Pruning of Neural Network using Generalized Bayesian Approach

Bhavesh Mayekar B.E.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Data Science)

Supervisor: Prof. Jason Wyse

August 2019

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Bhavesh Mayekar

August 13, 2019

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Bhavesh Mayekar

August 13, 2019

Acknowledgments

I would like to take this opportunity to express my gratitude to my supervisor Prof. Jason Wyse, for believing in me to accomplish this project. Without his support and motivation, this dissertation would not have been possible. Through his in-depth knowledge in Bayesian Statistics, he provided me with great advice and guidance.

I would also like to thank Prof. Arthur White, for teaching the course Applied Statistical Modelling and clearly explaining concepts of statistics and probability. I am so thankful to the Department of Computer Science and Statistics, Trinity College Dublin for providing me with infrastructure and environment to work on my thesis.

I would like to thank my family, for continuously motivating and helping me and also believing in me. Without their love and support this thesis would never be possible.

Bhavesh Mayekar

University of Dublin, Trinity College
August 2019

Pruning of Neural Network using Generalized Bayesian Approach

Bhavesh Mayekar , Master of Science in Computer Science

University of Dublin, Trinity College, 2019

Supervisor: Prof. Jason Wyse

The use and popularity of Neural networks have increased in the recent past due to its performance and efficiency to solve problems of different sectors. With the improvement in performance, there is an increase in the complexity of the structure. Small sized networks tend to show good generalization but are also are underfitted. On the other hand, large scale networks learn the data efficiently but lack generalization. In this research, a Neural network pruning algorithm influenced by general Bayesian inference has been introduced. The research is an attempt to prune the Neural network and also to study the activation patterns of weights for different class of input data. The proposed approach was verified on a simulated and an image dataset. On simulated dataset it was found that pruning was maximum when the data was easily separated, as the complexity of the data was increased, the pruning reduced substantially. The results from real-world data show that the base network was pruned by 40%. It was also possible to decipher the important nodes for the classication of classes. After pruning, loss of the network was reduced to some extent.

Summary

Neural Network have the capability of learning complex data. Neural network models are being used in various industries to learn data, mainly in classification or prediction tasks. These models contains number of neurons which helps in the processing of the network. With the increase in number of neurons, there is an increase in the energy consumption and processing time. In order to implement, these models on a small device there is a need to compress the network. In this research, a pruning algorithm for the compression of Neural Network has been introduced.

The pruning algorithm uses generalized Bayesian methods, to prune the redundant weights. Using this method, a Bayesian model was designed with loss function instead of likelihood function. An activation switch function was introduced which decides the pruning of weights. The algorithm makes use of Metropolis Hastings method, to propose samples for the activation switch. Every weight is assigned a switch, which decides the activation of a weight. The designed algorithm was verified on a simulated dataset and also on a real-world dataset. The results shows the significant amount of pruning. Also, the nodes responsible for the classification of 2 classes were identified.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Neural Networks

In this era of advanced technology, machine learning and deep learning models are being used in many applications. These models are generally used for prediction or pattern analysis. Over the period of time, deep learning models have been providing better performance which outnumbers human efficiency. It is a method in which the replication of the structure of a human brain is performed [1]. In this technique, a sequence of the algorithms is used to find a hierarchical representation of input information by simulating how the human brain senses a collection of sensory information to which it is subjected [2]. The foundation for deep learning methods was laid in the 1940s when perceptron was discovered [3]. Later in the 1980s with additional research perceptron was modified to multi-layer perceptrons. Later in the 2000s due to shortcomings of perceptrons and limited computational ability, Neural network model with multiple neurons was introduced. [4].

The Artificial Neural Network (ANN) is an approach to reproduce the mathematical representations of information processing which happens in the human brain on a computer. ANN is the network of multiple neurons/perceptrons connected to each other. ANN consists of 3 layers, namely, the input, hidden and the output layer. The input layer feeds the incoming data to the hidden layer, where all computation work is performed. The computed output is given out by means of the output layer. There can be single/multiple hidden layers between input and output layers. Neurons are present in these layers. Every connection between the neurons has a specific weight which is used for computation. There are two processing stages in the Neural Network (NN) algorithm. The weights are computed in a forward direction in the first phase, that is from from the input to the output layer. The entire task is known as Forward Propagation (FP). In the second step, weights are adjusted by means of Backward Propagation (BP).

In FP, once the input is fed to the network, the assignment of weights take place randomly. Consider, a NN model as shown below [5]: where,

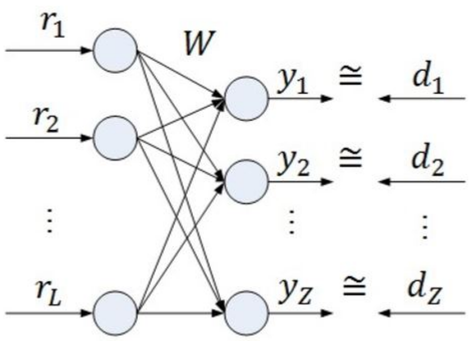

Figure 1.1: Artificial Neural Network model

$\bar{r} = [r_1, r_2, r_3, ..., r_L]^T \ \epsilon \ \mathbb{R}^L$
$\bar{r}$ is the input vector
$w = [\bar{w_1}, \bar{w_2}, \bar{w_3}, ..., \bar{w_z}]^T \ \epsilon \ \mathbb{R}^{LxZ}$
w is the weight matrix
$\bar{w_i} = [w_{i1}, w_{i2}, w_{i3}, ..., w_{iL}]^T \ \epsilon \ \mathbb{R}^L$
$\bar{w_i}$ is the i-th weight vector $\bar{y} = [y_1, y_2, y_3, ..., y_Z]^T \ \epsilon \ \mathbb{R}^Z$
$\bar{y}$ is the network output
$\bar{d} = [d_1, d_2, d_3, ..., d_Z]^T \ \epsilon \ \mathbb{R}^Z$
$\bar{d}$ is the expected network output and ‘T’ is a transpose operator.

The network output is given by the formula:

$$y_z = \sum_{i=1}^{L} r_i * w_{zi} = \bar{r}^T w_z \tag{1.1}$$

Once the weights are calculated using FP method, the error between expected output and the computed output is calculated. Total error in the network is the loss function of the network. The error value is given by,

$$E = \frac{1}{2} \sum_{z} (y_z - d_z)^2 \tag{1.2}$$

For a NN to be efficient E should be minimum. This task can be performed using Gradient Descent method. It emphasizes on calculating the partial derivative of E with respect to each weight in the network to calculate error. Once, the error is calculated for FP, gradient descent sets in the BP process. Using BP, weights are adjusted in such a way to find an optimum gradient where the loss function is minimum. The steps of the gradient descent function are decided by the step size function η also known as learning rate (LR). Multiple iterations take place to adjust the weights in the network. For every iteration the weights are adjusted using the formula as below:

$$\Delta w_k = -\eta_k \left(\frac{\partial E}{\partial w} \right) \tag{1.3}$$

here, E indicates error function value at k^{th} iteration; w is the weight of the links in the network whereas η_k is the step size or learning rate to minimize the error. The value of LR is not fixed and depends on multiple factors.

In this case of BP, the value of LR can be between 0 to 1. If η is very small, then gradient descent will take many iterations to converge to local minima. Also, if the value of η is very big the method will not converge as it will overshoot the optimum point. Therefore, it is needed to select LR carefully in order to make the algorithm converge and attain the state of minimum loss. Many research has been performed to find the optimum value of LR.

These two processes FP and BP generates a NN model. The model is trained for the input data and the training accuracy is computed. After k iterations of FP and BP, a final model with weights for all its links is generated. ANN model can have multiple numbers of the hidden layer with hidden nodes in it. Once the model is trained, for every connection between the nodes, weights are assigned which help to generate the output. For example, if there are 10 input nodes and 5 hidden nodes, the total weights will be (10 * 5) 50 between these to layers.

Once the model is generated, it is bulky with a huge amount of weights. Hence, there is a need to optimize the model. This optimization can be in terms of compressing the weights to decrease the network complexity. Such optimization can aid simplicity of the network, efficient performance on training and generalized network which avoids overfitting [6]. It includes the selection of a suitable number of inputs and outputs nodes, the determination of the number of weights between the nodes [7].

1.2 Research Aim and Motivation

ANN can perform a wide range of complex tasks like face recognition, voice assistants etc. but it involves high energy consumption. One of the drawbacks of ANN is that its computationally heavy and needs Graphical Processing Units (GPU) to process. Many industry applications have various resources at their disposal, hence high computation is not a problem. If any such models are to be implemented on a smartphone or any other devices with computational resources limitation, then it can result in low network performance. Moreover, executing such models on servers can result in transmission overhead. Hence, there is a need to compress ANN in order to make it energy efficient. There is a need to generate smaller NN models, which can be used on fewer memory devices. Many methods are proposed for the compression of the NN model, one of them is 'Pruning'.

When the ANN model is constructed, it has multiple weights. For the processing task like classifications, not all theses weights are important. The attempt has been made to understand this redundancy in the network. The aim of this research is to propose a concept of Bayesian update on weights to prune the Neural network. The redundant network weights are being pruned using the iterative approach without affecting the performance of the network. Also, the attempt has been made to understand the activation patterns of weights in a Neural network for different classes.

The basic architecture of ANN consists of multiple hidden layers which have a predefined set of hidden nodes present in it. All the links between the nodes are assigned with weights, once the network is constructed. Some of these weights are redundant and can be pruned in order to compress the network. In this research, the method for ANN compression has been described.

1.3 Dissertation Structure

The dissertation is organized as follows: the next sections describes the study of related work and literature review is performed. Chapter 2, talks about the Bayesian Inference in detail. Generalized Bayesian update, which is one of the supporting ideas of the research is described in Chapter 3, whereas Chapter 4 explains the loss function derivation of the network. Chapter 5 explains the Monte Carlo Markov Chain process along with Metropolis Hasting algorithm. The proposed method for the pruning of the NN and the algorithm is explained in Chapter 6. In Chapter 7, the details about the implementation of the proposed algorithm on simulated data set and the real-world dataset is described and results are explained. Chapter 8, concludes the research and give a necessary idea about future work.

