Neural Networks 83 (2016) 75–85

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Approximate Bayesian MLP regularization for regression in the presence of noise

Jung-Guk Park, Sungho Jo *

School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea

ARTICLE INFO

Article history:
Received 28 December 2015
Received in revised form 6 July 2016
Accepted 18 July 2016
Available online 10 August 2016

Keywords:
Bayesian method
Multilayer perceptron training
Non-smooth regression
Regularization
Weight-decay

ABSTRACT

We present a novel regularization method for a multilayer perceptron (MLP) that learns a regression function in the presence of noise regardless of how smooth the function is. Unlike general MLP regularization methods assuming that a regression function is smooth, the proposed regularization method is also valid when a regression function has discontinuities (non-smoothness). Since a true regression function to be learned is unknown, we examine a training set with our Bayesian approach that identifies non-smooth data, analyzing discontinuities in a regression function. The use of a Bayesian probability distribution identifies the non-smooth data. These identified data is used in a proposed objective function to fit an MLP response to the desired regression function regardless of its smoothness and noise. Experimental simulations show that the MLP with our presented training method yields more accurate fits to non-smooth functions than other MLP training methods. Further, we show that the suggested training methodology can be incorporated with deep learning models.

1. Introduction

A multilayer perceptron (MLP) is a universal approximator (Hornik, Stinchcombe, & White, 1989) that has the great ability to approximate any function. It has been widely used in many applications such as phenomenological simulations (Kucuk, Manohara, Hanagodimath, & Gerward, 2013; Piliougine, Elizondo, Mora-López, & Sidrach-de-Cardona, 2013), biological models (Chamjangali, Mohammadrezaei, Kalantar, & Amin, 2012) as well as deep learning applications (Raiko, Valpola, & LeCun, 2012) and hybrid system identification (Rolla, Bemporadb, & Ljunga, 2004; Yang, Wang, Wu, Lin, & Liu, 2015). In both learning theory and application, training data is generally assumed to contain noise, which implies that the MLP inevitably learns the noise as well. This makes performances of the MLP undesirable when presented with unseen data. Analyzed in machine learning communities, this problem is considered as overfitting; in order to solve this, various approaches have been proposed (Larsen & Hansen, 1994; Ludwig, Nunes, & Araujo, 2014). Especially, weight-decay or regularization methods (Connor, 2015; Foresee & Hagan, 1997; Pinzolas et al., 2006; Sum & Ho, 2009) successfully prevent connectionist models from overfitting. However, these methods assume that a regression function to be learned is smooth and expect that a learned MLP output response is smooth. In other words, this is not appropriate for non-smooth regression functions. Although researches aimed at overcoming the non-smoothness have been reported, for example, experiments on noise-free data (Llanas, Lantarón, & Sáinz, 2008), or noisy regression data (Bowman & Pope, 2008; Esposito, Marinaro, Oricchioa, & Scarpetta, 2000), these authors experimented with low-dimensional or small scale datasets, which is not applicable for generalizing data dimension and size in most cases.

Our work aims to overcome fitting the MLP response to real-world regression functions that contain discontinuities in noisy training data. The proposed regularization method, Approximate Bayesian Regularization, effectively distinguishes between non-smoothness and noise in training data. Then, the MLP more accurately approximates the non-smoothness by avoiding the presence of noise (Fig. 1 shows an example of the non-smooth data). Since estimating noise is very challenging, this paper proposes an alternative estimation through a Bayesian approach.

Our method yields a good fit even to a non-smooth regression function as well as a smooth function. The proposed regularization establishes two steps. The first is to successfully find discontinuities in a regression function by identifying non-smooth data in a noisy training set with probability. Identifying non-smooth data is based on a Bayesian probability distribution in which the evaluation is computationally feasible. The second is to design the MLP objective function with the identified data such that the MLP yields

* Corresponding author.
E-mail addresses: jgparknn@kaist.ac.kr (J.-G. Park), shjo@kaist.ac.kr (S. Jo).

http://dx.doi.org/10.1016/j.neunet.2016.07.010
0893-6080/

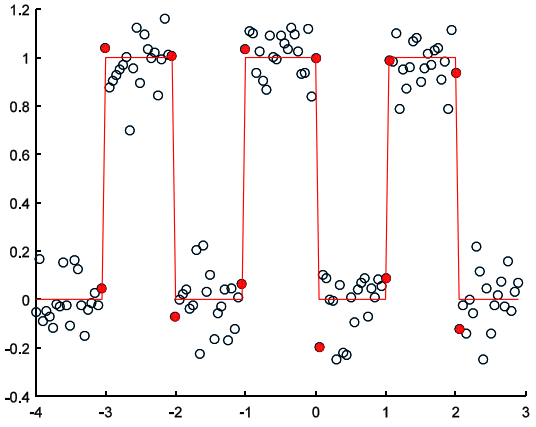

Fig. 1. Example of non-smooth data in the presence of noise. Training data and non-smooth data are denoted by black circles and red dots, respectively. The true regression function is denoted by the red line. Non-smooth data enable us to find the discontinuities in the regression function. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the desired fit to a regression function regardless of its smoothness and noise. Compared to general regularization methods which use the penalized objective function $E = E_{data} + \lambda_1 E_{pen1}$ where λ_1 is a regularization coefficient, and E_{data} and E_{pen1} denote a training error and a smooth penalty error, respectively, we suggest the novel MLP objective function which can be interpreted as $E = E_{data} + \lambda_1 E_{pen1} + \lambda_2 E_{pen2}$ where E_{pen2} and λ_2 denote a non-smooth penalty error and a non-smooth regularization coefficient, respectively.

To evaluate the performance of the proposed method, we compare its results with the results obtained by using Gauss–Newton Bayes regularization (GNBR), which is used in Oliver, Fuster-Garcia, Cabello, Tortajada, and Rafecas (2013), the Levenberg–Marquardt (LM) training (Hagan, Demuth, Beale, & Jess, 2014), and the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) training (Apostolopoulou, 2009). The LM training has been widely used for training an MLP because of its ability of optimization (Wilamowski & Yu, 2010); GNBR has a great regularization ability in the presence of noise, and has been employed in recent applications (Chamjangali et al., 2012). The experiments in this study are conducted with time series predictions, synthetic function approximations, and real-world regression datasets. Moreover, we show that our regularization method can be applied to deep learning models. The remainder of this paper is organized as follows: Section 2 details the proposed learning method, Section 3 presents our experimental results. Finally, Section 4 concludes this paper.

2. Probabilistic analysis of discontinuities in regression function

The proposed method first analyzes discontinuities in a regression function through evaluating a distribution of the change in a regression function[1] by using training set to identify non-smooth data.[2] We find the more striking changes in a function (i.e., those that largely deviate from the other changes) using the distribution. To identify the non-smooth data, we model a probability density function of the change in a regression function, which provides an analytic form. The next section describes identifying the non-smooth data that comprise its dataset $\hat{\mathbf{D}}$, and presents how our approach uses $\hat{\mathbf{D}}$ for fitting the MLP response to non-smooth data through a proposed objective function and a Bayesian framework.

2.1. Identifying non-smooth data in training set

We first assume that response data in a training set contains additive Gaussian noise:

$$t^{(i)} = f^{(i)} + e^{(i)} \tag{1}$$

where $f^{(i)} = f(x^{(i)})$ could be either a linear or nonlinear true regression function. $t^{(i)}$ represents the noisy output data corresponding to an input $x^{(i)} \in \mathbf{R}$ in a training dataset.[3] In the following statement in this subsection, note that $t^{(i)}$ is scaled between zero and one. i denotes the training data index, $1 \le i \le N$, and N is the number of training data. The value $e^{(i)}$ denotes Gaussian noise taken from a random variable[4] $\mathrm{e}^{(i)}$ that is identically and independently drawn from a normal distribution $\mathcal{N}(0, \sigma^2)$ with the unknown parameter σ^2.

As the regression function f is unknown, calculating the exact change in a regression function is infeasible. Instead, we propose a novel method to approximate the change in a regression function. For $1 \le i, j \le N$, let $f^{(i)}$ and $f^{(j)}$ denote a pair of regression function data with input data $x^{(i)}$ and $x^{(j)}$, respectively. Here, $x^{(i)}$ is the interested input data for identifying whether the around of $f^{(i)}$ is non-smooth or not. $x^{(j)}$ is the nearest neighbor point of $x^{(i)}$ among the training data such that $j \neq i$. Since the change in a regression function between the two points $f^{(i)}$ and $f^{(j)}$ is proportional to the finite difference approximation $df/dx|_{x=x^{(i)}} \approx \left(f(x^{(i)} + h) - f(x^{(i)})\right)/h$, the change in a regression function has information about the slope of the true regression function around $x^{(i)}$. The nearest neighbor $x^{(j)}$ is chosen in the same manner as choosing h being as minimum as possible.

Observing these principles, we model a probability distribution and identify non-smooth data. Let $\Delta\mathrm{f}^{(i)}$ be a random variable that takes the value, $\Delta\mathrm{f}^{(i)} = \Delta f^{(i)}$, as follows

$$\Delta f^{(i)} \triangleq \begin{cases} f^{(i)} - f^{(j)} & \text{if } x^{(i)} > x^{(j)} \\ f^{(j)} - f^{(i)} & \text{otherwise} \end{cases} \tag{2}$$

in which the first case of the definition satisfies the backward difference and the second case does the forward difference to ensure the valid approximation without knowledge of h. Further, we assume that $\Delta f^{(i)}$ is the population data and that the random variable $\Delta\mathrm{f}^{(i)}$ is independently and identically drawn from the normal distribution. Then, for $1 \le i \le N$,

$$\Delta\mathrm{f}^{(i)} \sim \mathcal{N}(\Delta\bar{f}, \sigma^2_{\Delta f}) \tag{3}$$

where the distribution has the mean $\Delta\bar{f} = \left(\sum_{i=1}^{N} \Delta f^{(i)}\right)/N$, and variance $\sigma^2_{\Delta f} = \left(\sum_{i=1}^{N} (\Delta f^{(i)} - \Delta\bar{f})^2\right)/N$. With our model in (1), we consider the noisy change in a regression function random variable $\Delta\mathrm{t}^{(i)}$ that takes the value

$$\begin{aligned} \Delta t^{(i)} &= t^{(i)} - t^{(j)} = (f^{(i)} + e^{(i)}) - (f^{(j)} + e^{(j)}) \\ &= \Delta f^{(i)} + e^{(i)} - e^{(j)} \end{aligned} \tag{4}$$

which is a linear combination of the three random variables: $\Delta\mathrm{f}^{(i)}$ in Eq. (3), $\mathrm{e}^{(i)} \sim \mathcal{N}(0, \sigma^2)$, and $\mathrm{e}^{(j)} \sim \mathcal{N}(0, \sigma^2)$. Therefore, its distribution is given as

$$\Delta\mathrm{t}^{(i)} \sim \mathcal{N}(\Delta\bar{t}, \sigma^2_{\Delta t}) \tag{5}$$

[1] In this paper, the change in a regression function is defined as $\Delta f = f(x+h) - f(x)$ where f is a true regression function.

