Turkish Journal of Forecasting vol. 01 no. 2 (2017) pp. 54-65

Content list available at JournalPark

Turkish Journal of Forecasting

Journal Homepage: tjforecast.xyz

# Bayesian Learning based Gaussian Approximation for Artificial Neural Networks

O. Kocadagli*

*Mimar Sinan Fine Arts University, Department of Statistics, Istanbul, TURKEY*

A R T I C L E  I N F O

*Article history*:

| | |
|---|---|
| Received | 27 October 2017 |
| Accepted | 25 December 2017 |
| Available online | 29 December 2017 |

*Keywords*:

Bayesian neural networks
Bayesian learning
Gaussian approach
Fixed hyperparameters
Gradient based algorithms

A B S T R A C T

In the nonlinear systems, the pre-knowledge about the exact functional structure between inputs and outputs is mostly either unavailable or insufficient. In this case, the artificial neural networks (ANNs) are useful tools to estimate this functional structure. However, the traditional ANNs with the sum squared error suffer from the approximation and estimation errors in the high dimensional and excessive nonlinear cases. In this context, Bayesian neural networks (BNNs) provide a natural way to alleviate these issues by means of penalizing the excessive complex models. Thus, this approach allows estimating more reliable and robust models in the regression analysis, time series, pattern recognition problems etc. This paper presents a Bayesian learning approach based on Gaussian approximation which estimates the parameters and hyperparameters in the BNNs efficiently. In the application part, the proposed approach is compared with the traditional ANNs in terms of their estimation and prediction performances over an artificial dataset.



## 1. Introduction

The Bayesian Neural Networks (BNNs) provide a flexible way to model the non-linear problems due to their capabilities to cope with the model complexity. Besides, they ensure a natural interpretation to the estimations and the predictions performed over the estimated models. For this reason, BNNs are very useful in the regression, time series, classification, density estimation problems, etc. In the context of Artificial Neural Networks (ANNs), Bayesian treatments of the learning are typically based on Gaussian approximation, ensemble learning and Markov Chain Monte Carlo (MCMC) simulations known as the full Bayesian approach. For ANNs, Gaussian approximation known as Laplace's method was introduced by Buntine and Weigend [1] and MacKay [2]. This approach is to model the posterior distribution by a Gaussian distribution, centered locally at a mode of posterior distribution of parameters [2]. The ensemble learning was introduced by Hinton and van Camp [3] in which the approximating distribution is fitted globally by minimizing a Kullback - Leibler divergence rather than locally. Within the context of Full Bayes, Neal [4] introduced advanced Bayesian simulation methods in which MCMC simulations are used to generate samples from the posterior distribution. However, MCMC techniques can be computationally expensive, and also suffer from assessing the convergence. For this reason, Neal [4] integrated Bayesian learning with Hybrid Monte Carlo (HMC) method introduced by Duane et al. [5] to overcome the mentioned shortcomings. Afterwards, Bayesian applications to ANNs was reviewed in [6, 7, 8] in detail.

In the literature, there are the remarkable studies in which are focused the specific problems related to ANNs from Bayesian perspective. For instance; Insua and Müller [9], Marrs [10], Holmes and Mallick [11] worked on the issue of selecting the number of hidden neurons with the growing and the pruning algorithms for the dimensionality problem in the ANNs. In these studies, they applied the reversible jump MCMC algorithm introduced by Green [12],

* Corresponding author.
*E-mail address*: ozankocadagli@gmail.com (Ozan Kocadagli)

Richardson and Green [13]. Freitas [14] incorporated the particle filters and the sequential Monte Carlo (MC) methods in the BNNs. Liang and Wong [15] proposed to the evolutionary MC algorithm which samples the parameters in the ANNs from the Boltzmann distribution using the mutation, the crossover and the exchange operations defined in the Genetic Algorithms (GAs). Chua and Goh [16] proposed a hybrid Bayesian back-propagation approach to the multivariate modeling in the ANNs. They used the stochastic gradient descent algorithm integrated with the evolutionary operators to produce the parameters in the ANNs. Liang [17] and Lord et al. [18] used the truncated Poison priors to determine the neuron numbers in the hidden layers, and estimated the parameters in the ANNs via the evolutionary MC algorithms proposed by Liang and Wong [15]. Lampinen and Vehtari [19], Vanhatalo and Vehtari [20] improved a hybrid and reversible MCMC algorithm that are based on Neal [4]. Marwala [21] adapted to mutation and crossover operators defined in GAs into Bayesian learning and estimated the parameters using Genetic MC algorithm. Mirikitani [22] proposed a probabilistic approach to recursive the second-order training of recurrent neural networks for improved time-series modelling in which the regularization hyperparameters leads to better generalization and stable numerical performance. Goodrich [23] developed a powerful methodology for estimating the full residual uncertainty in the network weights and making predictions by using a modified Jeffery's prior combined with a Metropolis MCMC method. Martens and Sutskever [24] resolved the long-outstanding problem of how to effectively train recurrent neural networks on complex and difficult sequence modelling problems which may contain long-term data dependencies. Niu et al. [25] adapted Hybrid MC, proposed by Neal [4] and Duane et al. [5]. Beam et al. [26] examined Hybrid Monte Carlo proposed by Neal [4] in the context of full Bayesian approach, and then they compared this procedure with the existing methods using artificial and real datasets. Kocadagli [27, 28, 29] integrated the hierarchical Bayesian learning with GAs and the fuzzy numbers to estimate the parameters in the ANNs.

