training data to a function $f(\vec{x},\vec{\omega})$, which can be expressed as a sum of functions with parameters, or weights $\vec{\omega}$. In the case of one-dimensional polynomial fitting, we can express $f(\vec{x},\vec{\omega})$ as a sum of J^{th}-order polynomials and coefficients of the following form:

$$f(\vec{x},\vec{\omega}) = \omega_0 + \omega_1 x + \omega_2 x^2 + ... + \omega_{J-1} x^{J-1} + \omega_J x^J = \sum_{j=0}^{J} \omega_j x^j. \tag{1.12}$$

For higher dimensions, we need a more complex function $f(\vec{x},\vec{\omega})$, capable of modeling the posterior probability, Eq.(1.6), for a general binary classification. Again, the classifier should learn from training data using successive training epochs. Within the machine learning literature, one such class of functions is the multi-layer *perceptron*, or Neural Network (NN) [13]. We use the explicit functional form

$$f(\vec{x},\omega) = a + \sum_{j=1}^{H} b_j \tanh(c_j + \sum_{i=1}^{P} d_{ji} x_i), \tag{1.13}$$

for the function f in Eq.(1.6). Our NN function is a logistic function whose exponential argument $f(\vec{x},\vec{\omega})$ is a superposition of hyperbolic tangent functions with free parameters, or weights, $\omega(a, b_j, c_j, d_{ij})$. Weights c_j and a are called *biases*, and correspond to the $i = 0$ and $j = 0$ summation terms, respectively. Figure (1.2) is a graphical representation of $f(\vec{x},\vec{\omega})$, Eq.(1.13), its weights and its H hidden nodes.

One of the potential difficulties with the NN model is the possibility of a too restrictive mapping due to the use of a specific set of network parameters $\vec{\omega_0}$ found via training. This can arise because our minimization procedure for $E(\vec{\omega})$ is a gradient descent within a parameter space of $HP + 2H + 1$ dimensions and $2^H H!$ equivalent points. Therefore, the parameter space $\vec{\omega}$ is a very complex landscape riddled with corrugated peaks and valleys. The high dimensionality of the parameter space presents a severe optimization challenge for any gradient descent minimization algorithm.

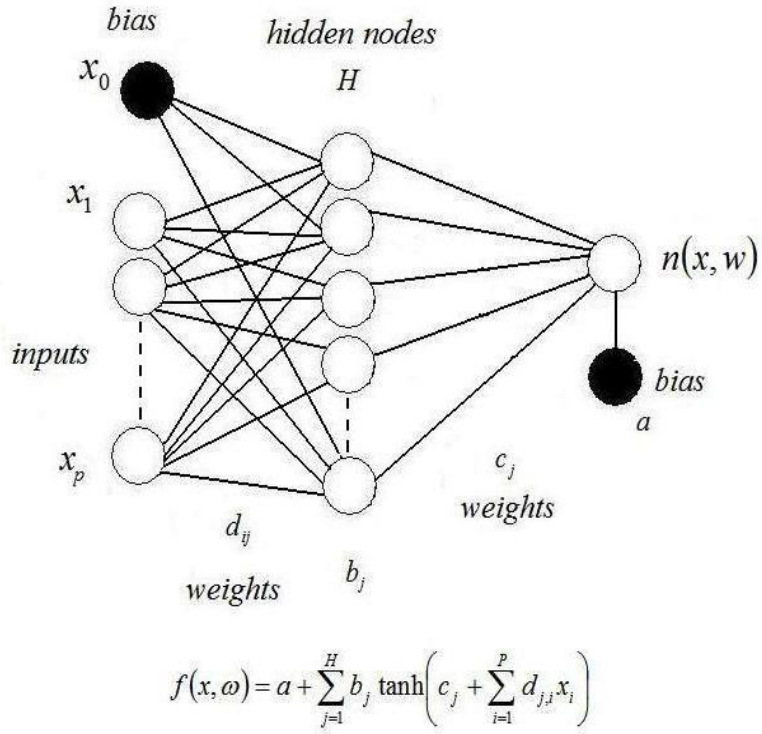

Figure 1.2: A Neural Network $n(\vec{x}, \vec{\omega})$ for binary classification.

