T.R.
MİMAR SİNAN FINE ARTS UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY

HYBRID BAYESIAN NEURAL NETWORKS WITH GENETIC ALGORITHMS AND FUZZY MEMBERSHIP FUNCTIONS

Ph.D. Thesis by

Ozan KOCADAĞLI, M.Sc.

Department of Statistics

Programme of Statistics

Supervisor: Prof. Dr. Nalan CİNEMRE
Co-Supervisor: Prof. Dr. Refik SOYER

March 2012

This Thesis titled HYBRID BAYESIAN NEURAL NETWORKS WITH GENETIC ALGORITHMS AND FUZZY MEMBERSHIP FUNCTIONS prepared by Ozan KOCADAĞLI is approved as Doctoral Thesis.

Prof. Dr. Nalan CİNEMRE

Thesis Advisor

This study is accepted as Doctoral Thesis in the Department of Statistics by Thesis Committee.

Chair: Prof. Dr. Nalan CİNEMRE

Member : Prof. Dr. Gülay KIROĞLU

Member : Prof. Dr. Aydın ERAR

Member : Assoc. Prof. Dr. Tuncay CAN

Member : Assoc. Prof. Dr. Eyüp ÇETİN

This Thesis is submitted in accordance with the requirements of Thesis of Mimar Sinan Fine Arts University, The Institute of Science and Technology.

HYBRID BAYESIAN NEURAL NETWORKS WITH GENETIC ALGORITHMS AND FUZZY MEMBERSHIP FUNCTIONS

ABSTRACT

The aim of this study is to improve the novel algorithms, which estimate the models that represent accurately to factors of dynamic and nonlinear systems in the fuzzy and stochastic environment. In dynamic systems, modeling with possibilistic and probabilistic distribution to uncertainties included in data set allows to more robust analysis. In nonlinear systems, the pre-knowledge about the functional structure between inputs and outputs is either unavailable or insufficient. In such situations, the neural networks are useful tools to determine the functional structure between inputs and outputs. However, the traditional neural networks with mean squared errors suffer from the approximation and estimation errors. These errors can be decreased by the Bayesian neural networks simultaneously, since Bayesian learning provides a consistent way to penalize the excessive complex models.

In this study, Monte Carlo (MC) algorithms used in Gaussian approach with recursive hyperparameters and full Bayesian approach of Bayes Neural Networks are hybridized with Genetic Algorithms (GA) and the fuzzy membership functions. Besides, to evaluate fuzzy uncertainty in MC and GA processes, the fuzzy membership functions are used. Thus, the novel hybrid Bayes learning approaches, which effectively estimate parameters and hyperparameters of Bayes Neural Networks, are improved. The software of the improved algorithms is written in MATLAB package program.

In application parts, the performances of the improved approaches are compared with ones of traditional approaches, and then outcomes are discussed.

GENETİK ALGORİTMALAR VE BULANIK ÜYELİK FONKSİYONLARIYLA HİBRİT BAYES YAPAY SİNİR AĞLARI

ÖZET

Bu çalışmanın amacı, dinamik ve doğrusal olmayan sistemlerin faktörlerini bulanık ve stokastik ortamda en iyi biçimde temsil edecek modellerin kestiriminde kullanılacak özgün algoritmaların geliştirilmesidir. Dinamik sistemlerde, veriler anlık olarak değerlendirildiklerinden verinin hem olasılık dağılımının hem de olabilirlik dağılımının birlikte ele alınması daha hassas sonuçların elde edilmesini sağlayacaktır. Ayrıca, doğrusal olmayan sistemlerde giriş ve çıkış değişkenleri arasındaki fonksiyonel yapı hakkındaki ön bilgi ya yoktur ya da çok azdır. Böyle durumlarda yapay sinir ağları giriş ve çıkış değişkenleri arasındaki fonksiyonel yapıyı belirlemek için oldukça kullanışlı araçlardır.

Bu çalışmada, Bayes yapay sinir ağlarının yinelenen hiper-parametreli normal yaklaşımında (Gaussian approach with recursive hyperparameters) ve tam Bayes (full Bayesian approach) yaklaşımında kullanılan Monte Carlo (MC) algoritmaları, bulanık üyelik fonksiyonları ve Genetik Algoritmalar (GA) ile hibritleştirilmiştir. Ayrıca, GA ve MC işlevleri içinde bulanık belirsizliği ölçmek için bulanık üyelik fonksiyonlarından yararlanılmıştır. Böylece, Bayes YSA nın parametre ve hiper-parametrelerini daha etkin bir biçimde kestirmek için hibrit Bayes öğrenim yaklaşımları geliştirilmiştir.

Uygulama bölümünde, Bayes yapay sinir ağları için önerilen öğrenme algoritmalarının performansları geleneksel yapay sinir ağlarınınkiyle karşılaştırılarak sonuçlar tartışılmıştır.

ACKNOWLEDGMENT

It would not have been possible to write this doctoral thesis without the help and support of the kind people around me, so I would like to give particular mention here only some of whom.

First and foremost, I would like to acknowledge the advice and guidance of Prof. Dr. Nalan CİNEMRE, my advisor in the Department of Statistics at Mimar Sinan F.A. University. I also thank the guidance and suggestions of Prof. Dr. Refik SOYER, my advisor in the Department of Decision Sciences at The George Washington University during my doctoral research in USA.

I would like to acknowledge The Scientific and Technological Research Council of TURKEY (TÜBİTAK) for Doctoral Research Scholarship that provided the necessary financial support for this research in the Department of Decision Sciences at The George Washington University in USA, and particularly Mimar Sinan F.A. University and The Council of Higher Education of TURKEY for the financial, academic and technical support.

My family has given me their hearty support throughout, as always, for which my mere expression of thanks even does not suffice, particularly I would like to thank my wife Aylin for her personal support and great patience at all times.

March, 2012 Ozan KOCADAĞLI

CONTENTS

INDEX OF FIGURES

INDEX OF TABLES

ABBREVIATIONS

GA: Genetic Algorithms

HMC: Hybrid Monte Carlo

MC: Monte Carlo

MCMC: Markov Chain Monte Carlo

SSE: Sum Squared Errors

MSE: Mean squared errors

CV: Cross-validation

OPERATORS AND FUNCTIONS

A'	Transpose of matrix A
$\|A\|$	Determinant of matrix A
$\|\|v\|\|$	Euclid distance of vector v from origin
$E(x)$	Expectation of the random variable x
$exp(\cdot)$	Exponential function
$\Gamma(\cdot)$	Gamma function
$log(\cdot)$	Natural logarithm
$arg\min_x Z(x)$	The argument z that minimizes the operand
$tanhx$	Hyperbolic tangent function

SYMBOLS

$\propto$	Proportional
$\approx$	Equal by asymptotically
$p(x)$	Distribution of x
$x \sim p(x)$	x is distributed according to $p(x)$
$p(y\|x)$	Conditional distribution of y given x
$p(x,y)$	Joint distribution of x and y
I_n	Identity matrix of dimensions $n \times n$
$Y_{1:N}$	Stacked vector => $Y_{1:N} \triangleq (Y_1, Y_2, \ldots, Y_N)$

1 INTRODUCTION

In Bayesian data analysis all uncertain quantities are modeled as probability distributions, and inference is performed by constructing the posterior conditional probabilities for the unobserved variables of interest, given the observed data sample and prior assumptions. The some references for Bayesian data analysis are (Berger, 1985; Bernardo and Smith, 1994; Gelman et al., 1995; Lindley, 2000).

The application of the Bayesian learning paradigm to neural networks results in a flexible and powerful nonlinear modeling framework that can be used for regression, density estimation, prediction and classification. Within this framework, all sources of uncertainty are expressed and measured by probabilities. This formulation allows for a probabilistic treatment of a priori knowledge, domain specification knowledge, model selection schemes, parameter estimation methods and noise estimation techniques.

For neural networks the Bayesian approach was pioneered in (Buntine and Weigend, 1991; MacKay, 1992; Neal, 1992) and reviewed in (MacKay, 1995; Neal, 1996; Bishop, 1995). With neural networks the main difficulty in model building is controlling the complexity of the model. It is well known that the optimal number of degrees of freedom in the model depends on the number of training samples, amount of noise in the samples and the complexity of the underlying function being estimated. For the standard neural networks techniques, determining the correct model complexity and setting up a network with the desired complexity are rather crude, and often computationally very expensive.

In the Bayesian approach these issues can be handled in a natural and consistent way. The unknown degree of complexity is handled by defining vague (non-informative) priors for the hyperparameters that determine the model complexity, and the resulting complexity is averaged over all model complexities weighted by their posterior probability given the data sample. The model can be allowed to have different complexity in different parts of the model by grouping the parameters that are exchangeable (have identical role in the model) to have a common hyperparameters.

Another issue in the neural network literature is estimation of parameters and hyperparameters of network. There are different parameter estimation procedures for Bayesian neural networks in literature. Gaussian Approximation (MacKay, 1992) and advanced Bayesian simulation approach (Neal, 1992) are well-known pioneers in Bayesian neural network jargon. In order to train Feed-Forward and Back-propagation neural networks by means of these approaches, Gradient, Evolutionary, Markov Chain Monte Carlo (MCMC), Simulated Annealing methods are mostly preferred. However, when the parameter space with high dimensions is searched by means of these methods, the certain shortcomings arise such as stuck in local optimums and training time. Therefore, analysts desire more active and fast search algorithms against the mentioned shortcomings. In circumstances, Genetic Algorithms (GA) provides rather fast search and accurate solutions for constrained or unconstrained nonlinear problems and parameter space with high dimensions.

Genetic Algorithms (GA) are developed by John Holland in 1970's are heuristic optimization methods based on concepts of natural evolution, and belongs to the larger class of evolutionary algorithms (Holland, 1975; Goldberg, 1989). To understand the process of natural evolution and to design software of artificial systems that retains the robustness of natural systems, they are mostly preferred in mentioned problems. However, GA has to utilize to complicated operators to ensure accurate solutions, so it needs to measure both possibilistic (or linguistic) and probabilistic uncertainty that are included in features of members in population. Possibilistic uncertainty concept first is suggested by Lotfi Asker Zadeh in 1965. According to Zadeh, making decision about processes that show nonrandom uncertainty, such as the uncertainty in natural language, has been demonstrated to be less than perfect. In these situations, the membership function is the main key to decision making when faced with uncertainty. *Essentially, such a framework provides a natural way of dealing with problems in which the source of imprecision is the absence of sharply defined criteria of class membership rather than the presence of random variables (Zadeh, 1965).* Thus, intuitively plausible semantic description of imprecise properties of data used in natural system might be made by fuzzy theory easily. For example, let us assign scores to each member of the current population by computing its fitness value. Then, in order to measure this kind of uncertainty, it can be assumed that these scores belong to any fuzzy set. In detail, the range of the scaled values affects the search performance of GA.

In this study, GA with fuzzy membership functions is integrated into both Gaussian approach with recursive hyperparameters and full Bayesian approach, and then fuzzy membership

functions are utilized to measure possibilistic (fuzzy) uncertainty in operators of GA and MCMC process. Thus, the novel hybrid Bayesian learning approaches are proposed to estimate network parameters and hyperparameters. Specifically, using full GA functions differently, the genetic Metropolis algorithm, which is proposed by Marwala (2007), is modified. For full Bayesian approach, the novel Monte Carlo Algorithm is improved by means of Gibbs sampling, Metropolis Hastings algorithm and GA with fuzzy membership functions.

Lastly, the performances of the proposed learning algorithms for Bayesian neural networks are compared with traditional neural networks. In order to estimate of parameters of the feedforward neural networks with sum square error, Steepest Descent, Steepest Descent with momentum Quasi-Newton, Levenberg-Marquardt Algorithm and GA are used.

1.1 Bayesian Neural Networks

There is a growing interest in the use of neural networks in engineering and statistics to model non-linear multivariate problems. Neural networks are capable of learning from examples, and finding meaningful solutions without the need to specify the relationship between variables, are the multilayer perceptron (MLP) networks, also known as "back propagation" or "feedforward". They are useful for analyzing problems that are either poorly defined or not clearly understood. The capability of neural networks to find meaningful patterns in large quantities of noisy data provides some advantages.

A neural network is composed of simple inter-connected nodes or neurons, and is usually used three or four layers in its structure. The intermediate layers are called hidden layers. The simple neural network in Figure 1.1 is constituted by three layers in which the hidden layer includes p neurons. The neuron connections have weights that are adapted (trained) to improve its overall performance. The neurons are arranged in layers and are connected so that the neurons in the hidden layer receive inputs from the preceding layer and sends out outputs to the following layer. External inputs are applied at the first layer and system outputs are taken at the last layer. Each hidden and output unit processes its inputs by multiplying each input by its weight, summing the product with adding bias and then processing the sum using a non-linear transfer function, or called as activation function, to produce a result. In here, biases are intuitively used as a threshold value, and control the inputs of activation function and outputs. The main idea is to modify the connection weights to reduce the errors between the actual output values and the target output values at a desired satisfactory level.

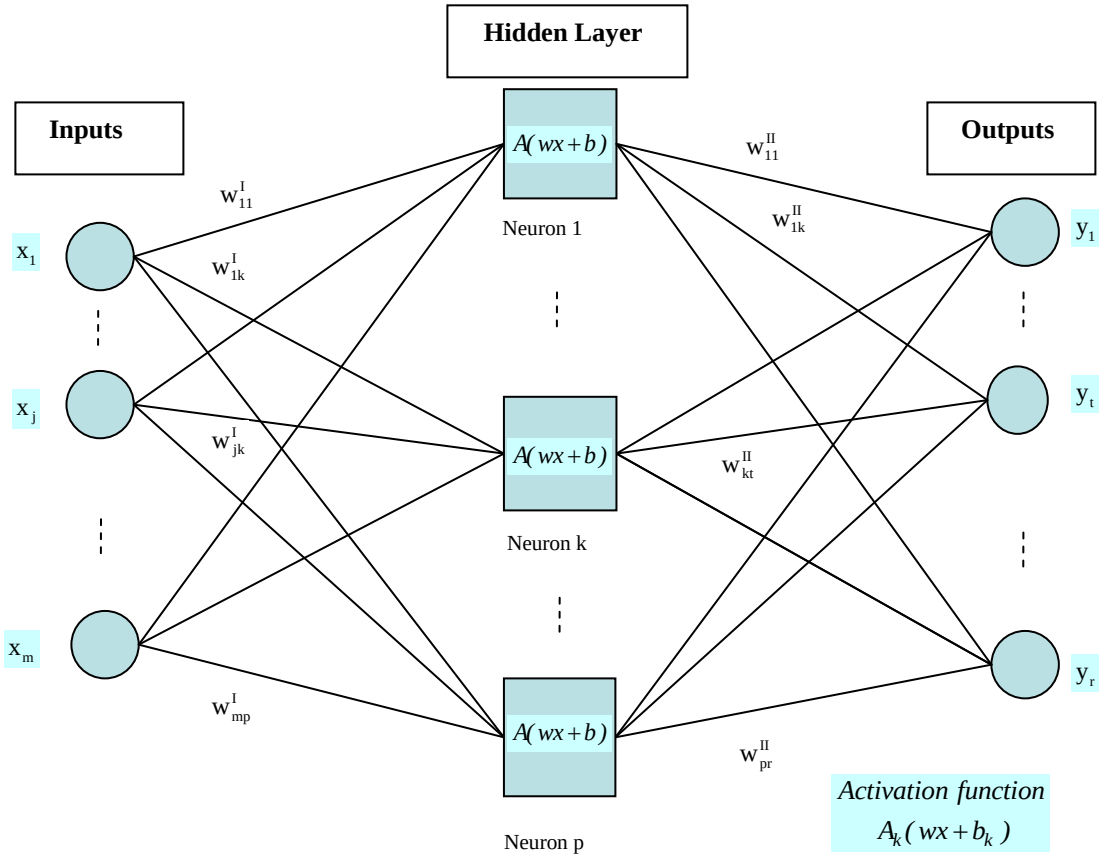


Figure 1.1. The simple neural network structure with one hidden layer

According to above network structure, the mathematical relation between inputs, neurons of hidden layer and *t -th* output for *i -th* observation can be formulated as follow:

$$f_t(w_k^I, w_t^{II}, x_i) = b_t^{II} + \sum_{k=1}^{p} w_{tk}^{II} A_k(w_k^I x_i + b_k^I) \qquad (1.1)$$

where $y_i = [y_{1i}\ y_{2i}\ ...\ y_{ri}]$ and $x_i = [x_{1i}\ x_{2i}\ ...\ x_{mi}]$ are target and input vectors of *i-th* observation respectively, $w_k^I = [w_{1k}^I\ w_{2k}^I\ ...\ w_{mk}^I]$ is the row vector that include weights between all inputs and *k-th* neuron in the hidden layer, $w_t^{II} = [w_{1t}^{II}\ w_{2t}^{II}\ ...\ w_{pt}^{II}]$ is the row vector that include weights between all neurons and *t-th* output, b_k^I and b_k^{II} are biases for *k-th* neuron and *t- th* output respectively, and $i = 1, 2, ..., N$; $j = 1, 2, ..., m$; $k = 1, 2, ..., p$; $t = 1, 2, ..., r$. In network w_k^I , w_t^{II} , b_k^I and b_t^{II} are parameters of network, and the activation function $A_k(w_k x_i + b_k^I)$ provides nonlinearity to neural network. In the literature, there are different types of activation functions related to its shape and structure. In this study, Hyperbolic Tangent, which is mostly used as activation function in the neural network literature since it is easily

calculated and differentiable, is preferred. In addition, Hyperbolic Tangent has range in the closed interval [-1, 1]:

$$A(x) = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

One of the most popular approaches to train neural networks has been to compute an estimator of the regression function by minimizing the following mean square empirical risk:

$$\mathrm{R}_e[\hat{f}(x,\hat{\theta})] = \frac{1}{N}\sum_{i=1}^{N}(y_{ti} - \hat{f}_t(x_i,\hat{\theta}))^2 \qquad (1.2)$$

where $\theta = \left\{ w_k^I , w_k^{II} , b_k^I , b_k^{II} \right\}$ covers all weights and biases.

The minimization is often performed by unconstrained gradient descent methods, such as back propagation or conjugate gradients (Bishop, 1995b). This approach causes two types of error, namely the approximation and estimation errors (Freitas, 2000).

The approximation error arises because the exact nonlinear behavior of the regression function is seldom known and, consequently, $f(x)$ has to be approximated by a combination of parameterized basis functions $\hat{f}(x,\theta)$. If the model structure has enough capacity to approximate the regression function, the approximation error will tend to zero as the number of parameters increases.

The estimation error is the result of our lack of knowledge about the conditional distribution $p(y|x,\theta)$. Likelihood methods do not attempt to estimate this distribution, but instead minimize the empirical risk in equation (1.2). One of the heuristic reasons for doing this is that the regression function minimizes the expected risk or L^2 norm (Freitas, 2000). This form can be explained as

$$f(\cdot) = arg \quad \min_{h(\cdot)\in T} R[h(\cdot)]$$

where $h(\cdot)$ denotes possible hypothesis and T represents the target space where the regression function lies. The estimator $\hat{f}(x,\hat{\theta})$ lies on hypothesis space H as shown in Figure 1.2. $R[h(\cdot)]$ corresponds to the mean square expected risk, given by

$$R[h(\cdot)] = \|y - h(x,\theta)\|^2_{L^2(p)} = E\left[(y - h(x,\theta))^2\right]$$

$$= \int (y - h(x,\theta))^2 \, p(y,\theta|x)\, dy\, d\theta$$

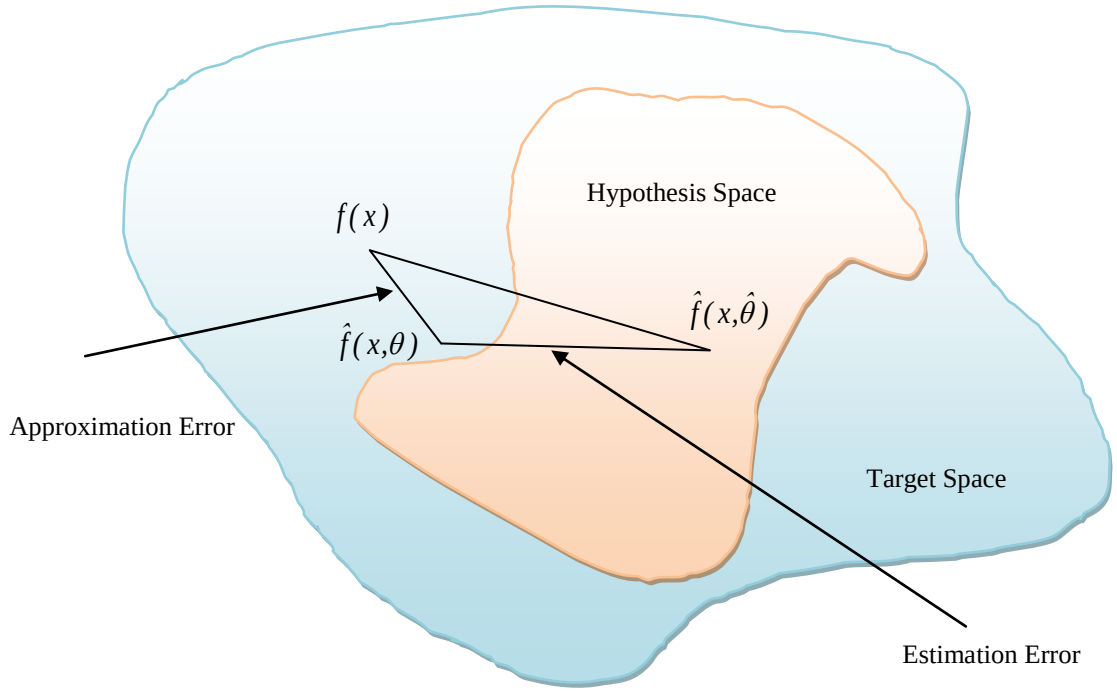


Figure 1.2 Approximation and estimation error

From a statistical point of view, the predictor $\hat{f}(x,\hat{\theta})$ obtained by empirical error minimization will approximate $\hat{f}(x,\theta)$ as the number of data increases without bound. It can also be expected that as the number of parameters increases, the expression for the empirical risk becomes more complex and, therefore, the estimation error can increase (Niyogi and Girosi, 1994; Freitas, 2000).

In detail, the approximation error is inversely related to the number of model parameters, while the estimation error is directly related to the number of parameters. This tradeoff is well known as the bias/variance tradeoff (White, 1989; Geman et al., 1992; Haykin, 1994; Bishop, 1995b; Sjoberg et al., 1995; de Freitas, 1997). The mean square error of the function $\hat{f}(x,\hat{\theta})$ as an estimator of the regression function may be decomposed into the following two terms:

$$\mathrm{E}\left(\left(\hat{f}(x,\hat{\theta})-f(x)\right)^2\right)=\underbrace{\mathrm{E}\left(\left(\hat{f}(x,\hat{\theta})-\mathrm{E}(\hat{f}(x,\hat{\theta})\right)^2\right)}_{\textbf{Variance}}+\underbrace{\left(\mathrm{E}\left(\hat{f}(x,\hat{\theta})\right)-f(x)\right)^2}_{\textbf{Bias}} \tag{1.3}$$

The bias term measures the distance between the average estimator and the regression function, while the variance term quantities the spread of the estimator with respect to the data distribution. In estimation process, to carry out good modeling performance, both the bias and the variance

have to be small. However, depending on increasing model complexity, the variance term (with estimation error) increases while the bias term decreases.

Within the learning paradigm discussed so far, an acceptable generalization performance can be achieved by balancing the bias and variance terms. Therefore, the only ways to reduce the bias and variance error terms simultaneously are to either increase the number of data or to model the noise characteristics and incorporate a priori knowledge about the form of the estimator (Freitas, 2000). This approach can be ensured by Bayesian learning paradigm inherently, since it provides probabilistic knowledge about related data.

Another way of making use of a priori knowledge is to impose the smoothness constraints on the model. This approach can be explained that small changes in the input should lead to small changes in the output. In addition, this scheme is known as regularization. By means of regularization, the infinite number of possible solutions of the learning problem can be reduced to one by balancing the bias and variance error terms simultaneously. To obtain a function that is simultaneously close to the data and smooth, the empirical modeling error criterion may be extended as follow:

$$R_r[\hat{f}(x,\hat{\theta})] = \frac{1}{N}\sum_{i=1}^{N}(y_{ti} - \hat{f}_t(x_i,\hat{\theta}))^2 + \nu\Omega \tag{1.4}$$

where ν is a positive parameter that serves to balance the tradeoff between smoothness and data approximation. A large value of ν places more importance on the smoothness of the model, while a small value of ν places more emphasis on fitting the data. The functional Ω penalizes excessive model complexity.

There are some approaches for function Ω (Hinton, 1987; Girosi et al., 1995), the most popular one of those is weight decay whose functional type is given by:

$$\Omega = \sum_i \theta_i^2 \tag{1.5}$$

where *w* is index of total parameter number.

One of the reasons for using weight decay is that superfluous parameters are forced to decay to zero. Previously, it was discussed that the generalization performance deteriorates if the number of parameters increases excessively. Using weight decay, only a few of the parameters contribute to the mapping and hence the generalization error decreases. From an intuitive point of view, the network outputs become approximately linear functions of the inputs for MLPs with

very small weights. Weight decay can be given a natural interpretation in the Bayesian framework (Bishop 1995b).

Other common approaches to controlling the complexity of the estimates include early stopping, training with noise, mixtures of networks and growing and pruning techniques. Early stopping has several shortcomings (Freitas, 2000):

• The amount of training data is usually halved,
• The estimator will be biased towards the validation set, thus requiring an extra test set,
• Early stopping relies on the assumption that the path taken through parameter space by the optimization algorithm passes through an acceptable solution. This is not always the case with multi-modal error surfaces.

Bishop Bishop (1995a), Leen (1995) and Wu and Moody (1996) showed that training with noise added to the inputs is equivalent to regularization. The minimization of the empirical risk (R_e) with noise, of small amplitude, added to the input data is equivalent to minimization of the regularized risk (R_r). It can be concluded that the heuristic basis for training with noise is that the noise will consider each data point and preclude the network from fitting individual data points precisely, and hence will reduce overfitting.

Several researchers have argued that the performance of estimators may be considerably improved by combining several estimators of different complexity and model structure (Jacobs, 1995; Perrone, 1995). If the individual models are trained so that their variance error terms are bigger than their bias error terms, then model combination may reduce the variance error component, as it involves averaging over all the estimates. It can be argued that combining models, in this way, is a brittle strategy. The resulting model still needs to be subject to the same model choice criteria as the individual models (Feritas, 2000).