In order to measure the model complexity and to estimate the parameters in the BNNs, the hybrid methods are frequently preferred against the non-linear problems having high dimensional parameter spaces due to their superior performance. In these approaches, the parameters in the BNNs are estimated by MCMC techniques integrated with the gradient optimization algorithms, the simulated annealing method or the evolutionary algorithms. HMC approaches with the gradient search algorithms provide desired performance against the mentioned problems because they cope with random-walk behaviour encountered in the MCMC treatments, and reduces the training time. There are the efficient optimization algorithms such as gradient descent with momentum, conjugate gradients, BFGS and Levenberg – Marquardt which are integrated with MCMC techniques. However, the gradient algorithms might not able to explore all space freely in the high dimensional parameter cases including many local optima, and it doesn't work without the derivative information. In terms of ANN applications, some shortcomings of these algorithms are discussed in [7]. In order to handle the non-linear functions without derivative information and reduce training time expended for the parameter estimation in the ANNs, recently the evolutionary algorithms are preferred as GAs, Particle Swarm Optimization, Ant and Bee Colony Algorithms, etc.

The aim of this study is to present a Bayesian learning based on Gaussian approximation with fixed hyperparameters which estimates the parameters in BNNs accurately, and overcome some shortcomings like over/lover fitting, variance/bias tradeoff. This article is organized as following. The second section includes the basic definitions of ANNs as well as the error functions and model complexity. The third section introduces Bayesian learning and Gaussian approximation. The last part is left to the applications in which Bayesian learning approach is compared with traditional ANNs in terms of their estimation and prediction performances over an artificial dataset.

## 2. Framework of Neural Networks

A simple ANN is composed of simple inter-connected nodes or neurons, and there are usually three or four layers in its structure. The intermediate layers are called hidden layers. In Fig. 1, the simple NN with three layers is demonstrated in which the hidden layer includes *p* neurons. The neurons are arranged in the 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. The external inputs exist at the first layer and system outputs come out from 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 an activation function, to produce a result. The neuron connections have weights that are adapted to improve its overall performance. In here, biases are intuitively used as a threshold value, and control the inputs of activation function and outputs.

According to network structure in Fig.1, the mathematical relation between inputs, neurons of hidden layer and *t-th* output for *i-th* observation can be formulated as follow:

$$f_t(x_i,\theta)=b_t^{II}+\sum_{k=1}^{p} w_{tk}^{II} A_k(w_k^I x_i+b_k^I) \qquad t=1, 2, \ldots r \tag{1}$$

where $y_i=[y_{1i}\ \ y_{2i}\ \ \ldots\ \ y_{ri}]$ and $x_i=[x_{1i}\ \ x_{2i}\ \ \ldots\ \ x_{mi}]$ are target and input vectors of *i-th* observation respectively; $w_k^I=[w_{1k}^I\ \ w_{2k}^I\ \ \ldots\ \ w_{mk}^I]$ is a row vector that include weights between all inputs and *k-th* neuron in the hidden layer; $w_t^{II}=[w_{1t}^{II}\ \ w_{2t}^{II}\ \ \ldots\ \ w_{pt}^{II}]$ is a 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; $\theta=\{w_k^I, w_k^{II}, b_k^I, b_k^{II}\}$ is a parameter vector that covers all weights and biases; $i=1, 2, \ldots, N$; $j=1, 2, \ldots, m$; $k=1, 2, \ldots, p$ and $t=1, 2, \ldots, r$.

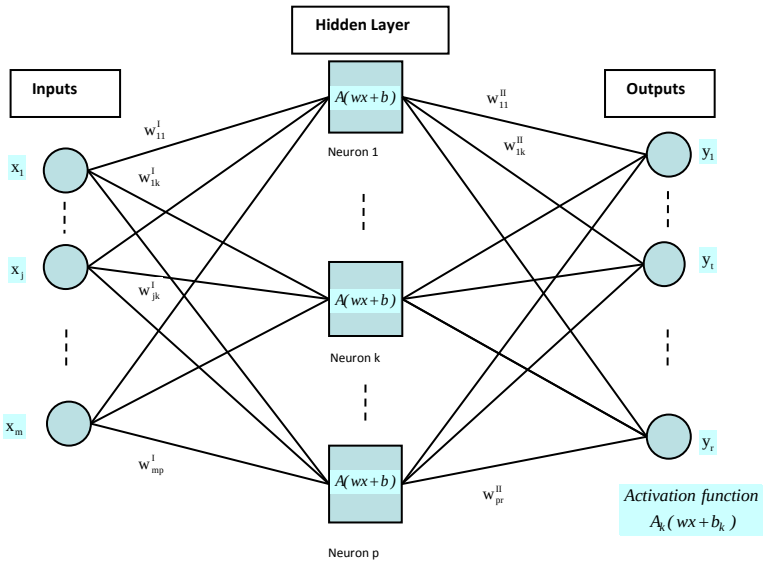



**Figure 0.** The simple NN structure with one hidden layer

Here, the activation function $A_k(w_k x_i+b_k^I)$ provides nonlinearity to the NN structure and scales its received inputs to its output range. In ANNs, the activation functions with different shape and structure are used according to data and model type. In this paper, as an activation function in the hidden layer, the hyperbolic tangent is preferred because it has the properties of differentiation and quick computability.