For an infinite number of training objects, our function, theoretically, could find the best set of parameters via gradient descent. However, we are faced with the reality of having only a finite set of objects and the possibility of finding a minimum of $E(\vec{\omega})$ that may perform poorly on data not in the training sample. To mitigate this, we consider a *Bayesian* approach to modeling our classifier.

1.5 Bayesian Neural Networks

The conventional NN training procedure yields one set of weights, or network parameters $\vec{\omega_0}$, which correspond to a mapping function $n(\vec{x}, \vec{\omega_0})$. This procedure is prone to over-training. Fortunately, one can overcome this by averaging over the entire $\vec{\omega}$-space (in practice, over a sample from this space),

$$\overline{n(\vec{x}'|t, \vec{x})} = \int n(\vec{x}', \vec{\omega}) p(\vec{\omega}|t, \vec{x}) d\vec{\omega}, \tag{1.14}$$

provided that an appropriate probability density $p(\vec{\omega}|t, \vec{x})$ can be assigned to each point $\vec{\omega}$ in that space.

A Bayesian Neural Network (BNN) is the result of $p(\vec{\omega}|t, \vec{x})$ integrating over the entire $\vec{\omega}$ -space. The BNN method will produce $\overline{n}$, a function that is a more stable and robust estimate of $P(C_S|\vec{x})$ that is less likely to be overtrained. The averaged network effectively reduces the tendency to over-fit data by automatically assigning a lower probability density to weight vectors $\vec{\omega}$ corresponding to unnecessarily large networks.

It should be mentioned that the number of weights need not be lower than the number of training data! This is because we are dealing with *nonlinear* functional fitting for which the rules of linear fitting do not apply. This is important because we want the dimensionality of our network parameter space to be as large as possible so that the space of functions, defined by some $\vec{\omega_0}$, include networks that are good approximations to the true mapping. Unfortunately, the dimensionality of $\vec{\omega}$-space can get extremely large, and so integrating over the parameter space is analytically impossible. Fortunately, we can use numerical methods to approximate integrals, specifically:

$$\int n(\vec{x'}, \vec{\omega}) p(\vec{\omega}|t, \vec{x}) d\vec{\omega} \approx \frac{1}{K} \sum_{k-1}^{K} y(\vec{x'}, \vec{\omega_k}), \tag{1.15}$$

where K is the number of points $\vec{\omega}$ sampled from $p(\vec{\omega}|t, \vec{x})$. In order to compute the average network function in Eq.(1.14) from the posterior density, we sample the posterior density using a Markov Chain Monte Carlo method.

1.6 Markov Chain Monte Carlo and Hybrid MCMC

Markov Chain Monte Carlo (MCMC) simulations use a Markov process to generate a sequence of "states" to model a probability density. The MCMC method treats the sampling of $p(\vec{\omega}|t,\vec{x})$ as a problem in statistical mechanics. In our case, the states are identified by points in the $\vec{\omega}$-space. In particular, the chain converges to the density $p(\vec{\omega}|t,\vec{x})$.

The Metropolis algorithm [14] is the classic MCMC model, where the transition probabilities from one state to another are ergodic [15] and obey detailed balance conditions

$$P(\vec{x})P(\vec{x}\to\vec{x}') = P(\vec{x'})P(\vec{x}'\to\vec{x}), \tag{1.16}$$

where $P(\vec{x})$ is the probability of the particle being in state $\vec{x}$ and $P(\vec{x}\to\vec{x}')$ is the transition probability from state $\vec{x}$ to $\vec{x}'$. Equation (1.16) simply states that the transition from state $\vec{x}$ to $\vec{x}'$, is equal to the transition from state $\vec{x}'$ to $\vec{x}$. For our purpose, we use a variation of the Metropolis algorithm in which a "particle" steps through the *network* parameter space in such a way that the points $\vec{\omega}$ are visited with a probability given by the posterior density $p(\vec{\omega}|t,\vec{x})$ [16]. We employ the Hybrid Markov Chain Monte Carlo algorithm (HMCMC) of Duane, Kennedy, Pendleton, and Roweth [17], which combines the Metropolis algorithm with the deterministic dynamics of a single particle under the influence of a complicated potential, described by

