BAYESIAN TRAINING OF NEURAL NETWORKS USING GENETIC PROGRAMMING

Tshilidzi Marwala

University of the Witwatersrand, School of Electrical and Information Engineering, Private Bag 3, Wits, 2050, South Africa
Abstract

A Bayesian training of neural networks using Markov chain Monte Carlo (MCMC) method based on genetic programming is proposed. The sampling space is conducted in binary space. The Markov chain is generated by performing a crossover between the current state and the previous state and then mutating the off-spring of the product of this crossover and reproducing the resulting state via Metropolis et al. criterion. The proposed method is tested on a numerically simulated data and is compared to classical MCMC method, which operates in floating-point space. The proposed method is observed to give better results than the network trained using traditional MCMC that operates in floating-point space, with regards to convergence to a stationary posterior distribution and network errors achieved.
1. INTRODUCTION

The use of Bayesian framework to train neural networks has been the subject of research over the last decade. Some of the techniques that have been applied thus far to train neural networks using Bayesian framework include Markov chain Monte Carlo (MCMC) method [1] and the hybrid Monte Carlo method [2]. Both these methods have been applied within the framework of Metropolis et al. algorithm [3]. Markov chain Monte Carlo method has been applied to improve the abilities of mathematical models to predict the dynamics and reliability of structures [4]. Punskaya et. al. [5] used MCMC for Bayesian curve fitting and applied that to signal segmentation, while Jalobeanu et al. [6] used MCMC to estimate regularisation parameters for satellite image restoration. Chib et al. [7] used Markov chain Monte Carlo methods for the inference of stochastic volatility models and tested the methodology on the S&P index.

Markov chain Monte Carlo method is essential because it has been used in diverse areas such as engineering and economics. All these applications that have been described above have one aspect in common, and this is that they have been applied without paying particular attention to the issue of achieving a global optimum posterior distribution function. Kendall and Montana [8] have noted that inside every Markov chain with measurable transition density there is a discrete state-space Markov chain struggling to escape from some local optimum distribution. This in essence indicates that the issue of global posterior distribution must not be taken for granted.

Several techniques have been implemented to achieve global optimum distributions such as simulated annealing [9]. There are other procedures such as evolutionary computing that have been proven to be able to identify global solutions in optimisation problems, and these include genetic algorithms (GA) [10]. This paper, therefore, proposes the use of genetic programming, of which GA is an example, to sample a posterior probability distribution of the network weights in the light of the data. The procedure proposed for Bayesian sampling, in this paper, operates in binary space and conventional evolutionary concepts of mutation, crossover and reproduction are applied. This procedure is then compared to the traditional Markov chain Monte Carlo method that samples states in floating-point space, by regressing a simulated function, which has an added noise.

2. NEURAL NETWORKS

This section describes neural networks, which are computational tools that are applied to simulate human brain. In this paper, neural networks are viewed as parameterised graphs that make
probabilistic assumptions about data. Learning algorithms are viewed as methods for finding parameter values that look probable in the light of the data.

In this paper, the multi-layer perceptron (MLP) architecture containing a hyperbolic tangent basis function in the hidden units and linear basis functions in the output units [11] are considered. A schematic illustration of the MLP is shown in Figure 1. This network architecture contains hidden units and output units and has one hidden layer. The bias parameters in the first layer are shown as weights from an extra input having a fixed value of $x_0=1$. The bias parameters in the second layer are shown as weights from an extra hidden unit, with the activation fixed at $z_0=1$. The relationship between output $y$ and input $x$, may be written as follows [11]:

$$y = f_{\text{outer}} \left( \sum_{j=1}^{M} w_{kj}^{(2)} f_{\text{inner}} \left( \sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

Here, $w_{ji}^{(1)}$ and $w_{j0}^{(1)}$ indicate weights in the first and second layers, respectively, going from input $i$ to hidden unit $j$ while $w_{j0}^{(1)}$ indicates the bias for the hidden unit $j$. Here $M$ is the number of hidden units, $d$ is the number of input units and $k$ is the index for the output units. In this paper, the function $f_{\text{outer}}(\cdot)$ is linear, while $f_{\text{inner}}(\cdot)$ is a hyperbolic tangent function. The Bayesian method identifies the distribution of weights in equation 1 that look probable in the light of data.