The functional structure in Eq. (1) can be estimated by modifying the connection weights to reduce the errors between the actual output and the target output values at a certain satisfactory level. To estimate an estimator of the regression function by training ANNs, one of the most popular approaches is minimizing the following mean square empirical risk:

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

Using this empirical risk brings about two types of error, namely the approximation and estimation errors. The approximation error arises because the exact nonlinear behaviour of the regression function is rarely known in the real-life problems. For this reason, $f(x)$ has to be approximated by a combination of the 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 can be interpreted as the lack of knowledge about the conditional distribution $p(y|x,\theta)$. In likelihood methods, the empirical risk in Eq. (2) is minimized instead of estimating this distribution. One of the heuristic reasons for doing this is that the regression function minimizes the expected risk or $L^2$ norm as follow [14]:

$$f(\cdot)=\arg\min_{h(\cdot)\in T} R[h(\cdot)] \tag{3}$$

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 , and $R[h(\cdot)]$ corresponds to the mean square expected risk, given by

$$\begin{aligned} R[h(\cdot)] &= \|y - h(x,\theta)\|^2_{L^2(p)} = E\left[(y - h(x,\theta))^2\right] \\ &= \int (y - h(x,\theta))^2 \, p(y,\theta|x)\, dy\, d\theta \end{aligned} \tag{4}$$

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. As the number of parameters increases, the expression for the empirical risk becomes more complex; hence, the estimation error can increase [30]. From here, it can be concluded that 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 is well known as the bias/variance tradeoff [31].

For an acceptable generalization performance, the bias and variance terms have to be balanced. 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 [14]. Bayesian learning ensures this approach inherently, since it provides probabilistic knowledge about related data.

Imposing the smoothness constraints on the model, known as regularization, is another approach. 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 estimate a function that is simultaneously close to the data and smooth, the following extension form instead of the empirical modelling error is used:

$$R_r[\hat{f}(x,\hat{\theta})] = \frac{1}{N}\sum_{i=1}^{N}(y_{ti} - \hat{f}_t(x_i,\hat{\theta}))^2 + \nu\Omega \tag{5}$$

where $\nu$ is a positive parameter that balances the tradeoff between smoothness and data approximation. Here, it can be interpreted that the larger values of $\nu$ have more impact on the smoothness of the model, while smaller ones have more impact on fitting the data. The function $\Omega$ penalizes the excessive model complexity. Weight decay and Laplace function approaches are well-known for function $\Omega$ . The most popular one of those is weight decay whose functional type is given by:

$$\Omega = \sum_i \theta_i^2 \tag{6}$$

where *i* is an index of total parameter number Mackay [2]. Other common approaches to controlling the complexity of the estimates include early stopping, training with noise, mixtures of networks and growing and pruning techniques. However, these approaches have substantial shortcomings discussed in the literature of ANNs [27, 28, 29, 32, 33, 34, 35, 36].

## 3. Bayesian Learning

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:

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{p(D)} = \frac{p(D \mid \theta)\, p(\theta)}{\int p(D \mid \theta)\, p(\theta) d\theta} \tag{7}$$

When the statistical information about the parameters is given, all features of interest by standard probability marginalization and transformation techniques can be obtained theoretically. By means of posterior distribution, the predictive density can be estimated as follow [27]:

$$p(y_{1:N+1} \mid x_{1:N+1}, y_{1:N}) = \int p(y_{1:N+1} \mid \theta, x_{1:N+1})\, p(\theta \mid x_{1:N}, y_{1:N}) d\theta \tag{8}$$

Thus, the forecast of interest can be evaluated as:

$$\mathrm{E}(y_{1:N+1} \mid x_{1:N+1}, y_{1:N}) = \int \hat{f}(\theta, x_{1:N+1})\, p(\theta \mid x_{1:N}, y_{1:N}) d\theta \tag{9}$$

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

### 3.1 Gaussian approximation for Bayesian learning

Gaussian approximation to ANNs is first proposed by MacKay [2]. According to this approximation, the uncertainty in the parameter space is assigned to a probability distribution representing the degree of belief in the different values of the parameter vector. By maximizing the posterior distribution over the parameters; the most probable parameters values can be determined. MacKay [2] show that maximizing the posterior distribution corresponds to minimizing the regularized error function that is very similar to the expression in Eq. (5). The posterior distribution is then used to evaluate the predictions of the trained network for new values of the input variables as well.

In Gaussian approximation, the data set is modelled from mapping in Eq. (1) under some additive noise process $\varepsilon$ [7, 27,28,29,38]:

$$y_{ti} = f_t(x_i, \theta) + \varepsilon \tag{10}$$

If $\varepsilon$ is modelled as zero-mean Gaussian noise with standard deviation $\sigma_{noise}$, then the probability of a data value given the parameter vector $\theta$ is:

$$p(y_i \mid x_i, \theta, \beta) = \frac{1}{(2\pi)^{1/2} \beta^{-1/2}} \exp\left( -\frac{\beta}{2} \left\{ y_{ti} - \hat{f}_t(x_i, \theta) \right\}^2 \right) \tag{11}$$

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

$$p(D \mid \theta, \beta) = \prod_{i=1}^{N} p(y_i \mid \theta, x_i) = \frac{1}{(2\pi)^{N/2} \beta^{-N/2}} exp\left( -\frac{\beta}{2} \sum_{i=1}^{N} \left\{ y_{ti} - \hat{f}_t(x_i, \theta) \right\}^2 \right) \tag{12}$$

More generally, the likelihood function can be written as

$$p(D \mid \theta, \beta) = \frac{1}{Z_D(\beta)} exp(-\beta E_D) \tag{13}$$

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 $\int dD = \int dy_1 dy_2 .... dy_N$. Here, the function $Z_D(\beta)$ is called a normalization constant, and it can be evaluated as [2].