[2] A regression input x^* and its output data pair $(x^*, f(x^*))$ are called non-smooth data if the function slope around $f(x^*)$ is relatively steeper than others.

[3] We first describe our one-dimensional model and explain the multivariate model later.

[4] This paper writes a random variable in roman type and its value in italic.

where $\Delta\bar{t} = \Delta\bar{f}$ and $\sigma^2_{\Delta t} = \sigma^2_{\Delta f} + 2\sigma^2$. Because the parameters $\Delta\bar{f}$, σ^2 and $\sigma^2_{\Delta f}$ are unknown, we should infer $\Delta\bar{t}$ and $\sigma^2_{\Delta t}$ using the training data. By applying the maximum likelihood (ML) estimation that asymptotically achieves the Cramér–Rao bound, the parameters in (5) are estimated as $\hat{\sigma}^2_{\Delta t} = \left(\sum_{i=1}^{N}(\Delta t^{(i)} - \Delta\hat{\bar{t}})^2\right)/N$ and $\Delta\hat{\bar{t}} = \left(\sum_{i=1}^{N}\Delta t^{(i)}\right)/N$. However, the mean squared error (MSE) of the variance estimator $\hat{\sigma}^2_{\Delta t}$ is given as

$$\mathbf{E}\left[\left(\hat{\sigma}^2_{\Delta t} - \sigma^2_{\Delta t}\right)^2\right] = \mathbf{E}\left[\left(\hat{\sigma}^2_{\Delta t} - E[\hat{\sigma}^2_{\Delta t}]\right)^2\right] + \left(\mathbf{E}[\hat{\sigma}^2_{\Delta t}] - \sigma^2_{\Delta t}\right)^2 = \frac{2(N-1)(\sigma^2_{\Delta t})^2}{N^2} + \left(\frac{(N-1)\sigma^2_{\Delta t}}{N} - \sigma^2_{\Delta t}\right)^2 \quad (6)$$

where $\mathbf{E}[.]$ denotes the expectation, which is proportional to the unknown and intrinsic noise variance σ^2 (recall that $\sigma^2_{\Delta t} = \sigma^2_{\Delta f} + 2\sigma^2$). Therefore, the ML estimator is not reliable for estimating $\sigma^2_{\Delta f}$.

As an alternative approach to properly deal with the problem of the effect of the noise variance σ^2, we model the probability distribution with a Bayes sense. To derive the probability distribution, let the mean parameter $\mu = \Delta\bar{t}$, the precision $\lambda = (\sigma^2_{\Delta t})^{-1}$, and the likelihood be

$$p(\Delta\boldsymbol{t}|u,\lambda) = \left(\frac{\sqrt{\lambda}}{\sqrt{2\pi}}\right)^N \exp\left[-\frac{\lambda}{2}\sum_{i=1}^{N}\left(\Delta t^{(i)} - \mu\right)\right] \quad (7)$$

where $\Delta\boldsymbol{t} = \{\Delta t^{(1)}, \Delta t^{(2)}, \ldots \Delta t^{(N)}\}$. Further, in terms of conjugate priors, the prior is given as

$$\begin{aligned} p(u,\lambda) &= p(\mu|\lambda)p(\lambda) \\ &= \mathcal{N}(u|u_0, (k_0\lambda)^{-1})Ga(\lambda|\alpha_0,\beta_0) \\ &= \frac{\sqrt{k_0}\beta_0^{a_0}}{\sqrt{2\pi}\,\Gamma(a_0)}\lambda^{a_0-\frac{1}{2}}\exp\left(-\frac{\lambda}{2}[k_0(\mu-\mu_0)^2 + 2\beta_0]\right) \\ &= \mathcal{N}G(u,\lambda|u_0,k_0,\alpha_0,\beta_0) \end{aligned} \quad (8)$$

in which the distribution for the mean parameter $p(\mu|\lambda)$ is assumed to be the normal distribution $\mathcal{N}(\mu|\mu_0, (k_0\lambda)^{-1})$. The hyperprior for precision distribution $p(\lambda)$ is the Gamma distribution with a rate parameter, $Ga(\lambda|\alpha_0,\beta_0)$. $\Gamma(x) = \int_0^\infty s^{x-1}e^{-s}ds$ is the Gamma function. α_0 and β_0 are hyperparameters of the precision. u_0 and k_0 are those of the mean. As $p(u,\lambda)$ is the normal-Gamma distribution, the posterior distribution is evaluated using (8) as

$$\begin{aligned} p(u,\lambda|\Delta\boldsymbol{t}) &= \frac{p(\Delta\boldsymbol{t}|u,\lambda)p(u,\lambda)}{\iint p(\Delta\boldsymbol{t}|u,\lambda)p(u,\lambda)d\mu d\lambda} \\ &\propto \lambda^{\frac{1}{2}(\alpha_0-1)}\exp\left[-\frac{1}{2}k_0\lambda(\mu-\mu_0)^2 - \beta_0\lambda\right] \\ &\quad \times \lambda^{\frac{1}{2}}\exp\left[-\frac{1}{2}\lambda\sum_{i=1}^{N}(\Delta t^{(i)}-\mu)^2\right] \\ &\propto \mathcal{N}\left(u\left|\frac{k_0\mu_0 + (N\Delta\bar{t})}{(k_0+N)}, \frac{1}{\lambda(k_0+N)}\right.\right)Ga \\ &\quad \times \left(\lambda|\alpha_0 + N/2, \beta_0 + \sum_{i=1}^{N}(t^{(i)} - \Delta\bar{t})^2/2\right. \\ &\quad \left. + [k_0N(\Delta\bar{t}-\mu_0)^2]/(2k_0+2N)\right). \end{aligned} \quad (9)$$

Thus, substituting $\beta_n = \beta_0 + \sum_{i=1}^{N}(\Delta t^{(i)} - \Delta\bar{t})^2/2 + [k_0N(\Delta\bar{t} - \mu_0)^2]/(2k_0+2N)$, $k_n = k_0 + N$, $\alpha_n = \alpha_0 + N/2$, and $u_n = [k_0\mu_0 + (N\Delta\bar{t})]/(k_0+N)$ into (9), the posterior distribution is a closed-form expression given as

$$p(u,\lambda|\Delta\boldsymbol{t}) = \mathcal{N}G(u,\lambda|u_n,k_n,\alpha_n,\beta_n). \quad (10)$$

To predict whether a new data Δt is non-smooth or not, we evaluate the posterior predictive distribution

$$p(\Delta t|\Delta\boldsymbol{t}) = p_{2a_n}\left(\Delta t|\mu_n, \frac{\beta_n(k_n+1)}{\alpha_n k_n}\right) \quad (11)$$

where the derivation comes from $p(\Delta t|\Delta\boldsymbol{t}) = p(\Delta t, \Delta\boldsymbol{t})/p(\Delta\boldsymbol{t})$, which is T-distribution with $2a_n$ degrees of freedom. The derivations of (10) and (11) are detailed in Murphy (2007). We decompose the target dataset into two different sets in order to infer the posterior predictive distribution and a singleton (a unit set) to be predicted. We make a set $\Delta\boldsymbol{t}_{i-} = \Delta\boldsymbol{t} - \{\Delta t^{(i)}\}$ for evaluating the distribution in (11) with $\Delta\bar{t} = \left(\sum_{i=1}^{N}\Delta t^{(i)}\right)/N$. The singleton $\{\Delta t^{(i)}\}$ is predicted either to be non-smooth or not. Finally, training data $(x^{(i)}, t^{(i)})$ and $(x^{(j)}, t^{(j)})$ are contained in the non-smooth dataset $\hat{\mathbf{D}}$ if $\Delta t^{(i)}$ is not in the interval $[-k, k]$ where k is given by

$$\int_{-k}^{k} p_{2a_n}\left(\Delta t|\mu_n, \frac{\beta_n(k_n+1)}{\alpha_n k_n}\right) = 0.95. \quad (12)$$

In a multivariate input case, $\mathbf{x} = (x_1, x_2, x_3 \ldots, x_M)$, we can approximate the partial derivative

$$\left.\frac{\partial t}{\partial x_d}\right|_{x_d^{(i)}} \approx \frac{(t^{(i)} - t^{(j)})}{\sqrt{\|x_d^{(i)} - x_d^{(j)}\|}} \approx (t^{(i)} - t^{(j)}) = \Delta t^{(i)} \quad (13)$$

where i is our interest to identify the non-smooth data and $j = \arg\min_q \|\mathbf{x}^{(i)} - \mathbf{x}^{(q)}\|$ ($\|.\|$ denotes Euclidean norm). The $\Delta t^{(i)}$ in the partial derivative approximation also is the same in (2).