The idea behind growing and pruning algorithms is to control the complexity of the estimator by eliminating and adding parameters to the estimator as the data is processed. Examples of this type of algorithm include “The Upstart Algorithm” Frean (1990), “The Resource Allocating Network” Platt (1991), “An Iterative Pruning Algorithm” Castellano at al. (1997) and “Sequential Learning Algorithm for Radial Basis Function” Huang at al. (2005).

1.2 Bayesian Learning

In within learning paradigm, the unnecessarily complex models shouldn't be preferred to simpler ones. This approach is called as Occam`s razor named after William of Occam (1285-1349), which actually is embodied automatically and quantitatively in Bayesian methods without penalty function, since complex model is automatically self-penalizing in Bayes's rule. But, the maximum likelihood model choice would lead us inevitably to implausible over parameterized models that generalize poorly (MacKay, 1992). In the Bayesian learning paradigm, a priori knowledge of all parameters is used together with likelihood, so that data fitting of a model corresponding to any parameter group is evaluated by likelihood as well.

The result of Bayesian learning is a probability distribution over model parameters that express our beliefs regarding how likely the different parameters values are. To start the process of learning, a prior distribution, $p(\theta)$, must be defined for the parameters, that expresses our initial beliefs about their values, before any data has arrived. When data $D = \{x_{1:N}, y_{1:N}\}$ is observed, this prior distribution updated to posterior one, using Bayes' rule:

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{p(D)} = \frac{p(D \mid \theta)\, p(\theta)}{\int p(D \mid \theta)\, p(\theta) d\theta} \tag{1.6}$$

or

$$p(M_l \mid D) = \frac{p(D \mid M_l)\, p(M_l)}{\sum_l p(D \mid M_l)\, p(M_l)} \qquad l = 1, 2, \ldots \tag{1.7}$$

where model M_l corresponds to any parameter group. Bayes's theorem is summarized as follow:

$$Posterior = \frac{Likelihood}{Normalization} Prior \tag{1.8}$$

Our subjective beliefs and views of uncertainty are expressed in the prior. Once the data becomes available, the likelihood allows us to support these beliefs. The resulting posterior distribution incorporates both our a priori knowledge and the information conveyed by the data, and then updates these beliefs.

The statistical information about the parameters and their number given the measurements and the prior, one can theoretically obtain all features of interest by standard probability marginalization and transformation techniques. The predictive density can be estimated as follow:

$$p(y_{1:N+1} \mid x_{1:N+1}, y_{1:N}) = \int p(y_{1:N+1} \mid \theta, x_{1:N+1})\, p(\theta \mid x_{1:N}, y_{1:N}) d\theta \tag{1.9}$$

and consequently forecast quantities of interest:

$$\mathrm{E}(y_{1:N+1} \mid x_{1:N+1}, y_{1:N}) = \int \hat{f}(\theta, x_{1:N+1})\, p(\theta \mid x_{1:N}, y_{1:N}) d\theta \tag{1.10}$$

In here, the prediction must be based on all possible values of the network parameters weighted by their probability in view of the training data.

In addition, parameter estimation is performed by Maximize Probability (MAP) or Minimum Variance Estimate (MVE) as well. In MAP, the solution is the largest mode (peak) of $p(\theta \mid x_{1:N}, y_{1:N})$. Besides, if the fixed priors are taken as uniform, then the resulting solution is the maximum likelihood estimate. In MVE, because of minimizing error function, $\int \left\| \theta - \hat{\theta} \right\|^2 p(\theta \mid x_{1:N}, y_{1:N}) d\theta$, the estimation of parameters is the expected value or conditional mean, $\mathrm{E}(\theta \mid x_{1:N}, y_{1:N})$.

In Bayesian analysis, inevitably there is an integration problem. The integrals appear whenever normalization, marginalization or expectations are attempted to carry out (Bernardo and Smith, 1994; Gelman et al., 1995). To solve these high-dimensional integrals, analysts can either resort to analytical integration, approximation methods, numerical integration or Monte Carlo simulation. Many real-world problems involve elements of non-Gaussianity, nonlinearity and non-stationarity, so these characteristic features preclude the use of analytical integration.

Approximation methods, such as Gaussian approximation and variational methods are easy to implement. In addition, they tend to be very efficient from a computational point of view. If, they do not take into account all the salient statistical features of the processes under consideration, thereby often leading to poor results.

Numerical integration in high dimensions is far too computationally expensive to be of any practical use.

Compared the other methods, MC methods are middle level. They lead to better estimates than the approximate methods. Although Bayesian simulations need extra computing requirements, the computational power of computers and some recent developments in applied statistics cope with this issue. MC methods are very flexible in that they do not require any assumptions about the probability distributions of the data. From a Bayesian perspective, MC methods allow one to compute the full posterior probability distribution.

There are lots of attempts to apply the Bayesian learning paradigm to neural networks. In the early nineties, Buntine and Weigend (1991) and Mackay (1992) showed that a principled Bayesian learning approach to neural networks can lead to many improvements. For instance, Mackay showed that by approximating the distributions of the weights with Gaussians and adopting smoothing priors, it is possible to obtain estimates of the weights and output variances and to automatically set the regularization coefficients. Neal (1992) introduced advanced Bayesian simulation methods, specifically the hybrid Monte Carlo method (Duane et al., 1987; Brass et al., 1993), into the analysis of neural networks. Rios Insua and Müller (1998), Marrs (1998) and Holmes and Mallick (1998) have addressed the issue of selecting the number of hidden neurons with growing and pruning algorithms from a Bayesian perspective. In particular, they apply the reversible jump MCMC algorithm of Green (Green, 1995; Richardson and Green, 1997) to feed-forward sigmoidal networks and radial basis function networks to obtain joint estimates of the number of neurons and weights. Freitas (2000) incorporated particle filters and sequential Monte Carlo methods into the analysis of Bayesian neural networks. Chua and Goh (2003) proposed a hybrid Bayesian back-propagation neural network approach to multivariate modeling. Liang (2005) and Lord at al. (2007) proposed the truncated Poison priors for neuron numbers in the hidden layers. Vanhatalo and Vehtari (2006) improved hybrid and reversible algorithm that are based on Neal (1992). Marwala (2007) adapted to mutation and crossover operators used in genetic algorithms into Bayesian learning, but, his algorithm structure slightly different from general genetic algorithm process. In literature, there are many implementations that are based on the mentioned studies above as well.

2 GAUSSIAN APPROXIMATION FOR BAYESIAN LEARNING

MacKay (1992) proposed Gaussian approximation for Bayesian neural networks. According to this approximation, the uncertainty in the weight space is assigned to a probability distribution representing the degree of belief in the different values of the weight vector. By maximizing the posterior distribution over the weights; the most probable parameters values can be determined. MacKay (1992) has shown that maximizing the posterior distribution corresponds to minimizing the regularized error function that is very similar in (1.4). The posterior distribution is then used to evaluate the predictions of the trained network for new values of the input variables as well.
In this approximation, the data set is modeled as deviating from this mapping under some additive noise process ε:

$$y_{ti} = f_t(x_i, \theta) + \varepsilon \tag{2.1}$$

If ε is modeled as zero – mean Gaussian noise with standard deviation σ_{noise}, then the probability of a data value given the parameter vector θ is:

$$p(y_i | x_i, \theta, \beta) = \frac{1}{(2\pi)^{1/2} \beta^{-1/2}} \exp\left(-\frac{\beta}{2} \sum_{i=1}^{N} \left\{ y_{ti} - \hat{f}_t(x_i, \theta) \right\}^2 \right) \tag{2.2}$$

where $\beta = 1 / \sigma_{noise}^2$ is called precision, and controls noise of variance. Provided data points are drawn independently from this distribution, so the likelihood can be constituted for N observations as follow:

$$p(D|\theta, \beta) = \prod_{i=1}^{N} p(y_i | \theta, x_i)$$

$$= \frac{1}{(2\pi)^{N/2} \beta^{-N/2}} exp\left(-\frac{\beta}{2} \sum_{i=1}^{N} \left\{ y_{ti} - \hat{f}_t(x_i, \theta) \right\}^2 \right) \tag{2.3}$$

More generally, the likelihood function can be written as

$$p(D|\theta, \beta) = \frac{1}{Z_D(\beta)} exp(-\beta E_D) \tag{2.4}$$

where $E_D = \frac{1}{2}\sum_{i=1}^{N}\left\{y_{ti} - \hat{f}_t(x_i,\theta)\right\}^2$, $Z_D(\beta) = \int exp(-\beta E_D)dD$ and $\int \partial D = \int dy_1 dy_2 ....dy_N$.

The function $Z_D(\beta)$ is called a normalization, and integrated as (MacKay, 1992; Bishop, 1995; Bishop, 2006) (*Appendix A*)

$$Z_D(\beta) = \int exp(-\beta E_D)dD = (2\pi)^{N/2}\beta^{-N/2} \tag{2.5}$$

In conventional maximum likelihood approach, the single best set of weight values (and Bias) is only determined by minimization of a suitable error function. In the Bayesian framework, however, probability distribution is considered over weight values as well. This distribution should reflect any the prior knowledge about network mapping that is expected to find. In generally, distribution of weights is assumed as an exponential form:

$$p(\theta) = \frac{1}{Z_\theta(\alpha)}exp(-\alpha E_\theta) \tag{2.6}$$

where $Z_\theta(\alpha)$ is a normalization factor given by $Z_\theta(\alpha) = \int exp(-\alpha E_\theta)d\theta$ which ensures that $\int p(\theta)d\theta = 1$.

As known tradeoff between variance and bias indicates that a smooth network function will typically have better generalization than one which is over-fitted to training data. This is one of the motivations for regularization techniques designed to encourage smooth network mappings. Such mappings can be expressed the following simple form for E_θ:

$$E_\theta = \frac{1}{2}\|\theta\|^2 = \frac{1}{2}\sum_{w=1}^{W}\theta_w^2 \tag{2.7}$$

where W is the total number of weights and biases in the network. This corresponds to the use of a simple weight-decay regularizer. Thus, a prior distribution of parameters is given by

$$p(\theta) = \frac{1}{Z_\theta(\alpha)}exp(-\frac{\alpha}{2}\|\theta\|^2) \tag{2.8}$$

where the normalization coefficient $Z_\theta(\alpha)$ is integrated as follow: (*Appendix A*)

$$Z_\theta(\alpha) = \int exp(-\alpha E_\theta)d\theta = (2\pi/\alpha)^{W/2} \tag{2.9}$$

where W is total parameter number. If $\|\theta\|$ is large, E_θ would large as well, and $p(\theta)$ would small, and so this choice of prior distribution ensure that weight values would be small rather than large. Since the parameter α itself controls the distribution of other parameters (weights and biases), it is called a hyperparameter. A major advantage of Gaussian distribution is the evaluation of normalization coefficients $Z_\theta(\alpha)$ and $Z_D(\beta)$ by analytical way. Many other choices for the prior can also be considered. Williams (1995) discusses a Laplacian prior, Buntine and Weigend (1991), entropy based prior, and Neal (1996) appropriate selection of priors of very large networks and Lampinen and Vehtari (2001) used inverse gamma priors for a hierarchal structure.

For all that, the regularizer coefficient E_θ provides some simplicity; this structure is inconsistence with certain scaling properties of network mappings. Because input and/or target variables are transformed by one of the linear transformations, then consistency requires that equivalent networks, which differ only by the linear transformation of weights, should be obtained. However, the regularizer coefficient E_θ does not satisfy this property, so a different regularizer type that is invariant under the linear transformations should be looked for. Such a regularizer is given by (Bishop, 1995, 2005)

$$E_{w^I, w^{II}} = \frac{\alpha_1}{2} \sum_{w^I \in W_1} (w^I)^2 + \frac{\alpha_2}{2} \sum_{w^{II} \in W_2} (w^{II})^2 \tag{2.10}$$

where W_1 and W_2 denotes the set of weights in the first and second groups respectively. This led to a consideration of weight decay regularizers in which there is a different regularization coefficient for weights in different groups. Thus, for two layers network, a prior form of weights can be given by

$$p(w) \propto \exp\left(-\frac{\alpha_1}{2} \sum_{w^I \in W_1} (w^I)^2 - \frac{\alpha_2}{2} \sum_{w^{II} \in W_2} (w^{II})^2 \right) \tag{2.11}$$

where biases are excluded from the summations. The prior with biases are improper since the bias parameter is unconstrained. The use of improper priors can lead to difficulties in selecting regularization coefficients and in model comparison within the Bayesian frame work, since evidence is zero (Bishop, 1995). Therefore, priors of biases are separately included from prior of weights.

More generally, priors in which the parameters (weights and biases) are divided into any number of groups Θ_ℓ are considered as follow:

$$p(\Theta) \propto exp\left(-\frac{1}{2}\sum_{\ell} \alpha_\ell \left\| \theta^\ell \right\|^2 \right) \tag{2.12}$$

where $E_\ell = \left\| \theta^\ell \right\|^2 = \sum_{W_\ell} \theta^2$ and ℓ is layer index. Thus E_θ can be substituted by

$$E_\Theta = [E_1 \; E_2 ... E_\ell] \tag{2.13}$$

Once a prior distribution and an expression for the likelihood are chosen, in order to find posterior distribution, Bayes` theorem in (1.6) can be used:

$$p(\theta \mid D) = \frac{1}{Z_S(\alpha, \beta)} exp(-\beta E_D - \alpha E_\Theta) = \frac{1}{Z_S(\alpha, \beta)} exp(-S_\theta) \tag{2.14}$$

where

$$S(\theta) = \beta E_D + \alpha E_\Theta \tag{2.15}$$

and

$$Z_S(\alpha, \beta) = \int \exp(-\beta E_D - \alpha E_\Theta) d\theta \tag{2.16}$$

When the different parameter groups are introduced, α_ℓ is vector in which all hyperparameters of parameter groups are included, otherwise it is only one hyperparameter for one parameter vector. In here, the main problem is to find the parameter vector θ_{MP} corresponding to the maximum of the posterior distribution. This can be found by minimizing the negative logarithm of (2.14). Since normalizing factor in (2.14) is independent of parameters (weights and bias), the exploring the maximum of the posterior distribution is equivalent to minimizing $S(\theta)$ given by (2.15). Thus, $S(\theta)$ can be written in the form:

$$S(\theta) = \frac{\beta}{2}\sum_{i=1}^{N} (y_{t,i} - \hat{f}_t(x_i, \theta))^2 + \frac{1}{2}\sum_{\ell} \alpha_\ell \left\| \theta^\ell \right\|^2 \tag{2.17}$$

The most probable value of parameter vector, denoted by θ_{MP}, can be found by the minimizing of the right-hand side in (2.17). Some consistence conclusions can be inferred from (2.17). For instance, the first term in (2.17) grows with *N* while the second term does not. If α_ℓ and β are fixed, then *N* increases, the first term becomes more and more dominant, until eventually the second term becomes insignificant. Therefore, the maximum likelihood solution is then a very

good approximation to the most probable solution θ_{MP} for large *N*. Conversely, for small data sets the prior term plays important role in determining the location of the most probable solution. In practice, in order to evaluate the probability distribution of network prediction, integrations over parameter space are required, so these integrals have to be analytically tractable. But, normalization factor $Z_S(\alpha, \beta)$ is not evaluated analytically. MacKay (1992) uses a Gaussian approximation for posterior distribution. This obtained by considering the Taylor expansion of $S(\theta)$ around its minimum value and retaining terms up to second order so that

$$S(\theta) = S(\theta_{MP}) + \tfrac{1}{2}(\theta - \theta_{MP})' A (\theta - \theta_{MP}) \tag{2.18}$$

where linear term vanished since around a minimum of $S(\theta)$ are being expanded. In here, A is Hessian matrix of error function in (2.17), with elements given by

$$\begin{aligned} A &= \nabla\nabla S_{MP} \\ &= \beta \nabla\nabla E_D^{MP} + \alpha I \end{aligned} \tag{2.19}$$

where α is diagonal matrix that includes α^{ℓ} of different groups. The expansion of (2.18) leads to posterior distribution, which is now a Gaussian function of the weights, is given by

$$p(\theta \mid D) = \frac{1}{Z_S^*(\alpha, \beta)} \exp\left(-S(\theta_{MP}) - \frac{1}{2}(\theta - \theta_{MP})' A (\theta - \theta_{MP}) \right) \tag{2.20}$$

where $Z_S^*(\alpha, \beta)$ is the normalization constant appropriate to the Gaussian approximation.

Under very general circumstances, a posterior distribution will tend to a Gaussian in the limit where the data points goes infinity. Another advantage of the Gaussian approximation is that it allows analytical process. By means of Gaussian approximation, the normalization constant in (2.20) is evaluated as (*Appendix B*)

$$Z_S^*(\alpha, \beta) = e^{-S(\theta_{MP})} (2\pi)^{W/2} |A|^{-1/2} \tag{2.21}$$

where A is Hessian matrix of error function in (2.17) .

Using the rules of probability, the distribution of outputs for a given new input vector x_{N+1}^{new} is given by

$$p(y_{t,N+1} \mid x_{N+1}^{new}, D) = \int p(y_{t,N+1} \mid x_{N+1}, \theta)\, p(\theta \mid D) d\theta \tag{2.22}$$

where $p(\theta|D)$ is posterior distribution of parameters, and D = { $x_{1:N}$, $y_{1:N}$ } is data set. The distribution $p(y_{t,N+1}|x_{N+1}, \theta)$ is simply the model for the distribution of noise on target data, for a fixed value of the parameter vector.

In order to evaluate the distribution of outputs in (2.22), Gaussian approximation in (2.20) for posterior distribution of weights can be used:

$$p(y_{t,N+1}|x_{N+1}, D) \propto \int \exp\left(-\frac{\beta}{2}\left\{y_{t,N+1} - \hat{f}_t(x_i, \theta)\right\}^2\right)\exp\left(-\frac{1}{2}(\theta-\theta_{MP})'A(\theta-\theta_{MP})\right)d\theta \qquad (2.23)$$

where any constant factors, which are independent of target y, are dropped. Besides, the width of the posterior distribution determined by the Hessian matrix A is sufficiently narrow that network function $\hat{f}_t(x_i, \theta)$ may be approximated by its linear expansion around θ_{MP}

$$\hat{f}_t(x_{N+1}, \theta) = \hat{f}_t(x_{N+1}, \theta_{MP}) + g_t'\Delta\theta \qquad t = 1, 2, \ldots, r \qquad (2.24)$$

where $g_t \equiv \nabla_\theta \hat{f}_t(x_{N+1}, \theta)\Big|_{\theta=\theta_{MP}}$ and $\Delta\theta = \theta - \theta_{MP}$.

Thus, (3.23) can be written by means of (3.24) as follow:

$$p(y_{t,N+1}|x_{N+1}^{new}, D) \propto \int \exp\left(-\frac{\beta}{2}\left\{y_{t,N+1} - \hat{f}_t(x_{N+1}, \theta_{MP}) - g_t'\Delta\theta\right\}^2 - \frac{1}{2}(\theta-\theta_{MP})'A(\theta-\theta_{MP})\right)d\theta \qquad (2.25)$$

The integral in (2.25) can be evaluated easily as follow: (MacKay, 1992; Bishop, 1995; Chua and Goh, 2003)

$$p(y_{t,N+1}|x_{N+1}^{new}, D) = \frac{1}{(2\pi\sigma_{t,N+1}^2)^{1/2}} exp\left(-\frac{\left\{y_{t,N+1} - \hat{f}_t(x_{N+1}, \theta_{MP})\right\}^2}{2\sigma_{t,N+1}^2}\right) \qquad (2.26)$$

where normalization factor is restored explicitly. Thus, it can be concluded that this distribution has a mean

$$\bar{y}_{t,N+1} = \hat{f}_t(x_{N+1}, \theta_{MP}) \qquad (2.27)$$

a variance given by

$$\sigma_{t,N+1}^2 = \frac{1}{\beta} + g' A^{-1} g \qquad (2.28)$$

In here, the standard deviation σ of the predictive distribution for $y_{t,N+1}$ can be interpreted as an error bar on the mean value $\bar{y}_{t,N+1}$. This error arises from two reasons, the first one is the intrinsic noise on the target data, corresponding to the first term in (2.28), and the second one is the width of posterior distribution of network parameters, corresponding to the second term in (2.28). Hence Bayesian approach allows the calculation of error bars on the network output, instead of just providing a single output.

So far it is assumed that the values of hyperparameters α and β are known, however there is little idea of suitable values for them. In here, the treatment of hyperparameters involves Occam's Razor since the values of hyperparameters, which give the best fit to the training data in a maximum likelihood setting, represent over-complex or over-flexible models that don't give the best generalization. In literature, two approaches are discussed to the treatment of hyperparameters, one of these performs the integrals over α and β analytically, and the second one, known as the evidence approximation, discussed by MacKay (1994), and it is computationally equivalent to the type II maximum likelihood method of conventional statistics (Berger, 1985; Bishop 1995; Kocadagli, 2011).

In this study, in order to estimate values of α and β, the first approximation, which involves marginalization, in other words integration over all possible values, is preferred. This can be done by the following marginal integral:

$$p(\theta \mid D) = \int\int p(\theta, \alpha, \beta \mid D)\, d\alpha\, d\beta$$

$$= \frac{1}{p(D)} \int\int p(D \mid \theta, \beta)\, p(\theta \mid \alpha)\, p(\alpha)\, p(\beta) d\alpha\, d\beta \qquad (2.29)$$

where $p(D \mid \theta, \alpha, \beta) = p(D \mid \theta, \beta)$ since likelihood term is independent of α, and similarly $p(\theta \mid \alpha, \beta) = p(\theta \mid \alpha)$ since the prior over the parameters are independent of β. Besides, because of independency of the two hyperparameters, it is assumed that $p(\alpha, \beta) = p(\alpha)\, p(\beta)$.

To evaluate the integral in (2.29), the specific choices for the priors are determined. If improper uniform priors with the logarithmic scale form are preferred where $p(ln\alpha) = 1$, $0 < ln\alpha < \infty$ and $p(ln\beta) = 1$, $0 < ln\beta < \infty$ and applied to exponential transformation with Jacobian, then priors can be transformed this way (Gill, 2008):

$$p(\alpha) = \frac{1}{\alpha} \quad \text{and} \quad p(\beta) = \frac{1}{\beta} \qquad 0 < \alpha < \infty, \quad 0 < \beta < \infty \qquad (2.30)$$

This choice leads to straightforward analytic integrals over the hyperparameters. Using (2.6) and (2.9), the integral over α in (2.29) can be evaluated as

$$
\begin{aligned}
p(\theta) &= \int_0^\infty p(\theta \mid \alpha)\, p(\alpha) d\alpha \\
&= \int_0^\infty \frac{1}{Z_\theta(\alpha)} exp(-\alpha E_\theta) \frac{1}{\alpha} d\alpha \\
&= (2\pi)^{-W/2} \int_0^\infty \exp(-\alpha E_\theta) \alpha^{\frac{W}{2}-1} d\alpha \\
&= \frac{\Gamma(W/2)}{(2\pi E_\theta)^{W/2}}
\end{aligned}
\tag{2.31}
$$

where Γ is standard Gamma function. The integration over β can be performed in exactly the same way with the result:

$$
P(D|\theta) = \int_0^\infty p(D \mid \theta, \beta)\, p(\beta) d\beta = \frac{\Gamma(N/2)}{(2\pi E_D)^{N/2}} \tag{2.32}
$$

Thus, the exact (rather than approximate) un-normalized posterior distribution of the weights can be constituted by means of (2.31) and (2.32) as follow:

$$
p(\theta \mid D) = \frac{1}{P(D)} \frac{\Gamma(N/2)}{(2\pi E_D)^{N/2}} \frac{\Gamma(W/2)}{(2\pi E_\theta)^{W/2}} \tag{2.33}
$$

The negative logarithm of this posterior, corresponding to error function, then takes form as

$$
-ln\, p(\theta \mid D) = \frac{N}{2} ln E_D + \frac{W}{2} ln E_\theta + constant \tag{2.34}
$$

From (2.14), the negative logarithm of posterior similarly can be written as

$$
-ln\, p(\theta \mid D) = \beta E_D + \alpha E_\theta + constant \tag{2.35}
$$

The derivative of (2.35) can be written as follow:

$$
-\nabla ln\, p(\theta \mid D) = \beta \nabla E_D + \alpha \nabla E_\theta \tag{2.36}
$$

The derivative of (2.34) similarly can be written as follow:

$$
-\nabla ln\, p(\theta \mid D) = \frac{N}{2} \frac{\nabla E_D}{E_D} + \frac{W}{2} \frac{\nabla E_\theta}{E_\theta} \tag{2.37}
$$

The equation in (2.37) can be arranged by means of transformations $\alpha_{eff} = \frac{W}{2E_\theta}$ and $\beta_{eff} = \frac{N}{2E_D}$ as follow:

$$-\nabla \ln p(\theta \mid D) = \beta_{eff} \nabla E_D + \alpha_{eff} \nabla E_\theta \tag{2.38}$$

In here, the equations constituted in (2.36) and (2.38) are equal each other, so that the efficient α_{eff} and β_{eff} can be written as follows:

$$\alpha_{eff} = \frac{W}{2E_\theta} \quad \text{and} \quad \beta_{eff} = \frac{N}{2E_D} \tag{2.39}$$

Consequently, minimization of the error function in (2.35) is equivalent to minimization of equality in (2.38) in which the values of α_{eff} and β_{eff} are continuously updated using re-estimation formulas in (2.39) at every iteration (MacKay, 1994; Bishop, 1995; Chua and Goh, 2003; Kocadagli, 2011). In order to optimize the error function at the any iteration, α_{eff} and β_{eff} in the previous iteration can be used. In addition, the optimal weights and biases might be directly estimated by using error function with fixed hyperparameters α and β in (2.17) as well. However, determining the optimal hyperparameters by trial and error doesn't make sense in terms of time cost and effective search. In literature, to minimize equality in (2.35); Gradient search, conjugate and evolutionary algorithms are preferred (MacKay, 1992; Goh and Chua, 2003; Marwela, 2007; Kocadagli, 2011). The problems of minimization of error function by recursive and fixed-hyperparameter approaches are given in application parts, and then the analysis of problems are discussed in detail.