$$Z_D(\beta) = \int exp(-\beta E_D) dD = (2\pi)^{N/2} \beta^{-N/2} \tag{14}$$

In the Bayesian framework, probability distribution defined over parameter values reflects any the prior knowledge about network mapping that is expected to estimate. Generally, distribution of parameters is assumed as an exponential form:

$$p(\theta) = \frac{1}{Z_\theta(\alpha)} exp(-\alpha E_\theta) \tag{15}$$

where $\mathrm{Z}_\theta(\alpha)$ 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 mentioned above, the tradeoff between variance and bias indicates that a smooth network function typically has 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 $E_\theta$ [7, 16, 27, 28,29].

$$E_\theta = \frac{1}{2}\|\theta\|^2 = \frac{1}{2}\sum_{w=1}^{W}\theta_w^2 \tag{16}$$

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

$$p(\theta) = \frac{1}{Z_\theta(\alpha)} exp(-\frac{\alpha}{2}\|\theta\|^2) \tag{17}$$

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

$$Z_\theta(\alpha) = \int exp(-\alpha E_\theta) d\theta = (2\pi/\alpha)^{W/2} \tag{18}$$

where $W$ is total parameter number. If $\|\theta\|$ is large, $E_\theta$ would large as well, and $p(\theta)$ would small. Since the parameter $\alpha$ itself controls the distribution of other parameters, it is called a hyperparameter. An advantage of using Gaussian distribution is the evaluation of normalization coefficients $Z_\theta(\alpha)$ and $Z_D(\beta)$ analytically. There are many alternatives for the priors. For instance, Laplacian priors were used by Williams [37]; the entropy based priors by Buntine and Weigend [1]; the appropriate selection of priors of very large networks by Neal [8], and the inverse gamma priors for a hierarchal structure by Lampinen and Vehtari [19].

Once a prior distribution and an expression for the likelihood are set, the posterior distribution can be constructed by using Bayes` theorem in Eq. (7) as follow [7, 16, 27, 28, 29]:

$$p(\theta|D) = \frac{1}{Z_S(\alpha,\beta)} exp(-\beta E_D - \alpha E_\theta) = \frac{1}{Z_S(\alpha,\beta)} exp(-S_\theta) \tag{19}$$

where

$$S(\theta) = \beta E_D + \alpha E_\theta \tag{20}$$

and

$$Z_S(\alpha,\beta) = \int \exp(-\beta E_D - \alpha E_\theta) d\theta \tag{21}$$

Here, the main problem is to find the parameter vector $\theta_{MP}$ corresponding to the maximum of the posterior distribution. This can be achieved by minimizing the negative logarithm of Eq. (19). Since normalizing factor in Eq. (19) is independent of parameters, the exploring the maximum of the posterior distribution is equivalent to minimizing $S(\theta)$ given by Eq. (20). Thus, $S(\theta)$ can be written as follow:

$$S(\theta) = \frac{\beta}{2}\sum_{i=1}^{N}(y_{t,i} - \hat{f}_t(x_i,\theta))^2 + \frac{\alpha}{2}\|\theta\|^2 \tag{22}$$

The most probable value of parameter vector, denoted by $\theta_{MP}$, can be found by the minimizing of the right-hand side in Eq. (22). From Eq. (22), some consistence conclusions can be inferred. For instance, the first term in Eq. (22) grows with $N$ while the second term does not. If $\alpha$ and $\beta$ are fixed, then $N$ increases, the first term becomes more and more dominant, until eventually the second term becomes insignificant. From here, it can be concluded that the maximum likelihood solution is then a very good approximation to the most probable solution $\theta_{MP}$ for large $N$. Conversely, for small data sets the prior term plays an important role in determining the location of the most probable solution [7, 16, 27, 28, 29, 38].

## 4. Applications

In order to compare the performance of BNNs with that of traditional ANNs, an artificial data set with two inputs and one target that was considered in [19]. The motivation behind choosing this data set is to examine the capability of ANNs and BNNs against the data having the non-linear and complex structure. In the analysis, dataset was divided

into two parts as training and test with 200 and 25 observations as demonstrated in Fig. 2 and Fig. 3 respectively. As the activation function in the hidden layer of ANNs, the tangent hyperbolic function given in Fig. 4 was used. To train the traditional ANNs, the gradient descent as known also the steepest descent and the gradient descent with momentum methods were handled. For Gaussian approximation with fixed hyperparameters of BNNs, BFGS algorithm was preferred due to its advantage in the evaluation of Hessian matrix. The performances of ANNs with different high dimensional parameter spaces were examined by using the different numbers of neurons in the hidden layers. In the analysis, the computer having the processor with Intel(R) Core(TM) i3 CPU 2.13GHz, 4GB RAM and "64 bit" operating system was used. The software of the proposed approaches was written in MATLAB 7.12 package program. The training results of the proposed and traditional ANNs are given in details below.