3. BAYESIAN METHOD

In this section a method of identifying the network weights described in equation 1 is outlined. The problem of identifying the network weights ($w_i$) and biases (with subscripts 0 in Figure 1) in the hidden layers is posed in Bayesian form as follows [11]:

$$P(w \mid D, x) = \frac{P(D \mid w, x)P(w \mid x)}{P(D \mid x)}$$

where $P(w \mid x)$ is the probability distribution function of the weight-space in the absence of any data, also known as the prior distribution function, and $D \equiv (y_1, \ldots, y_N)$ is a matrix containing the output data. The quantity $P(w \mid D, x)$ is the posterior probability distribution function after the data have been seen, $P(D \mid w, x)$ is the likelihood distribution function and $P(D \mid x)$ is the evidence and its function is to normalize the posterior probability distribution function. The MLP network trained by supervised learning does not model the distribution of the input data. This $x$ is a conditioning variable that always appears on the right-hand side of the probabilities. For the remaining part of this paper, $x$ is omitted to simplify the notation. Equation 2 may be expanded to give [11]:

4
\[ P(w \mid D) = \frac{1}{Z_s} \exp \left( -\beta \sum_n \sum_k (v_{nk} - y_{nk})^2 - \frac{\alpha}{2} \sum_j w_j^2 \right) \]  

(3)

where

\[ Z_s(\alpha, \beta) = \int \exp \left( -\beta \sum_n \sum_k (v_{nk} - y_{nk})^2 - \frac{\alpha}{2} \sum_j w_j^2 \right) dw \]

\[ = \left( \frac{2\pi}{\beta} \right)^{\frac{Nk}{2}} + \left( \frac{2\pi}{\alpha} \right)^{\frac{Nj}{2}} \]  

(4)

In equation 3, the first term in the exponent is the likelihood function and the second term is the prior information, \( n \) is the index for the training pattern, \( \beta \) is the data contribution to the error, \( k \) is the index for the output units and \( \alpha \) is the coefficient of the prior information. The second term in equation 3 may be viewed as the regularisation parameter and it penalises weights of large magnitudes [11]. The distribution in equation 3 is a canonical distribution [11]. Training the network using Bayesian approach automatically penalises highly complex models and is therefore able to select an optimal model without applying independent methods such as cross-validation and also gives a probability distribution of the output of the networks.

4. MARKOV CHAIN MONTE CARLO SIMULATION VIA METROPOLIS ALGORITHM

The application of Bayesian approach to neural networks, results in the probability distribution functions of the network outputs. From these distribution functions the average prediction of the neural network and the variance of that prediction can be calculated. The probability distributions of these network-weights are mathematically described by equation 3. From equation 3 and by following the rules of probability theory, the distribution of the output parameter, \( y \), is written as [2]:

\[ p(y \mid x, D) = \int p(y \mid x, w)p(w \mid D)dw \]  

(5)

Equation 5 depends on equation 3, and is difficult to solve analytically due to relatively high dimension of weight space. Thus the integral in equation 5 may be approximated as follows:

\[ \tilde{y} \approx \frac{1}{R+L-1} \sum_{i=I}^{R+L-1} F(w_i) \]  

(6)

Here \( F \) is the mathematical model that gives the output given the input, \( \tilde{y} \) is the average prediction of the Bayesian neural network, \( R \) is the number of initial states that are discarded in the hope of reaching a stationary posterior distribution function described in equation 3 and \( L \) is the number of retained states. Several methods have been proposed to simulate the distribution in equation 3 such
as Gibbs sampling [9], Metropolis et al. algorithm [3] and hybrid Monte Carlo method [12]. In this paper, MCMC method is implemented by sampling a stochastic process consisting of random variables \( \{w_1, w_2, \ldots, w_n\} \) through introducing random changes to weight vector \( \{w\} \) and either accepting or rejecting the state according to Metropolis et al. algorithm [3]:

In this paper we view this procedure as a way of generating a Markov chain with the transition from one state to another conducted using the Metropolis et al. algorithm. Metropolis et al. algorithm [3] has been used extensively to solve problems of statistical mechanics. In traditional Metropolis et al. algorithm, on sampling a stochastic process \( \{w_1, w_2, \ldots, w_n\} \) consisting of random variables, random changes to \( w_i \) are considered and are either accepted or rejected according to the following criterion:

\[
\text{if } P_{\text{new}} > P_{\text{old}} \text{ accept state } (w_{\text{new}}, P_{\text{new}}) \\
\text{else accept } (w_{\text{new}}, P_{\text{new}}) \text{ with probability } \exp\left(\frac{P_{\text{old}} - P_{\text{new}}}{\text{temp}}\right) \tag{7}
\]

From equation 7, it is observed that states with high probability form the majority of the Markov chain, and those with low probability form the minority of the Markov chain. Traditionally, the MCMC was conducted in floating-point space and this paper introduces genetic sampling of Bayesian networks, which is the subject of the next section.

5. MCMC USING GENETIC PROGRAMMING AND METROPOLIS ALGORITHM

Genetic programming takes features from natural evolution and uses these to computationally solve practical problems. Genetic algorithms are examples of genetic programming and a procedure that is inspired by these is introduced in this section. In this paper, some of the features of genetic computing are applied to sample the posterior distribution function in equation 3.

Genetic algorithms were inspired by Darwin’s theory of natural evolution. In natural evolution, members of the population compete with each other to survive and reproduce. Evolutionary successful individuals reproduce while weaker members die. As a result, the genes that are successful are likely going to spread within the population. This natural optimisation method has been successfully used to optimise complex problems [10].

In this paper features from genetic algorithm suggested by Holland are implemented [13]. This genetic algorithm uses a population of binary-string chromosomes [14]. Each of these strings is the discretised representation of a point in the search space and therefore has a fitness function that is
given by the objective function. On generating a new population three operators are performed: (1) crossover; (2) mutation; (3) and reproduction, and these operators are adopted in genetic MCMC sampling.

The crossover operator mixes genetic information in the population by cutting pairs of chromosomes at random points along their length and exchanging over the cut sections. This has a potential of joining successful operators together. Crossover occurs with a certain probability. In many natural systems, the probability of crossover occurring is higher than the probability of mutation occurring. Mutation will be described in the next section. Simple crossover technique [15] is used in this paper. For simple crossover, one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent. For example, when 11001011 undergoes simple crossover with 11011111 it becomes 11001111.

The mutation operator picks a binary digit of the chromosomes at random and inverts it. This has a potential of introducing to the population new information. Mutation occurs with a certain probability. In many natural systems the probability of mutation is low (i.e. less than 1%). In this paper binary mutation is used [14]. When binary mutation is used a number written in binary form is chosen and its value is inverted. For an example: 11001011 may become 11000011.

Reproduction takes successful chromosomes and reproduces them in accordance to their fitness functions. In this paper Metropolis et al. criteria, described by equation 7, is used as a reproduction method. By so doing the least fit members are therefore gradually driven out of the population of states that form a Markov chain.

The schematic illustration of the MCMC method trained using genetic programming is shown in Figure 2. In this figure an initial sample weight vector \( w \) \( _n \) is generated. Then the sample is converted into binary form using Gray method [14]. The sample is then mutated to form a new sample vector \( w \) \( _{n+1} \). The new weight vector \( w \) \( _{n+1} \) undergoes crossover with its predecessor \( w \) \( _n \) and mutates again to form a new network weight vector \( w \) \( _{n+2} \). The weight vector \( w \) \( _{n+2} \) is converted into floating-point and then its probability is calculated. This network weight vector is either accepted or rejected using the criterion in equation 7 and this is called reproduction. Thereafter, states \( w \) \( _{n+2} \) and \( w \) \( _{n+1} \) in binary form undergo crossover and are mutated to form \( w \) \( _{n+3} \). State \( w \) \( _{n+3} \) is then reproduced using Metropolis et al. criterion.
The genetic MCMC proposed in this section is different from the traditional GA in the following nature:

(a) The genetic MCMC does not generate a new population of genes at any given iteration (i.e. generation in the GA framework) as is the case in GA but it generates one sample at each iteration.

(b) The fitness function uses Metropolis criterion while in GA this is not the case.

(c) The genetic MCMC has a higher mutation rate than GA.