2.1 Genetic Monte Carlo Algorithms for Gaussian Approximation

According to (2.17) and the efficient parameters α_{eff} and β_{eff}, the optimal weights and biases that minimize to negative logarithm of posterior distribution in (2.17) can be estimated by means of algorithms such as Gradients, Evolutionary, the simulated annealing or MCMC. However, all gradient algorithms suffer from searching through parameter space with high dimensions, and don't work without the knowledge of derivative. The simulated annealing or MCMC algorithms are not useful to make accurate approximations in excessive parameter case. In high dimension case; Hybrid Monte Carlo (Neal, 1992), Sequential Monte Carlo, (Freitas, 2000), Hiybrid MC (Lampinen and Vahtari, 2000) and Genetic Monte Carlo (Marwela, 2007) approaches are proposed for effective searching through parameter space. In these studies, Hybrid Monte Carlo

(HMC) is based on the advanced full Bayesian simulations, and Genetic MC incorporates Gaussian approximation with the functions of GA. Especially, HMC makes efficient use of the gradient information to reduce random walk behavior. The gradient indicates in which direction one should go to find states with high probability. This approach is discussed in full Bayesian approach in next pages, and then the novel HMC algorithm is improved.

In here, a novel Genetic MC algorithm is proposed for Gaussian approximation with recursive hyperparameters. This novel approach is similar to Marwela (2007)'s approach point of using GA and Metropolis-Hastings. However, the proposed approach takes into account all functions of GA, and uses recursive hyperparameters approach for searching through parameter space as distinct from Marvela's approach. In Marwela's approach, to compare parameter vectors held by Metropolis Hastings algorithm in current and previous iterations, the candidate parameter vector in the current iteration is produced from parameter vector of previous iteration by crossover and mutation functions. However, this approach conflicts with diversity concept of general GA, since diversity of population has to be taken under control by keeping members with different features in the selection process of GA. Therefore, number of members produced in generations has to be greater than one. Otherwise, searching time through parameter space and chance of stuck in local optimums would increase unnecessarily. For the detailed information about GA functions and control parameters, Goldberg (1986), Michalewicz (1994) and MATLAB 7.12 are handy references. To overcome the mentioned shortcomings, the novel genetic Monte Carlo algorithms with recursive hyperparameter approach are proposed. The steps of novel genetic Monte Carlo algorithms for Gaussian approximation are summarized as follows:

i. Convert all parameters of error function in (2.17) to binary numbers (or real number), and then gather all parameters into one vector called chromosome in Genetic Algorithms.
ii. Create prospective vectors at the certain numbers (or initial population) by means of heuristic or random methods.
iii. Calculate scores of all individuals over error function in (2.17) called as fitness function in Genetic Algorithms (In this process, the fuzzy membership function is used to scale scores of candidate members).
iv. Select favorable individuals (or vectors) into mate pool for the next generation by heuristic or stochastic methods (Stochastic method is preferred).
v. Apply elite, crossover and mutation process to selected individuals called as parents in Genetic Algorithms. This process is called reproduction creates children for the next generation.

vi. If the stopping criteria are met, stop algorithm, otherwise turn to step iii, and then run the algorithm up to meet any stopping criterion.

vii. After stopping algorithm, keep the best parameter vector and then convert all binary numbers (or double vector) to any parameter into float numbers.

viii. Skip Metropolis Hastings algorithm, and then compare the previous and the current parameter vectors by means of posterior distribution in (2.14) .

The detailed information related to functions of Encoding, Selection, Elitism, Crossover, Mutation and Immigration are given in *Appendix C*. In step iii of GA cycle, each member of the current population by computing its fitness value is assigned to scores. The range of the scaled scores affects the performance of GA. In order to measure this kind of uncertainty included in scores, it can be assumed that these scores belong to any fuzzy set, and then fuzzy scores of population members can be evaluated by means of fuzzy membership functions. Thus, the imprecise properties of data used in scaled process might be intuitively defined by fuzzy membership function easily. In here, the membership functions that are similar to distance membership of Zimmerman (1978) can be defined (Lai and Hwang, 1992; Kocadagli and Cinemre, 2011) as follow:

$$\mu(skor_i) = 1/\left[\exp\left(\left(skor_i - \min\right)/(\max - \min)\right)\right]^p \quad p = 1, 2, 3;\ i = 1, 2, \ldots, population\ size \tag{2.40}$$

where max and min are maximum and minimum scores of population parameter respectively, and p that controls scale of the scores contributes to diversification in population.

The second alternative of fuzzy membership function can be defined as follow:

$$\mu(skor_i) = 1/\exp\left[\left(\left(score_i - \min\right)/(\max - \min)\right)^p\right] \quad p \in (0,1);\ i = 1, 2, \ldots, population\ size \tag{2.41}$$

where max and min are maximum and minimum scores of population respectively. In here, decision about selection of membership function can be determined by the trial and error. Then, the expectation number of members to be selected into mating pool is determined by following formulation:

$$E(score_i) = population\,size \times \frac{\mu(score_i)}{\sum \mu(score_i)} \quad i = 1, 2, \ldots, population\ size \tag{2.42}$$

The simple GA cycle can be summarized as the following chart in Figure 2.1:

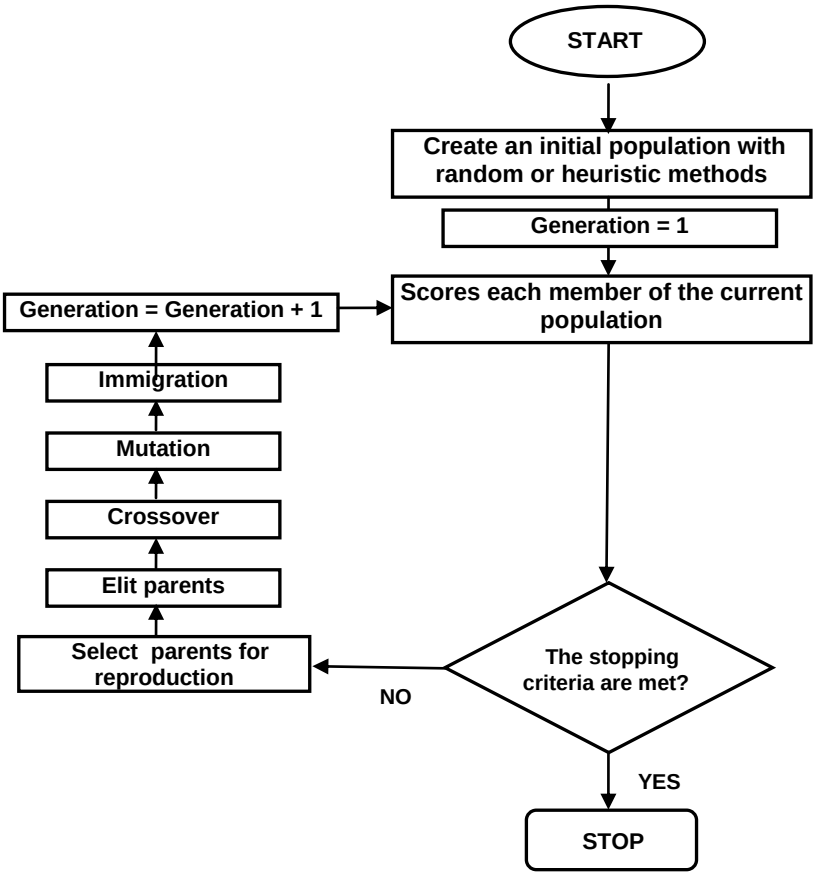


Figure 2.1 Simple Genetic Algorithm Cycle

In here, to compare candidate parameter vector obtained by GA with the best parameter vector kept in previous iteration of main algorithm, Metropolis-Hastings method (Metropolis at al., 1953; Hastings, 1970) is used. Metropolis-Hastings is a general MCMC method to generate samples $\left\{\theta^{iter}\right\}$ from a desired invariant probability distribution $\pi(\cdot)$ that may not be a standard distribution. $\left\{\theta^{iter}\right\}$ can be generated through a Markov Chain having $\pi(\cdot)$ as its stationary distribution (For detail, see *Appendix D*). In producing process, a candidate state θ^{iter+1} is drawn from a specified proposal distribution $Q(\theta^{iter+1}\left|\theta^{iter}\right.)$ that depends on the state θ^{iter} of the previous iteration, and then deciding whether accept or not the candidate state based on its probability density relative to that of the previous state with respect to invariant distribution $\pi(\cdot)$. That is, the densities of parameters determined in the previous and current iterations are compared with respect to $\pi(\cdot)$. According to original Metropolis Algorithm, a specified proposal distribution $Q(\cdot)$ must be symmetrical distribution, satisfying the condition

$Q(\theta^{iter} \left| \theta^{iter+1}) = Q(\theta^{iter+1} \right| \theta^{iter})$. However, the generalization by Hastings (1970) abolish this rule since it is also permissible for $Q(\theta^{iter+1}, \theta^{iter})$ not to depend on θ^{iter} at all in which case the algorithm is called Independence Chain Metropolis-Hastings. Because of this independency, the algorithm with a suitable proposal density function can reduce serial correlation between successive state and offer high efficiency than the original Metropolis algorithm, which is based on random walk. Besides, due to random walk problem, the original Metropolis algorithm can be very slow when applied to problems such as Bayesian learning of neural networks (Neal, 1996). Therefore, Neal (1992) proposed HMC to avoid random walk aspect of the exploration through parameter space.

In this study, to ensure that resulting sample of parameter vectors represents the required invariant distribution $\pi(\cdot)$, the posterior distribution of parameters in (2.14) is considered as invariant distribution. In order to sample parameters (weights and biases) from this posterior distribution, normalization constant $Z_S(\alpha, \beta)$ is omitted from the posterior distribution, and then the following function, which is called energy function (Bishop, 1995, Neal, 1996), can be found as follow:

$$E(\theta) = S(\theta) \tag{2.43}$$

In here, it can be seen that the negative logarithm of (2.14) is the error function in (2.17). Therefore, the candidate parameter vector obtained by means of GA search with recursive approach from the error function $S(\theta)$ is compared with that of previous iteration by Metropolis-Hastings algorithm as follows:

- if $p(\theta^* \mid D) \geq p(\theta^{iter-1} \mid D)$, accept candidate; otherwise ,
  - Draw $u \sim U(0,1)$, if $p(\theta^* \mid D) < p(\theta^{iter-1} \mid D)$, accept with probability $p(\theta^* \mid D) / p(\theta^{iter-1} \mid D) > u$; else,
- Reject candidate, and then let $\theta^{iter} = \theta^{iter-1}$

where θ^* and θ^{iter-1} are parameter vectors of candidate (of current iteration) and the selected at the previous iteration respectively. In iterative processes of GA, the updating the efficient parameters α_{eff} and β_{eff} is reconcile noise with over-fitting in error function simultaneously.

In MCMC the complex integrals in the marginalization are approximated via drawing samples from the joint probability distribution of all the model parameters and hyperparameters

(For detail, see *Appendix E*). For instance, the prediction of output for given new input vector x_{N+1}^{new} can be made by

$$\hat{y}_{k,N+1} = \mathrm{E}(y_{k,N+1} \mid x_{1:N+1},\, y_{1:N}) = \int \hat{f}_k(\theta,\, x_{1:N+1})\, p(\theta \mid x_{1:N}, y_{1:N})\, d\theta \tag{2.44}$$

Equation (2.44) corresponds to the expectation of the posterior predictive distribution, and this is approximated using a sample of vector θ^{iter} drawn from the posterior distribution of parameters:

$$\hat{y}_{k,N+1} \approx \frac{1}{M-L} \sum_{iter=L+1}^{M} \hat{f}_t(x_{N+1}, \theta^{iter}) \tag{2.45}$$

Note that samples from the posterior distribution are drawn during the "learning phase", which may be computationally very expensive, but predictions for the new data can be calculated quickly using the same stored samples (*M-L*) exception of those corresponding to iterations (*L*) in burn-in process. Thus, Markov Chain becomes more stationary in last (M –L) iterations.

The Genetic MC with recursive hyperparameters proposed for the Gaussian approach of Bayes neural networks are discussed in application part. Besides, the detailed knowledge related with Markov Chain is given in *Appendix E*.

3 FULL BAYESIAN APPROACH FOR NEURAL NETWORKS

For MLP networks, the Gaussian approach introduced by MacKay in the evidence framework (MacKay, 1992) or marginalization over hyperparameters (MacKay, 1994) (also called type II Maximum Likelihood approach (Berger, 1985)) is the first practical Bayesian method for neural networks where specific values are estimated for the hyperparameters.

In the evidence framework, the hyperparameters α and β are set to values that maximize the evidence of the model $P(D|\alpha,\beta)$ that is, the marginal probability for the data given the hyperparameters, integrated over the parameters $P(D|\alpha,\beta)=\int P(D|\theta,\alpha,\beta)P(\theta|\alpha,\beta)d\theta$.

The marginalization framework is to perform the integrations over hyperparameters analytically by means of $p(\theta|D)=\iint p(\theta,\alpha,\beta|D)\,d\alpha\,d\beta$.

At the both of these approaches, a Gaussian approximation is used the posterior of the parameters $P(\theta|D)$, to facilitate closed form integration, and thus the resulting posterior for θ is specified by the mean of the Gaussian approximation (i.e., one network with posterior mean weights).

In full Bayesian approach, any fixed values aren't estimated for any parameters or hyperparameters. Approximations are then needed for the integrations over the hyperparameters to obtain the posterior for the parameters and over the parameters to obtain the predictions of the model, as shown in (2.22). The correctness of the inference depends on the accuracy of the integration method; hence it depends on the problem whether approximation method is appropriate or not. Methods for approximating the integrations in neural network models include MCMC techniques for numerical integration, ensemble learning (Barber and Bishop, 1998), which aims to approximate the posterior distribution by minimizing the Kullback-Leibler divergence between the true posterior and a parametric approximating distribution, variational approximations (Jordan et al., 1998) for approximating the integration by a tractable problem,

and mean field approach (Winther, 1998), where the problem is simplified by neglecting certain dependencies between the random variables.

It is worth noticing, that also in full hierarchical Bayesian models there are large amounts of fixed prior knowledge, in the selection of the parametric form for the distributions (priors and noise models), that is based on uncertain assumptions. In such models, no guesses are made for exact values of the parameters or any smoothness coefficients or other hyperparameters, but guesses are made for the exact forms of their distributions. The goodness of the model depends on these guesses, which in practical applications makes it necessary to carefully validate the models, using Bayesian posterior analysis (Gelman et al., 1995), or cross-validation (Gelfand, 1996; Vehtari and Lampinen, 2000).

3.1 Role of Prior Knowledge in Statistical Models

As is known, the main difference distinguishing Bayesian approach from the Maximum Likelihood methods is the prior information. However, that the role of prior knowledge is equally important in the other approaches, including the Maximum Likelihood. In Bayesian approach, basically all generalization is based on the prior knowledge, since the training samples provide information only at those points, and the prior knowledge provides the necessary link between the training samples and the not yet measured future samples.

According to "No-free-lunch" (NFL) theorem, the attribution of priority is explained this way: "*if the class of approximating functions is not limited, any learning algorithm (i.e., procedure for choosing the approximating function) can as readily perform worse or better than randomly, measured by off-training set (OTS) error, and averaged over loss functions"* (Wolpert, 1996a,b). This theorem implies that it is not possible to find a learning algorithm that is universally better than random (Lampinen and Vehtari, 2001). In other words, if any priori do not assumed, the learning algorithm can't learn anything from the training data that would generalize to the off-training set samples.

The cross-validation (CV) method for model selection was analyzed by Wolpert and Macready (1995) and Wolpert (1996b). According to these studies, the NFL theorem can be applied to CV as well. That is, while CV is used to choose from a very large (actually infinite) set of models without priors on functions, this method doesn't guarantee any generalization at all.

In the Bayesian approach, the prior knowledge is defined as prior distributions for the model parameters, and hyperpriors for the parameters of the prior distributions. However, determining the relation between the actual domain knowledge of the experts and the priors for the model

parameters is not easy for complex models as the neural networks. Therefore, the incorporation of the very sophisticated knowledge with the prior of parameters may be difficult in practice. In such a situation, Bayesian approach provides the principled way to do inference when some of the prior knowledge is lacking or vague, the unknown attributes can be predicted by marginalization or integrating over the posterior distribution of the unknown variables.

In Bayesian literature, the use of "non-informative" priors is often referred as the "objective Bayesian approach", in contrast to informative (subjective) priors that correspond to the subjective choices. However, according to NFL theorem, this requires that the hypothesis space is already so constrained, that it contains the sufficient amount of prior information that is needed to be able to learn a generalizing model (Lemm, 1999). By using "non-informative" priors, the fixed, or guessed, choices can be moved to higher levels of hierarchical models.

Goel and Degroot (1981) showed that the training data contains little information of hyperparameters in the high level of hierarchy, so that the prior and posterior for the hyperparameters become more equal. Thus the models are less sensitive to the choices made in higher levels. In result, the higher level priors are in general less informative, and thus less subjective.

In complex statistical models, another complicated issue is determination of the correct number of degrees of freedom. The degrees of freedom depend on the number of the training samples, distribution of noise in the samples and the complexity of the underlying phenomenon to be modeled. Therefore, the complexity of the model cannot be defined by only one number, the total number of degrees of freedom, but instead the models have multiple dimension of complexity. In the Bayesian approach one can use a vague prior for the total complexity (called the effective number of parameters), and use a hierarchical prior structure to allow different complexity in different parts of the model. For example, the parameters may be assigned to different groups, so that in each group the parameters are assumed to have the same hyperparameter, while different groups can have different hyperparameters. Then a hyperprior is defined to explain the distribution of all the hyperparameters. MacKay (1994) and Neal (1996) defined each group of weights connected to the same input having common variance hyperparameters, while the weight groups can have different hyperparameters.

3.2 Network architectures

The networks are usually used to define models for the conditional distribution of a set of target values given a set of input values. There are three sorts of models, corresponding to three types of targets: real valued targets (regression models), binary valued targets (logistic regression), and class targets taking on values from a small a data set (a generalized logistic regression or softmax model). For regression and logistic regression models, numbers of target values are equal to the number of network outputs. For the soft-max model, there is only one target, the number of possible values for this target being equal to the number of network outputs.

The distribution of real valued targets, $y_{i,t}$ (*i-th* observation of *t -th* target), in case with inputs x_i can be modeled by independent Gaussian distribution with means given by the corresponding to network outputs, and with standard deviations given by the hyperparameters σ_t , the noise levels for targets. The probability density for a target given the associated inputs and the network parameters is then

$$p(y_{t,i}|x_i,\theta)=\frac{1}{(2\pi)^{1/2}\tau_t^{-1/2}}exp\left(-\frac{\tau_t}{2}\left\{y_{t,i}-\hat{f}_t(x_i,\theta)\right\}^2\right) \tag{3.1}$$

where $\tau_t=\sigma_t^{-2}$ is called as the precision. Alternatively, each case may have its own set of standard deviations, $\tau_{t,i}=\sigma_{t,i}^{-2}$, for heteroscedasticity problem (or situation including different variance). Neal (1992, 1996) proposed to use gamma distribution for precisions with means of μ_t and shape parameter α_{Noise} (or inverse gamma distribution for standard deviations):

$$p(\tau_{t,i}|\mu_t)=\frac{(\alpha_{Noise}/2\mu_t)^{\alpha_{Noise}/2}}{\Gamma(\alpha_{Noise}/2)}\tau_{t,i}^{\alpha_{Noise}/2-1}exp\left(-\tau_{t,i}\alpha_{Noise}/2\mu_t\right) \tag{3.2}$$

The distribution in (3.1) corresponds to the degenerate Gamma distribution with $\alpha_{Noise}=\infty$. Otherwise, integrating out $\tau_{t,i}$ gives t-distribution for the target with "degrees of freedom" α_{Noise}:

$$p(y_{t,i}|x_i,\theta)=\frac{\Gamma(\nu+1/2)}{\Gamma(\nu/2)\sqrt{\pi\nu}\sigma_t}\left[1+(y_{t,i}-\hat{f}_t(x_i,\theta)^2/\nu\sigma_t^2\right]^{(\nu+1)/2} \tag{3.3}$$

Let consider all observations of the *t-th* target in training set given the inputs, network parameters and the noise standard deviation, because of independency of all observations, the conditional distribution of *t- th* target can be constituted as follow:

$$p(y_{t,1}, y_{t,2} \ldots y_{t,N} \mid x_{1:N}, \theta, \tau_t) = \frac{1}{(2\pi)^{N/2} \tau_t^{-N/2}} exp\left(-\frac{\tau_t}{2} \sum_{i=1}^{N} \left\{ y_{t,i} - \hat{f}_t(x_i, \theta) \right\}^2 \right) \quad (3.4a)$$

where $\tau_t = \sigma_t^{-2}$ is the noise standard deviation for all observations of *t-th* target. Alternatively, the conditional distribution of *t -th* target can be used for heteroscedasticity problem as follow:

$$p(y_{t,1}, y_{t,2} \ldots y_{t,N} \mid x_{1:N}, \theta, \tau_{t,i}) = \prod_{i=1}^{N} \frac{1}{(2\pi)^{1/2} \tau_{t,i}^{-1/2}} exp\left(-\frac{1}{2} \tau_{t,i} \left\{ y_{t,i} - \hat{f}_t(x_i, \theta) \right\}^2 \right) \quad (3.4b)$$

In here, Gamma prior can be similarly defined as in (3.2) for heteroscedasticity. Thus, the conditional distribution of fix precision τ_t of noise is given by

$$p(\tau_t \mid x_{1:N}, y_{t,1:N}, \theta) \propto \tau_t^{(\alpha_{Noise}+N)/2-1} exp\left(-\frac{\tau_t}{2} \left[\alpha_{Noise} / \mu_t + \sum_{i=1}^{N} \left\{ y_{t,i} - \hat{f}_t(x_i, \theta) \right\}^2 \right] \right) \quad (3.5)$$

or all precisions of individual cases for heteroskedastic situation can be given as follow:

$$p(\tau_{t,1}, \ldots, \tau_{t,N} \mid x_{1:N}, y_{t,1:N}, \theta) \propto \prod_{i=1}^{N} \left[\tau_t^{\alpha_{Noise}-1/2} exp\left(-\tfrac{1}{2} \tau_{t,i} \left(\tfrac{\alpha_{Noise}}{\mu_t} + \left\{ y_{t,i} - \hat{f}_t(x_i, \theta) \right\}^2 \right) \right) \right] \quad (3.6a)$$

where the precisions of single case is independent each other, and defined as follow:

$$p(\tau_{t,i} \mid x_i, y_i, \theta) \propto \tau_t^{\alpha_{Noise}-1/2} exp\left(-\frac{1}{2} \tau_{t,i} \left[\tfrac{\alpha_{Noise}}{\mu_t} + \left\{ y_{t,i} - \hat{f}_t(x_i, \theta) \right\}^2 \right] \right) \quad (3.6b)$$

In here, it can be noticed that, if a single precision τ_t for all target values in (3.5) is used, either higher level hyperparameters linking τ_t should be defined or alternatively a single τ_t should have a t-distribution rather than a Gaussian (Neal, 1996; Lampinen and Vahtari, 2001).

After discussing hyperparameter for the noise level, the other issue is to consider the hyperparameter for prior distribution of parameters. In the simplest cases, any hyperparameter controls the standard deviation of all parameters in a certain group. Such a group might consist of weights on all connections from a particular unit to units, or the biases for all units of one type. In detail, let parameters in particular group be $w_k^I = [w_{1k}^I \;\; w_{2k}^I \;\; \ldots \;\; w_{mk}^I]$, which the row vector is, include the weights between all inputs and *k-th* neuron in the hidden layer. For this group, it might be assumed that all weights are independent, and have Gaussian distributions with mean zero and standard deviation σ_w. Alternatively, in order to determine the relation between any weight groups with input vectors, a distinctive weight group between any input vector with neurons in the hidden layer can be defined. That is, this distinctive weight group between input j

with the neurons in the hidden layer can be constituted as $w_j^I = [w_{j1}^I \;\; w_{j2}^I \;\; ... \;\; w_{jp}^I]$. It is convenient to represent this standard deviation in terms of corresponding to precision, defined to be $\tau_w = \sigma_w^{-2}$. The distributions of parameter groups for both definitions can be given as follows:

$$p(w_{1k}^I \;\; w_{2k}^I \;\; ... \;\; w_{mk}^I \,|\, \tau_w) = (2\pi)^{-m/2} \tau_w^{\,m/2} \exp\left(-\tau_w \sum_{j=1}^{m} (w_{jk}^I)^2 / 2 \right) \tag{3.7a}$$

or

$$p(w_{j1}^I \;\; w_{j2}^I \;\; ... \;\; w_{jp}^I \,|\, \tau_w) = (2\pi)^{-p/2} \tau_w^{\,p/2} \exp\left(-\tau_w \sum_{k=1}^{p} (w_{jk}^I)^2 / 2 \right) \tag{3.7b}$$

In this study, the weight groups defined in (3.7b) are preferred, thus the impact of any input over targets can be measured by variance of the related weight group. The detailed knowledge on defining parameter groups are given in the training procedure below.

The precision of any weight group is given a Gamma distribution with some mean ω_w and shape parameter specified by α_w, with density

$$p(\tau_w) = \frac{(\alpha_w / 2\omega_w)^{\alpha_w/2}}{\Gamma(\alpha_w / 2)} \tau_w^{\alpha_w/2 - 1} \exp\left(-\tau_w \alpha_w / 2\omega_w\right) \tag{3.8}$$

In here, if there is no hierarchical structure, then the values of ω_w and α_w can be considered the fixed. The otherwise, the hyper-prior for hyperparameter ω_w should be defined (the top level hierarchical).

The prior for τ_w is conjugate to its use in defining the distribution for w_j^I. The conditional distribution for the τ_w given w_j^I is also the Gamma form:

$$p(\tau_w \,|\, w_{j1}^I \;\; w_{j2}^I \;\; ... \;\; w_{jp}^I) \propto \tau_w^{\alpha_w/2 - 1} exp\left(-\tau_w \alpha_w / 2\omega_w\right) \tau_w^{\,p/2} exp\left(-\tau_w \sum_{k=1}^{p} (w_{jk}^I)^2 / 2 \right)$$

$$= \tau_w^{\alpha_w + p/2 - 1} exp\left(-\tau_w \left[\alpha_w / \omega_w + \sum_{k=1}^{p} (w_{jk}^I)^2 \right] \Big/ 2 \right) \tag{3.9}$$

From (3.9), the prior for τ_w can be interpreted as specifying imaginary parameter values α_w, whose average squared magnitude is $1/\omega_w$. Small values of α_w produce vague priors for τ_w.

The value τ_w can be drawn from the conditional distribution in (3.9) by means of Gibbs sampling updates, since given vector w_j^I. Besides, the value of τ_w is independent of the other parameters, hyperparameters, and target values.