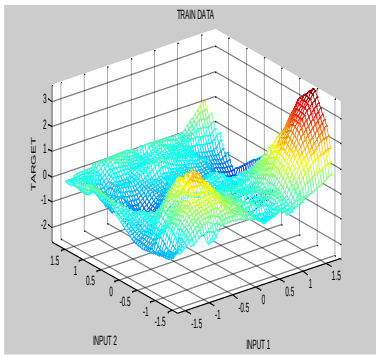



**Figure 2.** Training data

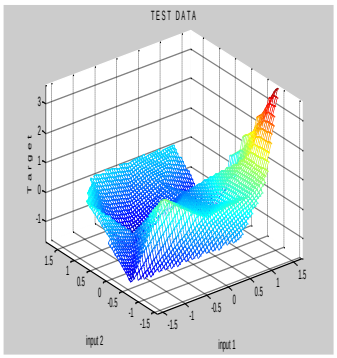



**Figure 3.** Test data

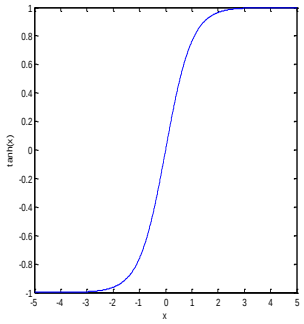



**Figure 4.** Tangent Hyperbolic

### 4.1. The steepest descent with sum squared error

To examine the performance of traditional approaches with the sum squared error, the parameters in the ANNs was estimated by the steepest descent optimization algorithm. The performance of the steepest descent was observed for different learning rate, neurons and iterations. According to training results in Table 1, the steepest descent algorithm achieves the best fit using 25 neurons and 0.005 learning rate for 1000 iterations. From Table 1, it can be concluded that if the neuron number is increased, then the algorithm requires more iterations for a good-fit, and both the mean squared errors (MSE) of training and test data substantially increase as well. For instance, the learning rate is fixed, and then the neuron number is taken as 100, the performance of proposed algorithm is not sufficient for a good fit, even the iteration number is increased to 2000 according to the results given in Table 1 and between Figures 5 - 8. At the same number of neurons, if the algorithm runs until 5000 iterations, both MSE of training and test data substantially decreases. If the iteration number is taken as 10000, then MSE of training data decreases, but MSE of test data substantially increases because of over-fitting. In such cases, the algorithm should be stopped at any iteration where MSEs of validation or test data begins to grow. However, it is known well that the early stopping approach might causes to the serious estimation errors when ANNs are trained by dataset with excessive noise and the high dimensional parameter spaces.

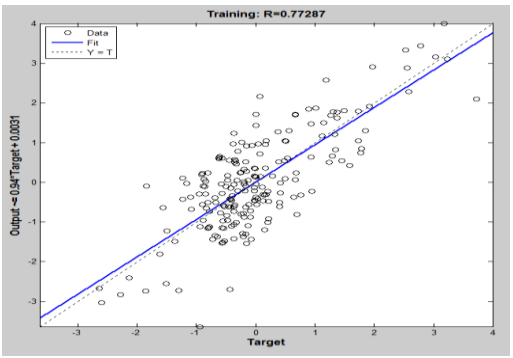



**Figure 5.** Correlation between targets and outputs

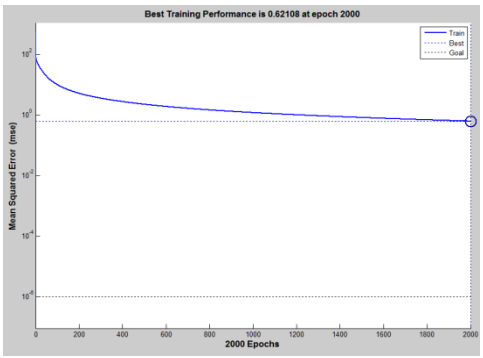



**Figure 6.** The performance of the Steepest Descent

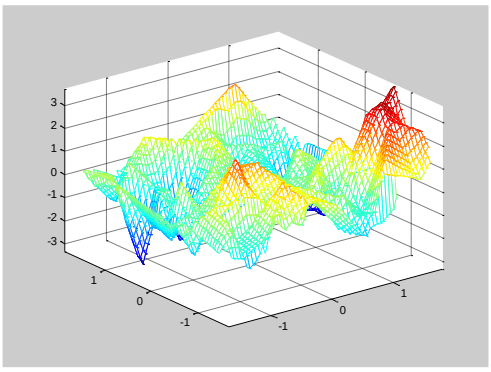


**Figure 7.** The performance for training data

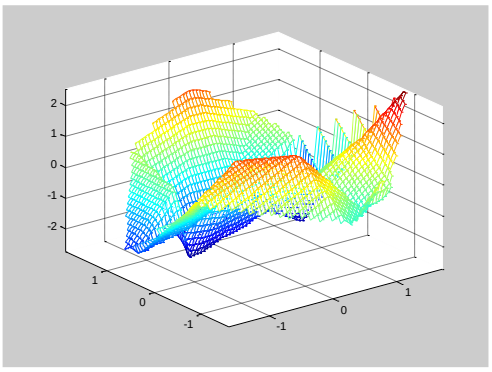


**Figure 8.** The performance for test data

**Table 1.** 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 | 0.723 | 0.622 | 0.759 | 5 |
| | 25 | 0.005 | 0.248 | 0.259 | 0.878 | 0.916 | 5 |
| | 25 | 0.01 | 0.188 | 0.237 | 0.904 | 0.923 | 5 |
| **1000** | 25 | 0.001 | 0.466 | 0.652 | 0.814 | 0.777 | 10 |
| | 25 | 0.005 | 0.159 | 0.145 | 0.920 | 0.948 | 10 |
| | 25 | 0.01 | 0.114 | 0.215 | 0.943 | 0.921 | 10 |
| **1000** | 50 | 0.001 | 0.836 | 1.447 | 0.696 | 0.628 | 11 |
| **1000** | 50 | 0.005 | 0.205 | 0.292 | 0.898 | 0.892 | 11 |
| **2000** | 100 | 0.001 | 0.621 | 1.288 | 0.772 | 0.640 | 25 |
| **5000** | 100 | 0.001 | 0.252 | 0.485 | 0.885 | 0.851 | 64 |
| **10000** | 100 | 0.001 | 0.097 | 0.527 | 0.953 | 0.868 | 125 |