The genetic MCMC is different from a standard MCMC in the following way:

(a) The random walk in the traditional MCMC is replaced by a procedure inspired by Darwin’s theory of evolution which entails cross-over, mutation and reproduction.

(b) It operates in floating-point space.

6. CASE STUDY

In this case study, Bayesian neural networks that are trained using genetic MCMC are used for regression problems. The same regression problem is solved using the classical MCMC which generate states in floating-point space and accept or reject the state using Metropolis et al. method. The results of the two methods are then compared.

The simulated data are generated from a noisy sine function with a standard deviation of 0.1. This is the same function that was used by Nabney [16]. Twenty data points are generated around $x=0.25$. Regression analysis is conducted for the domain $x=0 \text{ to } 1$. The MLP networks constructed have one input, five hidden units and one output units. The optimal number of hidden units was obtained by studying the relationship between the number of hidden units and the generalization error. This was conducted by setting the number of hidden units to fall between 1 and 8 and assessing the generalization error. The hidden layer activation functions are a hyperbolic tangent functions, while the output activation functions are linear functions.

On implementing Bayesian training the coefficient of the data contribution to the error $\beta$ is set to 100 while the prior coefficient $\alpha$ is set to 0.001. The manner in which these parameters fit into the Bayesian framework is described by equation 3. The number of retained states $L$ is 10000 while the number of discarded states $R$ is 200 and these values fit into the Bayesian framework through equation 6.
For the genetic part of the simulation the rate of mutation is 0.33 and the rate of crossover is 0.6. It should be noted that the rate of mutation proposed here is much higher than that of standard genetic algorithm. This shows that the proposed Bayesian method via genetic programming is a random search and therefore equivalent to the random walk that is executed in the standard Bayesian sampling. Indeed the proposed procedure is in principle equivalent to the standard random walk, however, it takes into account of the efficient sampling in binary space which has been observed in standard genetic algorithm. It must be noted that the rate of mutation chosen here is lower than the rate of crossover, which is in accordance to many natural systems. When implementing the genetic framework through genetic algorithm, 16-bit binary numbers are used. The bounds of the magnitudes of the components of the weight vectors are \([-4, 4]\).

7. DISCUSSION

The results obtained when Bayesian networks are trained using genetic programming are shown in Figure 3. The mean-square-error (MSE) obtained from this figure is 0.371. The results obtained when the Bayesian networks are trained using traditional MCMC, are shown in Figure 4. This gives an MSE of 0.55. When Figure 3 is compared to Figure 4, it is observed that genetic approach to MCMC performs better than the method that uses MCMC method that operates in floating-point space, because it gives lower average errors. The graph showing the errors as a function of samples accepted from state 1 to state 10000 is shown in Figure 5. This graph shows that the rate of convergence to a stationary posterior distribution is faster when using genetic approach than when using the standard MCMC method. This is because it has been proven that sampling through binary space is more efficient than sampling through floating-point space [15]. The reason for this is that sampling through binary space, is able to explore a larger part of the weight-space than if the process is conducted in floating-point space. The acceptance rate of states for MCMC method that operates in floating-point space is 0.80, while when using MCMC method based on genetic programming is 0.71. This indicates that the MCMC method based on genetic programming is able to explore states that form a Markov chain better than the MCMC method that operates in floating-point space.

8. CONCLUSION

In this paper genetic approach to MCMC sampling is introduced. The method is tested on simulated data. The results obtained are compared to those obtained from MCMC method that
operates in floating-point space. It is concluded that the MCMC method based on genetic programming gives better results than MCMC method that operates in floating-point space.
Reference


Figure 1. Feed-forward network having two layers of adaptive weights
Figure 2. A schematic illustration of genetic sampling for the MCMC implementation
Figure 3. Results obtained when Bayesian networks are trained via genetic programming
Figure 4. Results obtained when Bayesian networks are trained via traditional MCMC
Figure 5. Prediction errors versus samples.
List of Illustrations

Figure 1. Feed-forward network having two layers of adaptive weights

Figure 2. A schematic illustration of genetic sampling for the MCMC implementation

Figure 3. Results obtained when Bayesian networks are trained via genetic programming

Figure 4. Results obtained when Bayesian networks are trained via traditional MCMC

Figure 5. Prediction errors versus samples.