$$V(q) = -\ln p(\vec{\omega}|t,\vec{x}), \tag{1.17}$$

where the 'position' coordinates of the particle is $q \equiv \vec{\omega}$. Within Hamiltonian dynamics, $H = T + V$, where the "kinetic energy" of our particle can be described as $T(\vec{p}) = \frac{1}{2}\vec{p}^2$, with $\vec{p}$ a vector having one component for each dimension of the network parameter space. The particle "mass", along with the "Boltzmann constant k_B" , can be re-scaled to unity.

Like the Metropolis algorithm, the Hybrid MCMC algorithm ensures (at least for sufficiently long chains) that the density of sampled points is proportional to $\exp(-H)$. Conventional MC calculations create randomizations at each Markov chain step, thereby generating a pure random walk with, typically, low state change acceptance rates. On the other hand, because Hybrid MCMC calculations are partly deterministic and partly random, one can usually explore the state space more efficiently.

In our study, the Hybrid MCMC algorithm uses two types of transition: the 'deterministic' moves are called *microsteps*, and are used to sample states which H is constant; thus visited with equal probabilities. The 'stochastic' moves are called *macrosteps*. These moves change the value of the Hamiltonian with explore states with probabilities proportional to $\exp(-H)$ [18]. An illustration of this can be seen in Fig.(1.3). We thus create a Markov

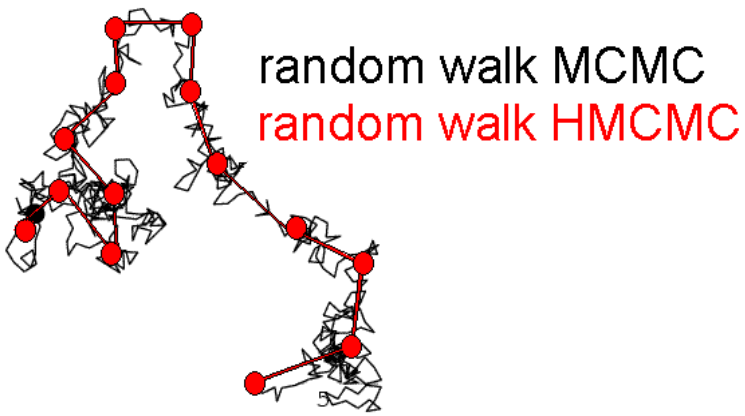

Figure 1.3: MCMC vs. HMCMC step size.

chain of networks $(\vec{\omega_1}, \vec{\omega_2}, ..., \vec{\omega_N})$ which converges to a sequence of points that constitute a sample from the density we wish to model, namely $p(\vec{\omega}|t, \vec{x})$. This method typically reduces the computation time needed to converge towards the desired probability density since it eliminates much of the random walk of the Metropolis algorithm. Since the correlation between each microstep is very high ~ 0.9, we save a point within the Markov chain (which in our case corresponds to a network) after every M macrosteps, so that we see low correlation between each saved step.

Unfortunately, a priori we are not sure how large M should be, nor how many macrosteps are needed to reach convergence. In the work of D0, typically, $M \approx 20$ and satisfactory performance is seen. But, we wish to understand quantitatively the correlation amongst the macrosteps. A good method of testing how much *non-randomness* occurs within our macrosteps is to study their *Autocorrelation* measurements upon the Markov chain.

CHAPTER 2

STUDIES

This chapter describes the studies undertaken to understand the correlation of a sequence of neural networks generated by the Hybrid MCMC method. But first, to illustrate the classification theory described in the previous chapter, we apply conventional neural network methods to a one dimensional event discrimination problem from high energy physics. We then extend the example to a Bayesian model with five inputs. We show that using a BNN classifier significantly improves the signal to noise ratio. Finally, we study the correlation properties of our BNN via autocorrelation calculations. From our findings, we propose a more robust Bayesian strategy to be used at the LHC in 2007 and beyond.