1.4 Related Work

One of the issues for network engineers is to determine the configuration of the network before training. The small-sized networks are faster and cost-effective to build and also possess generalization but main tend to underfit the training data. On the other hand, large networks end up being over-fitted with training data and have poor generalization capabilities. This problem can be solved by the construction of an optimal sized network. It can be done in two ways, either by having a constructive algorithm which deals in an incremental way of improving network configuration or by pruning algorithm. The incremental algorithm has a drawback that the network might get trapped at local minima [8]. The pruning algorithm starts with trainable network and ends up minimizing redundant weights or neurons. There are a few research done in this space to improve network efficiency.

In [9] 'Optimal Brain Damage (OBD)' procedure is explained. It uses the second derivative of the error with respect to the weight during backpropagation. The disadvantage of this approach is the computation of the Hessian matrix, which can be a costly affair. The other method 'Optimal Brain Surgery' proposed by [10] uses inverse the Hessian matrix to find the redundant weights to be pruned. It involves the simultaneous update of weights. These two methods are very effective but computationally expensive and have large storage requirement.

One of the processes in which, parallel pruning of least important weights is performed is described in [11]. It defines a sensitivity measure for every hidden node which is a parameter to understand the degree of importance for every weighted connection. This method does not involve a complex calculation of the Hessian matrix. The scope of this method is yet to be included in classification problems. Another sensitivity-based method is explained in [12]. The introduction of "Skeletonization" technique could remove weights which have least significant information to improve the learning performance. It compares the error of the network with and without the presence of the nodes. Thus, removing the units with the least error. For the error calculation, derivative computed in the BP process is used for generalization. The sensitivity analysis is also incorporated in the algorithm suggested in [13]. The introduced algorithm, Novel Pruning Feed-forward Neural Network (NP-FNN), determines the number of hidden nodes based on output sensitivity to the neurons. The pruning is depended on the training data and it can delete multiple redundant weights in a single iteration, thus reducing the run time.

A few other research, include generation of rule-based algorithm for pruning. In [14], an algorithm known as Neural Network Pruning for Function Approximation

(N2PFA) is described. It is a process, which focuses on pruning neurons than weights. Here, the effect of removal of neurons is studied and error is calculated. The moment at which error is deteriorated by a predefined value, the pruning stops. The drawback of the method is that if the tolerance value is large, then significant neurons are also pruned. An algorithm suggested in [15] aims at minimizing the sparsity of the weights generated in the BP process by pruning the weights using statistical quantities of the network. This algorithm posses fast processing as it uses first-order correlation data of weights and requires less storage as it does not involve storing Hessian matrix. The disadvantage of this approach is that the obtained network after pruning is decrepit and is not fault-tolerant.

The work in [16] talks about the pattern recognition algorithm which makes use of Bayesian decision boundary technique for pruning the network. The method deals with stem-and-leaf graphics technique to calculate the hidden neurons. It maintains the classification accuracy of the network after pruning. The credibility of this algorithm needs to be verified on multi-classification problems. Also, the study of the initial configuration of the network is needed.

Chapter 2

Bayesian Interface

2.1 Introduction

The study of statistics helps to understanding the uncertainty in the data, which is useful in decision-making process . The base technique of numerous research is to gather data, develop a hypothesis and test it as per the data collected, and based on the results draw conclusions. This is the traditional path and has been followed by many statisticians and is a form of frequentist interpretation [17] . One more popular approach for statistical analysis is based on Bayesian methods. In this method, the process is initiated with prior knowledge of data and updation of belief is performed. Hence, the data distribution is analyzed and the belief is formulated based on the information in a more formalized way

Consider an example of tossing a coin. The possible outcome in this task is either head or tail. Suppose, the task is to interpret the probability of the result is a tail. The frequentist method would be to accumulate data and use it to compute the probability of a tail. However, in the Bayesian approach, it would be necessary to first gather a prior idea about the fairness of coin and the data to find the probability of a tail. Since the coin is fair the prior probability of a tail is 0.5. After, analyzing the data, assume in 100 tosses 75 times the result was a tail. Hence, there is an update in the prior knowledge and the probability of getting a tail will be between 0.5 to 0.75 as per the analysis. On the other hand, the frequentist model will provide a probability of being a tail as 0.75. Thus, the Bayesian model, take into consideration some prior factors which affect the end result.

2.2 Frequentist Statistics

Assume the observed data to be y, such that $y = (y_1, y_2, y_3, \dots)$. Here y_1, y_2, y_3 indicate the individual observation. This data is independently and symmetrically distributed with a probability density function (pdf) f and parameter θ. The observed data y can be continuous or discrete, whereas θ can be scalar or a vector quantity. In this method, the clue is to derive inferences from frequent samples to recognize how frequency estimation of the parameter of interest performs. The likelihood function for such a scenario is constructed as follows:

$$L(\theta) = \prod_{i=1}^{n} f(y_i|\theta) \tag{2.1}$$

2.3 Bayesian Statistics

Bayesian interpretation is named after scientist Thomas Bayes, who had articulated specific cases of Bayes theorem in 1763. A few years after, it was Pierre Simon de Laplace who gave emphasis on using Bayesian theories for practical applications. He developed a relationship between pure mathematics and scientific reasoning by analyzing data using Bayesian methods. He emphasized upon learning from past till new data is obtained, hence getting better results and correlation. Consider two events A and B. The task is to impute the probability of B given, A has already occurred. This can be expressed as shown in the below expression [18].

$$p(B|A) = \frac{p(A|B)p(B)}{p(A)} \tag{2.2}$$

here, B represent unknown event and A is the known event. This is also known as Bayes Rule, Bayesian statistics is a branch of statistics that applies Bayes rule to solve uncertain events [19].

For Bayesian methods, the main requirement is the prior knowledge of the interested parameters. These parameters are considered as unknown random variables. In this context consider, y and θ to be random variables. The belief using joint distribution of θ and y is represented as following [5]

$$p(\theta, y) = p(y|\theta)p(\theta) \tag{2.3}$$

where,
$p(y|\theta)$ is the likelihood function,
$p(\theta)$ is the prior distribution.

Collection of data and prior information is the first step in Bayesian process. An appropriate Bayesian study will constantly include honest prior information, which will support interpretations about the correct value of the parameter and make sure there is no wastage of important information. Later, based on collected data, a likelihood model is created. After the model is generated, the first two steps are merged to generate a posterior distribution using Bayes theorem. This approach is observed as a continuous learning process, as the new data related to the problem is collected. Then the prior distribution is replaced by current posterior distribution and with the combination of the model, new posterior distribution is generated.

The above task can be better understood by a simple example. Consider, the task of finding the probability, whether a person has cancer or not. As mentioned before, the first step in any problem is to gather data. The sample space will consist of two outcomes, i.e. either a positive for cancer detection or a negative. It will be denoted by a random variable Y. The probability of a test is positive for cancer for a single person is $\{0, 1\}$, here $y = 0$ indicates a test for cancer being negative and $y = 1$ indicate a positive detection for cancer. In this example, Y is a random variable and y represent the instances of Y. θ is used to indicate the probability of the detection of positive cancer cases. In this example, Y is considered as a discrete function, hence its probability mass function is given as follows:

$$p(y|\theta) = \begin{cases} \theta, & \text{if } y = 0 \\ 1 - \theta, & \text{if } y = 1 \end{cases} \tag{2.4}$$

The next step is to develop a model. It is generated by using a likelihood function. The probability of positive cancer cases is calculated as a function of θ. Let $p(positive) = p(y = 1)$. Therefore in terms of θ, it can be represented as $p(y = 1|\theta) = \theta$, where $0 \leq \theta \leq 1$. Similarly for negative cases, it is represented using the following notations $p(negative) = p(y = 0|\theta) = 1 - \theta$, where θ will always lie between 0 and 1.

From the above inferences, the likelihood function is generated. It will give more information about the parameter θ and can be constructed using Bernoulli Distribution as follows:

$$L(\theta|y) = \theta^y (1 - \theta)^{(1-y)} \tag{2.5}$$

Here, y is fixed and θ lie between 0 and 1.

The next step is to generate a prior distribution. Let θ takes value from 0.1 to 1. The mean of a probability distribution is 0.5, whereas the value of θ ranges from 0.1 to 1. The last step is to collect data followed by an application of Bayes Rule. Multiple samples are taken to predict uncertain futuristic events considering the latest past events. Applying Bayes rule, posterior distribution is generated combining the prior data with likelihood model.

In Bayesian analysis, θ is unknown parameter whereas data is denoted by D . $p(\theta)$ denotes the prior distribution of θ and its likelihood function is given by $p(D|\theta)$. The marginal likelihood can be represented as $p(D)$ and $p(\theta|D)$ as the posterior distribution. [20] Bayes theorem states that "the probability of any event is the ratio between the value at which an expectation depending on the happening of the event ought to be computed and the value of the thing expected upon it's happening." [21]. In other words, posterior distribution is formalized as a function of the prior distribution and the likelihood function. It is represented as shown below:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{2.6}$$

Advantages of this approach are that it is a coherent statistical method and follows a predefined framework. It is effective in method to solve unknown problems provided prior data is available. This can also be a disadvantage as the method is concentrated on prior information. Also, it involves expensive calculations.

2.4 Beta-binomial Model

As from the previous example of detection of cancer. The data observed is $y_1, y_2, y_3, ..., y_n$ such that $y_i \in \{0, 1\}$. From this, prior is assumed as a Beta distribution $\beta(a, b)$. Let a and b be the hyper-parameters for the prior distribution. This is the belief about unobserved data and can be chosen. For initial consideration, set $a = b = 1$ for a Beta distribution model, which means that $p(\theta|a, b) = 1$ for any value of θ. This can be considered as non-informative prior, but if the data is sufficient then it does not have a significant impact.

The likelihood function can be considered as Bernoulli function with binomial data. The output value can be either 1 (positive for cancer detection) or 0 (negative for cancer detection). Hence, the likelihood function is constructed as follows:

$$L(y|\theta) = p(y|\theta) = \theta^y(1-\theta)^{1-y} \tag{2.7}$$

here $y = 1$, correspond positive cancer cases and θ is the probability of cancer prediction.