2.2. Non-smooth dataset for training

This section describes how the non-smooth dataset $\hat{\mathbf{D}}$ derived in the Section 2.1 is used to train an MLP. The proposed MLP objective function with $\hat{\mathbf{D}}$ is based on a Bayesian assumption, similar to the model in MacKay (1992),

$$p(t|\mathbf{x},\mathbf{w},\beta) = \frac{1}{Z_\beta}\exp\left[-\beta\,(t - y(\mathbf{x};\mathbf{w}))^2\right] \quad (14)$$

where $\mathbf{w}$ is an MLP weight vector, $Z_\beta = \int \exp\left[-\beta(t - y(\mathbf{x};\mathbf{w}))^2\right] dt$, and $y(\mathbf{x};\mathbf{w})$ denotes an MLP scalar output. $\mathbf{x} = (x_1, \ldots x_M)$ and t denote an input vector and the corresponding target variable, respectively. The MLP output is given as

$$y(\mathbf{x};\mathbf{w}) = \sum_{j=1}^{H} w_j^{(2)} g\left(\sum_{i=1}^{M}\left(w_{i,j}^{(1)}x_i\right) + w_{0,j}^{(1)}\right) + w_0^{(2)} \quad (15)$$

where H and M are the number of the hidden nodes and the input nodes of the MLP, respectively. $g(x)$ is the sigmoid function $g(x) = 1/(1+\exp(-x))$. x_i is the ith element of an input data vector, $w_{i,j}^{(1)}$ denotes the weight connecting between ith input data and jth hidden node, and $w_j^{(2)}$ denotes the weight connected to the output node from the jth hidden node. $w_{0,j}^{(1)}$ and $w_0^{(2)}$ are bias terms in the input-hidden and hidden-output layer, respectively. Therefore, the weight vector is encoded as $\mathbf{w} = (w_0^{(1)}, w_{1,1}^{(1)}, w_{1,2}^{(1)} \ldots w_{1,H,}^{(1)} \ldots w_{M,1}^{(1)}, w_{M,2}^{(1)} \ldots w_{M,H,}^{(1)} w_0^{(2)}, w_1^{(2)}, \ldots, w_H^{(2)})$.

The training data are assumed to be independent of each others, and so their joint probability is given as

$$p(\mathbf{D}|\mathbf{w},\beta) = \prod_{i=1}^{N} p(t^{(i)}|\mathbf{x}^{(i)},\mathbf{w},\beta) \quad (16)$$

where $\mathbf{D}$ denotes the training set $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^{N}$. The prior distribution of $\mathbf{w}$ is given as

$$p(\mathbf{w}|\alpha) = \frac{1}{Z_\alpha} \exp\left[-\alpha\left(\sum_{j=0}^{H}\sum_{i=0}^{M} w_{i,j}^{(1)} + \sum_{j=0}^{H} w_j^{(2)}\right)\right] \tag{17}$$

where $Z_\alpha = \int \exp\left[-\alpha\left(\sum_{j=0}^{H}\sum_{i=0}^{M} w_{i,j}^{(1)} + \sum_{j=0}^{H} w_j^{(2)}\right)\right] d\mathbf{w}$.

The proposed objective function is designed with these distributions and is represented as

$$E = \alpha E_W + \beta E_{\mathbf{D}} + \zeta E_{\hat{\mathbf{D}}} \tag{18}$$

in which $E_{\mathbf{D}} = \sum_{i=1}^{N}(t^{(i)} - y(\mathbf{x}^{(i)}; \mathbf{w}))^2$, $E_W = \sum_{j=0}^{H}\sum_{i=0}^{M} w_{i,j}^{(1)} + \sum_{j=0}^{H} w_j^{(2)}$, and $E_{\hat{\mathbf{D}}} = \sum_{i=1}^{N'}(t^{(i)} - y(\mathbf{x}^{(i)}; \mathbf{w}))^2$. Here, $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^{N'} = \hat{\mathbf{D}}$ and N' is the size of $\hat{\mathbf{D}}$. We suggest a probabilistic framework that evaluates a posterior distribution of the weight

$$p(\mathbf{w}|\mathbf{D}, \hat{\mathbf{D}}) = \int p(\mathbf{w}, \alpha, \beta, \zeta|\mathbf{D}, \hat{\mathbf{D}}) d\alpha d\beta d\zeta \tag{19}$$

where the distribution is assumed to be peaked around α_{MP}, β_{MP}, and ζ_{MP}. Thus, $p(\mathbf{w}|\mathbf{D}, \hat{\mathbf{D}})$ is approximated to $p(\mathbf{w}|\alpha_{MP}, \beta_{MP}, \zeta_{MP}, \mathbf{D}, \hat{\mathbf{D}}) \int p(\alpha, \beta, \zeta|\mathbf{D}, \hat{\mathbf{D}}) d\alpha d\beta d\zeta$. By using Bayes' rule, the distribution in the integral is evaluated as

$$p(\alpha, \beta, \zeta|\mathbf{D}, \hat{\mathbf{D}}) \propto p(\mathbf{D}, \hat{\mathbf{D}}|\alpha, \beta, \zeta) p(\alpha, \beta, \zeta) \tag{20}$$

with a flat prior. Taking the integral over the weights of neural networks $\mathbf{w}$ on the proposed distribution, we evaluate the likelihood in (20),

$$p(\mathbf{D}, \hat{\mathbf{D}}|\alpha, \beta, \zeta) = \int p(\mathbf{D}|w, \beta) p(\hat{\mathbf{D}}|\mathbf{w}, \zeta) p(\mathbf{w}|\alpha) d\mathbf{w} \tag{21}$$

where $p(\hat{\mathbf{D}}|\mathbf{w}, \zeta) = Z_\zeta^{-1} \exp\left[-\zeta \sum_{i=1}^{N'}(t^{(i)} - y(\mathbf{x}^{(i)}; \mathbf{w}))^2\right]$, Z_ζ is a normalization factor. $p(\mathbf{D}|w, \beta)$ and $p(\mathbf{w}|\alpha)$ are as in (16) and (17), respectively. The distribution in (21) is approximated by Laplace's method as

$$\begin{aligned} p(\mathbf{D}, \hat{\mathbf{D}}|\alpha, \beta, \zeta) &\approx (2\pi)^{W/2} \det(\mathbf{A})^{-1/2} \exp\left(-E^{MP}\right) \\ &= \pi^{W/2} \det(\zeta \mathbf{H}_{\hat{\mathbf{D}}} + \beta \mathbf{H}_{\mathbf{D}} + \alpha \mathbf{I})^{-1/2} \\ &\quad \times \exp\left(-E^{MP}\right) \end{aligned} \tag{22}$$

in which $E^{MP} = \alpha E_W^{MP} + \beta E_{\mathbf{D}}^{MP} + \zeta E_{\hat{\mathbf{D}}}^{MP}$. E_W^{MP}, $E_{\mathbf{D}}^{MP}$, and $E_{\hat{\mathbf{D}}}^{MP}$ are computed at the updated MLP weight vector $\mathbf{w}^{MP}$ with Levenberg–Marquardt method (the calculation of $\mathbf{w}^{MP}$ is presented in Appendix A). $\mathbf{A} = 2(\zeta \mathbf{H}_{\hat{\mathbf{D}}} + \beta \mathbf{H}_{\mathbf{D}} + \alpha \mathbf{I})$ where $\mathbf{H}_{\mathbf{D}} \approx \mathbf{J}_{\mathbf{D}}^{\mathrm{T}} \mathbf{J}_{\mathbf{D}}$. $\mathbf{J}_{\mathbf{D}}$ is the Jacobian matrix of $E_{\mathbf{D}}$ whose elements are $\mathbf{J}_{\mathbf{D}_{(i,j)}} = \partial y(\mathbf{x}^{(i\in G)}; \mathbf{w})/\partial w_j$ where $G = \{i|\mathbf{x}^{(i)} \in \mathbf{D}\}$ and w_j is one of all W MLP weights, $1 \le j \le W$. $\mathbf{H}_{\hat{\mathbf{D}}} \approx \mathbf{J}_{\hat{\mathbf{D}}}^{\mathrm{T}} \mathbf{J}_{\hat{\mathbf{D}}}$ and $\mathbf{J}_{\hat{\mathbf{D}}}$ is the Jacobian matrix of $E_{\hat{\mathbf{D}}}$. Its elements are $\mathbf{J}_{\hat{\mathbf{D}}_{(i,j)}} = \partial y(\mathbf{x}^{(i\in L)}; \mathbf{w})/\partial w_j$ where $L = \{i|\mathbf{x}^{(i)} \in \hat{\mathbf{D}}\}$. $\mathbf{I}$ and $\det(\mathbf{A})$ denote an identity matrix and the determinant of the matrix $\mathbf{A}$, respectively. We take the partial derivative of the approximated distribution in (22) to update α in (18) as follows

$$\alpha = \left(W - \sum_{i=1}^{W} \frac{\alpha}{\lambda_i^{\mathbf{D}} + \lambda_i^{\hat{\mathbf{D}}} + \alpha}\right) \Big/ 2E_W^{MP} \tag{23}$$

where $\lambda_i^{\mathbf{D}}$ and $\lambda_i^{\hat{\mathbf{D}}}$ denote eigenvalues of $\mathbf{H}_{\mathbf{D}}$ and $\mathbf{H}_{\hat{\mathbf{D}}}$, respectively. β is updated by according to

$$\beta = \left(N - \sum_{i=1}^{W} \frac{\lambda_i^{\mathbf{D}}}{\lambda_i^{\mathbf{D}} + \lambda_i^{\hat{\mathbf{D}}} + \alpha}\right) \Big/ 2E_{\mathbf{D}}^{MP}. \tag{24}$$

Table 1
Pseudo code for ABR for training MLP.