2.1 A 1-D example

In this example, we analyze a simulated data set comprised of 5,000 Supersymmetric (SUSY) and Standard Model (SM) events in equal numbers. The specific model used is called minimal supergravity (mSUGRA) [19]. We discriminate between the two classes of the events using a single variable, the missing effective energy, comprised of the missing transverse energy plus the transverse momentum of jets, or $M_{Eff} = M_{E_T} + \sum_{j=1}^{alljets} P_{T_j}$. If we train a neural network with equal numbers of signal and background events, the network function should model

$$P(C_S|x) = \frac{p(x|C_S)}{p(x|C_S) + p(x|C_B)}. \tag{2.1}$$

Using the probability densities illustrated in Fig.(2.1), we construct a neural network function $n(x, \vec{\omega})$ of the form given Eq. (1.5) with $f(x, \vec{\omega})$ given by Eq.(1.13). We use a feed-forward neural network based on JETNET 3.0 [20] using 1 input node, 10 hidden nodes, and 1 output node. The neural network undergoes 1,000 training epochs corresponding to the

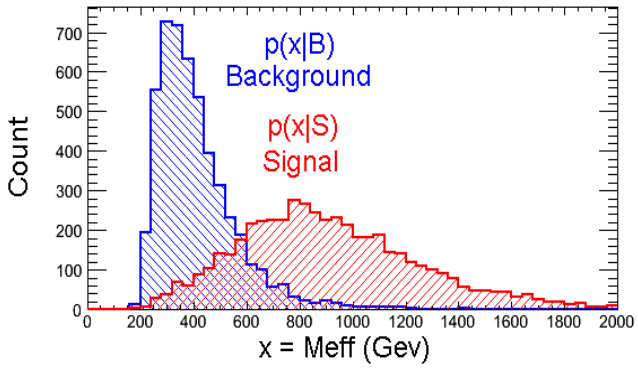

Figure 2.1: Probability densities used to generate class-conditional probabilities for the one dimensional example.

gradient of the error function, Eq.(1.7). Since $x = M_{Eff}$ is one dimensional, we can directly approximate $P(S|x)$ by computing (bin by bin) the following ratio

$$P(C_S|x) = \frac{H(x,S)}{H(x,S) + H(x,B)}, \tag{2.2}$$

where $H(x,S)$ and $H(x,B)$ are the histograms of the signal and background densities, respectively. The result of this calculation is shown in Figure (2.2) as the black dots.

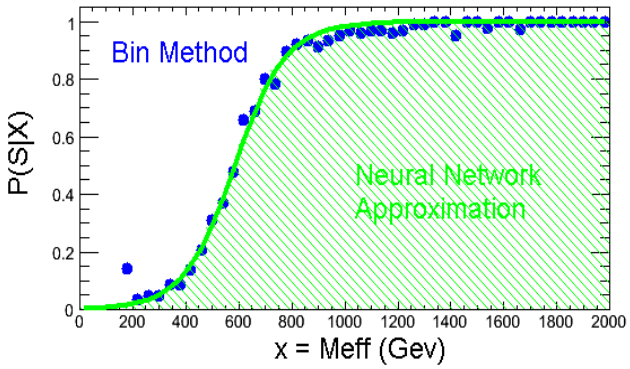

Figure 2.2: Approximating $P(S|x)$ for a one dimensional problem, using the histogram method described by Eq. (2.2) and a feed-forward neural network of 1 input node, 10 hidden nodes, and 1 output node. The points correspond to the histogram method of Eq.(2.2), while the solid curve corresponds to the 1-input variable JETNET neural network, trained by a gradient descent method called back-propagation.

Figure (2.2) illustrates how a neural network not only approximates $P(C_s|x)$ well, but also fits smoothly. However, $n(x,\omega)$ uses only one parameter point ω_0 from $\vec{\omega}$ -space. This particular ω_0 value is a local minimum of the risk function, Eq.(1.7). But, what if this 'best' network overfits the training data? And how can one quantify the quality of the fit of $n(x,\omega)$ to $P(S|x)$? It is precisely these questions (as well as others) that the Bayesian neural network method addresses.