Combining the prior and likelihood function posterior distribution is generated. It can be formulated as

$$p(\theta|y, a, b) \propto p(y|\theta)p(\theta|a, b) \tag{2.8}$$

Using Bayes theorem,

$$p(\theta|y,a,b) = \frac{p(y|\theta)p(\theta|a,b)}{p(y,a,b)} \tag{2.9}$$

Applying Bernoulli and beta distribution above equation can be modified as [21],

$$\begin{aligned} p(\theta|y,a,b) &= \theta^{y}(1-\theta)^{(a,b-y)}\frac{\theta^{(a-1)}(1-\theta)^{(b-1)}}{\beta(a,b)}/p(y,a,b) \\ &= \theta^{y}(1-\theta)^{(a,b-y)}\theta^{(a-1)}(1-\theta)^{(b-1)}/[\beta(a,b)p(z,a,b)] \\ &= \theta^{a'-1}(1-\theta)^{(b'-1)}/[\beta(a',b')] \end{aligned} \tag{2.10}$$

Hence, by correlation it can imputed that, $a' = \sum_{1=1}^{n} y_i + a$ and $b' = n - \sum_{i=1}^{n} y_i + b$. The form of posterior distribution is $p(\theta|y,a',b')$ which is similar to prior distribution of $p(\theta|y,a,b)$. Therefore, it is concluded that posterior also follows a beta distribution. If sample size n is much larger than $a+b$, then the hyper-parameters could have relatively less impact on the posterior distribution.

Chapter 3

Generalized Bayesian Update

Mathematically Bayesian inference state that it is required to study the probability distribution of all related parameters and analyzing the values of the parameters which are available. This helped in concluding the conditional probability of the parameter of interest. Bayesian approach works in two phases, first it is required to impute the marginal distribution of variables followed by the conditional probability distribution of the variables. The first phase is prior distribution, which takes into consideration the belief about parameters before observing the data. However, the likelihood which is the second phase aims at drawing insights from the observed data. This is the case of Bayesian inference described in Chapter 2, however, generalization tends to be more specific approach [22].

Classical Bayesian update generates the likelihood model which is depended on the distribution of the parameter of interest, let's say θ. There are many scenarios in which it is impractical to specify the distribution of θ. In such cases, the use of generalized Bayesian update assist in performing coherent Bayesian inference for the parameter θ, by making use of loss function instead of the likelihood function. This is a special case of Bayesian inference, used when the data does not follow any specific distribution. Hence, the loss function is used to bridge the gap. The work of [23] prove that loss function can be used as pseudo-likelihood function. This idea has been been used in the research.

Consider a sampling function $F_0(x)$ of any random unknown distribution. Let $p(\theta)$ represent the prior knowledge about θ which is the parameter of interest. Also, x is the sample from F_0 function, then it can be claimed that by any update on θ the posterior $p(\theta|x)$ gets updated [23], where

$$p(\theta|x) \propto L(\theta, x)p(\theta) \tag{3.1}$$

Many times the belief about data gets updated as the new information about θ is obtained through x. Hence these parameters are updated for the new function. This new function dictates the model when more information about the parameters is avialable. If there are multiple parameters say x_1 and x_2, the posterior function gets modified as it becomes a function of these parameters and can be represented as $p(\theta|x_1, x_2)$. This claims that posterior remains constant even if there is an update on (x_1, x_2) together or even individually.

In this study as proved by [23] the loss function is used as a pseudo negative likelihood function.The posterior distribution of a function is determined by updating prior and samples using a loss function. For this case, a negative likelihood function is defined as:

$$L(\theta) = -\int l(\theta, x)dF_0(x) \tag{3.2}$$

here, $l(\theta, x)$ denote some loss function. The idea of any Bayesian method is to have a maximum likelihood estimate, in this case, it can be done by minimizing the loss function. Hence, the negative sign is used to reduce the loss. As per the study on likelihood function, this case could be related to the use of negative log-likelihood function to the loss function. It can be represented as:

$$\log L(\theta) = -l(\theta, x) \tag{3.3}$$

Converting equation 3.3 to standard format :

$$L(\theta) = \exp(-l(\theta, x)) \tag{3.4}$$

From this, it can concluded that the performance of the model can not only be analyzed by checking the accuracy of the model but also by studying the loss function. Hence, it becomes necessary to minimize the loss function involving the parameter of interest θ. There have been studies to relate negative log-likelihood to the loss function. In [23] it is stated that after Bayesian update on the prior with the involvement of loss function, the statisticians must verify that the solution exists. Considering this criterion, there is a constraint on log function and it needs to be constructed as follows satisfying the following criteria:

$$0 < \int \exp\big\{ - l(\theta, x)\big\} p(\theta) d\theta < +\infty \tag{3.5}$$

From this, it is possible overcome the shortcoming of the Bayesian method as it requires a probabilistic model for the entire range of results which is subjected to all unknown parameters. However, this method needs the formalizing of loss functions, when the results for the variable of interest are available. It enables statisticians to give importance to the modeling of prime parameters.

Also, for updating the prior function $p(\theta)$ to generate a posterior function $p(\theta|x)$ given the input data x obtained from $F_0(x)$, there is a need to consider the marginal likelihood function. For the given sampling distribution $f(x, \theta)$ with prior $p(\theta)$, the marginal likelihood function is given as:

$$m(x) = \int f(x, \theta) p(\theta) d\theta \tag{3.6}$$

Thus Bayes theorem is modified as [24]:

$$p(\theta|x) = \frac{f(x, \theta) p(\theta)}{m(x)} \tag{3.7}$$

Chapter 4

Loss Function Derivation

The ideology used in this study is based on Bayesian methods. Using the Bayesian method, posterior distribution can be determined by studying the prior and likelihood function. Here, a lucid method has been introduced for Bayesian interface in which the update prior belief distribution for understanding posterior distribution is performed using the loss function. This approach is implemented in Neural networks to understand the posterior distribution of weights in the network. As discussed in the previous chapter, generalized Bayesian update is used to replace the likelihood function with the loss function. Current chapter relates the loss of an ANN to the loss function of the Bayesian model. The present prior information is related to initial weights of the network and testing data, based on this information it is possible to find the posterior behavior and prune the network.

$$\pi(w|Test) \propto l(Test|w).\pi(w) \tag{4.1}$$

Here, w indicates, weights assigned to the network, $Test$ indicates the test data to be used for testing the network. For all, n outputs, the loss can be defined as some function of all output data generated by the network for a specific configuration. It can be formulated as below,

$$l(\theta) = \prod_{i=1}^{n} f_\theta(x_i) \tag{4.2}$$

The log loss of the above equation 4.2 can given as follows,

$$\log l(\theta) = \sum_{i=1}^{n} \log f_\theta(x_i) \tag{4.3}$$

Consider a Neural network N with 1 input layer and output layer, coupled with n hidden layers. After, training the network with FP and BP process, the fully connected layered network is obtained. All the links in the network are assigned with significant weights which are used for the computational purpose. Let the input to the network be denoted by x, whereas the weights are denoted by w_{ij}^{n} where i indicates the source node, j indicates the destination node and n is the layer number. The fit of the model is decided by the loss function. Loss function is decided by the outputs which are misclassified. The loss function for this model can be given by $Q(\theta)$. The maximum likelihood of the model corresponds to reducing the loss function and can be related to negative log likelihood. This can be represented by the equation 4.3,

$$\max \log L(\theta) = \min \big[-\log l(\theta) \big] \tag{4.4}$$

From the above equation 4.4, min $\big[-\log(L(\theta)) \big]$ can be compared to the minimization of loss function $Q(\theta)$, i.e.

$$\min \big[-\log(l(\theta)) \big] = \min Q(\theta) \tag{4.5}$$

In order to convert the equation in standard format, there is a need to remove the log and get the equation in normal form and the loss function can be denoted as,

$$\exp(-Q(\theta)) = l(\theta) \tag{4.6}$$

From, the above justification, the new negative likelihood function can be expressed in terms of loss function of the network and is denoted as follows,

$$l'(\theta) = \exp(-Q(\theta)) \tag{4.7}$$

This can be considered as a classical case of using negative log likelihood function as a loss model. Since, a proper distribution for θ cannot be defined, the loss function is used in place of likelihood. In this approach the loss function will be used to test the hypothesis.

Chapter 5

Markov Chain Monte Carlo

5.1 Introduction

Bayes Theorem is used to find the maximum likelihood estimate (MLE) of θ, through which it is possible calculate posterior distribution $p(\theta|y)$ of θ with y as data. The method only focuses on θ however, other parameters of interest may be the distribution of the difference between different variables like θ_0, θ_1. Also, posterior distribution of $f(\theta)$, where $f(\theta)$ is some modification on θ, for example $f(\theta) = \exp(\theta)$. Such variables or parameters which are difficult to compute can be calculated using Monte Carlo methods. The central idea of Markov Chain Monte Carlo(MCMC) technique is to generate multiple samples of a continuous random variable having some known probability density function which can also be a posterior distribution. The generated sequence of sample data is then used estimate the integral to get the expectation or variance of the variable.

MCMC which is the combination of two processes, Monte-Carlo and Markov chain. Monte-Carlo methods aims at understanding the distribution by generating random samples for a given distribution. This approach gives better transparency and understanding of data as the mean or variance, is calculated from the generated samples instead of given data distribution. This can be helpful when data is not easily available. Hence, samples can be generated using this approach. By using Markov Chain technique of MCMC the generation of samples is done systematically by a chain of sequence. The distinct feature of the Markov Chain is that every sample generated is depended only on its previous sample [25]. These two properties form the building blocks of the MCMC method. Bayesian methods aims at finding the posterior distribution, which can be problematic at times to calculate. MCMC helps in the calculation of likelihood for different groups of samples, which can be later united with the prior to finding an analytical solution for posterior.