In addition, the distribution (3.9) given to a single parameter may also be t-distribution rather than a Gaussian. Since t-distribution can be represented as mixtures of Gaussian distribution with precision given by Gamma distributions, this can be implemented by extending the hierarchy downward, to include implicit precision variables associated with individual parameters (Neal, 1996).

The prior distributions for the parameters of network are defined in terms of hyperparameters. Conceptually, this implementation provides for one hyperparameter for every parameter, but these lowest-level hyperparameters are not explicitly represented. The mid-level hyperparameter control the distribution of a group of low-level hyperparameters. The high-level (or "common") hyperparameters control the distribution of mid-level hyperparameters, or of the low-level hyperparameters for parameters types with no mid-level hyperparameters. The same three level schemes can be used for noise levels in regression models.

The hierarchy process of network parameters can be defined as discussed above:

$$p(\tau_1) = \frac{(\alpha_0 / 2\omega_o)^{\alpha_0/2}}{\Gamma(\alpha_0/2)} \tau_1^{\alpha_0/2-1} exp\left(-\tau_1 \alpha_0 / 2\omega_0\right) \qquad \textit{Top-level} \tag{3.10}$$

$$p(\tau_w | \tau_1) = \frac{(\alpha_1 / 2\tau_1)^{\alpha_w/2}}{\Gamma(\alpha_1/2)} \tau_w^{\alpha_w/2-1} exp\left(-\tau_w \alpha_1 / 2\tau_1\right) \quad \textit{Middle-level} \tag{3.11}$$

$$p(w_{jk}^I | \tau_w) = (2\pi)^{-1/2} \tau_w^{1/2} exp\left(-\tau_w (w_{jk}^I)^2 / 2\right) \qquad \text{Low-}\textit{level} \tag{3.12}$$

where α_0 and ω_o in (3.10) are the fixed shape parameter and mean of precision τ_1; α_1 and τ_1 in (3.11) are shape parameter and mean of τ_w respectively, and w_{jk}^I, which has Gaussian distribution with mean zero and precision τ_w, is the weight value between *j-th* input and *k-th* neuron in hidden layer,. Since the parameters of α_0 and ω_o are defined in top-level, they have to be fixed. Neal (1996) proposed to the following scaling rules for ω_o, where ω is specified base precision and *g* is the number of source units (input or neuron number) :

$$\omega_o = \begin{cases} \omega n & for\ \alpha = \infty \\ \omega n\alpha/(\alpha-2) & for\ \alpha > 2 \\ \omega n \log n & for\ \alpha = 2 \quad (but\ \ fudged\ \ to\ \ \omega n\ \ if\ \ g < 3\ ) \\ \omega n^{2/\alpha} & for\ \alpha < 2 \end{cases} \tag{3.13}$$

Alternatively, each individual weight (or bias) may have its own precision corresponding to τ_{jk} with mean ω_{jk} and shape α_{jk}. But, (3.12) corresponds to degenerate distribution with $\alpha_{jk} = \infty$. Otherwise, t-distribution can be used for each weight:

$$p(w_{jk} \mid \sigma_{jk}) = \frac{\Gamma[(\alpha_{jk}+1)/2]}{\Gamma(\alpha_{jk}/2)\sqrt{\pi\alpha_{jk}}\sigma_{jk}} [1 + w_{jk}^2 / \alpha_{jk}\sigma_{jk}^2]^{(\alpha_{jk}+1)/2} \tag{3.14}$$

All distribution forms introduced so far are applicable to be modeled all weights and biases of the neural networks. Once defined priors and full conditional distributions of parameters and precisions at the prior level, to make inferences and predictions, the next step is to constitute the joint posterior distribution. In here, the normalization constant in the posterior distribution is omitted, so that the posterior distribution can be expressed as

$$p(\theta, \tau_{Noise}, \tau_\theta \mid D) \propto p(D \mid \theta, \tau_{Noise}, \tau_\theta)\, p(\theta, \tau_{Noise}, \tau_\theta) \tag{3.15}$$

$$= p(D \mid \theta, \tau_{Noise})\, p(\theta \mid \tau_{Noise}, \tau_\theta)\, p(\tau_{Noise}, \tau_\theta) \tag{3.16}$$

$$= p(D \mid \theta, \tau_{Noise})\, p(\theta \mid \tau_\theta)\, p(\tau_{Noise}, \tau_\theta) \tag{3.17}$$

$$= p(D \mid \theta, \tau_{Noise})\, p(\theta \mid \tau_\theta)\, p(\tau_{Noise})\, p(\tau_\theta) \tag{3.18}$$

where $\theta = \left(w^I, w^{II}, b^I, b^{II}\right)$, $\tau_\theta = (\tau_{w^I}, \tau_{w^{II}}, \tau_{b^I}, \tau_{b^{II}})$, $p(\tau_\theta) = p(\tau_{w^I})\, p(\tau_{w^{II}})\, p(\tau_{b^I})\, p(\tau_{b^{II}})$, $D = \left(x_{1:N}, y_{1:N}\right)$ and $p(\theta \mid \tau_\theta) = p(w^I \mid \tau_{w^I})\, p(w^{II} \mid \tau_{w^{II}})\, p(b^I \mid \tau_{b^I})\, p(b^{II} \mid \tau_{b^{II}})$.

In detail, $p(D \mid \theta, \tau_{Noise})$ and $p(\theta, \tau_{Noise}, \tau_w, \tau_b)$ in (3.15) are likelihood and prior that include the network parameters, noise and hyperparameters respectively. For the likelihood in (3.15), data is independent of precision vector of parameters; for the second distribution in (3.16), the parameter vector θ (weights and biases) is independent of noise precision, and for the third distribution in (3.17) precisions of noise and parameters are independent of each other's. In here, if three level hierarchy is used, $p(\tau_\theta)$ in (3.18) corresponds to $p(\tau_\theta \mid \tau_0) = p(\tau_{w^I} \mid \tau_{w^I}^0)\, p(\tau_{w^{II}} \mid \tau_{w^{II}}^0)\, p(\tau_{b^I} \mid \tau_{b^I}^0)\, p(\tau_{b^{II}} \mid \tau_{b^{II}}^0)$ where $\tau_0 = \left(\tau_{w^I}^0, \tau_{w^{II}}^0, \tau_{b^I}^0, \tau_{b^{II}}^0\right)$ the highest

level hyperparameters is. Similarly, $p(\tau_{Noise})$ in (3.18) corresponds to $p(\tau_{Noise} \,|\, \tau_{Noise}^{0})$ where τ_{Noise}^{0} the highest level hyperparameter is.

3.3 Training procedure

Neal (1996) has introduced to MCMC implementation of Bayesian learning for MLPs. Introduction to basic MCMC methods and many applications in statistical data analysis can be found in (Gilks et al., 1996), and more theoretical treatment in (Robert and Casella, 1999). In MCMC the complex integrals in the marginalization are approximated via drawing samples from the joint probability distribution of all the model parameters and hyperparameters. For instance, the prediction of output for given new input vector x_{N+1}^{new} can be made by

$$\hat{y}_{k,N+1} = \mathrm{E}(y_{k,N+1} \mid x_{1:N+1}, y_{1:N}) = \int \hat{f}_k(\theta, x_{1:N+1})\, p(\theta \mid x_{1:N}, y_{1:N})\, d\theta \tag{3.19}$$

The integral in (3.19) corresponds to the expectation of the posterior predictive distribution, and this is approximated using a sample of vector θ^{iter} drawn from the posterior distribution of parameters:

$$\hat{y}_{k,N+1} \approx \frac{1}{M-L} \sum_{iter=L+1}^{M} \hat{f}_t(x_{N+1}, \theta^{iter}) \tag{3.20}$$

Note that samples from the posterior distribution are drawn during the "learning phase", which may be computationally very expensive, but predictions for the new data can be calculated quickly using the same stored samples (*M-L*) exception of those corresponding to iterations (*L*) in burn-in process.

In the MCMC, samples are generated using a Markov chain that has the desired posterior distribution as its stationary distribution. For achieving this aim, HMC introduced by Duane et al., (1987) can be used. Therefore, Neal (1992) first applied HMC to Bayesian neural network for sampling the parameters, and Gibbs sampling (Geman and Geman, 1984) for hyperparameters. For the other possible sampling schemes see (Insua and Müller, 1998; de Freitas et al., 2000; Lampinen and Vahtari, 2001). HMC is an elaborate Monte Carlo method, which makes efficient use of the gradient information to reduce random walk behavior. The gradient indicates in which direction one should go to find states with high probability. But, the gradient information is not enough to explore all space freely at the high dimensions problem with lots of local minimum (or maximum). To overcome these problems, evolutionary or parallel algorithms as Genetic Algorithm can be used to estimate probable parameters, and

Metropolis-Hastings and Gibbs are used to estimate hyperparameters. Besides, fuzzy membership functions are used to clarify un-probabilistic uncertainty in GA and MCMC procedures. In this study, the proposed novel hybrid MCMC algorithm consists of following steps:

First, a simple hierarchical prior for the weights, called Automatic Relevance Determination (ARD) (MacKay, 1994; Neal, 1996) are defined. In ARD each group of weights connected to the same input *j=1, 2, ..., m* has common variance hyperparameters, while the weight groups can have different hyperparameters. Thus, the irrelevant inputs should have smaller weights in the connections to the hidden units than more important weights. The weights with separate hyperparameters of irrelevant inputs can have tighter priors that reduce such weights more effectively towards zero than having the common larger variance for all the input weights (Neal, 1996; Lampinen and Vehtari, 2001).

In more detailed, the sampling procedure can be introduced as follows:

- Set hierarchical structure for priors of parameters θ and precisions τ_θ:

$\tau_1 \sim Gam(\alpha_0, \omega_0)$	$\tau_1 \sim Gam(\alpha_0, \omega_0)$	$\tau_1 \sim Gam(\alpha_0, \omega_0)$	$\tau_1 \sim Gam(\alpha_0, \omega_0)$
$\tau_{w^I} \mid \tau_1 \sim Gam(\alpha_{w^I}, \omega_{w^I})$	$\tau_{w^{II}} \mid \tau_1 \sim Gam(\alpha_{w^{II}}, \omega_{w^{II}})$	$\tau_{b^I} \mid \tau_1 \sim Gam(\alpha_{b^I}, \omega_{b^I})$	$\tau_{b^{II}} \mid \tau_1 \sim Gam(\alpha_{b^{II}}, \omega_{b^{II}})$
$w^I \mid \tau_{w^I} \sim N(0, \tau_{w^I})$	$w^{II} \mid \tau_{w^{II}} \sim N(0, \tau_{w^{II}})$	$b^I \mid \tau_{b^I} \sim N(0, \tau_{b^I})$	$b^{II} \mid \tau_{b^{II}} \sim N(0, \tau_{b^{II}})$

(3.21)

In here, in order to set hierarchical structure for weights and biases, scheme discussed in (3.10) – (3.12) is used. Then, to take into account heteroscedasticity problem, its own set of precisions for each case of targets is determined as in (3.2). Thus, priors of precisions can be defined as follows:

$$\tau_1 \sim Gam(\alpha_1, \omega_1) \tag{3.22}$$

$$\tau_{t,i} \mid \tau_1 \sim Gam(\alpha_{Noise}, \omega_{Noise}) \tag{3.23}$$

- Draw τ_θ precision of parameters using its conditional when given parameter vector.

In this step, precisions of weights and biases have to be produced from its conditionals in (3.9) when given parameter vector of previous iteration as follow:

$$p(\tau_w^I \mid w_{j1}^I \; w_{j2}^I \; ... \; w_{jp}^I) \qquad \text{Gibbs Sampling} \tag{3.24}$$

where $\left\{ w_{j1}^{I} \;\; w_{j2}^{I} \;\; ... \;\; w_{jp}^{I} \right\}$ is weight vector of previous iteration that corresponds to connections defined from *j-th* input to all neurons in hidden layer.

$$p(\tau_w^{II} \left| w_{1t}^{II} \;\; w_{2t}^{II} \;\; ... \;\; w_{pt}^{II} \right.) \qquad \text{Gibbs Sampling} \tag{3.25}$$

where $\left\{ w_{1t}^{II} \;\; w_{2t}^{II} \;\; ... \;\; w_{pt}^{II} \right\}$ is weight vector of previous iteration that corresponds to connections defined from all neurons of hidden layer to target *t*.

$$p(\tau_b^{I} \left| b_1^{I} \;\; b_2^{I} \;\; ... \;\; b_p^{I} \right.) \qquad \text{Gibbs Sampling} \tag{3.26}$$

where $\left\{ b_1^{I} \;\; b_2^{I} \;\; ... \;\; b_p^{I} \right\}$ is bias vector of previous iteration that corresponds to biases defined for all neurons in the hidden layer.

- Draw noise precisions τ_{Noise} of each case using $p(\tau_{Noise} \left| x_{1:N}, y_{t,1:N}, \theta \right.)$ defined in (3.6b) by means of Metropolis - Hastings.
- Draw weights and biases using joint posterior distribution $p(\theta, \tau_{Noise}, \tau_\theta \mid D)$ in (3.18) by means of Hybrid Monte Carlo.

$$p(\theta, \tau_{Noise}, \tau_\theta \mid D) \propto p(D \mid \theta, \tau_{Noise})\, p(\theta \left| \tau_\theta \right.)\, p(\tau_{Noise})\, p(\tau_\theta) \tag{3.27}$$

3.3.1 Hybrid Monte Carlo by Genetic Algorithms

In order to apply the Hybrid Monte Carlo method, firstly the desired distribution in terms of a potential energy function must be formulated. This energy function is a function of network parameters that are purposed to sample from the joint posterior distribution. The potential energy function can be derived from the negative logarithm of (3.27) that is proportional to joint posterior distribution as follow:

$$\begin{aligned} E(\theta) &= -\log p(\theta, \tau_{Noise}, \tau_\theta \mid D) \\ &= -\log p(D \mid \theta, \tau_{Noise}) - \log p(\theta \left| \tau_\theta \right.) - \log p(\tau_{Noise}) - \log p(\tau_\theta) \\ &= -\log p(D \mid \theta, \tau_{Noise}) - \log p(\theta \left| \tau_\theta \right.) - F(\tau) \end{aligned} \tag{3.28}$$

where $F(\tau)$ is a function of the hyperparameters that are updated by Gibbs sampling as discussed above in each successive iteration. Thus, energy function changes whenever the

hyperparameters changes. In here, any terms that depend only on the hyperparameters can be omitted from energy function. The energy function is similar to the error function with weight decay penalty and negative logarithm of posterior in (2.14) used by Gaussian approximation. However, the main objective in Monte Carlo implementation of Bayesian learning is to sample the network parameters from the posterior distribution by simulations, rather than to find the minimum of error function.

In this study, the parameters are drawn from posterior distribution in (3.27) by means of Hybrid Monte Carlo, and the hyperparameters are updated by Gibbs sampling. Therefore, firstly the parameters in energy function are updated by Genetic Algorithms, and then parameters are sampled by Metropolis-Hastings in the each iteration of algorithm. This algorithm works in the following scheme:

i. Convert all parameters of error function in (3.28) to binary numbers (or real number), and then gather all parameters into one vector called chromosome in Genetic Algorithms.
ii. Create prospective vectors at the certain numbers (or initial population) by means of heuristic or random methods.
iii. Calculate scores of all individuals over error function called as fitness function in Genetic Algorithms (In this process, the fuzzy membership function is used to scale scores of candidate members).
iv. Select favorable individuals (or vectors) into mate pool for the next generation by heuristic or stochastic methods (Stochastic method is preferred).
v. Apply elite, crossover and mutation process to selected individuals called as parents in Genetic Algorithms. This process is called reproduction creates children for the next generation.
vi. If the stopping criteria are met, stop algorithm, otherwise turn to step iii, and then run the algorithm up to meet any stopping criterion.
vii. After stopping algorithm, keep the best parameter vector and then convert all binary numbers (or double vector) to any parameter into float numbers.
viii. Skip Metropolis-Hastings algorithm, and then compare the previous and the current parameter vectors by means of posterior distribution in (3.27).

The simple Genetic Algorithms Cycle can be summarized as following chart:

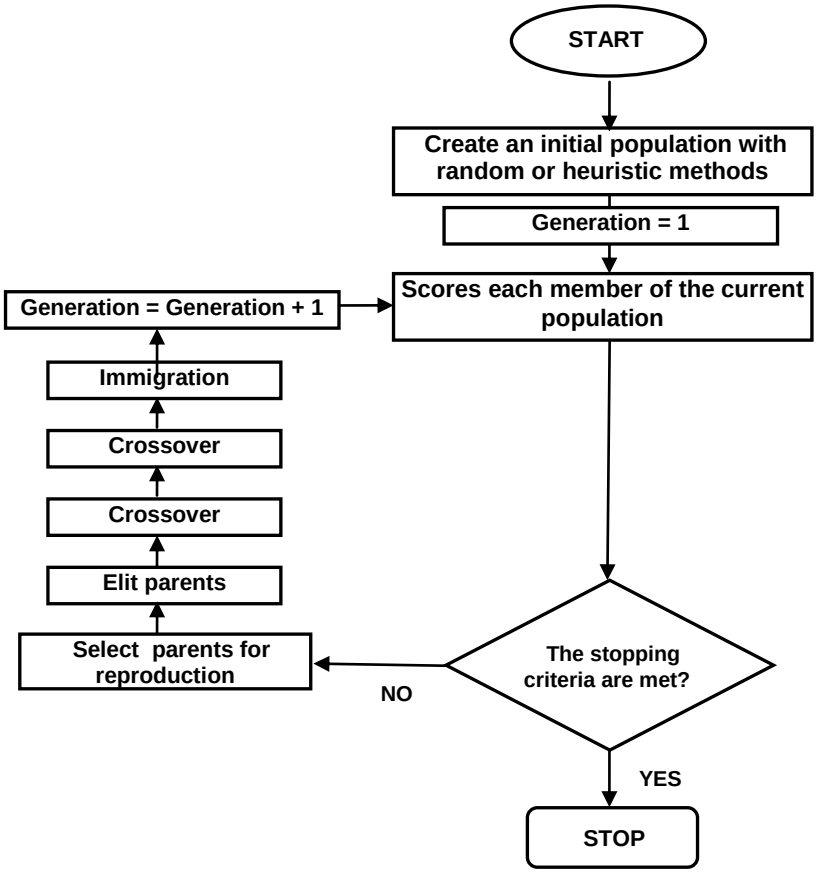


Figure 3.1. Simple Genetic Algorithm Cycle

In the Metropolis-Hastings algorithm, a candidate state drawn from the proposal distribution is compared with previous state by means of the desired or posterior distribution. In this study, the joint posterior distribution in (3.27) is to use in the comparison steps of Metropolis-Hastings algorithm. The metropolis-Hastings algorithm can be summarized as follow:

- if $p(\theta^{*} \mid D) \geq p(\theta^{iter-1} \mid D)$, accept candidate; otherwise ,
  - Draw $u \sim U(0,1)$, if $p(\theta^{*} \mid D) < p(\theta^{iter-1} \mid D)$, accept with probability $p(\theta^{*} \mid D) / p(\theta^{iter-1} \mid D) > u$; else,
- Reject candidate, and then let $\theta^{iter} = \theta^{iter-1}$.

In order to use HMC with GA, the algorithm has to be run for a certain iteration number. After the parameters produced in burn-in process is discarded, the estimations and predictions of outputs can be done by formulation in (3.20).

Alternatively, the simulated annealing introduced by Kirkpatrick at al. (Kirkpatrick at al., 1983) is used with Metropolis Hastings as well. According to the simulated annealing, the standard Metropolis Algorithm can be modified as follows:

- First apply negative logarithm to $p(\theta^* \mid D)$ and $p(\theta^{iter-1} \mid D)$ as $E^* = -\ln p(\theta^* \mid D)$ and $E^{iter-1} = -\ln p(\theta^{iter-1} \mid D)$
- If $E^* < E^{iter-1}$ accept candidate; otherwise,
  - Draw $u \sim U(0,1)$, if $E^* > E^{iter-1}$, accept with probability $\exp\left(-(E^* - E^{iter-1})/T\right) > u$ where T is a parameter generally referred to temperature; else,
- Reject candidate, and then let $\theta^{iter} = \theta^{iter-1}$.

For $T = 1$ the desired distribution is recovered, for $T >> 1$, however, the algorithm explores parameter space much more freely, and can readily escapes from the local error function minima. Simulated Annealing involves starting with a large value of T and then gradually reducing its value during the course of algorithm. Thus, the algorithm has change to settle into a region of high probability (Neal, 1992, 1996; Bishop, 1995b). Unfortunately, the application of simulated annealing to the Monte Carlo algorithm for the Bayesian treatment of neural networks by Neal (1992, 1994) was not found be essential (Bishop, 1995). In this study, alternatively the difference between minimum and maximum of energy function amounts obtained by burn-in process as temperature vale is proposed, and then this amount is gradually reduced to approximate "1" in the each iteration, thus, the impact of T values over the algorithm are observed. Besides, the following criterion with fuzzy membership function instead of $\exp\left(-(E^* - E^{iter-1})/T\right) > u$ criterion in the third step of the simulated annealing is proposed:

"Draw $u \sim U(0,1)$ if $E^i > E^{iter-1}$ accept θ^i with $\mu(E^i) > u$"

In here, $\mu(E^i)$ is a fuzzy membership function as follow:

$$\mu(E^i) = 1/\left[\exp\left(\left(E^i - E^{i-1}\right)/(E_{\max} - E_{\min})\right)\right]^p \quad p = 1, 2, 3 \ldots;\ i = 1, 2, \ldots, iteration \qquad (3.29)$$

$E_{\min}$ and $E_{\max}$ are minimum and maximum energy values that is obtained in iterations.

As a result, the introduced algorithm above not only allows to using the different functional structure, but also it can be used by any distributions exception of Normal and Gamma distributions. Besides, the membership functions used in MC and GA procedures can be determined by trial and error as well.

4 APPLICATIONS

In this study, in order to show performance of Bayesian Learning in training of neural networks, two applications are handled. To give visibility to analysis results, the data sets with two inputs and one target are preferred. The data set in the first application is based on which Vanhatalo and Vehtari (2006) used in their work, and the second one is produced artificially. The neural networks are trained by the improved and traditional approaches, and then analysis results are discussed in detail. In order to train neural networks, the computer that has processor with Intel(R) Core(TM) i3 CPU 2.13GHz, 4GB RAM and 64 bit operating system is used. The software of the improved approaches is written in MATLAB 7.12 package program, and then training of neural networks by traditional approaches is done by the functions of this program.

4.1 Application I

In here, to train Bayesian neural networks, the data set with 225 observations that include two inputs and one target is used, and then this data set is divided into two parts as training (200) and test (25) data demonstrated in Figure 4.1 and Figure 4.2 respectively. In terms of reliability of the analysis, the test data plays very crucial role to evaluate performances of traditional neural networks, since it is not introduced to the neural network together with training data in the training phase. According to Figure 4.1 and Figure 4.2, it can be seen that there is the nonlinear structure between inputs and target data. Because of this non-linear structure, the estimations and predictions done by the multiple linear regressions are not eligible. On the other hand, the neural networks provide the flexible and nonlinear model structure to make more realistic estimations and predictions. The necessary information related with structure of neural networks and optimization algorithms are given in the following implementations.

In implementations, the tangent hyperbolic function that is demonstrated as an example in Figure 4.3 is preferred as activation function, since it is differentiable, and easily calculated. In training stage, the performances of neural networks against problems with different dimensions are observed with trying out the different numbers of neurons in the hidden layer.

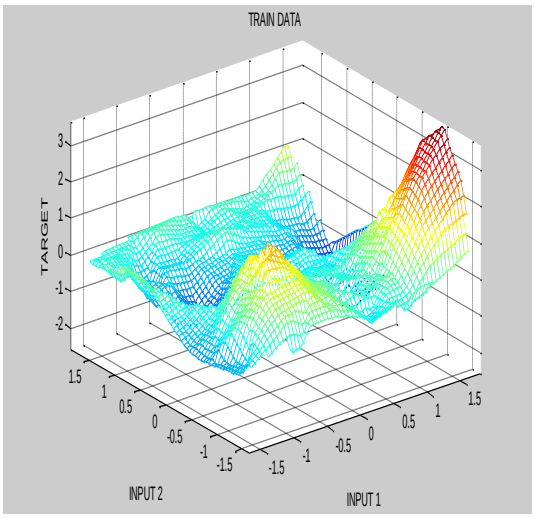


Figure 4.1. Training data

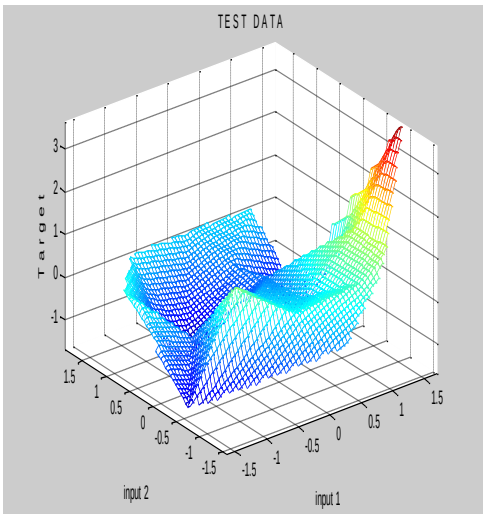


Figure 4.2. Test data

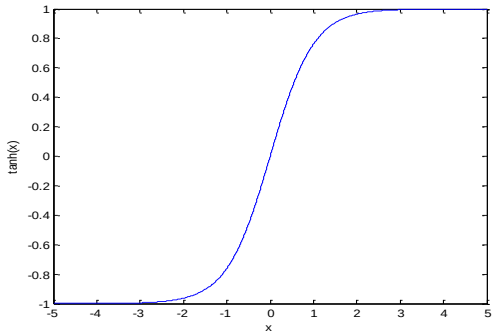


Figure 4.3. Tangent Hyperbolic Function

4.1.1 Gaussian approach with fixed hyperparameter

For Bayesian learning implementation of neural networks, firstly Gaussian approach with fixed hyperparameter is used. To train the neural network, Quasi-Newton optimization algorithm, which is improved by Broyden, Fletcher, Goldfarb, and Shanno (BFGS), is preferred. The advantage of this algorithm is that the approximations of Hessian matrix can be evaluated using the first derivatives without the second derivatives. In the first implementation, it was supposed that both Alpha and Beta hyperparameters are of equal importance, therefore, both hyperparameters were taken as 0.5 during training. After the neuron number in the hidden layer was determined as 25, the algorithm was run for 1000 iterations. In the second implementation, it was supposed that the noise amount included in data set is more important, so that Alpha and Beta hyperparameters were taken as 0.1 and 0.9 respectively. After the neuron number was determined as 25, the algorithm was run for 1000 iterations. The statistical indicators obtained at the end of the training process as correlation between targets and outputs, mean squared errors (MSE) and training time are given in Table and Figures below.