2.2 SUSY study

It is well known that the average of many measured quantities is generally a more accurate and robust estimate of the true value, than is a single measurement. Therefore, the idea of averaging over networks is sensible, since the average $\overline{n(\vec{x'}|t,\vec{x})}$ is more likely to be a robust classifier than any individual $n(\vec{x},\vec{\omega})$. While a single parameter point ω_0 may represent a local minimum of the error function, Eq.(1.7), averaging over $n(\vec{x},\vec{\omega})$ corresponds to averaging over $\vec{\omega}$- space, effectively smoothing out corrugations within the error function. For this section we illustrate the power of a Bayesian approach for approximating $P(\vec{x}|\vec{\omega})$ via an averaged network function $\overline{n(\vec{x},\vec{\omega})}$.

For this work, we considered the separation of mSUGRA events, characterized by a five dimensional variable $\vec{x} = (M_{E_T}, \sum_{j=1}^{alljets} P_{T_j})$, from SM background events. The inputs for $\vec{x}$ are the missing transverse energy due to neutrinos and neutralinos and transverse momenta of the four most dominant transverse momentum jets. The signal events were simulated using ISAJET 7.69 [21] using the mSUGRA model [19]. For $\overline{n(\vec{x},\vec{\omega})}$, we used 5 input nodes, 25 hidden nodes, and 1 output node, with training samples of 5,000 simulated events (half mSUGRA, half SM). We used Radford Neal's Flexible Bayesian Modeling (FBM) software [22] to create a Markov chain of 1,000,000 MCMC microsteps. Figure (2.3) shows a tiny signal to background ratio for the missing E_T distribution, which is, about 1 to 10,000. Using a Bayesian Neural Network, one can significantly enhance this ratio to 1 to 100, as illustrated in Figure (2.4).

Figure (2.4) shows a signal to background ratio of on the order of 1 to 100 for BNN

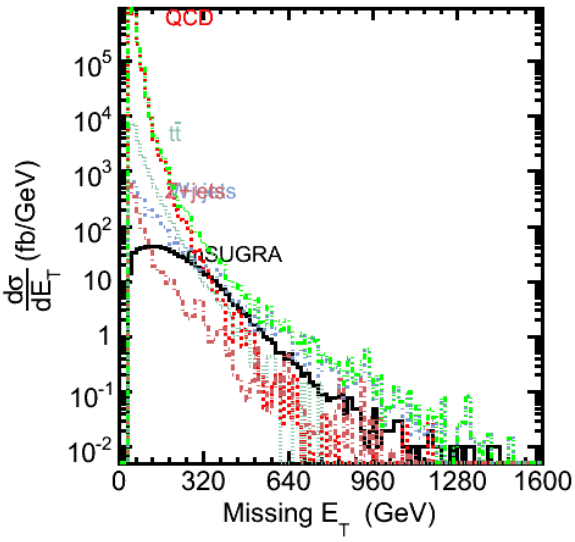

Figure 2.3: The energy spectra of the missing transverse energy. The black curve is the spectrum for mSUGRA events, while the other red curves are the spectra of SM (background) events. The disparity between the peaks shows the small signal to noise ratio that we wish to improve upon with Bayesian techniques.