Monte Carlo techniques were derived in Mexico in the 1940s. These methods were used to formulate the Metropolis algorithm in the 1950s. These researches were improvised by Hastings in 1970s using statistical approach. The influence of MCMC techniques in the field of mathematics was not stroked nearly till the 1990s [26]. There can be many reasons for this: limitation of computational power as MCMC techniques involve multiple calculations and it was not possible in the 1900s. For example, if there are 5 parameters and each parameter take 1000 values, which results in 1000^5 combinations. This was practically not possible. The other reasons can be acceptance of such high computational method and its practical usage. As technology evolved and computers became supercomputers, all these drawbacks seem to be minor problems. Now in the 21st century, with tremendous computational power possessed by computers, MCMC methods are now practically possible. An advanced study places MCMC methods in the best ten techniques that have an impact on the advancement of science and technology in 20th century [27].

An ideal Monte Carlo simulation will draw independent and identically distributed samples θ^i where i takes values from 1 to n, from a target probability density function $p(\theta)$ over $\{X\}$. But that is not the case every time as the correlation of samples depends on the sampling function. Most of the times the samples generated are correlated. The correlation of samples will be less if the sampler is efficient. The N samples are generated from posterior density $\pi(\theta|x)$. It is needed to determine the expected value of function of θ, $f(\theta)$. The posterior expectation $E\{f(\theta)\}$ of this function $f(\theta)$ is given by:

$$E\{f(\theta)\} = \int f(\theta)\pi(\theta|x)d\theta \tag{5.1}$$

The few drawbacks in the 5.1 is that θ can be high dimensional. Therefore, it becomes difficult to compute the integration function to get the estimate. An alternate approach could be to generate samples. Instead of computing the integration, samples can be generated from the posterior distribution to approximate the integral value. 'MCMC' method implements this idea to provide the estimated value of the function by generating samples from a distribution. Thus, the complex integration process can be avoided using the sampling process. This can be expressed as below:

$$E\big\{f(\theta)\big\} \approx \frac{1}{N}\sum_{i=1}^{N} f(\theta^i) \tag{5.2}$$

here, $E\big\{f(\theta)\big\}$ is the expected value of the function, $f(\theta^i)$ are the generated samples and N is the number of samples.

5.2 Markov Chains

The random sequence of samples $x_1, x_2, x_3, ...$ generated from a known distribution is known as Markov Chain. The necessary condition that should hold true is that $x_j|x_{j-1}$ is independent of for any value of x, that is $x_{j-1}, x_{j-2}, ...x_{j-n-1}$ for any value of j. In other words, every sample generated in the chain should depend on the previous sample and no other sample. So all the sample will determine their next sample respectively. Consider a transition kernel $s(x_j|x_{j-1})$. For every value of j the following condition holds true, $s(x_j|x_{j-1}, ..., x_{j-n-1}) = s(x_j|x_{j-1})$for some function s, given that x is a Markov Chain. Then, the probability density function p is given by,

$$p(x_j) = \int s(x_j|x_{j-1})p(x_{j-1})dx_{j-1} \tag{5.3}$$

For above density function p to hold true the transition kernel s should possess the following properties:

1. Irreducibility: If the chain is started, then there exists a probability of generating any possible value of x. In other words, it is possible to traverse from state i to

state j if there exists some $t \geq 0$ such that,

$$p(x_{ij}^n) = P(x_n = j | x_0 = i) > 0 \tag{5.4}$$

It states, it needs n steps to move from state i to state j with probability $p(x_{ij}^n)$. If this condition hold true for the movement from state i to j and state j to i, then both the states are communicable. Hence, Markov chains are irreducible, if both state can communicate.

2. Aperiodicity: The samples generated in the chain will not occur again after a fixed interval. All the samples generated will not follow any fixed frequency pattern, it will completely be random, for example $x_{(j+mk)} \neq x_j$, for some k , for all m.

3. Recurrency: If the chain of random number tends to infinity, then it will return to any set of x for infinite times.

5.3 Metropolis Hastings Algorithm

The Metropolis-Hastings (MH) algorithm is used to get approximate sample values for a distribution. Metropolis algorithm was studied and statistically generalized by Hastings in 1970 to remove the drawbacks of Monte Carlo methods. Hastings defined in his paper an approach to treat finite Markov Chain by implementing the concept of acceptance probability. When the samples are generated from the state r to r' , the acceptance probability is given by [28]:

$$\alpha\{r, r'\} = \min\left(1, \frac{P(r')}{P(r)} \frac{q(r|r')}{q(r'|r)}\right) \tag{5.5}$$

Here, $q(r|r')$conditional probability of proposing a state r' given r,

The MH algorithm is used to compute multiple samples from a target probability density π of any indeterminate parameter θ. The aim of the algorithm is to generate new samples τ, by using proposed density $q_k(\theta^t, \tau)$ and taking into consideration the current state of the chain θ^t. The new samples τ are the proposed samples. These newly generated samples τ are either accepted or rejected. The chain accepts the new state $\theta^{t+1} = \tau$ with a probability of acceptance lets say α, and if this probability criterion is not satisfied then the samples are rejected. Thus it possible to generate better and accurate samples using the MH algorithm since all the samples are not selected. The new samples τ are generated from its previous sample θ^t.

The aim is to generate new samples τ for a Markov chain having state θ^t and density function q_k. The detailed procedure of the algorithm are as follows:

1. Start by initializing the Markov chain with some sample θ^0.

2. If N samples are needed to be generated, then compute for $i = 0$ to $N - 1$

   - Calculate a random number from a uniform distribution between 0 and 1, $u \sim U_{[0,1]}$

   - Generate new sample τ from the previous sample, $\tau \sim q(\tau|\theta^t)$.

3. Next step is to compute the probability of acceptance α. It is formulated as follows:

$$\alpha(\tau|\theta^t) = min\left(1, \frac{p(\tau)q_k(\theta^t|\tau)}{p(\theta^t)q_k(\tau|\theta^t)}\right) \tag{5.6}$$

4. Now, the acceptance probability is compared with the uniform random number u generated in step 2. The new value of the sample, τ is accepted if the acceptance probability is greater than u, if not then the sample is rejected. Pseudo code for the same can be as follows:

$$\begin{aligned} if \quad & \alpha(\tau|\theta^t) > u \\ & do \quad \theta^{(t+1)} = \tau \\ else, \quad & do \quad \theta^{(t+1)} = \theta^t \end{aligned} \tag{5.7}$$

5. Return to step 1.

It is necessary to postulate the initial value θ^0, else it is selected on a random basis. There is also a need to specify a burn-in period. The burn-in period is the time in which the model attains stability, hence it is required to discard the samples generated in this period. This period can be specified by the user. If the burn-in period is very less, then unwanted samples can be selected resulting in biasing of the distribution, whereas if this period is large, it may result in wastage of potential samples. Thus, the selection of burn-in period has to be appropriate. Various researches are being carried out, to find the solution for the value of the burn-in period. The aim of this is to negate the biasing created by the initial value selected θ^0, hence the initial m samples are discarded.

MH process will provide a proposed distribution of q_k as close as possible to the target density function and increases the efficiency of the algorithm. If the proposed density function is symmetric, $q_k(\theta^t, \tau) = q(\tau, \theta^t)$, it will then result into quicker calculation of the acceptance function α. The effect of this sampling can be observed on the posterior distribution. If the proposed value results into an increase of the posterior distribution, then the sample will always be accepted. If the proposed distribution reduces the posterior, then the new sample will either be accepted or rejected, whereas if the posterior is significantly reduced by the proposed then the new samples will be rejected with higher probability.

MH algorithm can be better understood by an example. Consider a city with multiple schools. A survey needs to be conducted in a limited time with a maximum number of student participation. The number of students in each school is unknown. The task of the survey manager is to decide the school with maximum number of students. The number of students can be determined only after reaching school. Thus, the survey manager decides to randomly decide a school as a reference for the first survey. Then, he calculates the probability of moving to next school, if the proposed school has more students than the current, he moves to the new school.

Let P_{new} be the proposed student count in proposed school and P_{old} be the count of the student in the present school. Hence, the probability of proceeding $P_{proceed}$ is computed as:

$$P_{proceed} = \frac{P_{new}}{P_{old}} \tag{5.8}$$

Along with this, a uniform random number u is generated in the range of 0 to 1. If $P_{proceed}$ is greater than the random number u , then the survey team moves to the next school.

5.4 Random Walk

Considering the above example, if there are 10 schools in the city. The survey team is at school number 1 such that $\theta = 1$. The number of students in each school is assumed to be increasing linearly. Also, there are a few schools but outside the survey area hence the proposal to go to this school will always be rejected.

Suppose, the team is in school 2, $\theta_{old} = 2$ and decides to go to the next school $\theta_{new} = 3$. Assume, the population in school 3 is more than that of school 2, hence this move is accepted. Now the current school is school 3, $\theta_{old} = 3$. To calculate the probability of proceed to next school is calculated as $P_{proceed} = \frac{P(\theta_{new})}{P(\theta_{old})} = \frac{3}{4} = 0.75$. Then as per process a random number u is generated. If the value of u is greater than $P_{proceed}$ then the move is rejected. This process is repeated numerous times. This can be generalized as follows:

$$P_{proceed} = min\left(\frac{P_{new}}{P_{old}}, 1\right) \tag{5.9}$$

The above process is similar to the random walk and the next step is determined by the previous step and no other step. For every step, it is necessary to pre-define the maximum width of the step. To determine the width is the new problem if the width is large the samples generated will be far away from the current value, thus reducing the probability of being accepted. This may result in a halt of the chain on a single value for multiple iterations. However, if the width is very small, then the chain will need a more time to generate samples with low probability density and may result in under-sampling. Thus finding this value can be a topic some research. It is claimed in [29] that optimal acceptance rate should be around 0.25. MH algorithm is powered by this random walk and make complex simulation possible.

Chapter 6

Methodology

6.1 Proposed Method

In this research, the aim is to prune the Neural network by using Bayesian methods. The above described generalized Bayesian inference, which aims at using loss function as a negative log-likelihood function is used. Once, the Neural network is constructed, the model assigns weights for all the connections among the nodes. Some of these weights are important for the prediction of output class, whereas there are a few weights which are redundant. Compressing or turning these weights to 0 should not depreciate the model performance. Also, turning the weights to 0 will result in compression of the network.