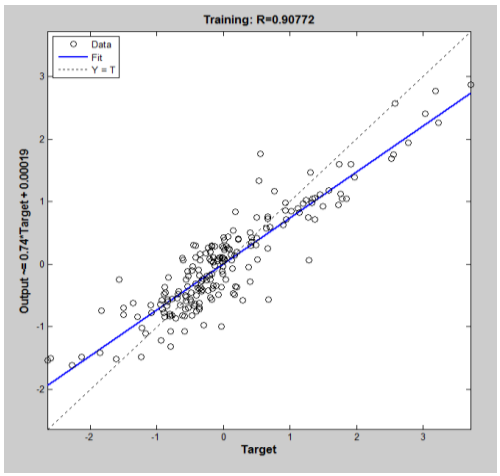


Figure 4.4. Correlation for Alpha=0.5, Beta=0.5

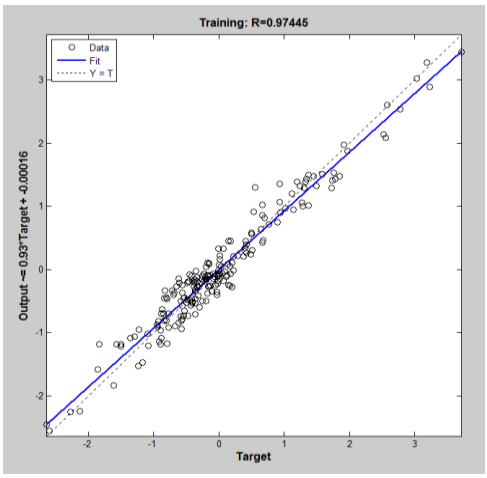


Figure 4.5. Correlation for Alpha=0.1, Beta=0.9

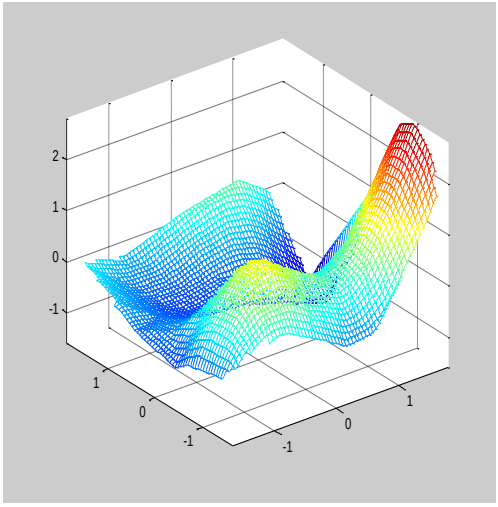


Figure 4.6. Training output for Alpha=0.5, Beta=0.5

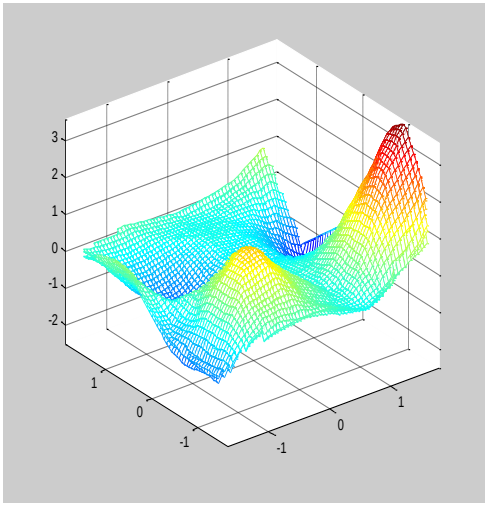


Figure 4.7. Training output for Alpha=0.1, Beta=0.9

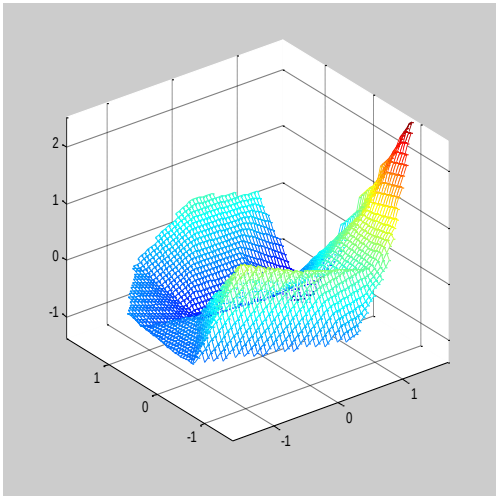


Figure 4.8. Test output for Alpha=0.5, Beta=0.5

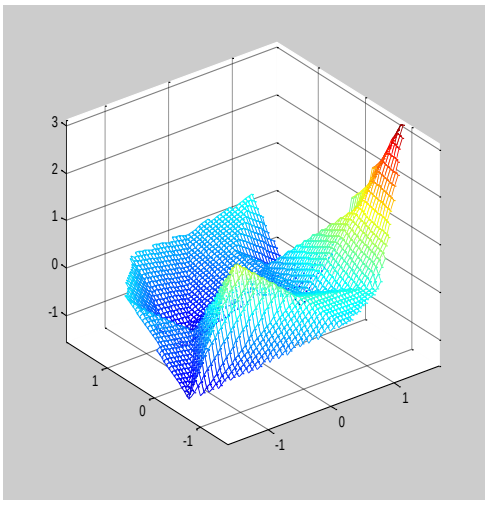


Figure 4.9. Test output for Alpha=0.1, Beta=0.9

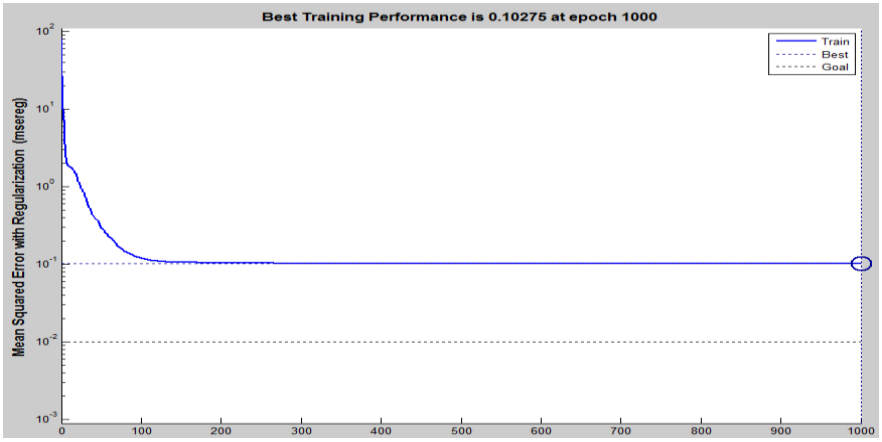


Figure 4.10. The performance for 1000 iterations

Table 4.1 The performance of the fixed hyperparameter approach

Hyperparameter	Neuron number	Training MSE	Test MSE	Training correlation	Test correlation	Training (Seconds)
Alpha= 0.5, Beta=0.5	25	0.1917	0.2543	0.9077	0.9242	29
Alpha= 0.3, Beta=0.7	25	0.0796	0.1142	0.9627	0.9711	29
Alpha= 0.1, Beta=0.9	25	0.0525	0.0631	0.9744	0.9826	27
Alpha= 0.01, Beta=0.99	25	0.0358	0.0432	0.9825	0.9865	27
Alpha= 0.01, Beta=0.99	50	0.0318	0.0502	0.9845	0.9826	41
Alpha= 0.01, Beta=0.99	100	0.0268	0.0537	0.9869	0.9831	103

From Figure 4.4 - Figure 4.9 and Table 4.1, it can be seen that the search method is more sensitive to both the training and test data for Beta = 0.9, so that, when Beta is taken as 0.9 instead of 0.5, the algorithm shows a better performance. As seen in Figure 4.10, at the end of 1000 iterations, the total error in (2.17) and MSE are 0.10275 and 0.0525 respectively. From Figure 4.10, it can be seen that the algorithm reach the best solution at the end of 300 iterations. However, in order to monitor performance of algorithms for different hyperparameters and neuron numbers, the iteration number is fixed as 1000. From Table 4.1, it can be seen that, as Beta is increased, the correlation coefficients take larger values, however, MSE decreases inversely. Besides, if Alpha and Beta are fixed as 0.01 and 0.99 respectively, and the neurons numbers are increased, then MSE of the training data decreases due to over-fitting to data, but, MSE of test data and training time increase as well. Where Beta is larger than Alpha, SSE term

in the error function of the Gaussian approach with fixed hyperparameter in (2.17) becomes more dominant than the second term. From here, it can be concluded that the hyperparameter Alpha plays an important role to decrease over-fitting to training data.

As a result, the Gaussian approach with the fixed hyperparameter performs good-fit to both training and test data where Beta is larger than Alpha. However, the algorithm with larger Beta values is enforced to overfitting the training data where the noise amount included in data set especially is very high. Therefore, determining the best Alpha and Beta hyperparameters is a problem in itself. In such a case, the best Alpha and Beta hyperparameters can be determined by the trial and error, but this method leads an excessive cost in terms of time.

4.1.2 Gaussian Approach with Recursive Hyperparameters

The optimal values of objective function in (2.17) can be searched by updating formulations in (2.39) mentioned in recursive approach instead of the fixed hyperparameter approximation. In recursive approach, the iterative Levenberg-Marquardt Algorithm, which is able to search through the parameter space rapidly, was preferred. The performances of recursive approach for different neuron numbers are given in Table 4.2. According to Table 4.2, this approach shows good-fit to both test and training data for different neuron and iteration numbers. Although the best fit is obtained using 25 neurons, the algorithm shows good performance for 50 and 100 neurons as well. While neuron number is taken as greater than 25, MSE of test and training data slightly increases. For 25 neurons, the outputs of analysis are given in Figure 4.11 – Figure 4.14. As seen in Figure 4.12, the algorithm reaches to the best fit at nearly 150-th iteration.

As result, the advantage of this approach is that hyperparameter Alpha and Beta are determined automatically, and then over-fitting problem can be solved inherently. Besides, Levenberg-Marquardt algorithm substantially decreases training time as seen in Table 4.2. The best training times of algorithm for 50 and 100 neurons are showed with “*” in the last second rows of Table 4.2

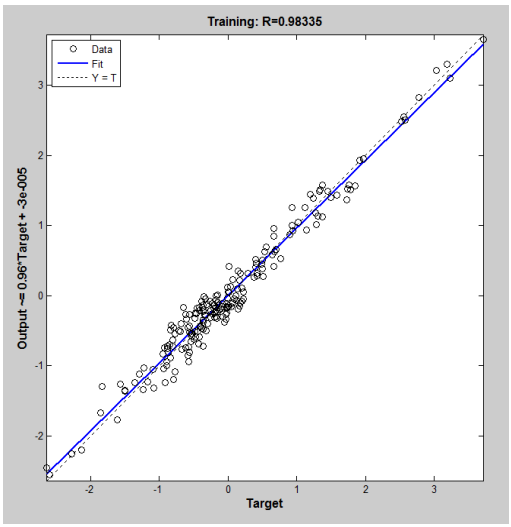


Figure 4.11. Correlation between Target and Output

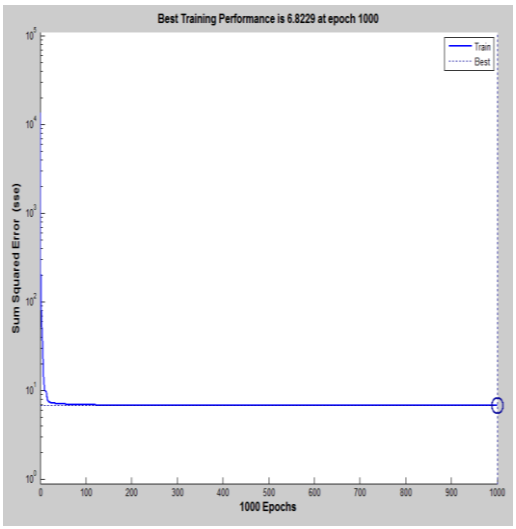


Figure 4.12. Recursive hyperparameter approach

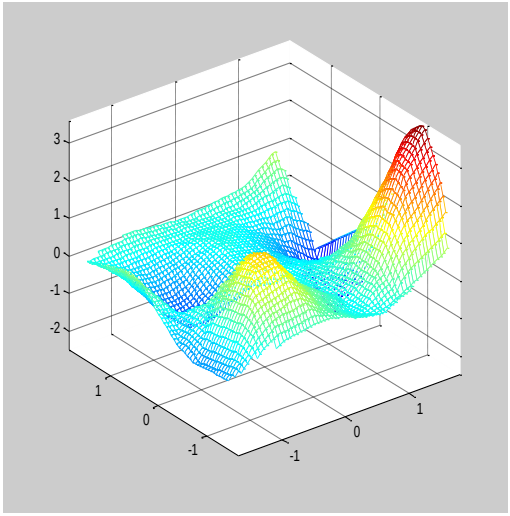


Figure 4.13. The performance of training data

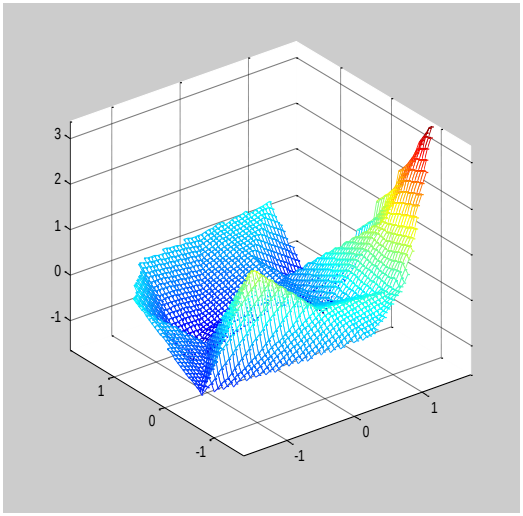


Figure 4.14. The performance of test data

Table 4.2 The performance of recursive hyperparameter approach

Iteration	Neuron number	Training MSE	Test MSE	Training correlation	Test correlation	Training (Seconds)
100	25	0.0365	0.0426	0.9822	0.9857	2
200	25	0.0342	0.0462	0.9833	0.9839	4
1000	25	0.0341	0.0422	0.9834	0.9855	20
1000/500*	50	0.0335	0.0425	0.9837	0.9854	29/14*
1000/400*	100	0.0339	0.0488	0.9835	0.9832	138/55*

4.1.3 The steepest descent with mean squared errors

To check the performance of traditional approaches with MSE, the parameters of neural network was estimated by the steepest descent. The performance of the steepest descent was observed for different learning rate, neurons and iterations. According to Table 4.3, the steepest descent algorithm reaches to the best fit by 25 neurons and 0.005 learning rate for 1000 iterations. At the end of the trials, it can be concluded that if the neuron number is increased, the algorithm needs to more iterations for good-fit, and the both MSE of training and test data substantially increase. For instance, the analysis outputs of algorithm for 2000 iterations, 100 neurons and 0.001 learning rate are given in Table 4.3, Figure 4.15 and Figure 4.18. According to results, the algorithm doesn't work well for 2000 iterations and 100 neurons. When the algorithm runs for 5000 iterations, both MSE of training and test data substantially decrease. However, if iteration number is taken as 10000, and then MSE of training data decreases, but MSE of test data substantially increase due to over-fitting. In such situations, the algorithm should be stop when MSE's of validation or test data start to arise. However, the early stopping approach causes to serious estimation errors when neural networks are trained by data with high dimensions and excessive noise.

As result, the steepest descent algorithms with MSE have problems such as the long training time, early stopping and over-fitting depending on parameter number of neural network. Besides, the steepest descent algorithms suffer from stuck in local optimums in high dimensions problems. In order to escape local optimums, the steepest descent algorithm with momentum can be used instead of traditional the steepest descent.

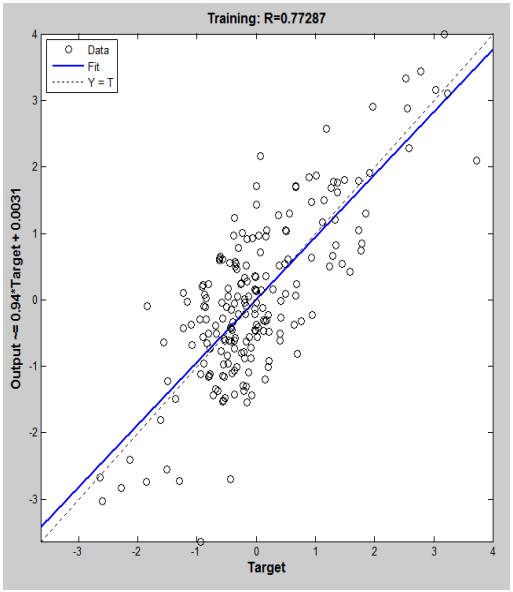


Figure 4.15. Correlation between Targets and Outputs

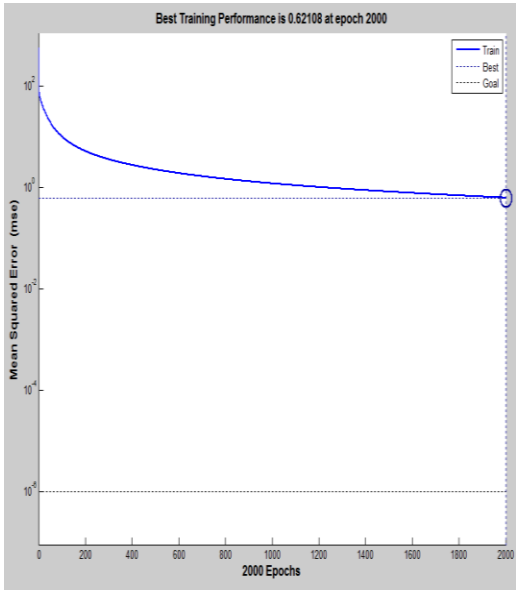


Figure 4.16. The performance of Steepest Descent

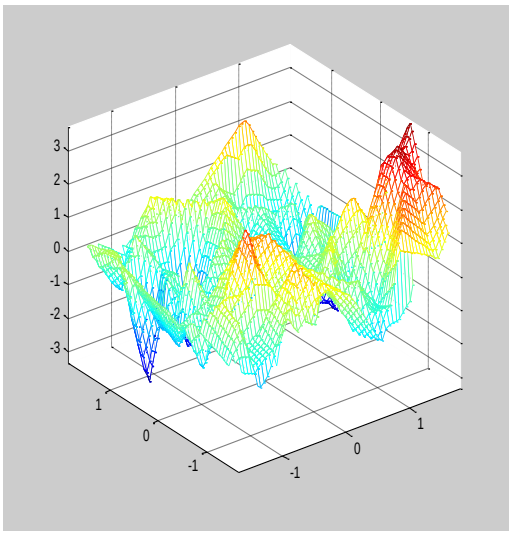


Figure 4.17. The performance for training data

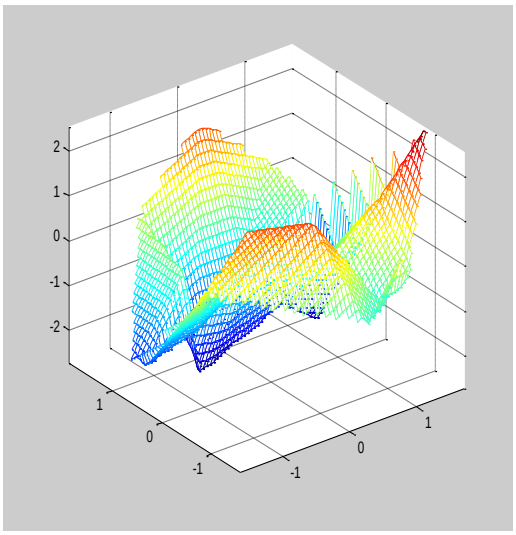


Figure 4.18. The performance for test data

Table 4.3 The performance of the steepest descent with MSE

Iteration number	Neuron number	Learning rate	Training MSE	Test MSE	Training correlation	Test correlation	Training (Seconds)
500	25	0.001	0.871	0.7232	0.6228	0.7599	5
	25	0.005	0.248	0.2594	0.8785	0.9161	5
	25	0.01	0.188	0.2374	0.9044	0.9230	5
1000	25	0.001	0.466	0.6528	0.8140	0.7777	10
	25	0.005	0.159	0.1453	0.9202	0.9480	10
	25	0.01	0.114	0.2150	0.9431	0.9211	10
1000	50	0.001	0.8367	1.4477	0.6962	0.6283	11
1000	50	0.005	0.2052	0.2926	0.8986	0.8925	11
2000*	100	0.001	0.6211	1.2882	0.7729	0.6404	25
5000*	100	0.001	0.2527	0.4850	0.8852	0.8517	64
10000*	100	0.001	0.0973	0.5273	0.9535	0.8685	125

4.1.4 The steepest descent with momentum

In here, the steepest descent algorithm was run for different neuron number, the learning rate and the momentum constant. The performance of algorithm is given in Table 4.4. According to Table 4.4, this algorithm reaches to the best fit using 0.005 learning rate and 0.5 momentum constant. While the neuron number is taken as 50, the best fit is obtained at 5000 iterations. By using same parameters, the algorithm gives the following results in Figure 4.19 – Figure 4.21 for 10000 iterations. From Table and Figures, it can be seen that the MSE of training data substantially decreases, and MSE of the test data slightly increases. However, the algorithm shows overfitting to both training and test data as seen in Figure 4.19 and Figure 4.20. According to Figure 4.21, the algorithm should be stopped before 9000-th iteration, otherwise, the overfitting problem becomes inevitable. Besides, while neuron number is taken as 100, the effectiveness of algorithm disappears, and MSE's of training and test data increase.

Table 4.4 The performance of the steepest descent with momentum

Iteration number	Neuron number	Learning rate	Momentum constant	Training MSE	Test MSE	Training correlation	Test correlation	Training (Seconds)
1000	25	0.001	0.1	0.6065	1.0486	0.7008	0.5820	11
	25	0.001	0.5	0.8359	1.0619	0.5725	0.5944	11
1000	25	0.005	0.1	0.2492	0.3640	0.8729	0.8712	11
	25	0.005	0.5	0.3241	0.3348	0.8300	0.8740	11
1000	25	0.01	0.5	0.2044	0.2424	0.8966	0.9207	11
	25	0.01	0.1	0.1686	0.1795	0.9151	0.9379	11
1000	50	0.001	0.1	1.3905	3.3276	0.6116	0.4420	12
5000	50	0.001	0.1	0.2307	0.2756	0.8890	0.9091	59
5000	50	0.005	0.5	0.1070	0.1047	0.9501	0.9658	65
10000	50	0.005	0.5	0.0761	0.1090	0.9653	0.9615	111
10000	100	0.005	0.5	0.1446	0.2708	0.9339	0.9265	126

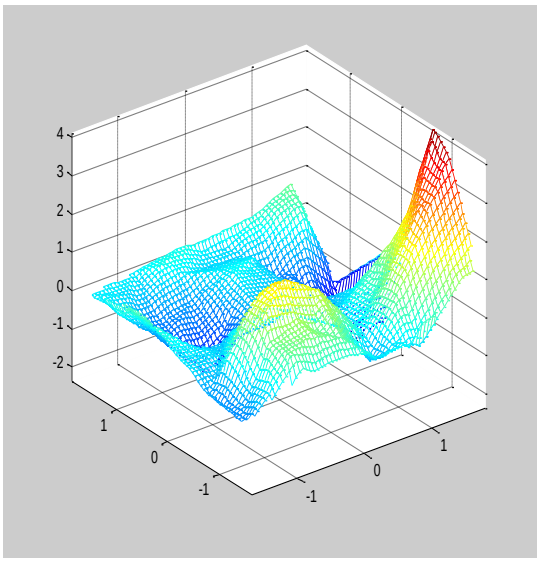


Figure 4.19. The performance for training data

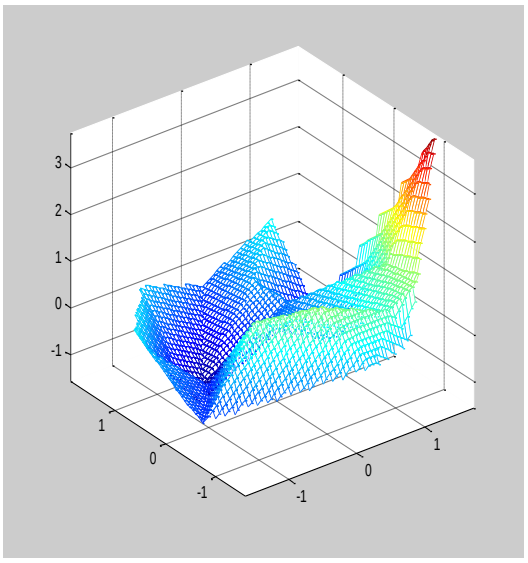


Figure 4.20. The performance for test data

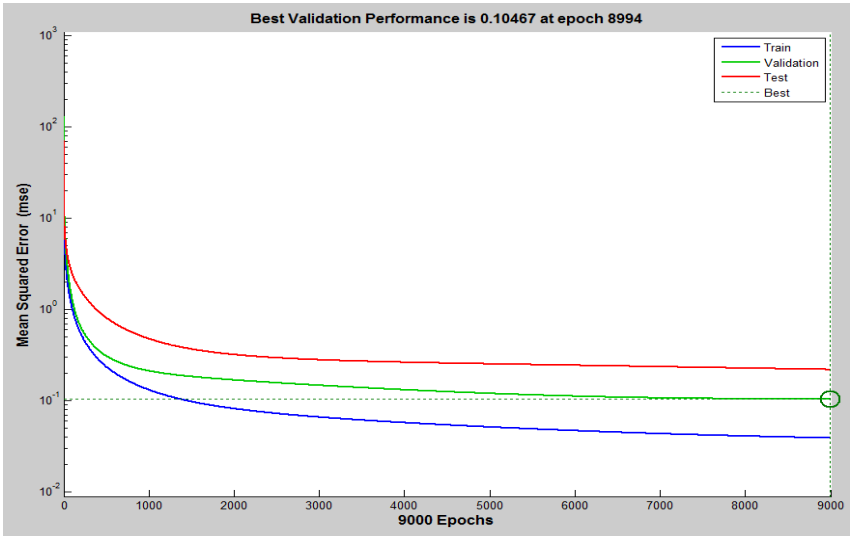


Figure 4.21. The performance of algorithm to iterations

As result, the steepest descent algorithm with momentum reduces to training time by using the combinations of learning rate and momentum constant, and overcomes to problem of stuck in local optimums. However, the early stopping approach should be used, since MSE causes overfitting problem. In addition, determining the best combination of the learning rate and momentum constant requires additional trials.