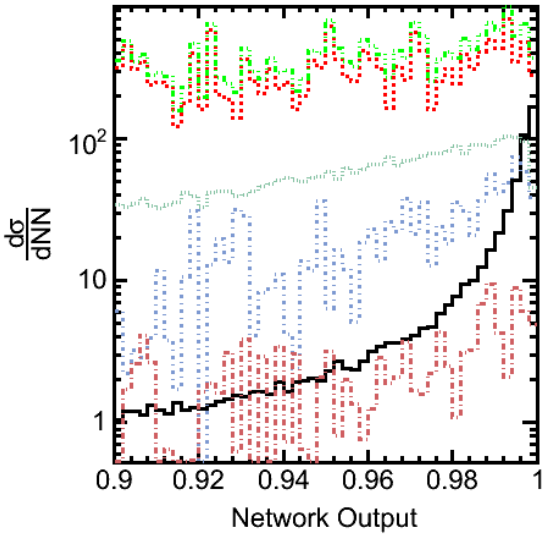

Figure 2.4: ME_T Spectra of the Bayesian neural network output for BNN output values > 0.9. The Bayesian technique for discrimination significantly enhances the signal to noise ratio to 1 to 100.

output values greater than 0.9, illustrating the power of a Bayesian classifier for signal-background discrimination. This method is currently being used by the D0 Collaboration at Fermilab and is to be used by CMS during the LHC era, soon to begin at CERN. As of now, the Bayesian classifier used by the D0 Single Top Group employs a HMCMC strategy that generates a Markov Chain in which networks are saved every 2,000 microsteps, that is, every 20 macrosteps. As noted above, averaging over MCMC points should yield the best approximation to $P(S|\vec{x})$, if these points are statistically independent. Therefore, our principle motivation is to ascertain whether or not the current strategy used by the d0 Single Top Group yields a sequence of networks with sufficiently low correlation.

2.3 Autocorrelations

The current strategy of the D0 group is to save networks every 2,000 MCMC steps. However, until the study reported here, it was not known whether this saving schedule was reasonable. Therefore, it is of some importance to see if this saving schedule produces strongly correlated networks, since averaging over highly correlated networks would limit the utility of a Bayesian approach. One way to characterize the 'randomness' of a Markov sequence is to perform autocorrelation calculations.

The autocorrelation is a measure of temporal correlations within a time series. Suppose, for a generated Markov chain, we wish to measure how points separated by some lag distance $d = |i - j|$, are correlated. We can use this quantity

$$A(d) = \frac{C_{i+d}}{C_i}, \tag{2.3}$$

where,

$$C_{i+d} =< (n_i - \bar{n})(n_{i+d} - \bar{n}) > . \tag{2.4}$$

For a lag of zero, the autocorrelation is unity. Equation (2.3) has the asymptotic behavior

$$A(d) \sim \exp(-\frac{d}{\tau}), \tag{2.5}$$

where τ is the correlation distance. In order to obtain a sequence of networks with low correlation, if is necessary to use a saving schedule with a lag much greater than the correlation distance τ.

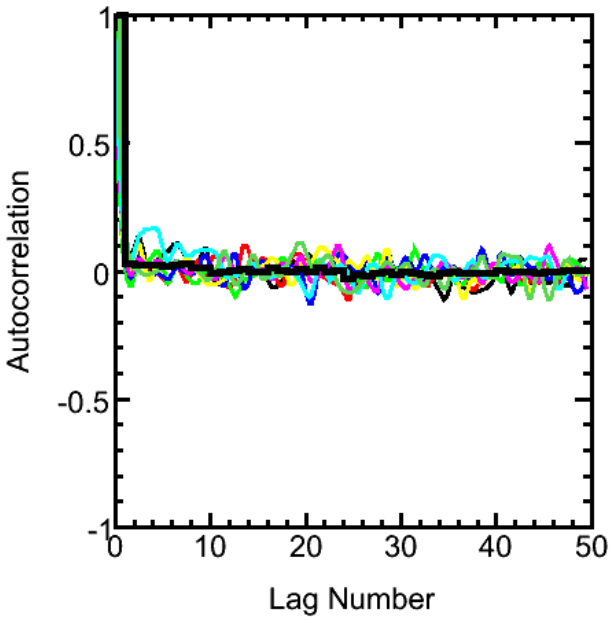

Figure 2.5: Autocorrelation behavior for $d = 2,000$ MCMC steps.