The impression is to introduce a new parameter lets say γ, it will act as an activation function to decide which weights will be turned off (0) or on (1). In this study, γ is one of the parameters of interest and can be related to θ mentioned in an earlier chapter. There are two-parameter of interest; weights of the ANN (w) and the switch variable (γ). These parameters will be used in the Bayesian model. The input to the Bayesian model will be the *test* data used to test the Neural network. Let the *test* data be denoted by T.

As discussed in chapter 3 on General Bayesian Update, the loss function will be used instead of likelihood. As the distribution of weights in the Neural network is not predefined and difficult to analyze. The posterior Bayesian model for the parameters is defined below:

$$\pi(w, \gamma|T) \propto -l(T|w, \gamma)\pi(w|\gamma)\pi(\gamma) \tag{6.1}$$

Here, γ will follow a Binomial distribution. The samples of γ can have value either 0 or 1. Let the probability of $\gamma = 0$ be p, then the probability of $\gamma = 1$ be $1 - p$ as per the binomial distribution. For n samples of γ, the binomial distribution will be given as $\beta(n, p)$, where $n \in (0, 1, 2, \dots)$ and $p \in (0, 1)$. If the weight is pruned the value of γ will be 0 else 1. Consider the loss function used in the above equation 6.1 $l(T, w, \gamma)$, the test data T is used to predict the correct labels here. The aim is to find the efficient distribution of w and γ. The goodness of w and γ will define the probability of correctness of the labels.

As proved in the earlier section, the loss function is used as a negative log-likelihood function. In this case, loss function can be computed using *test* data T, weights w and activation function γ which is represented as $l(T, w, \gamma)$. Since it is compared to negative likelihood function, the loss would be represented with a negative sign, that is $-l(T, w, \gamma)$. The task is to minimize the loss function. Loss will be primarily decided by γ. The generated samples of the activation switch γ will be multiplied with the corresponding weight doing element-wise multiplication. Using these weights model will classify the test data and loss function will be defined for the model. Inefficient values of γ will increase the value of loss function. Therefore, there is a need to find efficient samples of γ, which can reduce the loss and improve model efficiency.

As proposed, loss function will be expressed as a negative log likelihood function. It can be represented as follows:

$$l(T|w, \gamma) \propto \exp(-l(T; w, \gamma)) \tag{6.2}$$

Using above relation 6.2 pseudo-posterior function can be determined for this scenario. The posterior distribution of γ denoted as $\pi(\gamma|T, w)$ can be generated, using the prior knowledge $\pi(\gamma)$ and new likelihood function as a loss function. It is developed as follows:

$$\pi(\gamma|T,w) \propto \exp(-l(T,w,\gamma))\pi(\gamma) \tag{6.3}$$

The aim is to propose new value of γ as γ^*. There is a need to move from state γ to γ^*. To do this task, MCMC - Metropolis Hastings algorithm is used. Using MH algorithm the samples of γ are generated. Let say there are n weight, so it requires n samples of γ such that $\gamma = [\gamma^1, \gamma^2, \gamma^3, ...\gamma^n]$. Also, each γ_i correspond to the respective weight w_i. It is required to go from state γ^t to γ^{t+1}. A new activation sample γ^{t+1} is proposed such that $\gamma^{t+1} \sim p(\cdot|\gamma^t)$. The new value of γ^{t+1} is decided by its previous sample γ^t.

The first requirement of the MH algorithm is to generate acceptance probability function. The values of γ which satisfy this criterion will be accepted else can be rejected. Let the initial state of activation sample be denoted as γ^t and a new value be denoted as γ^*. Before deriving acceptance probability, it is necessary to compute the posterior probability of γ^*. By using 6.3 it can be derived as :

$$\pi(\gamma^*|T,w) \propto \exp(-l(T,w,\gamma^*))\pi(\gamma^*) \tag{6.4}$$

Using equations 6.3 and 6.4 the probability of acceptance is computed as follows:

$$p(\gamma^t,\gamma^*) = \frac{\pi(\gamma^*|T,w)}{\pi(\gamma^t|T,w)} = \frac{\exp(-l(T,w,\gamma^*))\pi(\gamma^*)}{\exp(-l(T,w,\gamma^t))\pi(\gamma^t)} \tag{6.5}$$

Solving the above equation 6.5 to simplified form:

$$p(\gamma^t,\gamma^*) = \exp[-l(T,w,\gamma^*) + l(T,w,\gamma^t)]\frac{\pi(\gamma^*)}{\pi(\gamma)} \tag{6.6}$$

Therefore γ^* is accepted as new γ^{t+1} with acceptance probability:

$$\alpha(\gamma^t,\gamma^*) = \min(1, p(\gamma^t,\gamma^*)) \tag{6.7}$$

After the samples are generated, the study of the distribution of new activation γ^* is performed. This inference will help in understanding active weights and pruned weights as all γ correspond to the weight w. The value of γ can be as follows:

$$\gamma_i = \begin{cases} 0, & \text{if } \textit{pruned} \\ 1, & \text{otherwise} \end{cases} \tag{6.8}$$

Here, $i = 1 \ \textit{to} \ n$.

To sum up the model, the samples of γ are generated for every weight w. The new sample γ^{t+1} can be computed from γ^t. The new sample γ^{t+1} is accepted as γ^* if it satisfies the probability of acceptance criteria. If this criterion is not satisfied for γ^*, then this sample is rejected and next sample is computed. Thus, the efficient samples are obtained, as with the introduction of new samples, the defined loss function keeps on decreasing and the efficiency of model increases.

6.2 Proposed Algorithm

The task in this research is to compress the Neural network by pruning redundant weights. For this task, an ANN model will be used that can classify images. The Neural network model will be implemented to classify two categories of images. The aim of image classification technique is to classify images correctly as per the classes. It requires a binary indicator which denotes the presence of an object in an image. The algorithm designed will be based on Generalized Bayesian Update and Proposed methodology described in previous sections.

The detailed method of the algorithm is as follows:

- Step 1: Perform data pre-processing task on image data set. The image information needs to be converted into the data compatible with the model.
- Step 2: Split data into train, test and validation data-set.
- Step 2: Develop a Neural network model N, to classify the images.
- Step 3: Perform model analysis and accuracy analysis.

- Step 4: Calculate the loss function value as L.
- Step 5: For each generated weights w_i of the model, assigns an activation switch function γ_i with its initial value as 1, such that $\gamma = [1, 1, 1, 1, ...]$ for all weights.
- Step 6: In this step, the decision regarding which weights can be pruned is taken by the analysis of activation switch function γ using MH algorithm. The new weights samples w^* are generated, such that $w_i^* = \gamma_i^* * w_i$. This will be done via, element-wise multiplication. Iterate the following steps as per the number of weights (for example, if the number of weights in all layers is 10,000, the iteration will be 10,000).
  - In each iteration, randomly turn off one of the activation switch ($\gamma_i = 0$). Hence, the corresponding weight will be pruned ($w_i = 0$) to generate w^*.
  - Using the new weights w^*, perform model analysis and calculate the new loss function L^* .
  - Now, it is required to decide whether to accept or reject these new weights. To do so generate a uniform random number between 0 and 1 as u.
  - Compute the probability of acceptance, α such that $\alpha = \min(1, e^{-L^*}/e^{-L})$.
  - If the acceptance probability for new weights w^* is greater than the uniform random number generated u, then the sample is accepted as new weight and the loss function is calculated L^* as L, that is $L = L^*$.
  - If the above criterion is not satisfied then the weights are rejected and the new samples for weights is computed, that is return to Step 6.
- Step 7: The new weights will be calculated as per the distribution of γ^* obtained from step 6. The model will have pruned weights as per the distribution.
- Step 8: Pruning is calculated and a new model N^* is generated.

The new model generated N^*, will have some of the weights as 0 ($w_i = 0$), that is they are pruned.

Chapter 7

Application of Algorithm

7.1 Simulated Dataset

The proposed methodology needs to be tested on small sample features. To verify the this approach, random pixel values were simulated for two class of images. Then the data was transferred in a CSV file. 600 sample images were generated for each of the two classes. The details of the pixel values are as specified below.

The class 1 images had pixel value with a normal distribution where mean $\mu_1 = +1$ and standard deviation $\sigma = 0.5$. For this configuration 600 samples were generated. There were 784 pixels, hence all the pixels had a value from this distribution. Similarly, for class 2 images, samples were generated with normal distribution having $\mu_2 = -1$ and standard deviation $\sigma = 0.5$. Likewise, all 1200 samples were generated and used for model building.

These generated images were then split into train, test and validation dataset in the ratio of 60:25:15. The training dataset was subjected to a feed-forward Neural network model N. The constructed model had 5 hidden nodes and 2 output nodes. 'Softmax' activation function was used at the output. The learning rate was fixed as 0.01. The model was trained with the training accuracy of 100% . Since, both the classes of images can be separated distinctly, the model could easily classify both the images.

On this trained model, test dataset was tested. The accuracy on test dataset was also 100%. Now, the proposed algorithm was implemented on the model. The algorithm provided new weights w^* and new network N^*. It was found that the new network was pruned by 99% compared to the base network N. This was the behavior observed, as the input values can be easily distinguished for both the classes. This was the case when both classes of images were uniquely separated.

The next step was to increase the standard deviation while simulating sample images, such that both the images would have some overlapping pixel values. The standard deviation was increased in steps of 0.5 until 40. One of the configuration of normal distribution for class 1 images was $\mu_1 = +1$ and $\sigma = 30$, whereas for class 2 images were generated with $\mu_2 = -1$ and $\sigma = 30$. Now the generated images for both the classes will have some overlapping values of the pixel. The model N_2 was constructed with the similar configuration as N_1. The training accuracy for model N_2 was calculated as 70%.

The generated model N_2 was then used for testing the test dataset. The accuracy on test dataset dropped down to 65%. This happened as the pixel values were not straight forward. Then algorithm was implemented on N_2 model to generate new model N_2^*. This method could prune 30% of the weights compared to the base model N_2 and providing an accuracy of 64% on the test dataset.