4.1.5 Genetic Monte Carlo Approach with Recursive hyperparameters

In order to be alternative against the traditional methods, the genetic MC with recursive hyperparameters, which is improved for Gaussian approach of Bayesian learning, can be used. In this approach, the GA cycle turns its inside till the generation number in any iteration of main algorithm. In addition, if it is desired, the simulations can be done by means of the fixed hyperparameters as well. In order to observe the performance of algorithm for the fixed hyperparameter approach, the simulations were done for Alpha 0.01 and Beta 0.99. For GA cycle; elitism, crossover, mutation, generation and population parameters were taken as 2, 0.8, 0.01, 100 and 100 respectively. To train the neural network with 25 neurons at the only 100 iterations, the Genetic MC worked for 1198 seconds, and the outputs of trial for 100 iterations (iteration x generation = 100x100) are given in Figure 4.22 - Figure 4.25 and Table 4.5.

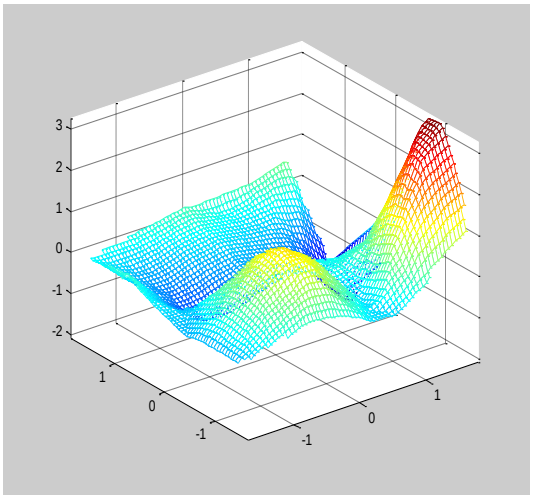


Figure 4.22. The performance for training data

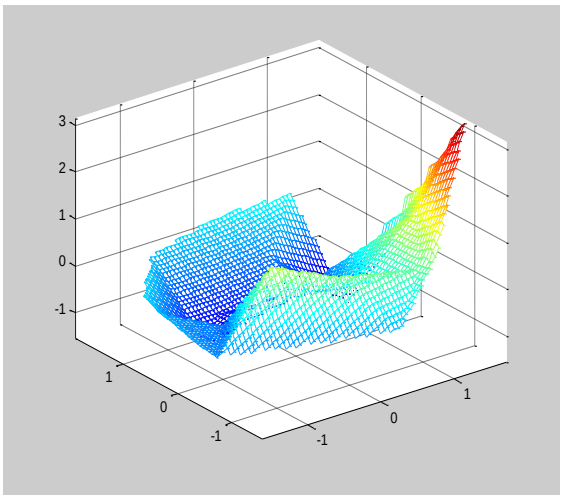


Figure 4.23. The performance for test data

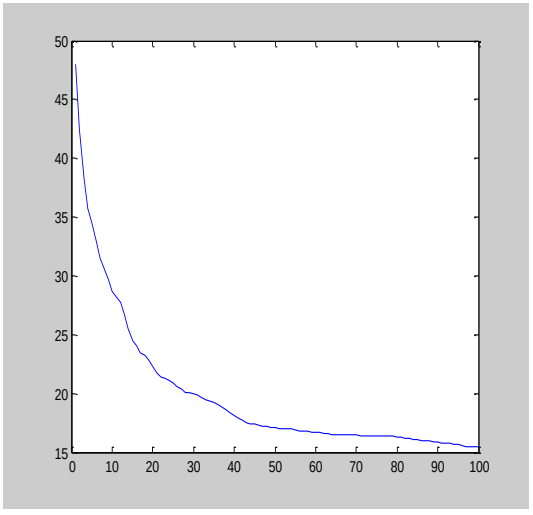


Figure 4.24. Error function versus iterations

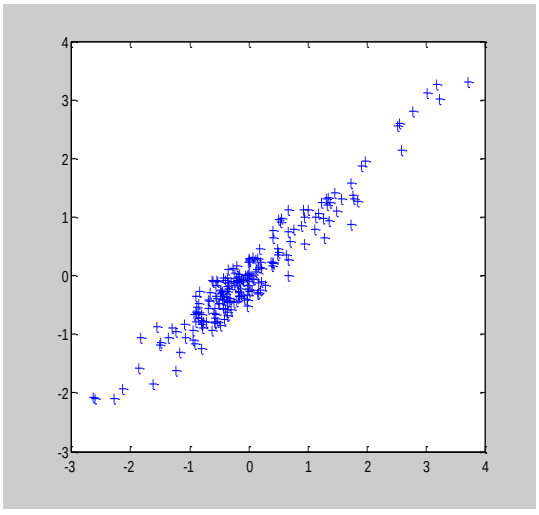


Figure 4.25. Corr. between targets and outputs

At the end of simulations, MSE's of the training and test data were obtained as 0.07 and 0.08, and the correlations between outputs and targets as 0.97 and 0.98 respectively. Besides, the performances of algorithm for different parameter values are given in Table 4.5. Alternatively, the generation of GA was taken as 1, thus the performance of algorithm was observed for only one generation. The result for only one generation is showed in the row with "*" in Table 4.5. According to results, this trial needs much more iterations, since generation number is taken as 1. However, this case doesn't cause an excessive cost in terms of training time because of being provided the required population size for the diversification in any generation.

Table 4.5 The performance of Genetic MC with the fixed hyperparameters

Iterations	Neuron number	Generation	Training MSE	Test MSE	Training correlation	Test correlation	Training (Seconds)
10	25	100	0.14	0.20	0.93	0.94	120
	25	200	0.07	0.12	0.96	0.96	218
	50	100	0.17	0.13	0.92	0.92	243
	50	200	0.14	0.17	0.93	0.94	441
100	25	100	0.07	0.08	0.97	0.98	1198
100	50	100	0.07	0.14	0.97	0.96	2674
1000*	25	1*	0.09	0.13	0.96	0.96	399

According to Table 4.5, while the neuron number is increased, MSE's of training and test data increase as well. However, if the generation and iteration numbers are increased together, MSE's can be reduced to a certain extent. As a result, the problems caused by the high parameter space can be solved by increasing generation number.

As discussed above, the hyperparameters can be automatically determined by means of the recursive hyperparameter approach instead of the fixed one. In order to use this approach; elite, crossover, mutation parameters of GA were determined as 2, 0.8, 0.01 respectively, and then the algorithm was run by different generations, population, neuron and iteration numbers. The performance of algorithm is given in Table 4.6. Besides, the neural networks with 25 neurons were trained by Genetic MC with 100 generation and 150 population numbers.

The performances of Genetic MC at the end of the 10 (iteration x generation = 10x100) and 100 iterations (100x100) are given in Figure 4.26 – Figure 4.31. From Figure 4.26 - Figure 4.27, it can be seen that while the iterations continues, Beta/Alpha ratio increases gradually. However, according to Figure 4.27, this ratio becomes more stable after 40-th iteration. From this trial, it can be concluded that the fitting to data is more dominant than weight decay for high Beta/Alpha ratio. According to Figure 4.28 - Figure 4.29, while the simulations continue, the posterior distribution becomes more stable. Therefore, the fitting obtained at end of 100 iterations is better than one obtained at the end of 10 iterations.

From Table 4.6, it can be seen that although MSE's increase for neuron number 50 and 100, the algorithm ensures to produce the consistent solutions. Besides, MSE's can be reduced to a certain extent by increasing both neuron and generation numbers together. Although the high generation and population numbers allow reducing MSE, the required time for simulations will increase. Therefore, the generation and population number should be kept at the reasonable level for good fitting.

As a result, Genetic MC approach ensures the consistent solutions at end of the short iterations, since hyperparameters are balanced automatically. Besides, it is possible to obtain more consistent results by increasing iteration number as well.

Table 4.6 The performance of Genetic MC with the recursive hyperparameters

Iterations	Neuron number	Generation	Population	Training MSE	Test MSE	Training correlation	Test correlation	Training (Seconds)
10	25	100	100	0.22	0.29	0.90	0.91	150
	25	150	200	0.17	0.25	0.92	0.93	440
	50	150	200	0.13	0.18	0.94	0.95	841
	100	150	200	0.16	0.23	0.92	0.92	1666
100	25	150	200	0.06	0.08	0.97	0.98	2650
100	50	150	200	0.14	0.21	0.93	0.93	8003

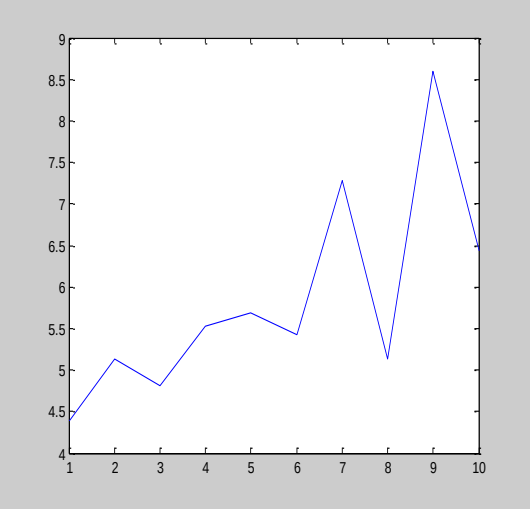


Figure 4.26. Beta/Alpha ratio for 10 iterations

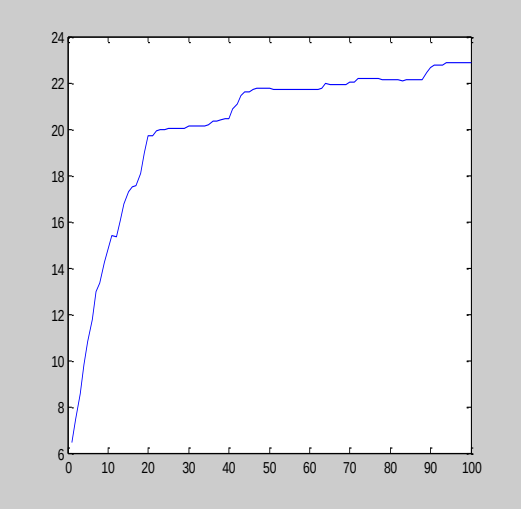


Figure 4.27. Beta/Alpha ratio for 100 iterations

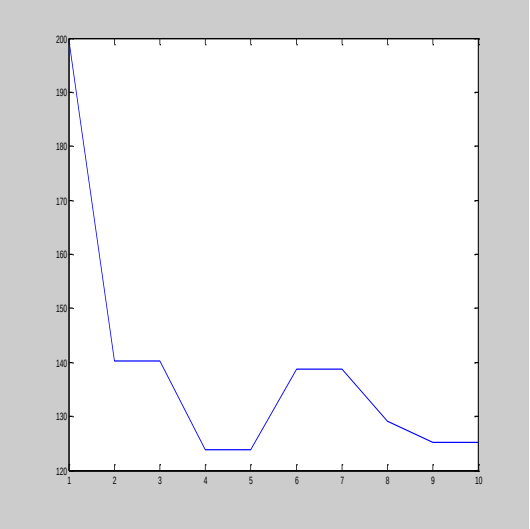


Figure 4.28. Errors for 10 iterations

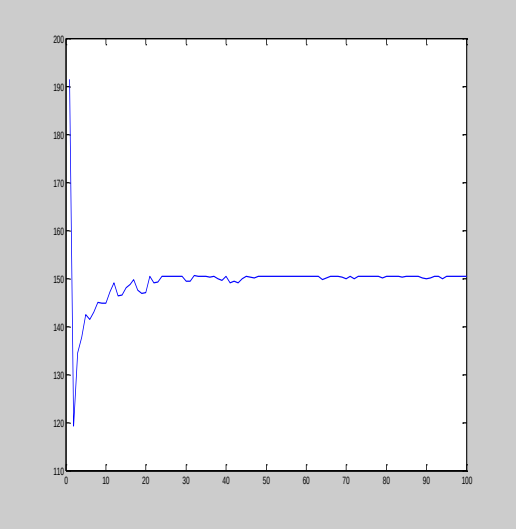


Figure 4.29. Errors for 100 iterations

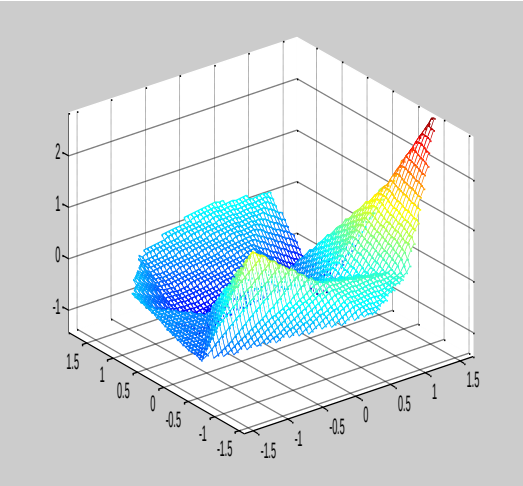


Figure 4.30 The test performance for 10 iterations

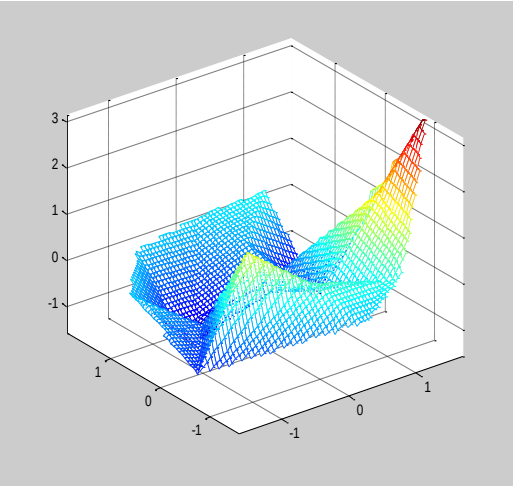


Figure 4.31. The test performance for 100 iterations

4.1.6 Hierarchal full Bayesian approach with hybrid Monte Carlo

In order to apply the hierarchal full Bayesian approach to data set used above, firstly the high-level hyperparameters have to be fixed. Therefore, the shape parameter Alpha was taken as 1, 2 and 10; mean of the weights, biases and noise precisions were taken as 1, 10 and 100 respectively. For instance, if the parameter Alpha and the precision mean are taken as 1 and 100 respectively, and then the scale parameter corresponds to 0.01 (mean=Alpha/Beta). For Alpha 1 and Beta 0.01, the priors of parameters become little bit informative. In Bayesian analysis, the non-informative priors are often preferred such as Gamma (0.1, 1) and Gamma (0.01, 1) as well. However, the simulations done by different priors showed that little bit informative priors produce more consistent results for network structure constituted in here. In simulations, the middle-level hyperparameters in (3.11) were fixed as 1, thus, the mean and Beta in the middle level of hierarchy was generated by simulations.

The elite, crossover, mutation, generation and population size parameters of GA were taken as 2, 0.8, 0.01, 50 and 100 respectively. After the neural networks were trained by only Hierarchal full Bayesian approach without GA updates for 1000 or 10000 iterations in the burn-in process, the neural network parameters were estimated by Hybrid Monte Carlo with GA until a sufficient iteration number. The performance of the hybrid MC with GA was tested by different hyperparameter, iteration and neuron number as well. The simulation results are given in Table 4.7 for different parameters. In addition, changing of parameter precisions for different Alpha and Beta combinations, the stationarity of posterior distribution and noise level to iterations, the performances of fitting to training and test data are demonstrated in Figure 4.32–Figure 4.47.

According to Table 4.7, it can be said that the algorithm produces consistent results for different burn-in and iteration numbers. For instance, after hierarchal full Bayesian approach without GA were run for 1000 iterations in burn-in process, hybrid MC with GA was run for 10 iterations (totally 10x150 = 1500 iterations together with GA generations), and the good fitting to both training and test data was obtained at the end of training process with 10 iterations. From Table 4.7, it can be seen that, while iterations continue, MSE's of the training and test data decrease distinctly. If the iteration number is kept at an adequate level, the hybrid MC maintains to produce the consistent results, even if the neuron number is increased.

After the hybrid MC was run for 10000 iterations without GA in burn-in process, and 1000 iterations with GA in the training process, the negative logarithm of posterior and noise level

were obtained as demonstrated in Figure 4.32 and Figure 4.33. From, these Figures, it can be seen that after burn-in process both the negative logarithm of posterior distribution and the noise level become increasingly stationary. Similarly, the impact of hybrid MC over the negative logarithm of posterior and noise level for 1000 iterations in burn-in process, and 10 iterations in training process are demonstrated in Figure 4.34 and Figure 4.35. According to these Figures, although Markov chain is not stationary without GA update in burn-in process, it returns to the stationary structure by means of hybrid MC with GA in a short time.

Table 4.7 The performance of hybrid Genetic MC

Burn-in	Iteration	Alpha – Mean	Neuron	Generation	Population	Training MSE	Test MSE	Training correlation	Test correlation	Training (seconds)
1000	**10**	2, 1	25	150	150	0.11	0.17	0.96	0.96	470
		2, 1	50	150	150	0.18	0.16	0.91	0.94	930
		1, 1	50	150	150	0.11	0.14	0.95	0.96	1012
1000	**100**	10, 10	25	150	150	0.06	0.06	0.97	0.98	4293
1000	**100**	10, 10	50	150	150	0.07	0.14	0.97	0.96	7821
10000	**1000**	2, 1	25	150	150	0.03	0.04	0.98	0.99	34209

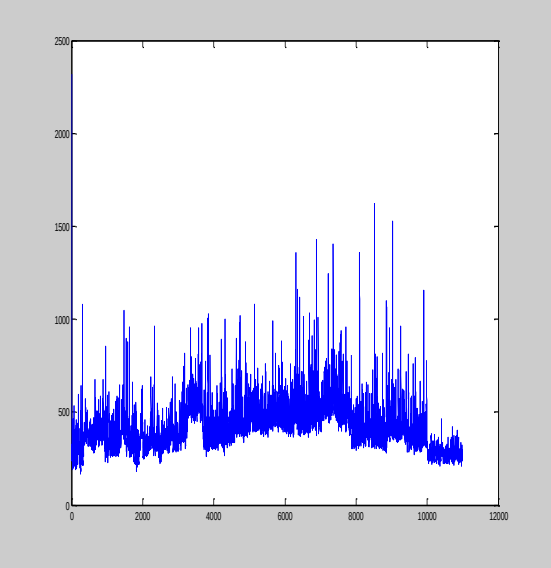


Figure 4.32. Negative log of posterior to iterations

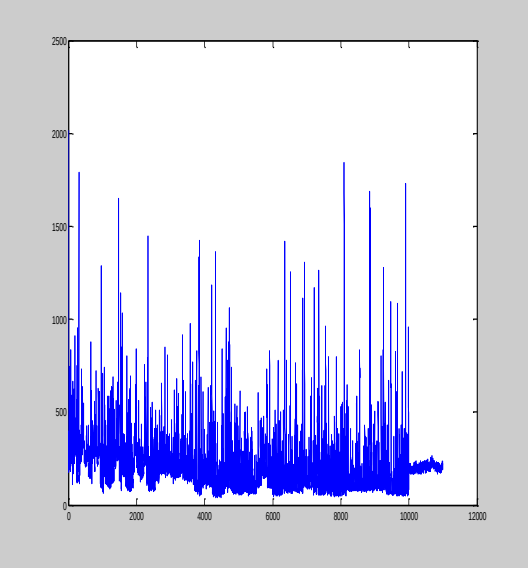


Figure 4.2. Noise level to iterations

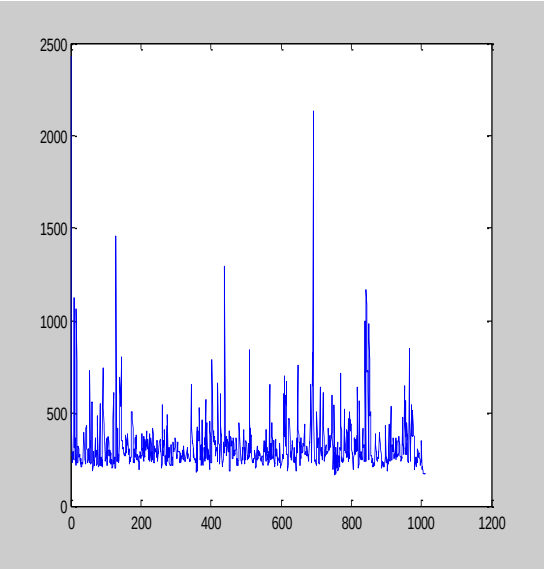


Figure 4.34. Negative log of posterior in burn-in stage

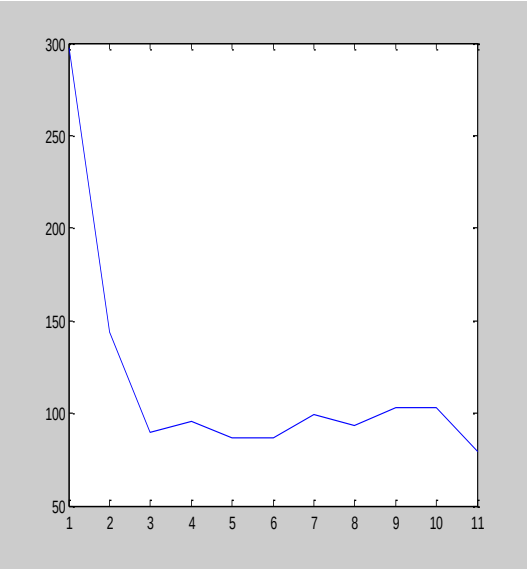
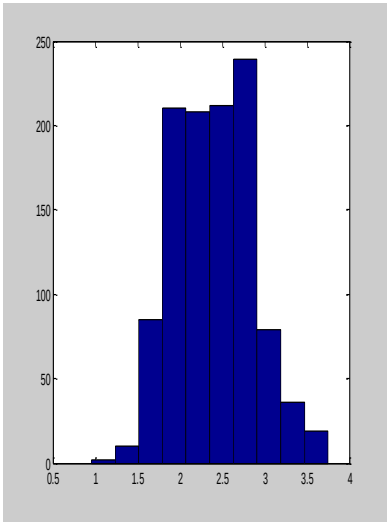
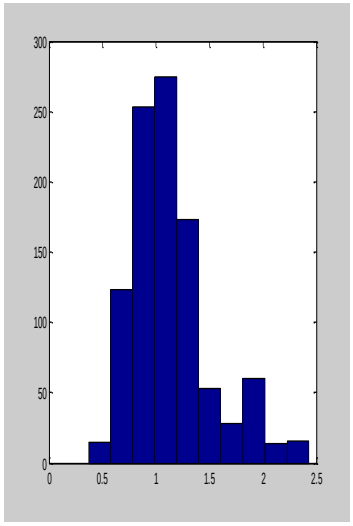


Figure 4.3. Negative log of posterior in learning stage

After simulations with 1000 iterations in burn-in process and 10 iterations in training process; the first, second and third lagged autocorrelations of Markov chain demonstrated in Figure 4.35 are evaluated as -0.17, -0.12 and -0.21 by formulation in (E.6) introduced in *Appendix E* respectively. According to these values, the autocorrelations between the successive iterations are negative, thus, independence ratio for the first, second and third lagged are evaluated as 0.66, 0.41 and -0.02 by formulation in (E.5) respectively. In the training stage, the independence ratios between successive elements of chain are the satisfactory level, since the autocorrelations are shorter than 1. However, in burn-in stage seen in Figure 4.34; the first, second and third lagged autocorrelations are evaluated as 1.80, 2.42 and 2.89, thus, the independence ratios are obtained as 1.80, 2.42 and 2.89. From here, it can be concluded that, in burn-in process, there are high positive autocorrelations between successive elements of chain without GA updates; so that; this situation causes the high independence ratios.

In order to produce precision values by simulations, Alpha value in the high level hyperparameter of the first weight group in the input layer was taken as 1, 2 and 10; means for weights, biases and noises were taken as 1 and 10 respectively. After simulations with 1100 iterations, the precisions of the first quantities for different Alpha and mean values were estimated from neural network with 25 neurons. The estimated precisions are given in Figure 4.36 – Figure 4.38. According to Figure 4.36, when the precision mean of priors is taken as 10, the mean of the first weight group is obtained approximately 2.25 at end of the simulations. As seen in Figure 4.37 and Figure 4.38, when means of precision priors is taken as 1, the mean of the first weight group is obtained approximately 1.





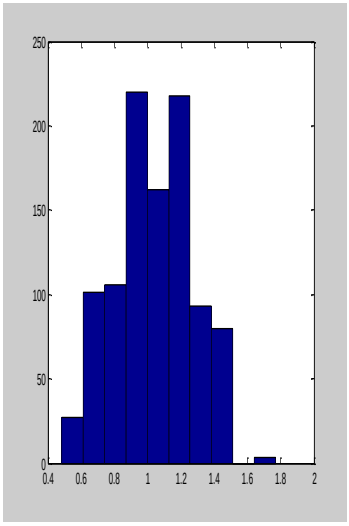
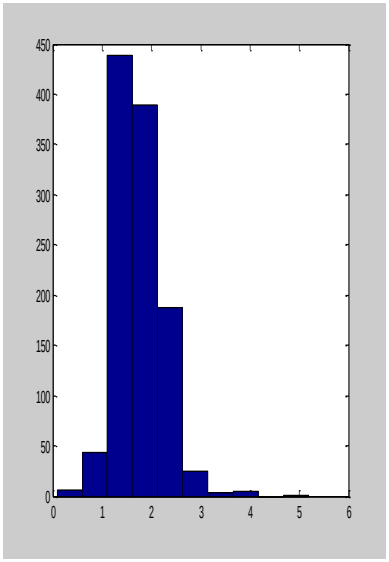
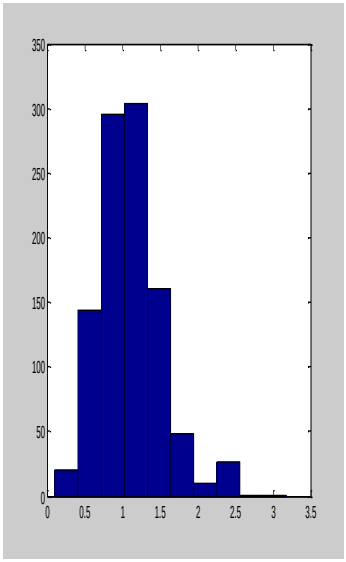


Figure 4.36. Alpha = 10, Mean = 10 **Figure 4.4. Alpha = 2, Mean = 1** **Figure 4.38. Alpha = 1, Mean = 1**

The hyperparameter values assigned for the first weigh group were taken for the second weight group, and then the simulation results were given in Figure 4.39 and Figure 4.41. According to Figure 4.39, when the precision mean of priors is taken as 10, the mean of the second weight group is obtained approximately 2. As seen in Figure 4.40 and Figure 4.41, when means of precision priors is taken as 1, the mean of the second weight group is obtained approximately 1.