## 4.2. The steepest descent with momentum

In the second implementation; ANNs were trained by the steepest descent with momentum. The estimation results of ANNs are given in Table 2. According to Table 2, this algorithm achieved the best fit using 0.005 learning rate and 0.5 momentum constant. While the number of neurons was fixed as 50, the best fit was obtained at the end of the 5000 iterations. By using the same number of neurons, the algorithm produced the following results given between Figures 9 – 11 at the end of the 10000 iterations. From Table and Figures, it can be seen that while the MSE of training data substantially decreases, MSE of the test data slightly increases inversely. Besides, the algorithm caused the overfitting to both training and test data as seen in Figures 9 and 10. According to Figure 11, the algorithm should be run until 9000-th iteration; otherwise, the overfitting problem becomes inevitable. Besides, if the neuron number is fixed as 100, the effectiveness of algorithm disappears, and MSEs of training and test data arise even the iteration number is increased.

**Table 2.** 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 | 0.606 | 1.048 | 0.700 | 0.582 | 11 |
| | 25 | 0.001 | 0.5 | 0.835 | 1.061 | 0.572 | 0.594 | 11 |
| **1000** | 25 | 0.005 | 0.1 | 0.249 | 0.364 | 0.872 | 0.871 | 11 |
| | 25 | 0.005 | 0.5 | 0.324 | 0.334 | 0.830 | 0.874 | 11 |
| **1000** | 25 | 0.01 | 0.5 | 0.204 | 0.242 | 0.896 | 0.920 | 11 |
| | 25 | 0.01 | 0.1 | 0.168 | 0.179 | 0.915 | 0.937 | 11 |
| **1000** | 50 | 0.001 | 0.1 | 1.390 | 3.327 | 0.611 | 0.442 | 12 |
| **5000** | 50 | 0.001 | 0.1 | 0.230 | 0.275 | 0.889 | 0.909 | 59 |
| **5000** | 50 | 0.005 | 0.5 | 0.107 | 0.104 | 0.950 | 0.965 | 65 |
| **10000** | 50 | 0.005 | 0.5 | 0.076 | 0.109 | 0.965 | 0.961 | 111 |
| **10000** | 100 | 0.005 | 0.5 | 0.144 | 0.270 | 0.933 | 0.926 | 126 |

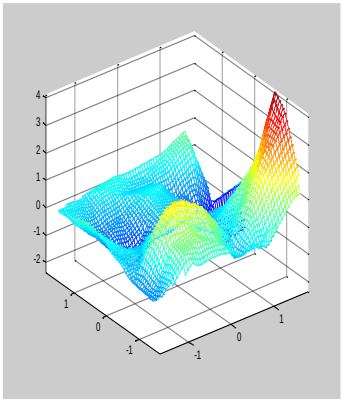


**Figure 9.** The performance for training data

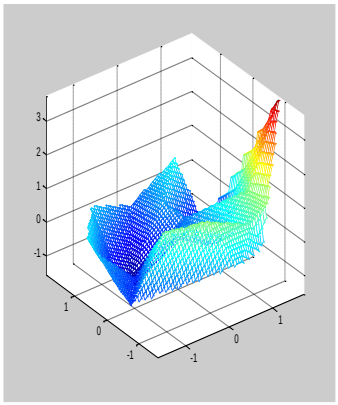


**Figure 10.** The performance for test data

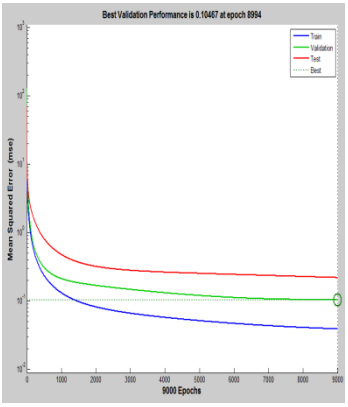



**Figure 11.** The performance for iterations

### 4.3. Gaussian approach with fixed hyperparameters

In this implementation, for Bayesian learning implementation to ANNs, Gaussian approach with fixed hyperparameter is used. To do this, Quasi-Newton known as Broyden, Fletcher, Goldfarb, and Shanno (BFGS) optimization algorithm 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. Besides, the number of neurons in the hidden layer was determined as 25, and then the algorithm run until 1000 iterations. In the second implementation, it was supposed that the noise amount included in data set is more important, so the hyperparameters Alpha and Beta were taken as 0.1 and 0.9, respectively. At the end of training processes in which the number of iteration was fixed as 1000, the statistical indicators were found as Table 3 and between Figures 12 - 18 below.

**Table 3.** The performance of the fixed hyperparameter approach

| Hyperparameters | Neuron number | Training MSE | Test MSE | Training correlation | Test correlation | Training (Seconds) |
|---|---|---|---|---|---|---|
| Alpha= 0.5, Beta=0.5 | 25 | 0.191 | 0.254 | 0.907 | 0.924 | 29 |
| Alpha= 0.3, Beta=0.7 | 25 | 0.079 | 0.114 | 0.962 | 0.971 | 29 |
| Alpha= 0.1, Beta=0.9 | 25 | 0.052 | 0.063 | 0.974 | 0.982 | 27 |
| Alpha= 0.01, Beta=0.99 | 25 | 0.035 | 0.043 | 0.982 | 0.986 | 27 |
| Alpha= 0.01, Beta=0.99 | 50 | 0.031 | 0.050 | 0.984 | 0.982 | 41 |
| Alpha= 0.01, Beta=0.99 | 100 | 0.026 | 0.053 | 0.986 | 0.983 | 103 |