Following the D0 collaboration, we generate 1,000,000 Markov microsteps, constructed as follows: 20 repetitions of 100 MCMC microsteps followed by a randomization, repeated 500 times. We average over the last saved 100 networks and find a rapid drop in correlation, almost to zero, for lag distances greater than $d = 1$ corresponding to 2,000 saved microsteps, as seen in Fig.(2.5). We conclude, that indeed, the strategy currently used by the D0 Single Top Group yields networks with almost no correlation. We now ask ourselves the following: Could one save points more frequently? If one were to shorten the lag distance to, say 100 MCMC microsteps, would we get the same, if not better results?

Figure (2.6) suggests a possible new saving schedule, namely, saving every 200 MCMC microsteps. This proves to be just as good, if not better, than saving every 2,000 MCMC microsteps! We have therefore shown that the current strategy used by D0 is a conservative one, and that one could choose to save more often, thereby yielding more networks over which to average. The more saved networks over which one averages, the more robust the classifier.

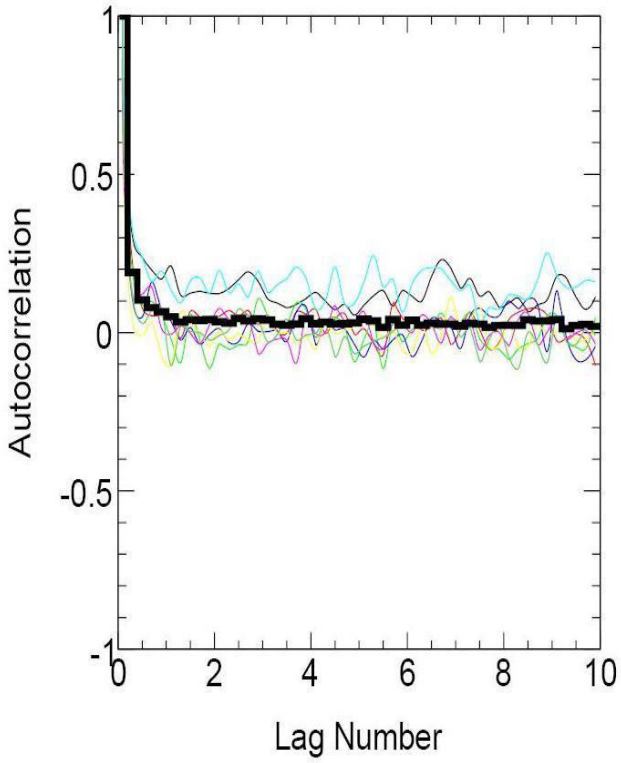

Figure 2.6: Autocorrelation behavior for $d = 100$ MCMC microsteps.

CHAPTER 3

CONCLUSION

The Bayesian Neural Network method averages over an ensemble of networks in order to find a more precise estimate of the posterior probability $P(S|\vec{x})$ than is typically available using conventional Neural Networks. In order for this to be useful, however, it is necessary to understand quantitatively the correlation properties of the numerical method used to integrate over the parameter space. In this thesis, we have shown that within a Markov chain of networks, saving a network every 2,000 MCMC steps is a very conservative strategy for creating an accurate classifier. In fact, an equally accurate algorithm which saves a network after every 200 MCMC steps performs just as well, as seen in the autocorrelation calculations we have reported. Finally, we suggest that saving a network more frequently could yield a more robust Bayesian classifier, because one would be averaging over more networks.

REFERENCES

[1] A. Turing. Computing machinery and intelligence. *Mind.*, 59(236):435–460, 1950. 1.1

[2] T. Kimotoand K. Asakkawa, M. Yoda, and M. Takeoka. Stock market prediction system with modular neural networks. *Neural Networks, IJCNN International Joint Conference*, pages 1–6, 1990. 1.1

[3] J. Khan, M. Ringer J.S. Wei, L.H. Saal, and M. Landanyi. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nat. Med.*, pages 673–679, 2001. 1.1

[4] S. Kononenko. Inductive and bayesian learning in medical diagnosis. *Applied Artificial Intelligence*, pages 317–337, 1993. 1.1

[5] C. O'Laughlin and P. Thagard. Autism and coherence: A computational model. *Mind & Language*, 15(4):375–392, 2000. 1.1