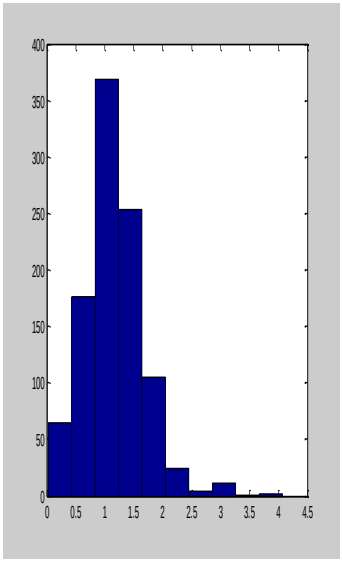


Figure 4.39. Alpha = 10, Mean = 10 **Figure 4.40. Alpha = 2, Mean = 1** **Figure 4.5. Alpha = 1, Mean = 1**

The estimated precisons of weight group between the hidden layer and output layer are given in Figure 4.42 and Figure 4.44. According to Figure 4.42, when the precision mean of priors is

taken as 10, the mean of the related weight group is obtained approximately 2. As seen in Figure 4.43 and Figure 4.46, when means of precision priors is taken as 1, the mean of the related weight group is obtained approximately 1.25.

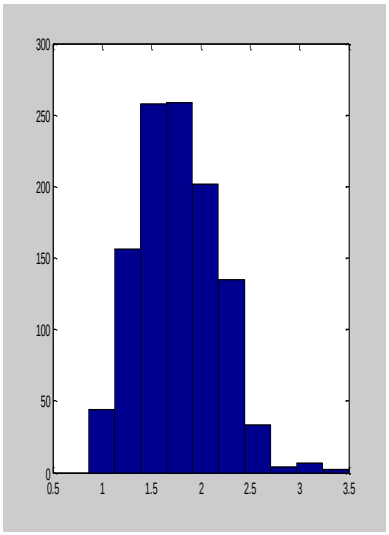


Figure 4.6. Alpha = 10, Mean = 10

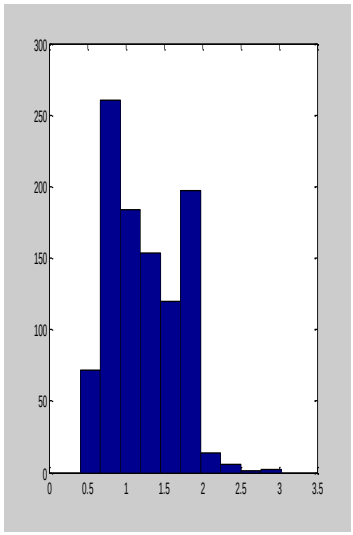


Figure 4.43. Alpha = 2, Mean = 1

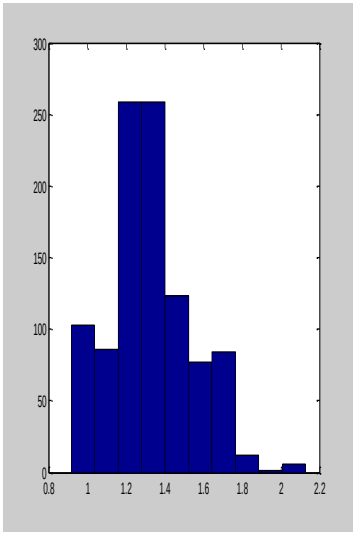


Figure 4.44. Alpha = 1, Mean =1

Lastly, the estimated precisons of bias group between the hidden layer and output layer are given in Figure 4.45 and Figure 4.47. According to the following Figures, the estimated precisions of bias group are shorter than ones of weights groups for same hyperparameters.

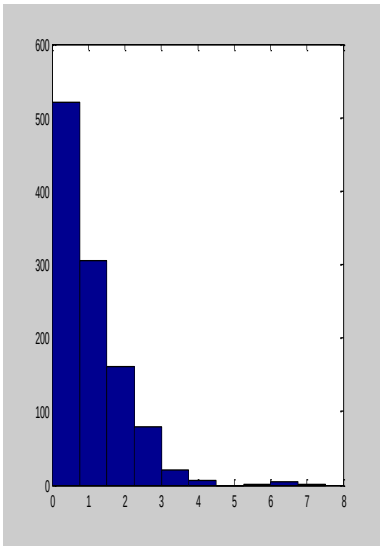


Figure 4.45. Alpha = 10, Mean = 10

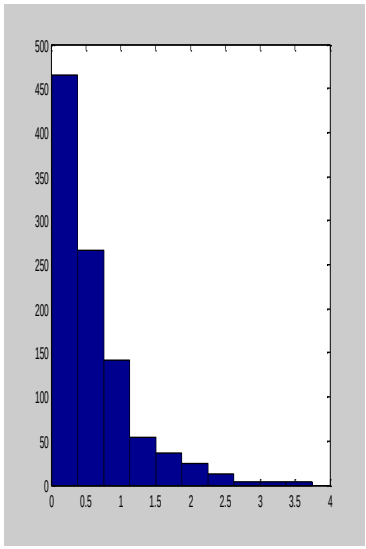


Figure 4.46. Alpha = 2, Mean = 1

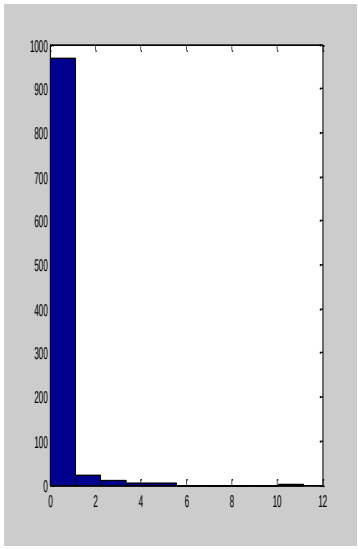


Figure 4.7. Alpha = 1, Mean = 1

4.2 Application II

In here, data set with two inputs and one target as demonstrated in Figure 4.48 is preferred. In order to apply the hierarchal full Bayesian approach to this data set, firstly Alpha and means in the high level introduced in (3.10) - (3.12) were fixed as 1 and 10 respectively. Similarly, the shape parameter at the mid-level was fixed as 1 as well. In hidden layer, the neuron number was is determined as 20, and the tangent hyperbolic function was preferred for the activation function. The elite, crossover, mutation, generation and population size parameters of GA were taken as 2, 0.8, 0.01, 100 and 100 respectively. After Hybrid MC was run without GA updates for 1000 iterations in the burn-in process, Hybrid MC with GA was run for 10 iterations (totally 10x100 = 1000 iterations together with GA generations). At end of simulations lasting 220 seconds, the algorithm produced the following results demonstrated in Figure 4.49 – Figure 4.54.

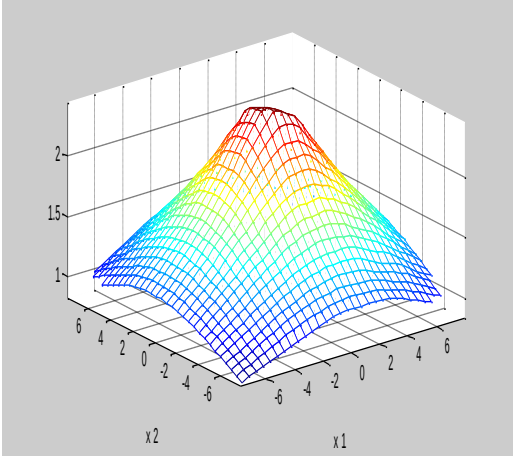


Figure 4.8. Targets

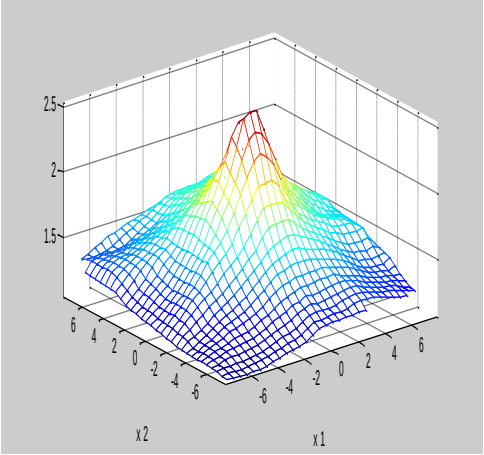


Figure 4.49. Outputs

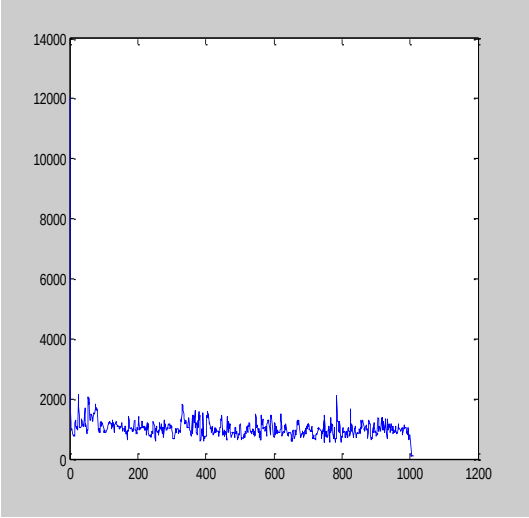


Figure 4.50. Negative log of posterior in burn-in stage

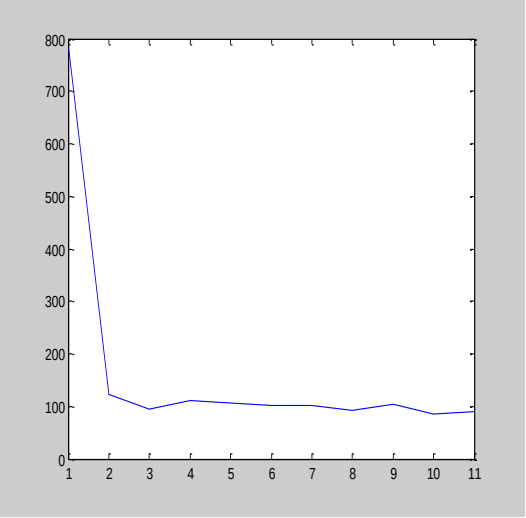


Figure 4.91. Negative log of posterior in learning stage

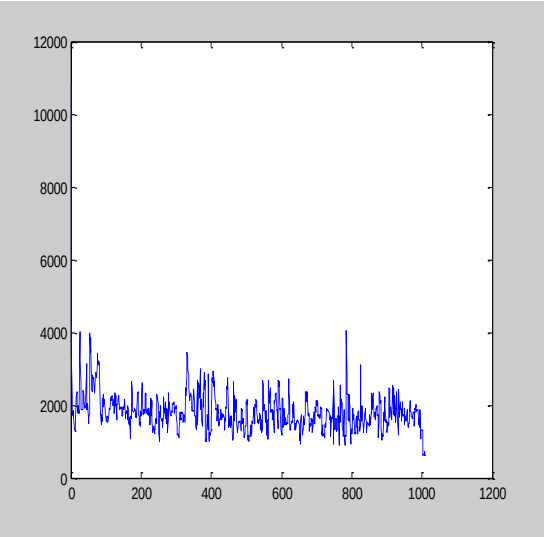


Figure 4.52. Noise level in burn-in stage

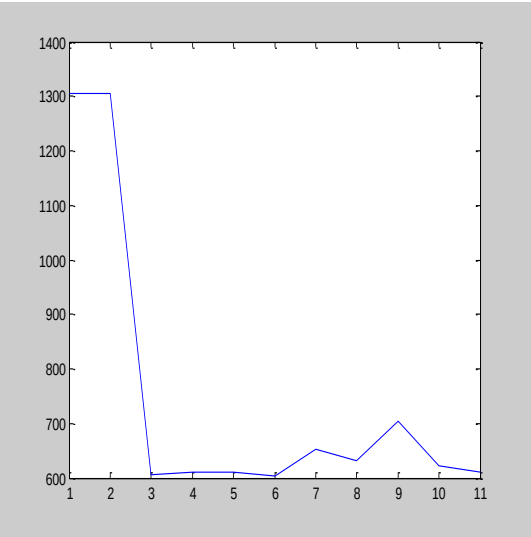


Figure 4.53. Noise level in learning stage

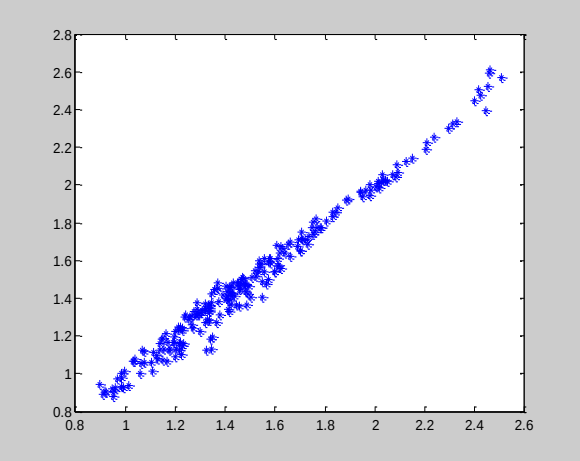


Figure 4.54. Correlation between targets and outputs

In Figure 4.50 – Figure 4.54, the impact of Hybrid MC over the negative logarithm of posterior and noise are demonstrated. From Figures above, it can be seen that after burn-in process both the negative logarithm of posterior and noise become increasingly stationary. Besides, the correlation between targets and outputs are 0.99. The first, second and third lagged autocorrelations of Markov chain in Figure 4.51 are evaluated as -0.27, 0.09 and -0.16 by formulation in (E.6) respectively. According to these values, the autocorrelations between successive iterations are negative, thus, the independence ratio for the first, second and third lagged are evaluated as 0.44, 0.63 and 0.29 by formulation in (E.5) respectively. The independence ratio between successive elements of chain are satisfactory level since the autocorrelations are shorter than 1. However, the independence ratios for the first, second and third lagged in burn–in process are obtained as 1.52, 1.87 and 2.12. From here, it can be

concluded that there is high positive autocorrelations between successive elements of chain without GA updates; therefore, this situation causes the high independence ratios.

Consequently, the Hybrid MC with GA ensures to consistent results for the short iterations. In addition, if iteration number is increased, then obtaining better results would be possible as well.

5 CONCLUSION and SUGGESTIONS

As discussed above, there are some complicated problems encountered in training process of a traditional neural network with MSE. The one of these problems is dividing into data set as training, validation and test. In training process, the performance of the neural network should be checked over training, validation and test data simultaneously. However, the dividing process is not necessary in Bayesian neural networks, since they have ability to learn over whole of data. The other problems caused by MSE are stuck in local minimums due to using gradient searches, the estimation and approximation errors. Hybrid Bayesian neural networks solve not only the problems discussed above, but also they provide an opportunity for a robust statistical inference. However, while number of the estimated parameters and hyperparameters increases, more fast and effectiveness algorithms are necessary for the advanced Bayesian simulations.

In this study, the novel Hybrid MC methods with GA are improved for the normal approach and full Bayesian approach used in the Bayesian neural networks. To measure un-probabilistic (fuzzy) uncertainty, the fuzzy membership functions are used in both the Gaussian and full Bayesian approaches. Thus, the algorithms that accurately estimate the parameters and hyperparameters of neural networks are improved. However, the problems such as determination of neuron and the layer numbers, the selection of the activation function are not included into this study.

From implementations, it can be concluded that the proposed approaches ensure the consistent results for problems, and allow more effective and fast simulations that overcome random walk caused by classic MCMC.

REFERENCES

Barber, D. and Bishop, C. M., 1998. Ensemble Learning in Bayesian Neural Networks. in Bishop, C. M., Editor, Neural Networks and Machine Learning, Volume 168 of NATO ASI Series: Computer and Systems Sciences, Springer-Verlag, 215-237.

Berger, J. O., 1985. Statistical Decision Theory and Bayesian Analysis. Springer Series in Statistics. Springer, 2nd edition.

Bernardo, J. M. and Smith, A. F. M., 1994. Bayesian Theory, John Wiley & Sons.

Bishop, C. 1995a. Training with Noise is Equivalent to Tikhonov Regularization, *Neural Computation*, **Vol. 7(1)**, 108-116.

Bishop, C. M., 1995b. Neural Networks for Pattern Recognition, Oxford University Press.

Brass, A., Pendleton, B. J., Chen, Y. and Robson, B., 1993. Hybrid Monte Carlo Simulations Theory and Initial Comparison with Molecular Dynamics, *Biopolymers*, **Vol. 33(8)**, 1307-1315.

Buntine, W. L. and Weigend, A. S., 1991. Bayesian Back-Propagation. Complex systems, **Vol. 5(6)**, 603–643.

Castellano, G., Fanelli, A.M. and Pelillo, M., 1997. An Iterative Pruning Algorithm for Feedforward Neural Networks, *Neural Networks, IEEE Transactions on*, **Vol. 8**, 519- 531.

Chua C. G. and Goh A. T. C., 2003. Nonlinear Modeling with Confidence Estimation using Bayesian Neural Networks, **International Journal for Numerical and Analytical Methods in Geomechanics, int. J. Numer. Analy. Meth. Geomech., Vol. 27**, 651–667.

de Freitas, J. F. G., 1997. Neural Network Based Nonparametric Regression for Nonlinear System Identification and Fault Detection. Master's Thesis, University of The Witwatersrand, Johannesburg.

de Freitas, J. F. G., 2000. Bayesian Methods for Neural Networks, *Phd. Thesis*, Trinity College, University of Cambridge and Cambridge University Engineering Department.

Duane, S., Kennedy, A. D., Pendleton, B. J. and Roweth, D., 1987. Hybrid Monte Carlo. Physics Letters B, **Vol. 195(2)**, 216-222.

Fahlman, S. E. and Lebiere, C., 1988. The Cascade-Correlation Learning Architecture, intouretzky, D. S., Editor, *Proceedings of The Connectionist Models Summer School*, San Mateo, CA, **Vol. 2**, 524-532.

Frean, M., 1990. The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks, *Neural Computation,* **Vol. 2(2)**, 198-209.

Gelfand, A. E., 1996. Model Determination using Sampling-Based Methods. in Gilks, W. R.,Richardson, S., and Spiegelhalter, D. J., Editors, Markov Chain Monte Carlo in Practice. Chapman &Hall, 145–162.

Gelman, A., Carlin, J. B., Stern, H. S. and Rubin, D. R., 1995. Bayesian Data Analysis. Texts in Statistical Science, Chapman &Hall.

Gilks, W. R., Richardson, S. and Spiegelhalter, D. J., 1996. Markov Chain Monte Carlo in Practice. Chapman &Hall.

Geman, S. and Geman, D., 1984. Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images, *IEEE Transactions on Pattern Analysis and Machineintelligence,* **Vol. 6(6)**, 721-741.

Geman, S., Bienenstock, E. and Doursat, R., 1992**.** Neural Networks and The Bias/Variance Dilemma, Massachusetts institute of Technology, Vol. 4, No. 1, 1-58.

Gill, J., 2008. Beyasian Methods: A Social and Behavioral Sciences Approach, Second Edition, Chapman &Hall/CRC, 145-146.

Girosi, F., Jones, M. and Poggio, T., 1995. Regularization Theory and Neural Networks Architectures. Neural Computation, **Vol. 7(2),** 219-269.

Goel, P. K. and Degroot, M. H., 1981. Information About Hyperparameters in Hierarchical Models, *Journal of the American Statistical Association*, **7 Vol.** 6(373), 40–14.

Green, P. J., 1995. Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination, *Biometrika*, **Vol. 82**, 711-732.

Goldberg, D. E., 1989. Genetic Algorithms in Search: Optimization and Machine Learning, Addison-Wesley, USA, 1 - 7.

Hastings, W. K., 1970. "Monte Carlo Sampling Methods using Markov Chain and Their Applications", *Biometrica,* **Vol. 57(1)**, 97-109.

Haykin, S., 1994. Neural Networks: A Comprehensive Foundation. Macmillan College Publishing Company.

Hinton, G., 1987. Learning Translation invariant Recognition in Massively Parallel Networks, Lecture Notes in Computer Science, Volume 258, 1987, DOI:10.1007/3-540-17943-7.

Holland, J. H., 1975, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor.

Holmes, C. C. and Mallick, B. K., 1998. Bayesian Radial Basis Functions of Variable Dimension, *Neural Computation*, **Vol. 10(5)**, 1217-1233.

Huang, G. B., Saratchandran, P. and Sundararajan, N., 2005. A Generalized Growing and Pruning RBF (GGAP-RBF) Neural Network for Function Approximation, *Neural Networks, IEEE Transactions on*, **Vol.16**, 57-67.

Jacobs, R. A., 1995. Methods for Combining Experts Probability Assessments, *Neural Computation*, **Vol. 7(5)**, 867-888.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K., 1998. An introduction to Variational Methods for Graphical Models, Kluwer Academic Publishers.

Kirkpatrick, S., C. D. Gelatt, and Vecchi M. P., 1983. Optimization by Simulating Annealing, *Science*, **Vol. 220 (4598)**, 671-680.

Kocadagli, O., 2011 Bayesian Learning of Neural Networks using Genetic Algorithms, Young Statatisticians Meeting, Trinity College, August 19-21, Dublin, Ireland.

Kocadağlı, O. and Cinemre, N., 2011. A Fuzzy Nonlinear Model Approach with CAPM for Portfolio Optimization, *Istanbul University Journal of Business*.

Lampinen, J. and Vehtari, A., 2001. Bayesian Approach for Neural Networks – Review and Case Studies, *Neural Networks,* **Vol. 14(3)**, 7-24.

Le Cun, Y., Denker, J. S. and Solla, S. A., 1990. Optimal Brain Damage, intouretzky, D. S., Editor, *Advances in Neural information Processing Systems*, San Mateo, CA. **Vol. 2**, 598 - 605.

Lemm, J. C., 1996. Prior information and Generalized Question,. Technical Report AIM 1598, CBCLP 141, Massachusetts institute of Technology, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, Department of Brain and Cognitive Sciences.

Liang, F., 2005. Bayesian neural networks for nonlinear time series forecasting. Stat. Comput, **Vol. 15 (1)**, 13–29.

Lindley, D., 2000. Journal of The Royal Statistical Society, The Philosophy of Statistics, The Statistician, 293-337.

Lord, D., Xie, Y. and Zhang Y., 2007**.** Predicting Motor Vehicle Collisions using Bayesian Neural Network Models: An Empirical Analysis, *Elsevier, Accident Analysis and Prevention*, **Vol. 39**, 922–933.

Mackay, D. J. C., 1992. A Practical Bayesian Framework for Backpropagation Networks, *Neural Computation*, **Vol. 4(3)**, 448–472.

Mackay, D. J. C., 1994. *Hyperparameters: Optimize or integrate Out? in G. Heidbreder (Ed), Maximum Entropy and Bayesian Methods*, Santa Barbara 1993, Dordrecht: Kluwer.

Mackay, D. J. C., 1995. Probable Networks and Plausible Predictions - A Review of Practical Bayesian Methods for Supervised Neural Networks, Network: *Computation in Neural Systems*, **Vol. 6(3)**, 469–505.

Marrs, A. D., 1998. An Application of Reversible-Jump MCMC to Multivariate Spherical Gaussian Mixtures, in Jordan, M. I., Kearns, M. J., and Solla, S. A., Editors, Advances in Neural Information Processing Systems, **Vol. 10**: 577-583.

Marwala, T., 2007, Bayesian Training of Neural Networks using Genetic Programming, *Pattern Recognition Letters 28*, 1452–1458.

Metropolis, N., A., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller, E. 1953, "Equations of State Calculations by Fast Computing Machines", *Journal of Chemical Physics* **Vol. 21 (6)**, 1087–1092.

Michalewicz, Z., 1994, Genetic Algorithms + Data Structures = Evolution Programs, 2nd Edition, Springer-Verlag, Berlin, 340.

Neal, R. M., 1992. Bayesian Training of Back-Propagation Networks By The Hybrid Monte Carlo Method. Technical Report CRG-TR-92-1, Dept. of Computer Science, University of Toronto.

Neal, R. M., (1996. Bayesian Learning for Neural Networks, Springer.

Niyogi, P. and Girosi, F., 1994. On the Relationship between Generalization Error, Hypothesis Complexity, and Sample Complexity for Radial Basis Functions, Technical

Report AIM-1467, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, MA.

Perrone, M. P., 1995. Averaging/Modular Techniques for Neural Networks, in Arbib, M. A., Editor, The Handbook of Brain Theory and Neural Networks, MIT Press, ss. 126-129.

Platt, J., 1991. A Resource Allocating Network for Function interpolation, *Neural Computation*, **Vol 3**: 213-225.

Richardson, S. and Green, P. J., 1997. On Bayesian Analysis of Mixtures with an Unknown Number of Components, *Journal of the Royal Statistical Society B*, **Vol. 59(4)**, 731- 792.

Rios insua, D. and Muller, P., 1998. Feedforward Neural Networks for Nonparametric Regression, Technical Report 98.02, Institute of Statistics and Decision Sciences, Duke University.

Robert, C. P. and Casella, G., 1999. Monte Carlo Statistical Methods, Springer Text in Statistics, Springer-Verlag.

Sjoberg, J., 1995. Non-Linear System Identification with Neural Networks, *Phd Thesis*, Department of Electrical Engineering, Linkoping University, Sweeden.

Vanhatalo, J. and Vehtari, A., 2006. MCMC Methods for MLP-network and Gaussian Process and Stuff–A documentation for Matlab Toolbox MCMCstuff, Laboratory of Computational Engineering, Helsinki University of Technology.

Vehtari, A. and Lampinen, J., 2000. on Bayesian Model Assessment and Choice using Cross Validation Predictive Densities, Technical Report B23, Laboratory of Computational Engineering, Helsinki University of Technology.

White, H., 1989. Learning in Artificial Neural Networks: A Statistical Perspective, *Neural Computation*, **Vol. (1)**, 425.464.