Pseudo code for ABR for training MLP.	
notations	N : size of training data, M : dimension of input data. i : index of training data, $1 \le i \le N$. j : index of the nearest neighbor of i th training input data (i.e., $j = \arg\min_q \|\mathbf{x}^{(i)} - \mathbf{x}^{(q)}\|$) with $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots x_M^{(i)})$. $t^{(i)}$: scalar-valued training output data.
Step 1.	$\Delta t = \varnothing$. For $1 \le i \le N$ For $1 \le d \le M$ If $x_d^{(i)} > x_d^{(j)}$ compute $\Delta t^{(i)} = t^{(i)} - t^{(j)}$. otherwise compute $\Delta t^{(i)} = t^{(j)} - t^{(i)}$. $\Delta t = \Delta t \cup \{\Delta t^{(i)}\}$.
Step 2.	For $1 \le i \le N$ Make two sets, $\{\Delta t^{(i)}\}$ and $\Delta t_{i-} \triangleq \Delta t - \{\Delta t^{(i)}\}$. Evaluate the distribution in (11) using Δt_{i-}. Determine whether $(\mathbf{x}^{(i)}, \Delta t^{(i)})$ belongs to $\hat{\mathbf{D}}$ using the criterion in (12) with $\{\Delta t^{(i)}\}$.
Step 3.	Initialize $\alpha = 0$, $\beta = 1$, and $\zeta = 0$ in (18). Initialize the MLP weights.
Step 4.	Update the MLP weights using the LM rule for the objective function in (18) (the LM updating rule is described in appendix).
Step 5.	Check negative eigenvalues of $\mathbf{H}_{\mathbf{D}}$ and $\mathbf{H}_{\hat{\mathbf{D}}}$ in (22) and set them to zeros. Then, update α, β, and ζ according to (23)-(25) and return to *Step 4* until the LM update rule terminates.

The weight parameter ζ with respect to $E_{\hat{\mathbf{D}}}$ is updated as

$$\zeta = \left(N' - \sum_{i=1}^{W} \frac{\lambda_i^{\hat{\mathbf{D}}}}{\lambda_i^{\mathbf{D}} + \lambda_i^{\hat{\mathbf{D}}} + \alpha}\right) \Big/ 2E_{\hat{\mathbf{D}}}^{MP}. \tag{25}$$

In (23)–(25), the eigenvalues can be negative since $\mathbf{H}_{\mathbf{D}}$ and $\mathbf{H}_{\hat{\mathbf{D}}}$ are not computed at the minimum of $E_{\mathbf{D}}$ and $E_{\hat{\mathbf{D}}}$. Thus, an Approximate Bayesian Regularization (ABR) approximates the solution by setting the negative eigenvalues to zero. The ABR pseudo-code is summarized in Table 1. The convergence of the training error with the proposed objective function is proved in Appendix A.

3. Experimental results

The experimental simulations consist of time series predictions, synthetic datasets, and real-world regression problems. The proposed method is compared with the other methods, GNBR used in Oliver et al. (2013), LM in Hagan, Demuth, Beale, and Jess (2014), and L-BFGS[5] employed in Apostolopoulou (2009). The initial MLP weights are randomly selected from the range $[-1, 1]$ and each method is evaluated five times with given training data and the number of the MLP hidden nodes. We provide the same initial weights for each training method. The MLP architecture given in (15) is trained with a maximum of 500 epochs. We conduct all experiments on Matlab 2014. We assume that there is no evidence

[5] We implement the L-BFGS training by using the code available at https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html.

for the hyperparameters in (8). Therefore, the prior distribution of the hyperparameter u_0 is zero. α_0 is set to be small in order to ensure that the prior precision is very vague. We also set k_0 to a very small value, 10^{-5}, which sufficiently supports the large uncertainty in u_0. Setting β_0 is suggested as

$$\beta_0 = v_1 \exp\left[-\frac{v_0}{N}\sum_{i=1}^{N}\left(\Delta t^{(i)}\right)^2\right] \tag{26}$$

which is proportional to the degree of the smoothness in training output data. We empirically found that $v_1 = 10^4$ and $v_0 = 170$ (note that only synthetic function1 and synthetic function3 in Section 3.2 were used to choose the value of v_0, v_1. Also, the hyperparameter β_0 could be tuned by either an expert knowledge or nested cross-validation).

3.1. Time series prediction

The regression functions used in most hybrid system identification problems (Lauera, Blochb, & Vidalc, 2011) are discontinuous. Thus, some of these problems provide a good example of discontinuous (or non-smooth) regression function. Note that our experiments focus on time series prediction rather than on hybrid system identification (i.e., finding submodels in a hybrid system). The functions used in this experiment are used in the earlier works (Paoletti, Roll, Garulli, & Vicino, 2010; Yang et al., 2015). Time series functions in this section either contain a small amount of noise or are entirely noise-free. These input–output data are modeled by

$$y(k) = F(y(k-1), y(k-2), \ldots, u(k), u(k-1), \ldots) + \epsilon(k) \tag{27}$$

where F denotes an MLP, k and $\epsilon(k)$ denote the time step and the noise term, respectively. y and u are time series data and extra input data, respectively. Time series prediction function1 (TS-Func1) is obtained by

$$\begin{aligned} y(k) &= 0.8y(k-1) + 0.4u(k-1) - 0.1 \\ &\quad + \max\{-0.3y(k-1) + 0.6u(k-1) + 0.3, 0\}. \end{aligned} \tag{28}$$

Time series prediction function2 (TS-Func2) is modeled as

$$\mathbf{w}(k+1) = \begin{cases} \begin{bmatrix} 0 & \sin(\mathbf{w}(k)_1) \\ 0 & \lambda_1 \end{bmatrix} w(k), \quad y(k) = [1\,0] \times 10 \\ \quad \times \sin(w(k)) \quad \text{if } \mathbf{h}^T\mathbf{w}(k) + v < 0 \\ \begin{bmatrix} 0 & \sin(\mathbf{w}(k)_1) \\ \lambda_2 \dfrac{h1}{h2} & \lambda_2 - \dfrac{h_1}{h_2} \end{bmatrix} \mathbf{w}(k) \\ \quad + \begin{bmatrix} a * e(k) \\ -0.1 + a * e(k) \end{bmatrix}, \\ \quad y(k) = [\kappa_1 - 10] \times 10\cos(\mathbf{w}(k)) \quad \text{otherwise.} \end{cases} \tag{29}$$

TS-Func1 and TS-Func2 are obtained from Eqs. (22) and (29) in the article by Yang et al. (2015) in which details of the variables are also explained. Other time series prediction functions are TS-Func3

$$y(k) = \begin{cases} 0.8y(k-1) - 0.64y(k-2) - 0.4\sqrt{3}u(k-2) \\ \quad \text{if } y(k-1) \geq 0,\ y(k-2) \geq 0 \\ 0.64y(k-2) + 0.4\sqrt{3}u(k-2) \\ \quad \text{if } y(k-1) < 0,\ y(k-2) \geq 0 \\ 0.8y(k-1) - 0.64y(k-2) + 0.4\sqrt{3}u(k-2) \\ \quad \text{if } y(k-1) < 0,\ y(k-2) < 0 \\ 0.64y(k-2) - 0.4\sqrt{3}u(k-2) \\ \quad \text{if } y(k-1) \geq 0,\ y(k-2) < 0. \end{cases} \tag{30}$$

TS-Func4

$$\mathbf{x}(k+1) = \begin{cases} \begin{bmatrix} 0 & 1 \\ 0 & \gamma_1 \end{bmatrix} \mathbf{x}(k), \quad y(k) = [1\,0]\mathbf{x}(k) \\ \quad \text{if } \mathbf{h}^T\mathbf{x}(k) + w < 0 \\ \begin{bmatrix} 0 & 1 \\ \gamma_2 \dfrac{h_1}{h_2} & \gamma_2 - \dfrac{h_1}{h_2} \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} \\ \quad y(k) = [\gamma_1 - 1]\mathbf{x}(k) \quad \text{otherwise,} \end{cases} \tag{31}$$

and TS-Func5

$$y(k) = \begin{cases} 0.4y(k-1) \quad \text{if } 0.72y(k-1) + 1 < 0, \\ \quad 2y(k-1) + y(k-2) + 1 < 0 \\ 0 \quad \text{if } 0.72y(k-1) + 1 \geq 0,\ 1.8y(k-1) + 1 < 0, \\ \quad 2y(k-1) + y(k-2) + 1 < 0 \\ -0.1 \quad \text{if } y(k-1) = 0,\ y(k-2) - \dfrac{1}{9} \leq 0 \\ -0.5y(k-1) - 0.1 \\ \quad \text{if } 2y(k-1) + y(k-2) + 0.2 = 0. \end{cases} \tag{32}$$

Details of the variables are presented in Eqs. (44)–(46) in Paoletti et al. (2010). All of these datasets contain non-smooth data. Fig. 2 shows plots of the regression functions in TS-Func1 and TS-Func2. It also presents a comparison of the performance of LM, GNBR, L-BFGS, and ABR with 10-fold cross-validation. The training and validation errors are evaluated by the mean square error (MSE), $\sum_{i=1}^{N}(t^{(i)} - y(\mathbf{x}^{(i)}; \mathbf{w}^{MP}))^2/N$, where y and $\mathbf{w}^{MP}$ denote the MLP output and the trained MLP weight vector, respectively.

Fig. 3 shows plots of the regression functions in TS-Func3, TS-Func4, and TS-Func5. It compares the performance of LM, L-BFGS, GNBR, and ABR with 10-fold cross-validation. Since TS-Func3 includes four-dimensional data, we use principal components to plot the regression surface. In Figs. 2 and 3, GNBR produces underfitted MLP responses of the regression functions because GNBR yields a smooth output of the MLP. The performance of L-BFGS is not good to optimize the training error as well as the test error. ABR yields a best fit to each dataset. In cases with a small amount of non-smooth data (e.g., in Figs. 2(a) and 3(a) and (b)), the performance of LM is similar to that of ABR.