7.1.1 Result Analysis

As the standard deviation for the sample generation was increased, it was found that pruning of the network substantially decreased. For simple sample data with a standard deviation of 1, the pruning of 99% was achieved, whereas when the standard deviation was increased to 40, the pruning of the network was observed to be 29.18%. The Table 7.1 below, shows some values for various levels of standard deviation. The results for all values of standard deviation is shown in Appendix A.1. The Figure 7.1 shows the effect of increasing standard deviation on pruning.

Standard deviation	Pruning(%)	Loss Before Pruning	Loss After Pruning	Reduction in loss after pruning(%)
1	99.336	0.133	0.133	0
2	98.137	0.146	0.146	0
5	89.872	0.144	0.144	0
10	58.954	1.936	1.851	4.399
15	40.994	6.804	6.445	5.278
20	36.556	15.699	15.501	1.262
25	35.969	22.053	20.608	6.554
30	29.795	28.56	26.003	8.952
35	36.862	30.167	27.399	9.173
40	29.183	37.764	36.245	4.023

Table 7.1: Simulation Results

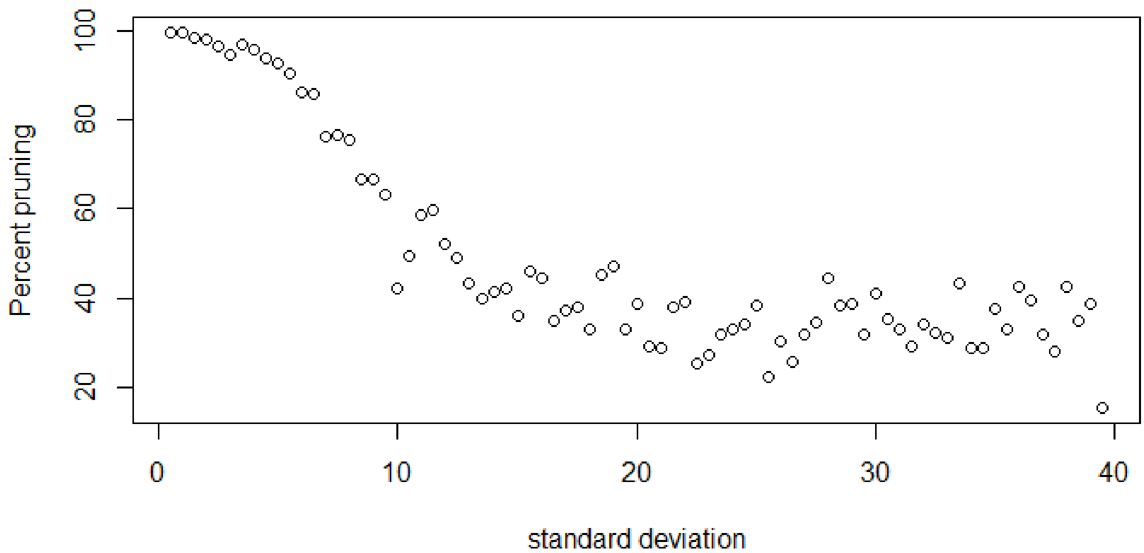

Figure 7.1: Plot of standard deviation vs pruning

It was observed that, even after increasing the complexity of data, the network is being pruned. The pruning decreases slowly after $\sigma = 10$ and it suggests the possibility of pruning a complex network. It also suggests that there is some measurable redundancy in the network. When the data is simple, that is when it can be separated using a single hyperplane, pruning is maximum. The network needs very few parameters to differentiate two classes. On the other hand, as the standard deviation for sampling images was increased, the samples got more complex and both classes had overlapping samples. Thus, such images cannot be differentiated using hyperplane, hence the network needs more parameters. This justifies the decrease in pruning with an increase in the standard deviation of samples. Thus, the hypothesis was tested on the simulated samples of data. The following sections will provide detail information with real-world complex dataset.

7.2 Real-world Dataset

7.2.1 Dataset

The dataset tested in this research is downloaded from Kaggle dataset library which is an open-source website. This library consists of various datasets for classification tasks. An image dataset was selected as the ANN model will be built to classify images. The dataset consists of 10000 images. Every image had either a cat or a dog image explicitly. Only one figure of a cat or a dog is present in the image. All the images present were colored images and in jpg format. The sample images are shown in Figure 7.2 below.

7.2.2 Data Pre-processing

Before implementing a model on these images, it needs to be processed in a format which a model can decode. Initially, all the images were re-sized to 28 x 28 pixels. The images were later normalized using image contrast normalization techniques, to improve the robustness against illumination changes. At last, the images were converted to gray-scale images in order to avoid information overload in each pixel. Now all the

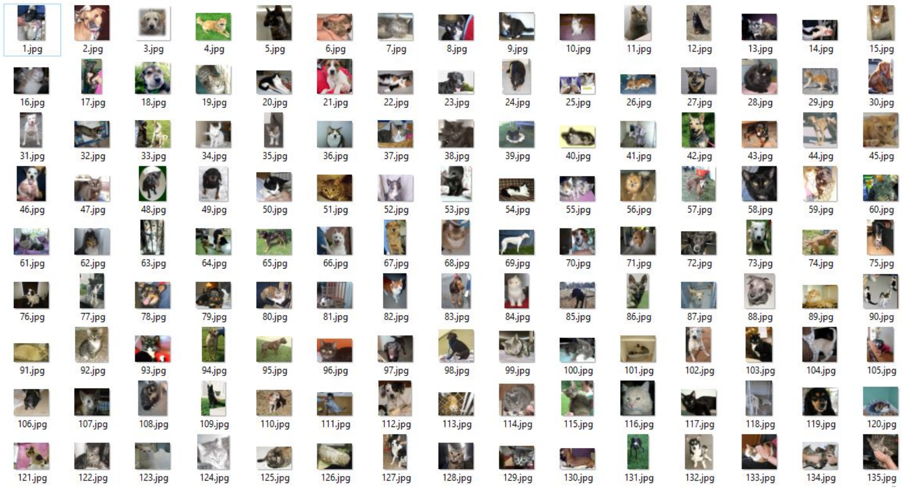

Figure 7.2: Samples of cats and dogs from Image dataset

pixels will have a value in the shades of gray. Later, the pixel value of these images was transferred into a CSV file. The CSV file contained 784(28x28) columns which correspond to each pixel value. The other task performed after transferring pixel data to CSV file is category labeling. Along with the pixel value, another column of the class label was added. It contained information about the category of the image. Since, the 2 types of images namely cat and dog were examined, it had 2 categories 0 and 1 which corresponds to cat and dog, respectively.

7.2.3 Train-Test Split

The first step performed after the dataset was retrieved was to remove all the duplicates. The images were grouped by 'date modified' and 'user' and the scan for duplicate images was performed. It was found all the images were unique. To analyze the model, the entire dataset was divided into 3 parts namely training, test and validation dataset in the ratio 60:25:15. The general convention is 60:20:20 split, since the pruning will be mainly decided using test data, in this study an edge of 5% data has been given to the test data compared to validation dataset. Thus, the training dataset consists of 6000 images, whereas testing dataset comprised of 2500 images and validation dataset had around 1500 images. The idea of having a validation dataset was to confirm the credibility of the model.

7.2.4 Evaluation Metrics

The dataset used in this task has two classes of images. The two categories are evenly distributed hence, 'accuracy' was used as one of the measures of evaluation. It will measure the ratio of correct predictions given the total predictions. The metric of prime interest in this study is 'loss function'. The efficiency of the model will be defined by the value of loss function it generates for a specific combination of weights.

7.2.5 Model Building and Analysis

The Neural network model was built using R software by means of Apache MXNet package. MXNet package was selected as it provides high performance and low-level control with the features like automatic differentiation and optimized predefined layers. It has efficient computational power compared to NeuralNet package in R.

The first step in model building is to make data compatible to the MXNet requirement, hence the training dataset was first split into two data frames, one contained only class column whereas others contained all pixel columns. These two data frames were then converted into a matrix as MXNet requires X and Y category data in the form of a matrix. The hyper-parameters selected for the model are as follows.

- Hidden layers: 1 hidden layer consisting of 10 hidden nodes was implemented in the model. Each node had incoming links 784 weights which corresponds to the input pixels data.
- Output nodes: Since the dataset contained 2 class of images, output nodes were selected as 2.
- Activation: 'Relu' activation function was used between the hidden layer and output layer, whereas for the output layer 'Softmax' activation was used. Hence,

the output layer had 2 nodes.

- Learning rate: For the better performance of the model, the 'ADAM' optimizer technique was used. It is a customized algorithm for gradient-based optimization of stochastic objective functions [30].

- Epoch and batch size: The model was constructed with a batch size of 50 and the number of training epoch was 100.

Using the above mentioned parameters, the Neural network model N was constructed. It was discovered that the training accuracy of the model was around 79.31%. The entire network was now ready for testing. The fully functional network consists of 7860 weights. This seems to be a computationally large network for the task of classifying cats and dogs. The main aim is to prune the network without affecting the performance. For understanding the difference in the performance of the compressed network, the fully functional Neural network should be tested on test data. In the initial test on the test data, the model had an accuracy of 62.37% with the initial network loss L of 362.96.

The idea is to prune the network and study the dominant nodes for classification of cats and dogs respectively. The weights corresponding to each node were denoted by $w = [w_1, w_2, w_3, \ldots, w_n]$. For each of these node an activation switch function was introduced such that $\gamma = [\gamma_1, \gamma_2, \gamma_3, ..., \gamma_n]$ corresponding to each node. Initially, all the γ are initialized to 1. The idea is to identify the redundant weights, so that the value of γ for that weight can be made to 0. This is a replica of switch pattern in order to turn on or off the switch with respect to the data. The new weights w^* of the network can simply be given by the product of w and γ^*.

After initializing γ_i to 1, the aim was to propose a new γ^* such that it provides the value of new weights w^*. These new weights are calculated using a Bayesian approach. Initially, prior information available was initial weights w, initial activation γ, test data, loss function. It is possible to find the posterior value of the weights based on the data. As explained in the preceding chapters, MH algorithm was used to generate samples. With initial loss as L and new loss as L^*, the probability of acceptance was calculated as

$\min(1, \exp(-L^*)/\exp(-L))$. This introduced algorithm was implemented on 3 variants of the test dataset. The first dataset was used for understanding pruning effect on the network, whereas the other two datasets is used to throw light on activation patterns of nodes for cats and dogs respectively.