Williams, P. M., 1995. Bayesian Regularization and Pruning using A Laplace Prior, *Neural Computation*, **Vol. 7 (1)**, 117-143.

Winther, O., 1998. Bayesian Mean Field Algorithms for Neural Networks and Gaussian Processes, *Phd Thesis*, University of Copenhagen.

Wolpert, D. H., 1996a. The Existence of a Priori Distinctions between Learning Algorithms, *Neural Computation*, **Vol. 8(7)**, 1391–1420.

Wolpert, D. H., 1996b. The Lack of A Priori Distinctions between Learning Algorithms. *Neural Computation*, **Vol. 8(7)**, 1341–1390.

Wu, L. and Moody, J., 1996. A Smoothing Regularizer for Feedforward and Recurrent Neural Networks, *Neural Computation*, **Vol. 8(3)**, 461.489.

Lai, Y. and Hwang, C. L., 1992. Fuzzy Mathematical Programming, Siprenger-Verlag, Berlin, 80-88.

Zadeh, L., 1965. Fuzzy Sets, *inf. Control*, **Vol. 8**, 338 – 353.

Zimmermann, H. J., 1978. Fuzzy Programming and Linear Programming with Several Objective Functions, *Fuzzy Sets and Systems*, **Vol. 1**.

APPENDIXES

Appendix A. Evaluating normalization factor of prior and noise

Evaluating the Gaussian Integral:

$$I = \int_{-\infty}^{\infty} exp(-\tfrac{\lambda}{2} x^2)dx \tag{A.1}$$

$$I^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} exp(-\tfrac{\lambda}{2}(x^2 + y^2))dxdy \tag{A.2}$$

Transforming the polar coordinates as $x = r\,cost$ and $y = r\,sint$ in (A.2), and then using $r^2 = u$ the square of integral can be evaluated as follow:

$$I^2 = \int_{0}^{\infty}\int_{0}^{2\pi} exp(-\tfrac{\lambda}{2}(r^2))rdrdt$$

$$= \pi\int_{0}^{\infty} exp(-\tfrac{\lambda}{2}u)dr$$

$$= 2\pi/\lambda \tag{A.3}$$

Thus, taking square root of the integral in (A.13, it is obtained as follow:

$$I = \int_{-\infty}^{\infty} exp(-\tfrac{\lambda}{2} x^2)dx = \left(2\pi/\lambda\right)^{1/2} \tag{A.4}$$

For evaluating integral in (2.5) and (2.9):

$$I^N = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} exp\left(-\frac{\beta}{2}\sum_{i=1}^{N}\left\{y_i - \hat{f}(x_i,\theta)\right\}^2\right)dD \qquad dD = dy_1 dy_2 ... dy_N \tag{A.5}$$

Transforming $y_i - \hat{f}(x_i,\theta) = s_i$ and $dy_i = ds_i$ into integral (A.5), and then integral in (A.5) can be arranged as follow:

$$I^N = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} exp(-\frac{\beta}{2}\sum_{i=1}^{N} s_i^2)dS \qquad dS = ds_1 ds_2 ... ds_N \tag{A.6}$$

According to feature of exponential function and by means of equality in (A.4), the integral in (A.6) can be written as follow:

$$I^N = \prod_{i=1}^{N}\int_{-\infty}^{\infty} exp(-\tfrac{\beta}{2} s_i^2)ds_i$$

$$= \left(2\pi/\beta\right)^{1/2} ... \left(2\pi/\beta\right)^{1/2}$$

$$= \left(2\pi / \beta \right)^{N/2} \tag{A.7}$$

Integral in (2.9) can be solved by similar way above

$$I^W = \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} exp(-\frac{\alpha}{2}\|\theta\|^2)d\theta \qquad d\theta = d\theta_1 d\theta_2 \ldots d\theta_W$$

$$= \left(2\pi / \alpha \right)^{W/2} \tag{A.8}$$

Appendix B. Evaluating normalization factor of posterior distribution

In order to solve the following normalization factor of posterior distribution in (2.20),

$$Z_S(\alpha, \beta) = \int \exp\left(-S(\theta_{MP}) - \tfrac{1}{2}(\theta - \theta_{MP})' A(\theta - \theta_{MP})\right) d\theta \tag{B.1}$$

where A is Hessian matrix ($A = \nabla\nabla S_{MP}$), let's transform $\theta - \theta_{MP}$ = w, $d\theta = dw$ and $S(\theta_{MP}) = h'w$ into (B.1), and then integral in (B.1) is arranged as follow:

$$Z_S(\alpha, \beta) = \int \exp\left(-h'w - \tfrac{1}{2} w'Aw\right) dw \tag{B.2}$$

In here, it is convenient to work in terms of the eigenvectors of matrix A,

$$Au_k = \lambda_k u_k \tag{B.3}$$

Since matrix A is real and symmetric, eigenvectors can be chosen form to a complete orthonormal set as follow:

$$u_k' u_l = \delta_{kl} \tag{B.4}$$

The vector w can be expanded a linear combination of eigenvectors,

$$w = U \times \alpha = \sum_{i=1}^{W} \alpha_k u_k \tag{B.5}$$

Thus, the integration over the parameter values $dw_1, dw_2, ..., dw_W$ can be replaced by integration $\partial\alpha_1, \partial\alpha_2, ..., \partial\alpha_W$. The Jacobian of this change of variables given by

$$J = \det\left(\frac{dw_i}{d\alpha_k}\right) = \det(u_{ki}) \tag{B.6}$$

where u_{ki} is the i*th* element of vector u_k. Thus, $J^2 = \det(I) = 1$ and $|J| = 1$. Using the orthonormality of vector u_k , it can be obtained

$$w'Aw = \sum_{i=1}^{W} \lambda_k \alpha_k^2 \tag{B.7}$$

Besides, the projections of h onto the eigenvectors of matrix A can be defined as follow:

$$h_k = h'u_k \qquad k=1,2, \ldots, W \tag{B.8}$$

From (B.5) and (B.8), the first term of exponential function in (B.2) can be evaluated as follow:

$$h'w = h'U\alpha = \left(h_1\, h_2 ... h_w\right) \times \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ . \\ \alpha_W \end{pmatrix} = (\alpha_1 h_1 + \alpha_2 h_2 + ... \alpha_W h_W) \tag{B.9}$$

According to (B.7) and (B.9), integral in (B.2) can be written a set of decoupled integrals over $\partial\alpha_1, \partial\alpha_2, ..., \partial\alpha_W$ as follow:

$$Z_S(\alpha, \beta) = \prod_{k=1}^{W} \int_{-\infty}^{\infty} exp(-\frac{\lambda_k \alpha_k^2}{2} + h_k \alpha_k) d\alpha_k \tag{B.10}$$

In here, the term in exponential function can be written this way:

$$-\frac{\lambda_k \alpha_k^2}{2} + h_k \alpha_k = -\frac{\lambda_k}{2}\left(\alpha_k - \frac{h_k}{\lambda_k}\right)^2 + \frac{h_k^2}{2\lambda_k} \tag{B.11}$$

Let's change integration variables to $\tilde{\alpha}_k = \alpha_k - \frac{h_k}{\lambda_k}$ and then apply to equalities in (A.4) and (A.7), the integral in (B.10) is reduced the following form:

$$Z_S(\alpha, \beta) = (2\pi)^{W/2} |A|^{-1/2} exp(\sum_{k=1}^{W} \frac{h_k^2}{2\lambda_k}) \tag{B.12}$$

If A^{-1} is applied to both sides of (B.3), it can be seen that A^{-1} has eigenvalues λ_k^{-1} as follow:

$$A^{-1} u_k = \lambda_k^{-1} u_k \tag{B.13}$$

Thus, using (B.4) and (B.8), the term of exponential function in (B.12) can be replaced with

$$h'A^{-1}h = \sum_{k=1}^{W} \frac{h_k^2}{2\lambda_k} \tag{B.14}$$

Using this result, the final result can obtained as follow:

$$\begin{aligned} Z_S(\alpha, \beta) &= (2\pi)^{W/2} |A|^{-1/2} exp(½ h'A^{-1}h) \\ &= (2\pi)^{W/2} |A|^{-1/2} exp(-S(\theta_{MP})) \end{aligned} \tag{B.15}$$

Appendix C. Functions of Genetic Algorithm

Encoding:

The parameters of the fitness function should be any data type. Therefore, population type can be real numbers or bit string (binary numbers).

Population size:

Population size determines how many individuals (members) there are in each generation. With a large population size, the genetic algorithm searches the parameter space more thoroughly, thereby reducing the chance that the algorithm will return a local minimum that is not a global minimum. However, a large population size also causes the algorithm to run more slowly.

Creating populations:

In this level the initial population for GA is created. In this study, Uniform distribution preferred which creates a random initial population with a uniform distribution with respect to range of parameters.

Initial population:

Initial population has rows with population size and columns with number of decision variables. Besides, Initial population take initial scores with respect to fitness function.

Selection of parents:

In this step, the genetic algorithm chooses parents for the next generation related to its scores evaluated over fitness function. In this study, the stochastic uniform procedure, which lays out a line in which each parent corresponds to a section of the line of length proportional to its scaled value, is preferred. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on. The first step is a uniform random number less than the step size.

Reproduction:

Reproduction means how the genetic algorithm creates children for the next generation, which include processes of Elitism, crossover, mutation and immigration.

Elite count:

Elite count is the number of individuals that are guaranteed to survive to the next generation. Elite count is a positive integer less than or equal to the population size, and it can be determined

to in accordance with magnitude of population size, or made a decision about this count by trial and error and considering diversification.

Crossover fraction:

This ratio defines the fraction of the next generation, other than elite children, that are produced by crossover. For example, let's define this ratio as 0.9, and then this value shows that 90% percent of population in next generation is produced by crossover since 10% that of population are transferred from current population.

Mutation:

Mutation enables genetic algorithm to make small random changes in the individuals in the population to create mutant children. Thus, mutation provides genetic diversity and enables the genetic algorithm to search a broader space. In this study, Gaussian distribution approach is preferred for mutation.

Gaussian distribution approach adds a random number drawn from a Gaussian distribution with mean 0 to each entry of the parent vector. The standard deviation of this distribution is determined by the parameters Scale and Shrink.

The Scale parameter determines the standard deviation at the first generation. If initial range is set to be a vector *w* with two rows and number of variables columns, the initial standard deviation at *i-th* coordinate of the parent vector is given by Scale x [*w*(i,2) - *w*(i,1)].
The Shrink parameter controls how the standard deviation shrinks as generations go by. If initial range is set to be a vector with two rows and number of variables columns, the standard deviation at *i-th* coordinate of the parent vector at the *j-th* generation, $\sigma_{i,j}$, is given by the recursive formula,

$$\sigma_{i,j} = \sigma_{i,j-1}\left(1 - Shrink\frac{j}{Generations}\right)$$

If you set Shrink to 1, the algorithm shrinks the standard deviation in each coordinate linearly until it reaches 0 at the last generation is reached. A negative value of Shrink causes the standard deviation to grow. In this study, both Scale and Shrink is taken as 1.

Crossover:

Crossover combines the genes (bit or real number) of chromosomes (weight vector) to form the child from its parents. Thus, members have different features joins into population. In this study, scattered procedure is preferred, which creates a random binary vector and selects the genes

where the entry of vector is a 1 from the first parent, other else from the second parent. For example, if w_1 and w_2 are the parents, and random binary vector is [11001000], the procedure returns the following child:

$$w_1 = [13031980] \qquad w_2 = [27081982]$$

$$[11011000]$$

$$\text{Child} => [13031982]$$

Migration

This procedure ensures how individuals move between subpopulations. When migration occurs, the best individuals from one subpopulation replace the worst individuals in another subpopulation. Individuals that can migrate from one subpopulation to another are copied mutually. But, they are not removed from the source subpopulation. In here, migration takes place toward the last subpopulation. That is, the *i-th* subpopulation only migrates into the (i+1)-*th* subpopulation.

Stopping criteria:

There are some options to terminate algorithm:

Generations: defines the maximum number of iterations for the genetic algorithm to perform.

Time limit: it can be fixed the maximum time genetic algorithm runs before stopping.

Fitness limit: The algorithm stops if the best fitness value is less than or equal to the value of Fitness limit.

Stall generations: The algorithm stops if the weighted average change in the fitness function value over stall generations is less than Function tolerance.

Stall time limit: The algorithm stops if there is no improvement in the best fitness value for an interval of time specified by stall time.

Function tolerance: The algorithm runs until the cumulative change in the fitness function value over stall generations is less than or equal to function tolerance.

Appendix D. Markov Chains

Markov Chains:

In particular, it can be shown that if a Markov chain has an invariant distribution π, that is if $\theta^t \sim \pi$ then $\theta^{t+1} \sim \pi$, it follows that:

$$\lim_{t\to\infty} K^t(\theta^0, \theta^t \in A) = \pi(\theta^t \in A) \tag{D.1}$$

where θ^0 is initial state of the chain and $K(\theta^0, \theta^t \in A)$ is a mechanism by means of which the chain moves from one sample to the next. For example, for discrete state spaces, $K(\theta^t, \theta^{t+1} \in A)$ will simply be a transition matrix, while for more general state spaces, $K(\theta^t, \theta^{t+1} \in A)$ will correspond to various types of transition kernels. This convergence result states that if we iterate the transition mechanism many times, the samples produced by it will be approximately distributed according to the invariant distribution.

It is intuitive to introduce Markov chains on finite (discrete) state spaces, where θ can only take *s* possible values $\theta \in \Theta = \{\alpha_1, \alpha_2, ..., \alpha_s\}$. The stochastic process θ is called a Markov chain if:

$$\pi(\theta^0, \theta^1, ..., \theta^M) = \pi(\theta^0) \prod_{t=1}^{M} \pi\left(\theta^t \left| \theta^{t-1}\right.\right) \tag{D.2}$$

where t is the time index. The chain is *homogeneous* if there is a transition matrix:

$$\mathrm{T} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ & & \ddots & \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix} \tag{D.3}$$

such that the transition probabilities $\pi(\theta^t = \alpha_j \left| \theta^{t-1} = \alpha_i \right.) = p_{ij}$ for all t. That is, the evolution of the chain in a space Θ depends solely on the current state of the chain and a fixed transition matrix. Note that the transition matrix is a mapping $T : s \times s \to [0,1]$ with $\sum_{j=1}^{s} p_{ij} = 1$ for any *i*.

After several iterations (multiplications by *T*), $\pi(\theta^t) = \pi(\theta^0)T^t$ converges as follows:

$$\pi(\theta^1) = \pi(\theta^0)T \Rightarrow \pi(\theta^2) = \pi(\theta^0)T^2 \Rightarrow \pi(\theta^3) = \pi(\theta^0)T^3 \Rightarrow \cdots \Rightarrow \pi(\theta^t) = \pi(\theta^t)T^t \tag{D.4}$$

In general state spaces, $(\Theta, \mathrm{B}(\Theta))$, Markov chains are defined as sequences of random variables $\{\theta^t : t = 0,1,2,...\}$, whose evolution in the space Θ, depends solely on the current state of the

chain and a transition kernel. Here, the events correspond to the σ-algebra generated by a countable collection of subsets of Θ, for example intervals in parameter space $\Re^p$. The transition kernel can be interpreted as a matrix with an infinite number of elements, where the elements in each row add up to one. It satisfies:

$$\begin{aligned}\pi(\theta^{t+1} \in A \mid \theta^0, \theta^1, ..., \theta^t) &= \pi(\theta^{t+1} \in A \mid \theta^t) \\ &= \int_A K(\theta^t, \partial\theta^{t+1}) \end{aligned} \tag{D.5}$$

For all measurable sets $A \in \mathrm{B}(\Theta)$, that is, the kernel is a mapping $K : \Theta \times \mathrm{B}(\Theta) \to [0,1]$ with following properties:

- For any fixed set $A \in \mathrm{B}(\Theta)$, $K(\cdot, A)$ is measurable.
- For any fixed set $\theta \in \Theta$, $K(\theta, \cdot)$ is a probability measure.

The analysis is restricted to time-homogeneous Markov chains, that is, the transition kernel is fixed over time. Time varying kernels are, however, necessary when dealing with MCMC optimization algorithms such as simulated annealing (Geman and Geman, 1984; de Freitas, 2000).

As illustrated in the discrete case, once we know the initial distribution of the chain, say $\pi(\partial\theta^0)$, the transition kernel fully determines the behavior of the chain. However, in general state spaces, the iterative multiplications by the transition matrix are replaced by the following recursion:

$$\begin{aligned}\pi(\theta^0 \in A_0) &= \int_A \pi(d\theta^0) \\ \pi(\theta^0, \theta^1 \in A_0 \times A_1) &= \int_{A_0} \int_{A_1} \pi(d\theta^0) K(\theta^0, d\theta^1) \\ &\vdots \\ \pi\left((\theta^0, \theta^1, ..., \theta^M) \in A_0 \times A_1 \times \cdots A_M\right) &= \int_{A_0} \cdots \int_{A_M} \prod_{t=1}^{M} \pi(d\theta^0) K(\theta^{t-1}, d\theta^t) \end{aligned} \tag{D.6}$$

By marginalizing and choosing the initial condition $A_0 = \left\{\theta^0\right\}$, distribution $K^M(\theta^0, \theta^t \in A) \triangleq \pi(\theta^t \in A \mid \theta^0)$ is given by:

$$K^1(\theta^0, \theta^1 \in A) = K(\theta^0, \theta^1 \in A)$$

$$K^2(\theta^0, \theta^2 \in A) = \int_\Theta K(\theta^0, d\theta^1 \in A) K(\theta^1, \theta^2 \in A)$$

$$\vdots$$

$$K^M(\theta^0, \theta^M \in A) = \int_\Theta K(\theta^0, d\theta^1 \in A) K^{M-1}(\theta^1, \theta^M \in A) \tag{D.7}$$

In general the Chapman-Kolmogorov equation is obtained:

$$K^{L+M}(\theta, A) = \int_\Theta K^L(\theta, d\alpha) K^M(\alpha, A) \tag{D.8}$$

when considering discrete state spaces, the iterated application of the transition kernel (multiplication by the transition matrix) converges to an invariant distribution. In general state spaces, it is also possible to demonstrate that the iterated application of the transition kernel (equation (D.7)) converges to an invariant distribution.

Appendix E. Monte Carlo Integration

Monte Carlo Integration using Markov Chains:

Let's define posterior probability density for parameters as $p(\theta \mid x_{1:N}, y_{1:N})$, and the expectation of $\hat{f}(\theta, x_{1:N+1})$ as follow:

$$\mathrm{E}(y_{1:N+1} \mid x_{1:N+1}, y_{1:N}) = \int \hat{f}(\theta, x_{1:N+1})\, p(\theta \mid x_{1:N}, y_{1:N}) d\theta \tag{E.1}$$

In order to find the best guess for y_{N+1} under squared error loss, the integral in (E.1) is used. However, y_{N+1} can be evaluated by using samples from $p(\theta \mid x_{1:N}, y_{1:N})$ instead of solving analytically complicated integral in (E.1) as follow:

$$E\left[f(\theta) \right] = \hat{y}_{N+1} \approx \frac{1}{M} \sum_{t=1}^{M} \hat{f}(x_{N+1}, \theta^t) \quad t=1, 2, \ldots, M \tag{E.2}$$

where $\theta^1, \theta^2, \ldots, \theta^M$ are generated by a process that results in each of them having the distribution $p(\theta \mid x_{1:N}, y_{1:N})$. In simple Monte Carlo methods, θ^t are independent. But, generating these parameters is often infeasible since $p(\theta \mid x_{1:N}, y_{1:N})$ is a complicated distribution. However it may be possible to generate a series of dependent parameters. The Monte Carlo formulation in (E.2) is still an unbiased estimate of $p(\theta \mid x_{1:N}, y_{1:N})$ even when θ^t are dependent, and as long as this dependence is not too great, the estimate still converge to true value as M increases by law of large numbers.

Such a series of dependent parameters may be generated using a Markov Chain that has $p(\theta \mid x_{1:N}, y_{1:N})$ its stationary distribution. A chain is generated by transition from current state θ^t to next one θ^{t+1} by means of transition distribution $T(\theta^{t+1} \mid \theta^t)$. In here, $p(\theta \mid x_{1:N}, y_{1:N})$ is called as an invariant (or stationary) distribution $\pi(\theta)$. If θ^t has distribution given by $\pi(\theta^t)$, then θ^{t+1} will same distribution. This invariance condition can written as follows

$$\pi(\theta^{t+1}) = \int T(\theta^{t+1} \mid \theta^t) \pi(\theta^{t+1}) d\theta \tag{E.3}$$

Invariance with respect to π is implied by the stronger condition of detailed balance as follow:

$$T(\theta^{t+1} \mid \theta^t) \pi(\theta^t) = T(\theta^t \mid \theta^t) \pi(\theta^{t+1}) \tag{E.4}$$

If a chain satisfies this detailed balance, it's called be reversible. A Markov chain that is ergodic has a unique invariant distribution, its equilibrium distribution. Because of dependencies between the θ^t, the number of parameters needed for Monte Carlo estimates to reach a certain level of accuracy may be larger than would be require if θ^t were independent. The chain may also require a long time to reach a point where distribution of the current state is good approximation to equilibrium distribution.

The effect of dependencies on the accuracy of a Monte Carlo estimate can be quantified in terms of the autocorrelations between the values of $f(\theta^t)$ in (E.2) once equilibrium has been reached. If $f(\theta)$ has finite variance, the variance of estimate of $f(\theta)$ will be $Var\left[f(\theta)\right]/M$ if θ^t's are independent. When θ^t are dependent and M is large, the variance of the estimate is $Var\left[f(\theta)\right]/(M/\tau)$ where τ is a measure of inefficiency due to the presence of dependencies as follow:

$$\tau = 1 + 2\sum_{s=1}^{\infty}\rho(s) \tag{E.5}$$

Here, $\rho(s)$ is the autocorrelation of $f(\theta)$ at lag s, defined by

$$\rho(s) = \frac{E\left[\left(f(\theta^t) - E\left[f(\theta)\right]\right)\left(f(\theta^{t-s}) - E\left[f(\theta)\right]\right)\right]}{Var[f(\theta)]} \tag{E.6}$$

Iteration number *t* doesn't affect above definition since it is assumed that equilibrium has been reached. For Markov chains used to sample complex distributions π, these autocorrelations are typically positive, leading to a value for τ greater than one. If τ is less than one, in which case the dependencies between $f(\theta^t)$ are at fair level to increase the accuracy of the estimate.

If π is equilibrium distribution of ergodic Markov chain, chain converges to its distribution as rapidly as possible, and in which the states visited once equilibrium distribution is reached are not highly dependent.

Appendix F. Gibbs Sampling

In order to sample from a distribution over a multi-dimensional parameter $\theta = \{\theta_1, \theta_2, ..., \theta_p\}$, Gibbs sampling is applicable. If directly sampling from distribution π is infeasible, one component of θ is generated by conditional distribution of π when the values for all the other components of θ are given. A Markov chain can be simulated in which θ^{t+1} is generated from θ^t as follows:

Pick θ_1^{t+1} from full conditional $\pi\left(\theta_1 \middle| \theta_2^t, \theta_3^t, ..., \theta_P^t\right)$ given $\theta_2^t, \theta_3^t, ..., \theta_P^t$

Pick θ_2^{t+1} from full conditional $\pi\left(\theta_2 \middle| \theta_1^{t+1}, \theta_3^t, ..., \theta_P^t\right)$ given $\theta_1^{t+1}, \theta_3^t, ..., \theta_P^t$

.
.
.

Pick θ_j^{t+1} from full conditional $\pi\left(\theta_j \middle| \theta_1^{t+1}, ..., \theta_{j-1}^{t+1}, \theta_{j+1}^t, ..., \theta_P^t\right)$ given $\theta_1^{t+1}, \theta_2^{t+1}, ..., \theta_{j-1}^{t+1}, \theta_{j+1}^t, ..., \theta_P^t$

.
.
.

Pick θ_p^{t+1} from full conditional $\pi\left(\theta_p \middle| \theta_1^{t+1}, \theta_2^{t+1}, ..., \theta_{P-1}^{t+1}\right)$ given $\theta_1^{t+1}, \theta_2^{t+1}, ..., \theta_{P-1}^{t+1}$

where $\pi\left(\theta_j \middle| \theta_1, ..., \theta_{j-1}, \theta_{j+1}, ..., \theta_P\right)$ is a full conditional distribution for θ_j given $\theta_1, ..., \theta_{j-1}, \theta_{j+1}, ..., \theta_P$ under distribution π. In here, the new value for θ_{j-1}^{t+1} is used immediately when picking a new value for θ_j.

Such transitions will leave desired distribution, π, invariant if all the steps making up each transition leave π invariant. Since step j leaves θ_k for $j \neq k$ unchanged, the marginal distribution for these components is certainly invariant. Besides, these transitions do not necessarily lead to ergodic Markov chain since the joint distributions of all θ_j must also be desired distribution if simulation is started by a desired distribution π.

RESUME

Name and Surname: Ozan KOCADAĞLI

Place of Birth: Ayvalık, TURKEY

Birth date: 13.03.1980

Education

B.S.: Mimar Sinan Fine Arts University, Faculty of Arts and Sciences, Mathematics (2004)

M.S.: Mimar Sinan Fine Arts University, The Institute of Science and Technology, Department of Statistics (2006)

Doctoral research: The George Washington University, The Institute for Integrating Statistics in Decision Sciences, Washington D.C., USA (2010 - 2011)

Ph.D.: Mimar Sinan Fine Arts University, Institute of Science and Technology, Department of Statistics (2012)

Affiliations:

Fuzzy System Associations

The International Statistical Institute (ISI)

Bernoulli Society (BS)

International Society for Business and Industrial Statistics (ISBIS)

Grants and Fellowships:

- The Fifth General Conference on Advanced Mathematical Methods in Finance, AMaMeF 2010, Bled, Slovenia. (Grants)
- The Scientific and Technological Research Council of Turkey (TÜBİTAK), International Ph.D. Research Fellowship, 2010–2011, The George Washington University, Washington D.C., USA.
- Joint Meeting of y-BIS and jSPE, 2012, Lisbon, Portugal. (Grants)
- Mimar Sinan F.A. University, Scientific Project Support, 2012.

Work experiences: Mimar Sinan Fine Arts University, Faculty of Arts and Sciences, Department of Statistics, Research Assistant (2006 - continue)

Contact info:

Phone: 090 212 246 00 11 – 5511

E-mail: ozankocadagli@msgsu.edu.tr ozankocadagli@gmail.com