3.2. Synthetic datasets

Synthetic datasets are used to evaluate the regularization process in the MLP architecture. The first simulation denoted by SF-1 (synthetic function1) considers a regression function given by

$$f_1(x) = 0.5 + 0.4\sin(x) \tag{33}$$

for which the test dataset is $\{(x^{(i)}, f_1(x^{(i)}))\}_{i=1}^{N}$, $N = 100$, and $x^{(i)}$ is randomly queried within $[-3, 3)$. The training set is $\{(x^{(i)}, t^{(t)})\}_{i=1}^{N}$ where $t^{(i)} = f_1(x^{(i)}) + e^{(i)}$ and the additive random noise $e^{(i)}$ is drawn from the normal distribution $\mathcal{N}(0, 0.2^2)$. Fig. 4(a) shows the test and training set for SF-1. The second simulation denoted by SF-2 (synthetic function2) is

$$f_2(x) = \sin(x)/x \tag{34}$$

in which case the test dataset is $\{(x^{(i)}, f_2(x^{(i)}))\}_{i=1}^{N}$, $N = 50$, $x^{(i)}$ is randomly queried within $[-10, 10]$. The training set $\{(x^{(i)}, t^{(i)})\}_{i=1}^{N}$ is obtained where $t^{(i)} = f_2(x^{(i)}) + e^{(i)}$ with $e^{(i)} \sim \mathcal{N}(0, 0.02^2)$. Fig. 4(b) illustrates the test and training set of SF-2. The third simulation denoted by SF-3 (synthetic function3) is the step-function

$$f_3(x) = \begin{cases} 2 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases} \tag{35}$$

where the test dataset is generated from $x^{(i)} \in [-50, 50]$, $1 \leq i \leq 100$. The training set is acquired by using the test data with the

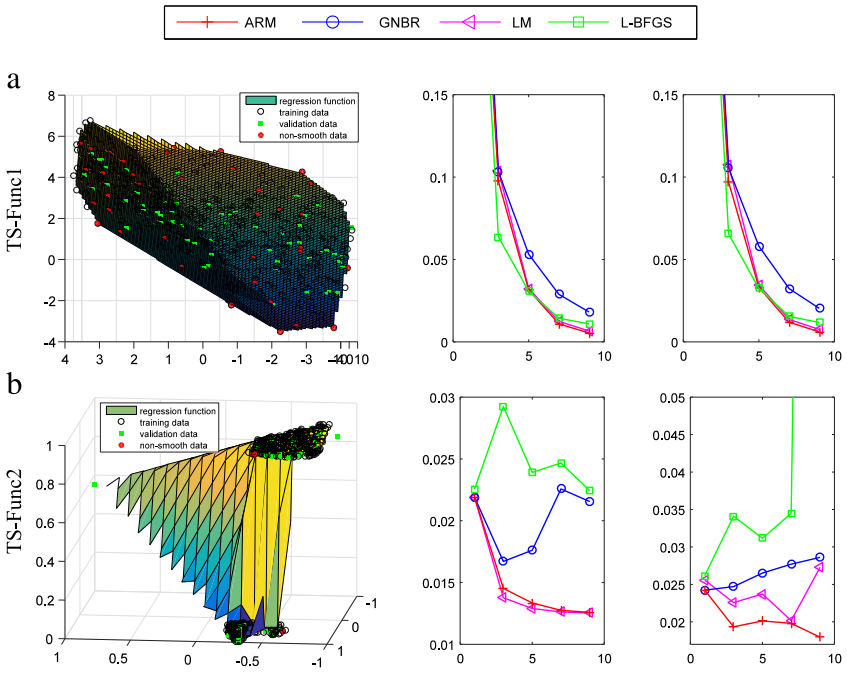

Fig. 2. Regression data for time-series functions. (a) TS-Func 1. (b) TS-Func 2. Non-smooth data identified by ABR are denoted by dots. The second and third columns present the average training mean squared error (MSE) and cross-validated MSE, respectively.

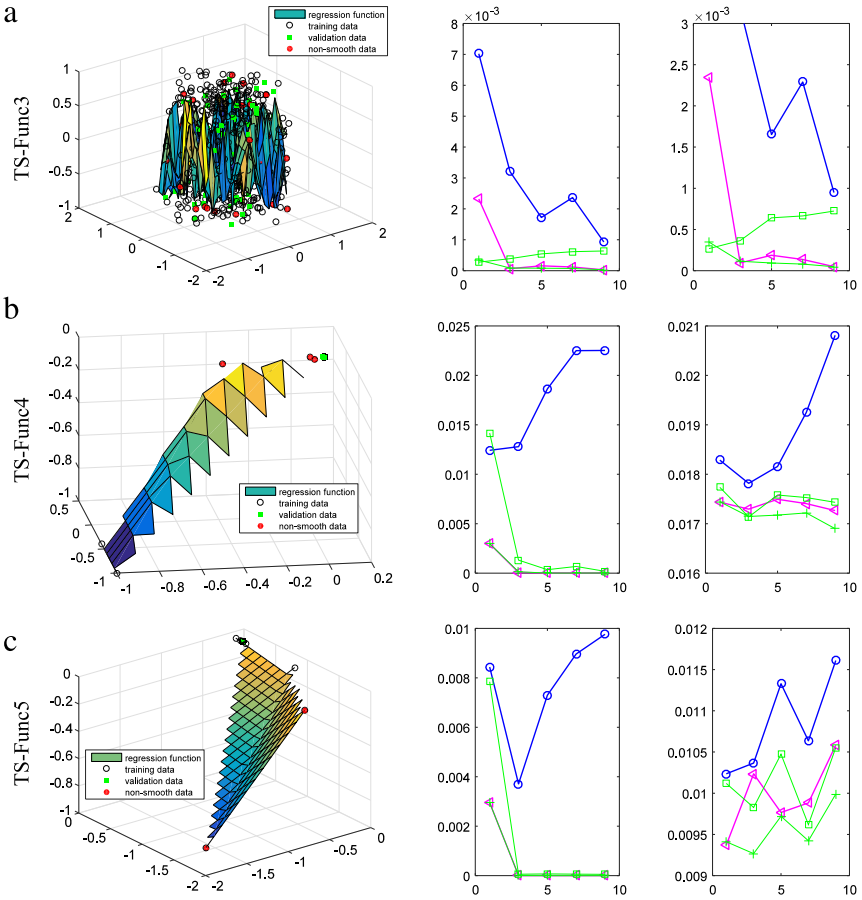

Fig. 3. Regression data for time-series functions. (a) TS-Func 3. (b) TS-Func 4. (c) TS-Func 5. Non-smooth data identified by ABR are denoted by dots. The second and third columns contain the average training mean squared error (MSE) and cross-validated MSE, respectively.

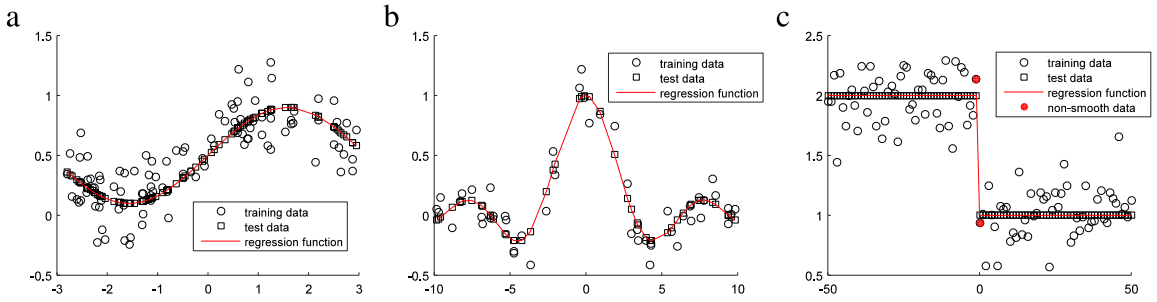

Fig. 4. Regression data for synthetic functions. (a) SF-1. (b) SF-2. (c) SF-3.

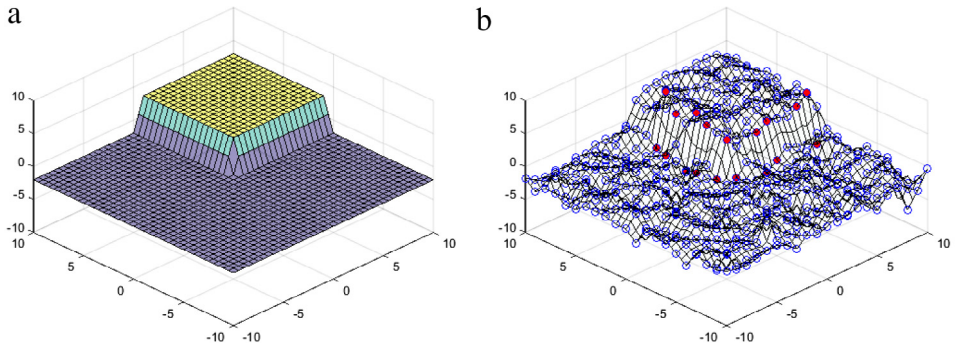

Fig. 5. Graphical representation of SF-4. (a) Surface of the true regression function. (b) Training data and identified non-smooth data are indicated by blue circles and red dots, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

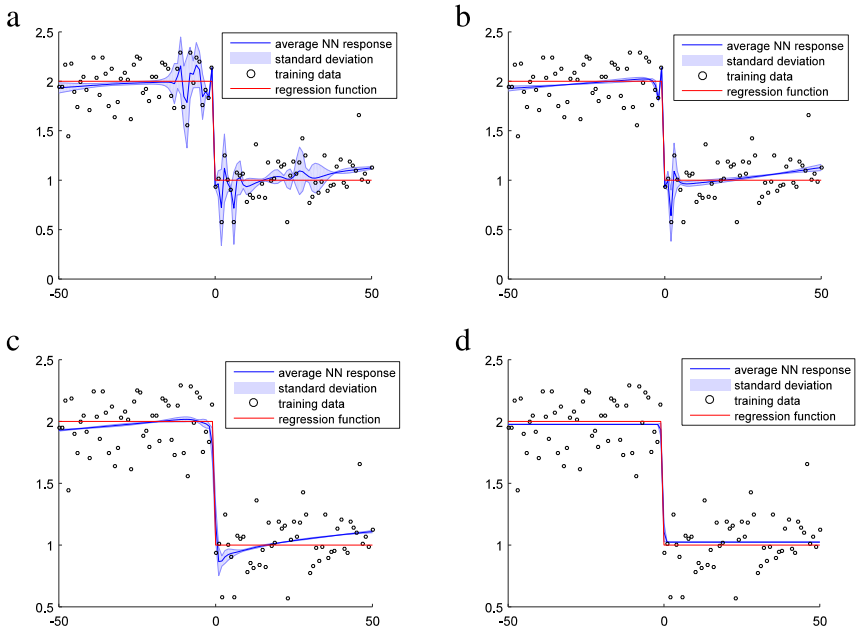

Fig. 6. SF-3 approximation. MLP response trained by LM, L-BFGS, GNBR and ABR. (a) LM. (b) L-BFGS. (c) GNBR. (d) ABR.