7.2.6 Study of Pruning Rate

The dataset comprised of both cat and dog images. This test dataset is mainly used for understanding the pruning rate of the network. The original model N was tested on this dataset, which provided an accuracy of 62.37% with the initial network loss L of 362.963. After implementing the proposed algorithm on this dataset, the new distribution of γ^* was obtained. It resulted in better understanding of pruned weights. The new weights w^*, were given as $w_i^* = w_i * \gamma_i^*$. These new weights accounted for the new network N^*. The new network provided the accuracy of 61.78% whereas the loss was reduced by 11.51%. The newly generated Neural network model N^* was also tested on the validation dataset. It provided a validation accuracy of 61.37%. When all the samples were generated the initial network N was compared with new network formed N^* for the weights. It was found that network N^* was pruned by 41.95%. It was observed that the majority of the pruned weights were present between the input layer and the hidden layer. Few weights between the hidden layer and the output layer were pruned.

These result can be summarized below, the Table 7.2 shows the number of weights pruned in every layer. The Table 7.3 shows the performance of the model before and after pruning.

	Total weights	Pruned weights	Pruned weights (%)
Input Layer to Hidden Layer	7840	3296	42.04
Hidden Layer to Output Layer	20	2	10
Total Pruning	7860	3298	41.95

Table 7.2: Layerwise weights pruning

Pruning (%)	Before Pruning Accuracy (%)	After Pruning Accuracy (%)	Before Pruning Loss	After Pruning Loss	Loss Reduction (%)	Validation accuracy (%)
41.95	62.37	61.78	362.963	321.172	11.513	61.37

Table 7.3: Pruning Results

7.2.7 Study of Activation Pattern for Cats

The test and validation dataset used for this task consisted only of cats images. It is used to study the activation patterns of weights which are used for classification of cats images after pruning. The reference model N was built using training dataset with accuracy and loss value as 65.21% and 171.18 respectively. Here, the new model N^* was constructed using the activation function values γ^*, which are generated using the proposed algorithm. The outcome of the algorithm provided the new Neural network model N^*. Since only cat samples were present in the test dataset, the algorithm generated samples of γ^* which can correctly classify cats. The model N^* provided an accuracy of 100%. Also, the loss function value was reduced by 99.99%. This new model N^* was tested for validation dataset, the validation accuracy was found to be 99%.

In order to understand the activation patterns, the pruning in the new network N^* obtained after operations was determined. The Neural network N^* was pruned by 71.39% compared to the base network N. After examining the pruned weights, it was found that pruned weights ($w_i^* = 0$) were present in both the types of links between the input layer to hidden layer and also in the hidden layer and output layer. The distribution was studied to find out that weights from hidden layer nodes number $1, 4, 5, 6, 9, 10$ to output nodes can be switched off, as the weights $w_1^h = 0, w_4^h = 0, w_5^h = 0, w_6^h = 0, w_9^h = 0, w_{10}^h = 0$. It was observed that weights corresponding to the nodes $2, 3, 7, 8$ are used for the classification of cats. These weights and node are significant in the classification of cats.

The summarized results for this implementation are shown below. The Table 7.4 shows the layerwise pruning while detecting class 0 (cats). The Table 7.5 shows the performance of model for class 0 (cats) data.

	Total weights	Pruned weights	Pruned weights (%)
Input Layer to Hidden Layer	7840	5599	71.41
Hidden Layer to Output Layer	20	12	60
Total Pruning	7860	5611	71.39

Table 7.4: Layerwise weights pruning for cats

Pruning (%)	Before Pruning Accuracy (%)	After Pruning Accuracy (%)	Before Pruning Loss	After Pruning Loss	Loss Reduction (%)	Validation accuracy (%)
71.39	65.21	100	171.18	0.003	99.997	99

Table 7.5: Pruning Results for class 0 (cats)

The activation pattern nodes for the classification of class 0 (cat) is shown below in Figure 7.3. These are the activated nodes in the hidden layer responsible for detecting cats .

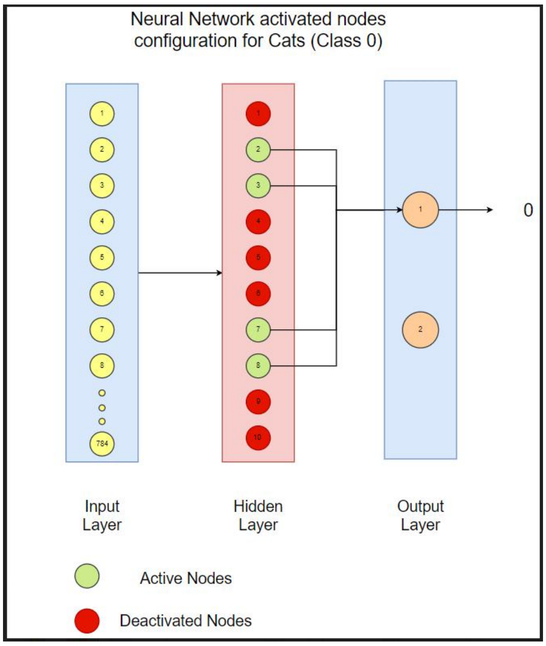

Figure 7.3: Activated nodes for cats

7.2.8 Study of Activation Pattern for Dogs

In this section, the testing and validation dataset consisted of images restricted to dogs only. This dataset was used to study the activation patterns of weights for dogs. Here, the original network N as the base network with accuracy and loss value as 63.19% and 181.203 respectively. As per the method, the new model N^* was generated using the proposed algorithm. Similar to the case of cats, since the test dataset consists of dog images, the algorithm, accept the samples of γ^* which can classify dogs correctly. The optimized model N^* provided an accuracy of 100%. The loss function was decreased by 99.99%. This model was also validated on validation dataset, and the validation accuracy was found to be 99%.

To study about activation patterns for dogs, there is need to understand the pruned weights of the network N^*. After the careful examination and completion of execution of the algorithm, it was concluded that the network N^* was pruned by 72.92%. The pruned weights were distributed in all the layers. The pruning of weights showed that in hidden layer nodes number $2, 3, 4, 8, 10$ can be neglected, as the weights from these nodes to output node are 0, that is $w_2^h = 0, w_3^h = 0, w_4^h = 0, w_8^h = 0, w_{10}^h = 0$. It implies that the weights corresponding to the nodes $1, 5, 6, 7, 9$ are prime nodes for the classification of dogs.

This approach results are summarized as below. The Table 7.6 shows the layerwise pruning while detecting class 1 (dogs). The Table 7.6 shows the performance of model for class 1 (dogs) data.

	Total weights	Pruned weights	Pruned weights (%)
Input Layer to Hidden Layer	7840	5722	72.98
Hidden Layer to Output Layer	20	10	50
Total Pruning	7860	5732	72.92

Table 7.6: Layerwise weights pruning for dogs

The activation pattern nodes for the classification of class 1 (dog) is shown below in Figure 7.4. This figure, the nodes responsible for detecting dogs.

Pruning (%)	Before Pruning Accuracy (%)	After Pruning Accuracy (%)	Before Pruning Loss	After Pruning Loss	Loss Reduction (%)	Validation accuracy (%)
72.92	63.19	100	181.203	0.001	99.999	99

Table 7.7: Pruning Results for class 1 (dogs)

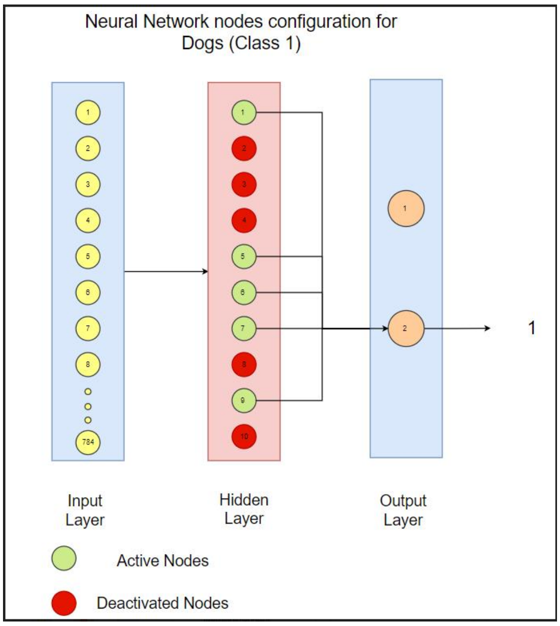

Figure 7.4: Activated nodes for dogs

7.2.9 Result Analysis

It can be seen from the results, the network is pruned by significant value. The pattern recognition analysis shows the nodes important for the classification of cats and dogs. When the cats and dogs images are given to the model, individually the network is pruned by almost 70%. As seen from the activated nodes, the nodes for classification of both the classes are different. Only node number 7 is common for both classes. Hence, it is the most important node. Also, it can be seen that node 4, is the least significant node. Hence, the results provides us the node importance and considerable pruning.

Chapter 8

Conclusion and Future work

8.1 Conclusion

In this research, a formal pruning algorithm to compress hidden units in a multi-layered ANN is introduced. This algorithm is influenced by general Bayesian update methods and MCMC-Metropolis Hastings algorithm. It can efficiently prune the redundant weights in the network. The method makes use of the redundancy of the network to propose the configuration of the new weight. The proposed methodology is general purpose, and it could be applied to any distinct network configurations. As suggested from the two applications described in Chapter 7, it was observed that the pruning is maximum when the data is classified easily with less loss of the model, for example, the images could be separated using a single hyper-plane. As the complexity of data is increased in the simulated example, that is the data can no more be separated by a hyper-plane as both the class of data have overlapping values. It was noticed that the pruning gets reduced considerably.

This method is a promising proof of concept. As the pruning results of the application example 2 showed the pruning of 40%. The weights of the network which are significant for the classification of two classes are retained. The obtained results roughly suggest that the pruning in an ANN model is depended on the complexity of the training data and the complexity of the network configuration. Besides, pruning it was possible to study the parts of the network which are activated for a different type of input data. The results clearly shows that all nodes are not required for the classification of 2 classes individually. Both the class of data have a different configuration of hidden nodes in the network.