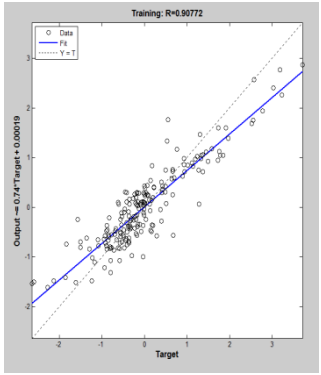



**Figure 12.** Alpha=0.5, Beta=0.5

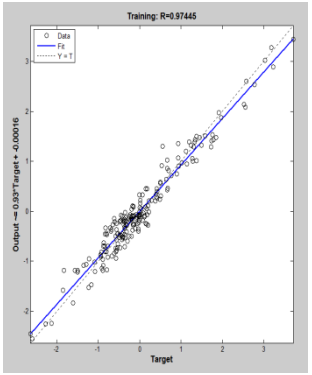



**Figure 13.** Alpha=0.1, Beta=0.9

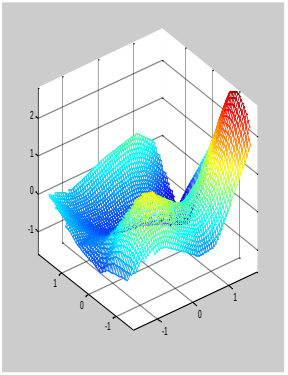


**Figure 14.** Alpha=0.5, Beta=0.5

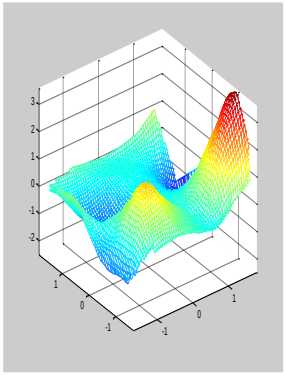
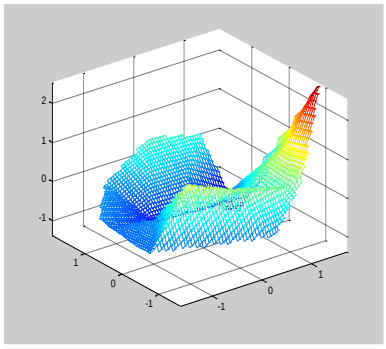
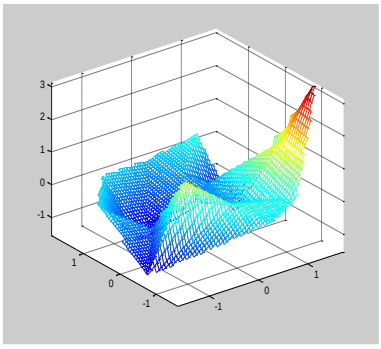


**Figure 15.** Alpha=0.1, Beta=0.9





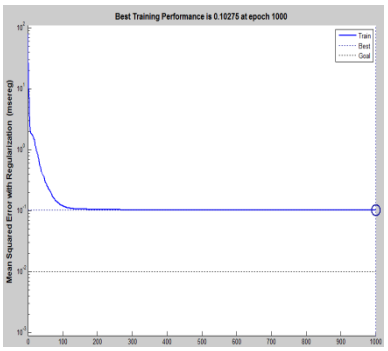



**Figure 16.** Test output for Alpha=0.5, Beta=0.5 **Figure 17.** Test output for Alpha=0.1, Beta=0.9 **Figure 18.** Performance for 1000 iterations

From Table 3 and Figures 12 – 18, it can be seen that the algorithm performs a better performance when Beta is taken as 0.9 instead of 0.5 because search method will be more sensitive to the training data for greater Beta values. As seen in Fig. 18, at the end of 1000 iterations, the total error in Eq. (22) and MSE are 0.10275 and 0.0525, respectively. From Fig. 18, it can be seen that the algorithm reaches the best solution at the end of 300 iterations. However, in order to monitor the performance of algorithms for different hyperparameters and neuron numbers, the iteration number is fixed as 1000. From Table 3, it can be seen that the correlation coefficients have got the larger values as Beta is increased; however MSE decreases inversely. 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 because of over-fitting to data, but MSE of test data and training time increase inversely. Where Beta is greater than Alpha, SSE term in Eq. (22) 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.

## 5. Results and Discussion

According to the analysis results, the steepest descent algorithm with sum squared error requires some criteria such as early stopping and determining a suitable learning rate and controlling over/lower-fitting depending on the number of parameters. When the algorithm runs for a long time, the over-fitting problem appears. In this case, despite MSE of training data decreases, MSE of test data increases inversely. If the training process is stopped too early, then lower-fitting is inevitable. Besides, increasing iteration number together with the neuron number cannot decrease MSEs of training and test data. In addition, the dimension of the solution spaces becomes more complex when the number of neurons is increased. Hence, the risk of stuck in the local optima increases as well. In order to escape from the local optima, the steepest descent with the momentum can be preferred.

In the second implementation, the steepest descent algorithm with momentum reduces to training time and might overcome to the problem of stuck in the local optima by using the effective combinations of learning rate and momentum. However, training ANNs with this algorithm requires the early stopping approach to decrease overfitting caused by using MSE. In addition, determining the best combination of learning rate and momentum constant requires additional trials as well.