same amount of random noise as in SF-1, as shown in Fig. 4(c). The synthetic multivariate function4 (SF-4) is

$$f_4(x, y) = 5f_4(x)f_4(y) - 2, \tag{36}$$

where $f_4(x) = 1$ if $x \geq 0$, $f_4(x) = 0$ if $x < 0$, $f_4(y) = 1$ if $y \geq 0$, $f_4(y) = 0$ if $y < 0$. The range of x and y is $-10 \leq x, y \leq 10$. A total of 441 data points are generated in order to comprise the test dataset. Gaussian noise $e^{(i)} \sim \mathcal{N}(0, 1)$ is applied to the test output data $\{((x^{(i)}, y^{(i)}), (f_4(x^{(i)}, y^{(i)})))\}_{i=1}^{N}$ to give the training set $\{((x^{(i)}, y^{(i)}), f_4(x^{(i)}, y^{(i)}) + e^{(i)})\}_{i=1}^{N}$. Fig. 5(a) shows the test data of SF-4 and Fig. 5(b) illustrates the training data and non-smooth plot of SF-4 identified by ABR. Fig. 6 shows the average response and standard deviation 3σ of the MLP responses of SF-3 trained by LM, L-BFGS, GNBR, and ABR in Table 3.

For SF-1 and SF-2 experiments, Fig. 7(a) and (b) compare the MSE of ABR with those of GNBR, LM, and L-BFGS, respectively. As the number of hidden nodes in the MLP increases, the test error produced by LM and L-BFGS increases. For both SF-1 and SF-2, the performance of ABR is equivalent to that of GNBR, as there is no non-smooth data. (c) Shows a performance comparison for SF-3. Here, ABR produces the lowest error among the four training schemes since ABR reflects the error term of the non-smooth data shown in Fig. 4(c). Fig. 7(d) plots the training and test MSEs of the four training methods for SF-4. ABR produces the lowest error

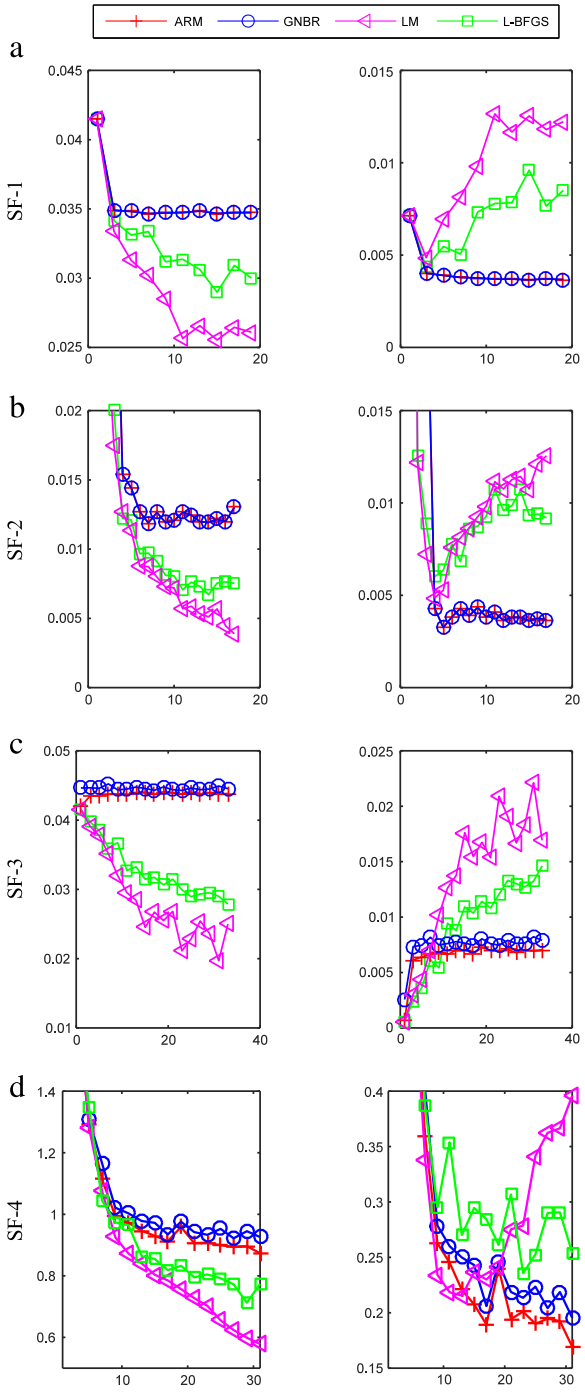

Fig. 7. Synthetic function approximations. Performance comparison of LM, GNBR, L-BFGS, and ABR. The first and second columns contain the average training mean squared error (MSE) and test MSE, respectively.

with the identified non-smooth data shown in Fig. 5(b). Table 3 presents the results for the MLP as the lowest training MSE among the different hidden node settings for each training method. ABR gives the best generalization error (i.e., the performance for unseen data) by using the training error. Its error is reported with the mean and standard deviation.

3.3. Real-world datasets

We performed experiments on real-world data by selecting five datasets from the UCI repository (Belsley, Kuh, & Welsch, 2007; Graf, Kriegel, Schubert, Poelsterl, & Cavallaro, 2011; Little, McSharry, Hunter, & Ramig, 2009; Tsanas & Xifara, 2012; Tüfekci, 2014).

Table 2
Specification of real-world regression datasets.

Datasets	Size	Input dimension
ELEC (Tüfekci, 2014)	9568	4
PARKINSONS (Little et al., 2009)	5875	16
COOLING LOAD (Tsanas & Xifara, 2012)	768	8
HOUSING (Belsley et al., 2007)	506	13
CT-SLICE (Graf et al., 2011)	53 500	385

Table 2 summarizes the specifications of the real-world datasets, ELEC, PARKINSONS, COOLING LOAD, HOUSING and CT-SLICE. The performance of ABR is compared with that of LM, L-BFGS, and GNBR by using a 10-fold cross-validation.

Fig. 8 shows the MLP performances trained by LM, L-BFGS, GNBR and ABR. The results, presented in Fig. 8, show that ABR generally produces the lowest cross-validated MSE. Even though L-BFGS often delivers a good training performance in terms of both the training MSE and cross-validated MSE in small hidden MLP node settings, it does not guarantee a lower cross-validated MSE when the larger number of hidden MLP nodes is specified. In addition, L-BFGS also sometimes yields under-fitting in the training sets or the unstable performance in the test datasets. Table 4 lists the MLPs selected on the basis of generating the lowest training MSE among their hidden nodes setting. The results show that ABR yields the lowest cross-validation error among the other training methods for unseen data.

In CT-SLICE which comprises relatively high-dimensional input data than the other datasets, first we train the deep network of 385-100-50-100-385 layered stacked autoencoder by using L-BFGS. Then, the first three layers are used as the pre-trained model. The next 50-10-1 layered architecture are trained by LM, GBNR, L-BFGS and ABR. Randomly selected 5350 data in CT-SLICE are used. The output data of CT-SLICE are scaled to a range of 0–100. The performance of the four training methods is evaluated by using 5-fold cross-validation. Table 5 lists the MLPs trained by the four methods.

Overall on three kinds of experiments (time series predictions, synthetic function approximations, and real-world datasets), LM yields a quite good fit to non-smooth data with no noise presented in Section 3.1. However, even though it would learn the desired response, it also includes noise when learning with the synthetic functions and real-world data. Hence, it causes overfitting as explained in Sections 3.2 and 3.3. On the other hand, GNBR is robust against the effect of noise so that it reduces overfitting, but it can result in under-fitting when approximating the non-smooth data, as explained in Sections 3.1 and 3.3. In general, the performance of L-BFGS is worse than the performances of LM, GNBR, and ABR, which is reported in the experiment sections. In contrast, ABR produces a good fit to both the smooth and non-smooth data regardless of the effect of noise.

4. Conclusions

This paper introduced ABR as a novel regularization method designed for non-smooth regression functions. The performance of ABR was compared to that of LM, GNBR, and L-BFGS with the various datasets. The proposed method is computationally efficient; therefore, it can be used in practical applications. The posterior predictive distribution of our method was analytically evaluated and shown to be valid in the real-world datasets as well as synthetic datasets. Our simulation results showed that the probabilistic assumptions of ABR are suitable for the identification of non-smooth data and for training the MLP.

Table 3
Performance comparison of LM, GNBR, L-BFGS and ABR on synthetic datasets.