[6] A.B. Teoh, A. Goh, and N. Ngo. Random multispace quantization as an analytic mechanism for biohashing of biometric and random identity inputs. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(12):435–460, 2006. 1.1

[7] B. Nisbet. *Nutrition and feeding strategies in Protozoa.* Croom Helm Publishing Co., London, first edition, 1984. 1.1

[8] The D0 Collaboration. Search for single top quark production using likelihood discriminants at d0 in run ii. D0 note, Fermi National Accelerator Laboratory (Fermilab), 2005. 1.1

[9] G.H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. *Uncertainty in Artificial Intelligence*, pages 338–345, 1995. 1.1

[10] Cms timeline. Technical report, Compact Muon Solenoid at CERN, 2006. 1.1

[11] H.B. Prosper. Boosting: Or how to make a silk purse out of a sow's ear. Technical report, D0 Note, 2005. 1.2

[12] C.M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, second edition, 1998. 1.3

[13] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. 1.4

[14] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953. 1.6

[15] B.A. Berg. *Markov Chain Monte Carlo Simulations and Their Statistical Analysis (With Web-Based FORTRAN Code)*. World Scientific, London, first edition, 2004. 1.6

[16] B. Mehling, D. W. Heermann, and B.M. Forrest. Hybrid monte carlo method for condensed-matter systems. *Physical Review B*, 45(2):679–685, 1992. 1.6

[17] S. Duane, A.D. Kennedy, B.J. Pendleton, and D. Roweth. Hybrid markov chain monte carlo. *Physics Letters B*, 195:216–222, 1987. 1.6

[18] R. Neal. Probabilistic inference using markov chain monte carlo methods. Dept. of Computer Science Technical Report 140 pages, University of Toronto, 1993. 1.6

[19] S. Eidelman et al. Supersymmetric partilc searches. *Phys. Lett. B*, 592(1), 2004. 2.1, 2.2

[20] C. Peterson and T. Rognavaldsson. Jetnet - a versatile artifical neural network package cern-th.7135/94. *unpublished*, 1994. Program obtainable via anonymous ftp from thep.lu.se in directory pub/Jetnet/. 2.1

[21] F. Paiges and S. Protopopescu. Isajet 7.69. Signal generating software, 2002. 2.2

[22] R. Neal. Software for flexible bayesian modeling and markov chain sampling. Technical Report v12, University of Toronto, 2004. 2.2

BIOGRAPHICAL SKETCH

Skyler R. Saucedo

Research Interests

I am interested in using Bayesian statistics and machine-learned neural networks to analyze data sets with the intent of data mining, forecasting, and pattern recognition. My current work utilizes a multivariate statistical method for data classification, whose goal is to amplify a low signal to background fraction within a given statistical distribution. This particular method is currently being used by the D0 group at Fermilab and will be used by Physicists at CERN in Geneva, Switzerland. In the past I have used Finite Difference methods to model Non-Linear pattern forming systems.

Education

M.S. Physics Florida State University, Tallahassee, Florida (December 2006) Masters Thesis, "Bayesian Neural Networks for Classification"

B.S. Physics with Mathematics minor Trinity University, San Antonio, Texas (May 2005) Senior Thesis, "A Numerical Simulation of the Dynamics of the Nonlinear Ginzburg-Landau Equation."

Publications (arXiv.org search S.R. Saucedo)

1. Amplitude Modulation and Relaxation-Oscillation of Counterpropagating Rolls within a Broken Symmetry Electrocovection Strip. D. R. Spiegel, E. R. Johnson, S. R. Saucedo Physical Review E 73, 036317 (2006). http://arxiv.org/PS_cache/cond-mat/pdf/0604/0604676.pdf

2. Dynamics of Laser-Induced Electroconvection Pulses. N. C. Giebink, E. R. Johnson, S. R. Saucedo, E. W. Miles, K. K. Vardanyan, D. R. Spiegel, and C. C. Allen. Physical Review E69, 066303 (2004). http://arxiv.org/ftp/cond-mat/papers/0404/0404004.pdf