In the conventional usage of Gaussian approach with the fixed hyperparameters, the algorithm performs good-fit to both training and test data where Beta is greater than Alpha. In addition, the algorithm with larger Beta values is enforced to overfitting to training data. However, overfitting to training data might reduce fitting to the test data. Therefore, determining the efficient Alpha and Beta combinations plays important roles to handle above mentioned problems and achieve to estimate more robust models. For this reason, to figure out the efficient combinations of hyperparameters, the algorithm should be run different combinations of these hyperparameters by the trial and error.

## 6. Conclusions

According to the analysis results performed over an artificial data set, there are significant advantages of BNNs based Gaussian approach with the fixed hyperparameters against the traditional ANNs in terms of balancing the model complexity, the learning ability and the estimating more robust models. However, determination of the efficient hyperparameters would cause time consumption for some excessive nonlinear and high dimensional cases. In such cases, Gaussaian approximation with the recursive hyperparameter or full Bayesian approach provides an alternative way to handle related problems accurately. In the future direction, to overcome the above mentioned problems we are planning to improve more robust Bayesian learning approaches and measure their performances

over more complicated artificial and real datasets. In addition, developed approaches will be compared the other artificial intelligent techniques in terms of reliability and efficiency.

## References


[1] W. L. Buntine, A. S. Weigend, Bayesian Back-Propagation, Complex Systems 5(6) (1991), 603–643.

[2] D. J. C. Mackay, A Practical Bayesian Framework for Back Propagation Networks, Neural Computation 4(3) (1992), 448–472.

[3] G. E. Hinton, D. V. Camp, Keeping Neural Networks Simple by Minimizing The Description Length of The Weights, In Proceedings of the Sixth Annual Conference on Computational Learning Theory, (1993), pp. 5-13.

[4] R. M. Neal, 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, (1992).

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

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

[7] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press (reprinted 2010), 1995.

[8] R. M. Neal, Bayesian Learning for Neural Networks, New York, Springer, 1996.

[9] D. Rios Insua, P. Muller, Feed-forward Neural Networks for Nonparametric Regression, Technical Report 98.02., Institute of Statistics and Decision Sciences, Duke University, (1998).

[10] A. D. Marrs, An Application of Reversible-Jump MCMC to Multivariate Spherical Gaussian Mixtures. Advances in Neural Information Processing Systems 10 (1998), 577-583.

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

[12] P. J. Green, Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination, Biometrika, 82 (1995), 711-732.

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

[14] J. F. G. Freitas, Bayesian Methods for Neural Networks, PhD. Thesis, Trinity College University of Cambridge and Cambridge University Engineering Department, UK, 2000.

[15] F. Liang, W.H. Wong, Real-Parameter Evolutionary Monte Carlo with Applications to Bayesian Mixture Models. J. Am. Stat. Assoc. 96 (454) (2001), 653–666.

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

[17] F. Liang, Bayesian Neural Networks for Nonlinear Time Series Forecasting, Statistics and Computing, 15(1), (2005), 13–29.

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

[19] J. Lampinen, A. Vehtari, Bayesian Approach for Neural Networks-Review and Case Studies, Neural Networks, 14(3) (2001), 7-24.

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

[21] T. Marwala, Bayesian Training of Neural Networks using Genetic Programming, Pattern Recognition Letters, 28 (2007), 1452-1458.

[22] D.T. Mirikitani, Recursive Bayesian Recurrent Neural Networks for Time-Series Modeling, IEEE Transactions on Neural Networks, 21 (2) (2010), 262-274.

[23] M. S. Goodrich, Markov Chain Monte Carlo Bayesian Learning for Neural Networks, Selected Papers at MODSIM World 2010 Conference and Expo, NASA/CP-2011-217069/PT1 (2011), 268-290.

[24] J. Martens, I. Sutskever, Learning Recurrent Neural Networks with Hessian-Free Optimization, Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA, (2011).

[25] D. Niu, H. Shi and D. D. Wu. Short-term load forecasting using Bayesian neural networks learned by Hybrid Monte Carlo Algorithm, Applied Soft Computing, 12(6), (2012), 1822–1827.

[26] A L. Beam, A. Motsinger-Reif and J. Doyle (2014). Bayesian Neural Networks for Genetic Association Studies of Complex Disease, arXiv:1404.3989 [q-bio.GN], Cornell University Library.

[27] O. Kocadagli, Hybrid Bayesian Neural Networks with Genetic Algorithms and Fuzzy Membership Functions, PhD. Thesis, Department of Statistics, Mimar Sinan F.A. University, Istanbul, Turkey, 2012.

[28] O. Kocadagli and B. Aşıkgil. Nonlinear Time Series Forecasting with Bayesian Neural Networks. Expert Systems with Applications, 41(15), (2014), 6596-6610.

[29] Kocadagli, O., A Novel Hybrid Learning Algorithm for Full Bayesian Approach of Artificial Neural Networks, Applied Soft Computing, Elsevier, 35, (2015), 52 – 65.

[30] P. Niyogi, F. Girosi, 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, (1994).

[31] S. Geman, E. Bienenstock, R. Doursat, Neural Networks and the Bias/Variance Dilemma. Massachusetts Institute of Technology, 4(1) **(**1992), 1-58.

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

[33] M. P. Perrone, Averaging/Modular Techniques for Neural Networks, In: Arbib, M. A. The Handbook of Brain Theory and Neural Networks, MIT Press, 1995.

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

[35] G. Castellano, A.M. Fanelli, M. Pelillo, An Iterative Pruning Algorithm for Feedforward Neural Networks, Neural Networks, IEEE Transactions on, 8 (1997), 519- 531.

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

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

[38] C. M. Bishop, Pattern Recognition and Machine Learning, Springer Science + Business Media, LLC, 2006.