Training method	Model and error	Dataset			
		SF-1	SF-2	SF-3	SF-4
LM	# Hidden nodes	21	17	15	27
	Training MSE	0.0248 ± 0.0017	0.0039 ± 9.6310e−004	0.0245 ± 0.0025	0.6239 ± 0.0194
	Test MSE	0.0134 ± 0.0018	0.0126 ± 9.0932e−004	0.0176 ± 0.0025	0.3625 ± 0.0316
GNBR	# Hidden nodes	21	7	17	27
	Training MSE	0.0347 ± 3.7992e−005	0.0087 ± 5.8052e−004	0.0443 ± 2.717e−004	0.9218 ± 0.0120
	Test MSE	**0.0037 ± 1.7249e−005**	**0.0081 ± 0.0012**	0.0075 ± 2.8249e−004	0.2041 ± 0.0118
L-BFGS	# Hidden nodes	15	14	19	29
	Training MSE	0.02898 ± 0.0022	0.00671 ± 0.000976	0.0306 ± 0.0014	0.7123 ± 0.0469
	Test MSE	0.0096 ± 0.00227	0.01068 ± 0.00159	0.0114 ± 0.0014	0.2906 ± 0.0456
ABR	# Hidden nodes	21	7	1	27
	Training MSE	0.0347 ± 3.7992e−005	0.0087 ± 5.8052e−004	0.042 ± 7.0617e−009	0.8942 ± 0.0131
	Test MSE	**0.0037 ± 1.7249e−005**	**0.0081 ± 0.0012**	**0.00071 ± 3.0752e−009**	**0.1947 ± 0.0062**

[a] MSE is represented as average and standard deviation. The best performance for test dataset is indicated in bold.

Table 4
Performance comparison of LM, GNBR, L-BFGS and ABR on real-world datasets (ELEC, PARKINSONS, COOLING LOAD, and HOUSING).

Training method	Model and error	Dataset			
		Elec	Parkinsons	Cooling load	Housing
LM	# Hidden nodes	21	11	25	27
	Training MSE	15.8986 ± 0.3248	74.4861 ± 5.0689	0.6261 ± 0.5408	4.9867 ± 6.6989
	Cross-validated MSE	16.6746 ± 1.2613	84.7613 ± 5.6860	1.5068 ± 0.7206 0	30.66 ± 10.8380
GNBR	# Hidden nodes	21	11	25	29
	Training MSE	15.9743 ± 0.3960	74.8636 ± 5.3027	0.3806 ± 0.1750	0.7073 ± 0.2535
	Cross-validated MSE	16.7909 ± 1.3099	84.7818 ± 5.6093	**1.0912 ± 0.5795**	28.6857 ± 11.2753
L-BFGS	# Hidden nodes	21	11	25	29
	Training MSE	5.5877 ± 0.91699	5.4767 ± 0.7181	0.5164 ± 0.9109	0.00018364 ± 0.00042428
	Cross-validated MSE	41.804 ± 23.774	85.0216 ± 296.8121	174.76 ± 1030.3635	20.095 ± 7.2461
ABR	# Hidden nodes	21	11	23	21
	Training MSE	15.8733 ± 0.3612	73.2864 ± 4.7014	0.3975 ± 0.1285	5.6021 ± 6.3053
	Cross-validated MSE	**16.5136 ± 1.2881**	**81.9557 ± 5.9263**	1.1175 ± 0.6042	**19.0972 ± 11.3319**

[a] MSE is represented as average and standard deviation. Each MSE is evaluated by 10-fold cross-validation. The best performance for cross-validation is indicated in bold.

Table 5
Performance comparison of LM, GNBR, L-BFGS and ABR with stacked autoencoder on CT-SLICE.

Training method	Training MSE	Cross-validated MSE
LM	0.0221 ± 0.00124	0.047 ± 0.01027
GNBR	0.02347 ± 0.00264	0.04021 ± 0.00219
L-BFGS	0.04642 ± 0.00427	0.05302 ± 0.00336
ABR	0.02418 ± 0.00261	**0.03742 ± 0.00172**

[a] MSE is represented as average and standard deviation. Each MSE is evaluated by 5-fold cross-validation. The best performance for cross-validation is indicated in bold.

The proposed method finds the regularization coefficients without the need of using a validation dataset. The notable characteristic of ABR is that it fits the MLP response to the desired response (true regression function) regardless of the effect of noise for both smooth and non-smooth data points. Moreover, ABR is able to be used for training a deep learning model and to provide more accurate regression performances.

The weakness of ABR is that it sometimes converges to a local optimum of its objective function as shown by the training MSE for 19 hidden nodes in Fig. 7(c). This is challenging in non-convex optimization communities. In future work, properly initializing the MLP weights or designing a robust optimization method for the non-convex objective function can be more analyzed.

Acknowledgments

This work was supported by the Technology Innovation Program, 10045252, funded by the Ministry of Trade, Industry and Energy in Republic of Korea. The authors would like to thank the anonymous reviewers for their helpful and insightful comments.

Appendix A. Proof of training error convergence in (18)

Let $E = E(\mathbf{w})$ be the objective function in (18), $\mathbf{w}^{(k)}$ be a weight vector of the MLP in kth epoch ($0 \le k \le l$) and $\mathbf{H}^{(k)}$ be the Hessian matrix of E at $\mathbf{w}^{(k)}$. Then, with Taylor expansion, the objective function can be approximated as $E(\mathbf{w}) \approx Q(\mathbf{w})$ where

$$\begin{aligned} Q(\mathbf{w}) = {} & \alpha E_W(\mathbf{w}^{(k)}) + \beta E_{\mathbf{D}}(\mathbf{w}^{(k)}) + \zeta E_{\hat{\mathbf{D}}}(\mathbf{w}^{(k)}) \\ & + (\mathbf{w} - \mathbf{w}^{(k)})^{\mathrm{T}} \left[\alpha \nabla E_W(\mathbf{w}^{(k)}) + \beta \nabla E_{\mathbf{D}}(\mathbf{w}^{(k)}) \right. \\ & \left. + \zeta \nabla E_{\hat{\mathbf{D}}}(\mathbf{w}^{(k)})\right] \\ & + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{(k)})^{\mathrm{T}} \mathbf{H}^{(k)} (\mathbf{w} - \mathbf{w}^{(k)}). \end{aligned} \tag{A.1}$$

To find the minimum of Q, $\nabla Q(\mathbf{w}) = \mathbf{0}$ and

$$\begin{aligned} \nabla Q(\mathbf{w}) = {} & \alpha \nabla E_W(\mathbf{w}^{(k)}) + \beta \nabla E_{\mathbf{D}}(\mathbf{w}^{(k)}) + \zeta \nabla E_{\hat{\mathbf{D}}}(\mathbf{w}^{(k)}) \\ & + \mathbf{H}^{(k)}(\mathbf{w} - \mathbf{w}^{(k)}). \end{aligned} \tag{A.2}$$

In the first epoch, $k = 0$, $\beta = 1, \alpha = 0$, and $\zeta = 0$ according to *Step* 3 in Table 1. The MLP weight vector is updated as $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \left(\mathbf{H}^{(k)}\right)^{-1} \left[\nabla E_W(\mathbf{w}^{(k)})\right]$ where $\mathbf{H}^{(k)} \approx \mathbf{J}_{\mathbf{D}}^{(k)\mathrm{T}} \mathbf{J}_{\mathbf{D}}^{(k)}$, $\mathbf{J}_{\mathbf{D}}^{(k)} = \nabla E_{\mathbf{D}} \cdot \mathbf{J}_{\mathbf{D}}^{(k)}$ is N by W Jacobian matrix whose elements are $\mathbf{J}_{\mathbf{D}(i,j)}^{(k)} = \partial y(\mathbf{x}_{i\in G}; \mathbf{w}^{(k)})/\partial \mathbf{w}_j$ and $G = \{i | \mathbf{x}_i \in \mathbf{D}\}$. By the Levenberg–Marquardt (LM) method (Hagan & Menhaj, 1994), it finds the LM parameter $\mu^{(k)}$ such that $\mathbf{H}^{(k)} = \mathbf{J}_{\mathbf{D}}^{(k)\mathrm{T}} \mathbf{J}_{\mathbf{D}}^{(k)} + \mu^{(k)}\mathbf{I}$ is positive-definite. Thus, $\mathbf{w}^{(1)}$ must satisfy $E(\mathbf{w}^{(1)}) < E(\mathbf{w}^{(0)})$ if there exists $\mu^{(1)}$ returned by the LM method. In *Step* 5 in Table 1, the parameters α, β, and ζ are updated as the approximated solution through