This algorithm could prune the network to a significant extent with a considerable decrease in the loss function of the pruned network. The advantages of the methods are that the method generates considerable pruning, loss of the network is decreased, the accuracy of the network is not affected significantly and also it is an adaptive method which exploits network redundancy. On the other hand, some of the disadvantages of this algorithm could be the method can be expensive as it involves multiple iterations to prune the weights. As the network gets complex, the weights will increase and so is the cost of the algorithm.

8.2 Future work

The engaging extensions of the work could be: (i) to implement this method on convolution Neural network and check the performance, (ii) introduction of method to estimate and prune the weights during back-propagation, (iii) to test the network performance on highly complex Neural network, involving multiple hidden layers like VGG and ResNet. (iv) Also to study on the weights configuration of the network and its geometric patterns.

Bibliography

[1] R. Wason, "Deep learning: Evolution and expansion," *Cognitive Systems Research*, vol. 52, pp. 701–708, 2018.

[2] A. Sevik, P. Erdogmus, and E. Yalein, "Font and turkish letter recognition in images with deep learning," in *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, pp. 61–64, Dec 2018.

[3] W. B. McCulloch, W.S. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[4] O. Elitez, *HANDWRITTEN DIGIT STRING SEGMENTATION AND RECOGNITION USING DEEP LEARNING A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF MIDDLE EAST TECHNICAL UNIVERSITY*. 2015.

[5] T. Zhang, C. L. P. Chen, C. Wang, and S. C. Tam, "A new dynamic optimal learning rate for a two-layer neural network," in *2012 International Conference on System Science and Engineering (ICSSE)*, pp. 55–59, June 2012.

[6] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295 – 307, 1988.

[7] Sang-Hoon Oh, "Improving the error backpropagation algorithm with a modified error function," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 799–803, 1997.

[8] Tin-Yau Kwok and Dit-Yan Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Transactions on Neural Networks*, vol. 8, pp. 630–645, May 1997.

[9] Y. Lecun, J. Denker, and S. Solla, "Optimal brain damage," vol. 2, pp. 598–605, 01 1989.

[10] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, pp. 293–299 vol.1, March 1993.

[11] P. V. S. Ponnapalli, K. C. Ho, and M. Thomson, "A formal selection and pruning algorithm for feedforward artificial neural network optimization," *IEEE Transactions on Neural Networks*, vol. 10, pp. 964–968, July 1999.

[12] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Advances in Neural Information Processing Systems 1* (D. S. Touretzky, ed.), pp. 107–115, Morgan-Kaufmann, 1989.

[13] Han Honggui and Qiao Junfei, "A novel pruning algorithm for self-organizing neural network," in *2009 International Joint Conference on Neural Networks*, pp. 1245–1250, June 2009.

[14] R. Setiono and W. K. Leow, "Pruned neural networks for regression," in *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*, PRICAI'00, (Berlin, Heidelberg), pp. 500–509, Springer-Verlag, 2000.

[15] Weishui Wan, K. Hirasawa, Jinglu Hu, and Chunzhi Jin, "A new method to prune the neural network," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 6, pp. 449–454 vol.6, July 2000.

[16] M. R. Silvestre and L. L. Ling, "Optimization of neural classifiers based on bayesian decision boundaries and idle neurons pruning," in *Object recognition supported by user interaction for service robots*, vol. 3, pp. 387–390 vol.3, Aug

2002.

[17] J. Chin, “Bayesian Analysis,” 2018.

[18] D. Sorensen and D. Gianola, *Likelihood, Bayesian, and MCMC Methods in Quantitative Genetics.* Springer, New York, NY, 2002.

[19] Z. Pawlak, “Rough sets, decision algorithms and bayes' theorem,” *European Journal of Operational Research*, vol. 136, no. 1, pp. 181–189, 2002.

[20] N.-Z. Sun and A. Sun, *Model calibration and parameter estimation: For environmental and water resource systems.* 01 2015.

[21] T. Bayes, “An essay towards solving a Problem in the Doctrine of Chances,” 2018.

[22] A. P. Dempster, “A generalization of bayesian inference,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 30, no. 2, 2008.

[23] P. G. Bissiri, C. C. Holmes, and S. G. Walker, “A general framework for updating belief distributions,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 78, p. 1103–1130, Feb 2016.

[24] J. M. Bernardo, M. H. DeGroot, D. V. Lindley, and A. F. M. Smith, “Bayesian statistics 2.,” *Biometrics*, vol. 43, p. 257, Mar 1987.

[25] D. van Ravenzwaaij, P. Cassey, and S. D. Brown, “A simple introduction to markov chain monte–carlo sampling,” *Psychonomic Bulletin & Review*, vol. 25, pp. 143–154, Feb 2018.

[26] C. Robert and G. Casella, “A short history of markov chain monte carlo: Subjective recollections from incomplete data,” *Statistical Science*, vol. 26, p. 102–115, Feb 2011.

[27] I. Beichl and F. Sullivan, “The metropolis algorithm,” *Computing in Science Engineering*, vol. 2, pp. 65 – 69, 02 2000.

[28] C. P. Robert and G. Casella, “The metropolis—hastings algorithm,” in *Monte*

Carlo Statistical Methods, pp. 231–283, Springer, 1999.

[29] G. O. Roberts, A. Gelman, and W. R. Gilks, “Weak convergence and optimal scaling of random walk metropolis algorithms,” *The Annals of Applied Probability*, vol. 7, no. 1, pp. 110–120, 1997.

[30] D. ”Kingma and J. Lei Ba, *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*.

Appendix A

Appendix

A.1 Simulation Results

The below table shows the complete results of the simulation application. The standard deviation was increased from 0.5 to 40 in the steps of 0.5 and corresponding pruning was analyzed

Standard Deviation	Pruning (%)	Before Pruning Loss	After Pruning Loss	Reduction in loss (%)
1	99.336	0.133	0.133	0
1.5	99.082	0.136	0.136	0
2	98.137	0.146	0.146	0
2.5	97.015	0.133	0.133	0
3	96.173	0.140	0.140	0
3.5	95.587	0.142	0.142	0
4	94.796	0.143	0.143	0
4.5	88.112	0.145	0.145	0
5	89.872	0.144	0.144	0
5.5	87.526	0.167	0.167	0
6	89.617	0.173	0.173	0

6.5	85.000	0.176	0.176	0
7	86.276	0.254	0.254	0
7.5	80.000	0.220	0.220	0
8	72.934	0.479	0.407	15.013
8.5	58.827	0.590	0.590	0.000
9	53.444	0.275	0.275	0.000
9.5	67.985	2.111	1.931	8.520
10	58.954	1.936	1.851	4.399
10.5	54.770	2.180	2.180	0
11	54.107	3.030	2.943	2.869
11.5	47.041	4.282	4.013	6.288
12	55.536	4.438	3.973	10.481
12.5	51.837	4.066	2.467	39.340
13	43.367	4.464	3.536	20.784
13.5	48.597	5.930	5.615	5.315
14	44.898	5.581	4.859	12.931
14.5	46.582	6.066	5.925	2.319
15	40.994	6.804	6.445	5.278
15.5	52.372	8.427	8.113	3.720
16	45.918	9.046	8.319	8.038
16.5	43.546	8.357	7.402	11.435
17	37.985	8.530	8.291	2.798
17.5	41.454	8.561	7.499	12.403
18	45.842	14.989	13.659	8.873
18.5	30.204	12.708	11.693	7.983
19	33.418	12.314	10.777	12.483
19.5	34.056	14.982	13.320	11.096
20	36.556	15.699	15.501	1.262
20.5	49.413	20.379	18.742	8.033
21	32.653	13.931	13.483	3.218
21.5	32.577	18.029	15.429	14.422

22	33.214	20.646	18.624	9.793
22.5	24.949	18.640	16.833	9.699
23	37.245	17.834	16.965	4.870
23.5	31.148	17.195	16.487	4.116
24	31.224	21.481	19.136	10.917
24.5	39.388	21.690	20.234	6.713
25	35.969	22.053	20.608	6.554
25.5	31.378	22.200	20.575	7.322
26	32.219	22.189	20.240	8.784
26.5	33.316	21.199	19.215	9.361
27	34.464	22.883	20.926	8.553
27.5	28.418	23.601	21.287	9.807
28	26.352	24.354	20.283	16.716
28.5	26.709	27.014	24.878	7.908
29	33.546	25.088	21.445	14.520
29.5	19.694	25.412	20.054	21.085
30	29.795	28.56	26.003	8.952
30.5	31.939	26.435	24.177	8.543
31	36.480	27.407	24.209	11.671
31.5	30.281	22.512	20.468	9.078
32	35.230	26.808	23.497	12.350
32.5	24.974	24.734	22.303	9.829
33	36.378	30.185	27.469	8.999
33.5	24.235	28.416	24.417	14.076
34	38.750	32.156	30.110	6.365
34.5	28.801	31.989	29.345	8.265
35	36.862	30.167	27.399	9.173
35.5	32.168	30.891	28.510	7.708
36	25.459	28.811	25.204	12.519
36.5	37.066	30.858	26.749	13.316
37	32.117	31.345	28.360	9.523

37.5	35.791	34.163	32.327	5.377
38	32.832	32.395	28.724	11.332
38.5	36.480	33.745	29.685	12.030
39	21.786	32.506	29.818	8.269
39.5	23.087	35.632	33.532	5.894
40	29.183	37.764	36.245	4.0233

Table A.1: Complete Simulation Results

A.2 List of Abbreviations

ANN	Artificial Neural Network
NN	Neural Network
FP	Forward Propagation
BP	Backward Propagation
GPU	Graphical Processing Units
OBD	Optimal Brain Damage
NP-FNN	Novel Pruning Feed-forward Neural Network
N2PFA	Neural Network Pruning for Function Approximation
PDF	Probability Density Function
MLE	Maximum Likelihood Estimate
MCMC	Markov Chain Monte Carlo
MH	Metropolis-Hastings