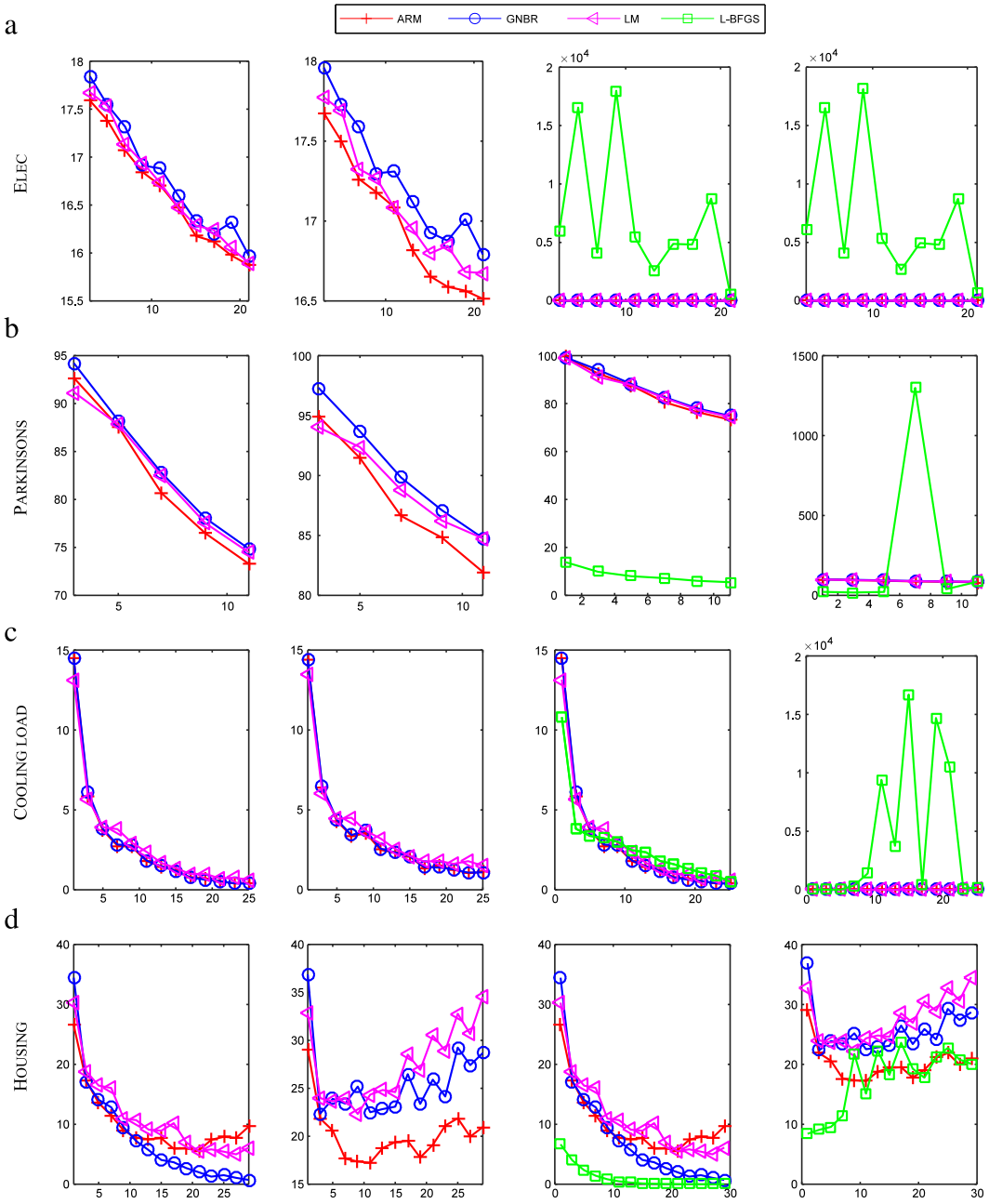

Fig. 8. Real-world datasets (ELEC, PARKINSONS, COOLING LOAD, and HOUSING). Performance comparison of LM, GNBR, L-BFGS, and ABR. The first and third columns contain the average training mean squared error (MSE). The second and fourth columns present the cross-validated MSE.

(23)–(25). In the following epochs, for $k > 0$,

$$\mathbf{w}^{MP} = \mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \left(\mathbf{H}^{(k)}\right)^{-1} \times \left[\alpha\nabla E_W(\mathbf{w}^{(k)}) + \beta\nabla E_{\mathbf{D}}(\mathbf{w}^{(k)}) + \zeta\nabla E_{\hat{\mathbf{D}}}(\mathbf{w}^{(k)})\right] \quad \text{(A.3)}$$

$$\begin{aligned}\text{where } \mathbf{H}^{(k)} &= \alpha\nabla^2 E_W(\mathbf{w}^{(k)}) + \beta\nabla^2 E_{\mathbf{D}}(\mathbf{w}^{(k)}) \\ &\quad + \zeta\nabla^2 E_{\hat{\mathbf{D}}}(\mathbf{w}^{(k)}) + \mu^{(k)}\mathbf{I} \\ &= \alpha\nabla^2 E_W(\mathbf{w}^{(k)}) + \beta\nabla^2\mathbf{H}_{\mathbf{D}} + \zeta\nabla^2\mathbf{H}_{\hat{\mathbf{D}}} + \mu^{(k)}\mathbf{I} \\ &\approx \alpha\mathbf{I} + \beta\mathbf{J}_{\mathbf{D}}^{(k)\mathrm{T}}\mathbf{J}_{\mathbf{D}}^{(k)} + \zeta\mathbf{J}_{\hat{\mathbf{D}}}^{(k)\mathrm{T}}\mathbf{J}_{\hat{\mathbf{D}}}^{(k)} + \mu^{(k)}\mathbf{I}\end{aligned}$$

where $\mathbf{J}_{\hat{\mathbf{D}}} = \nabla E_{\hat{\mathbf{D}}}$ is N' by W Jacobian matrix and its elements are $\mathbf{J}_{\hat{\mathbf{D}}(i,j)}^{(k)} = \partial y(\mathbf{x}_{i\in L}; \mathbf{w}^{(k)})/\partial\mathbf{w}_j$ where $L = \{i | x_i \in \hat{\mathbf{D}}\}$. $y(\mathbf{x}; \mathbf{w})$ is explained in (15). If $\mu^{(k)}$ exists such that $\mathbf{H}^{(k)}$ is positive-definite satisfying $E_{\mathbf{D}}(\mathbf{w}^{(k+1)}) < E_{\mathbf{D}}(\mathbf{w}^{(k)})$, then $\mathbf{w}^{(k+1)}$ is an approximated solution that minimizes (18) with $E_{\mathbf{D}}(\mathbf{w}^{(k+1)}) < E_{\mathbf{D}}(\mathbf{w}^{(k)})$. ■

Appendix B. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.neunet.2016.07.010.

References

Apostolopoulou, M.S. (2009). A memoryless BFGS neural network training algorithm. In *7th IEEE international conference on industrial informatics, INDIN 2009* (pp. 216–221).

Belsley, D. A., Kuh, E., & Welsch, R. E. (2007). *Regression diagnostics: Identifying influential data and sources of collinearity*. (pp. 244–261). Wiley.

Bowman, A. W., & Pope, B. (2008). Ismail, detecting discontinuities in nonparametric regression curves and surfaces. *Statistics and Computing*, *16*, 377–390.

Chamjangali, M. A., Mohammadrezaei, M., Kalantar, Z., & Amin, A. H. (2012). Bayesian regularized artificial neural network modeling of the anti-protozoal activities of 1-methylbenzimidazole derivatives against T. Vaginalis infection. *Journal of the Chinese Chemical Society*, *59*, 743–752.

Connor, P. (2015). A biological mechanism for Bayesian feature selection: Weight decay and raising the LASSO. *Neural Networks*, *67*, 121–130.
Esposito, A., Marinaro, M., Oricchioa, D., & Scarpetta, S. (2000). Approximation of continuous and discontinuous mappings by a growing neural RBF-based algorithm. *Neural Networks*, *13*, 651–665.
Foresee, F.D., & Hagan, M.T. (1997). Gauss-Newton approximation to Bayes learning, In *Int. conf. on neural networks, Vol. 3* (pp. 1930–1935).
Graf, F., Kriegel, H.-P., Schubert, M., Poelsterl, S., & Cavallaro, A. (2011). 2D image registration in CT images using radial image descriptors. In *Medical image computing and computer-assisted intervention, MICCAI, Vol. 14* (pp. 607–614).
Hagan, M. T., Demuth, H. B., Beale, M. H., & Jess, O. D. (2014). *Neural network design* (2nd ed.). PWS Publishing, (chapter 12).
Hagan, M. T., & Menhaj, M. B. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, *5*(6), 989–993.
Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366.
Kucuk, N., Manohara, S. R., Hanagodimath, S. M., & Gerward, L. (2013). Modeling of gamma ray energy-absorption buildup factors for thermoluminescent dosimetric materials using multilayer perceptron neural network: A comparative study. *Radiation Physics and Chemistry*, *86*, 10–12.
Larsen, J., & Hansen, L.K. (1994). Generalization performance of regularized neural network models. In *Proc. of the IEEE workshop on neural networks for signal processing, Vol. 5* (pp. 42–51).
Lauera, F., Blochb, G., & Vidalc, R. (2011). A continuous optimization framework for hybrid system identification. *Automatica*, *47*(3), 608–613.
Little, M. A., McSharry, P. E., Hunter, E. J., & Ramig, L. O. (2009). Suitability of dysphonia measurements for telemonitoring of Parkinson's disease. *IEEE Transactions on Biomedical Engineering*, *56*(4), 1015–1022.
Llanas, B., Lantarón, S., & Sáinz, F. J. (2008). Constructive approximation of discontinuous functions by neural networks. *Neural Processing Letters*, *27*, 209–226.
Ludwig, O., Nunes, U., & Araujo, R. (2014). Eigenvalue decay: A new method for neural network regularization. *Neurocomputing*, *124*, 33–42.
MacKay, D. J. C. (1992). A practical Bayesian framework for backpropagation networks. *Neural Computation*, *4*(3), 448–472.
Murphy, K. P. (2007). *Conjugate Bayesian analysis of the Gaussian distribution, Technical Report*.
Oliver, J. F., Fuster-Garcia, E., Cabello, J., Tortajada, S., & Rafecas, M. (2013). Application of artificial neural network for reducing random coincidences in PET. *IEEE Transactions on Nuclear Science*, *60*(5), 3399–3409.
Paoletti, S., Roll, J., Garulli, A., & Vicino, A. (2010). On the input–output representation of piecewise affine state space models. *IEEE Transactions on Automatic Control*, *55*(1), 60–73. IEEE Transactions on Automatic Control.
Piliougine, M., Elizondo, D., Mora-López, L., & Sidrach-de-Cardona, M. (2013). Multilayer perceptron applied to the estimation of the influence of the solar spectral distribution on thin-film photovoltaic modules. *Applied Energy*, *112*, 610–617.
Pinzolas, M., et al. (2006). A neighborhood-based enhancement of the Gauss-Newton Bayes regularization training method. *Neural Computation*, *18*, 1987–2003.
Raiko, T., Valpola, H., & LeCun, Y. (2012). Deep learning made easier by linear transformations in perceptrons. In *Proceedings of the 15th international conference on artificial intelligence and statistics*.
Rolla, J., Bemporadb, A., & Ljunga, L. (2004). Identification of piecewise affine systems via mixed-integer programming. *Automatica*, (40), 37–50.
Sum, J., & Ho, K. (2009). SNIWD: Simultaneous weight noise injection with weight decay for MLP training. *Neural Information Processing*, *5863*, 494–501.
Tsanas, A., & Xifara, A. (2012). Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, *49*, 560–567.
Tüfekci, P. (2014). Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, *60*, 126–140.
Wilamowski, B. M., & Yu, H. (2010). Improved computation for Levenberg–Marquardt training. *IEEE Transactions on Neural Networks*, *21*(6), 930–937.
Yang, Y., Wang, Y., Wu, J., Lin, X., & Liu, M. (2015). Progressive learning machine: A new approach for general hybrid system approximation. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(9), 1855–1874.