Regularization Parameter Estimation for Feedforward Neural Networks

Ping Guo, Michael R. Lyu and C.L. Philip Chen

P. Guo is with the Department of Computer Science, Beijing Normal University, Beijing, 100875, P. R. China. E-mail: pguo@elec.bnu.edu.cn.

M.R. Lyu is with the Department of Computer Science & Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong. E-mail: lyu@cse.cuhk.edu.hk.

C.L.P. Chen is with the Department of Computer Science & Engineering, Wright State University, Dayton, OH, 45435, USA. E-mail: pchen@cs.wright.edu.

Abstract

Under the framework of the Kullback-Leibler distance, we show that a particular case of Gaussian probability function for feedforward neural networks reduces into the first order Tikhonov regularizer. The smooth parameter in kernel density estimation plays the role of the regularization parameter. Under some approximations, an estimation formula is derived for estimating regularization parameters based on training data sets. The similarity and difference of the obtained results are compared with other's work. Experimental results show that the estimation formula works well in the sparse and small training sample cases.

Keywords

Tikhonov Regularizer, Regularization Parameter Estimation, Small Training Data Set.

I. Introduction

It is well known that the goal of training neural networks is not to learn an exact representation of the training data itself, but rather to build a statistical model of the process which generates the data. In practical applications of a feedforward neural network, if the network is over-fit to the noise on the training data, especially for the small-number training samples case, it will memorize training data and give poor generalization. To control an appropriate complexity of the network can improve generalization. There are two main approaches for this purpose: model selection and regularization. Model selection for a feedforward neural network requires choosing the number of hidden neurons and thereof connection weights. The common statistical approach to model selection is to estimate the generalization error for each model and to choose the model minimizing this error[1],[2]. Regularization involves constraining or penalizing the solution of the estimation problem to improve network generalization ability by smoothing the predictions[3],[4]. Most common regularization methods include weight decay[5] and addition of artificial noise to the inputs during training[6],[7].

Regularization method is widely used for smoothing output[8],[9]. A value of the regularization parameter is determined by using the statistical techniques such as cross-validation[10], bootstrapping[11], and Bayesian method[12]. Most work uses a validation set to select the regularization parameter[13],[14],[15],[16]. This requires to split a given data set into training and validation sets. The optimal selection of the regularization parameter on the validation set sometimes depends on how to partition the data set. For a small-number data set, we usually use leave-one-out cross-validation method. However, a recent study shows that cross-validation performance is not always good in the selection of linear models[17].

In this paper, under the framework of the Kullback-Leibler (KL) distance[18],[19] we show that a particular case of the system entropy reduces into the first order Tikhonov regularizer. The smoothing parameter in the kernel density function plays the role of the regularization parameter. Under some approximations, an estimation formula can be derived for estimating the regularization parameter based on the training data set. There is a lot of research work in smoothing parameter estimation of kernel density function[21],[22],[23]; however, in this paper we only focus on comparing the obtained result with the maximum a *posteriori* (MAP) framework[12]. Experimental results show that the newly derived estimation formula works well in the sparse and small training sample cases.

II. System Probability Function

When given a data set $D = \{\mathbf{x}_i, \mathbf{z}_i\}_{i=1}^{N}$, we consider that the data can be modelled by a probability function. In one particular design, we can let kernel density of the given data set D be $p_h(\mathbf{x}, \mathbf{z})$, and on the other hand, the mapping architecture is denoted as a joint probability function $P(\mathbf{x}, \mathbf{z})$ on the data set D. The relative entropy or Kullback-Leibler distance for this particular system is denoted by $J(h, \Theta)$ cost function, where Θ stands for a parameter vector, then the quantity of interest is the "distance" of these two probability functions, which can be measured as follows[18],[19]:

$$\begin{aligned} J(h,\Theta) &= \iint p_h(\mathbf{x},\mathbf{z}) \ln \frac{p_h(\mathbf{x},\mathbf{z})}{P(\mathbf{x},\mathbf{z})} d\mathbf{x} d\mathbf{z} \\ &= -\iint p_h(\mathbf{x},\mathbf{z}) \ln P(\mathbf{z}|\mathbf{x},\Theta) d\mathbf{x} d\mathbf{z} \\ &\quad + \iint p_h(\mathbf{x},\mathbf{z}) \ln \frac{p_h(\mathbf{x},\mathbf{z})}{P_0(\mathbf{x})} d\mathbf{x} d\mathbf{z}, \end{aligned} \tag{1}$$

where we use the notation of Bayes theorem,

$$P(\mathbf{x}, \mathbf{z}) = P(\mathbf{z}|\mathbf{x}, \Theta) P_0(\mathbf{x}). \tag{2}$$

$P(\mathbf{z}|\mathbf{x}, \Theta)$ is a parameter conditional probability and $P_0(\mathbf{x})$ is a prior probability function.

We define

$$J_1(h, \Theta) \equiv - \iint p_h(\mathbf{x}, \mathbf{z}) \ln P(\mathbf{z}|\mathbf{x}, \Theta) d\mathbf{x} d\mathbf{z}, \tag{3}$$

$$\begin{aligned} J_2(h) &\equiv \iint p_h(\mathbf{x}, \mathbf{z}) \ln p_{h0}(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z}, \\ p_{h0}(\mathbf{x}, \mathbf{z}) &\equiv \frac{p_h(\mathbf{x}, \mathbf{z})}{P_0(\mathbf{x})}. \end{aligned} \tag{4}$$

$J_1(h, \Theta)$ is related to network parameter vector Θ, and smoothing parameter $h = \{h_x, h_z\}$. $J_2(h)$ can be considered as the negative cross entropy of data distribution functions, and it is only related to the smoothing parameter h.

Now Eq. (1) becomes

$$J(h, \Theta) = J_1(h, \Theta) + J_2(h). \tag{5}$$

We can assign a prefixed kernel function $K(\cdot)$ and smoothing parameters h_x, h_z for nonparametric density estimation[20],[21] of $p_h(\mathbf{x}, \mathbf{z})$ for a given discrete training data set D, where the kernel density function[21] is

$$\begin{aligned} p_{h_x}(\mathbf{x}) &= \frac{1}{N} \sum_{x_i \in D} K_{h_x}(\mathbf{x} - \mathbf{x}_i), \\ K_{h_x}(\mathbf{x} - \mathbf{x}_i) &= \frac{1}{h_x^d} K(\frac{\mathbf{x} - \mathbf{x}_i}{h_x}), \end{aligned} \tag{6}$$

where N represents the number of samples in the data set D, d is the dimension of a random variable $\mathbf{x}$, and the joint distribution $p_h(\mathbf{x}, \mathbf{z})$ in this work is designed as

$$p_h(\mathbf{x}, \mathbf{z}) = \frac{1}{N} \sum_{\mathbf{x}_i, \mathbf{z}_i \in D} K_{h_x}(\mathbf{x} - \mathbf{x}_i) K_{h_z}(\mathbf{z} - \mathbf{z}_i). \tag{7}$$

The mostly used kernel density function is Gaussian kernel,

$$K_h(\mathbf{r}) = G(\mathbf{r}, 0, h\mathbf{I}_d) = \frac{1}{(2\pi h)^{d/2}} \exp\{-\frac{||\mathbf{r}||^2}{2h}\}. \tag{8}$$

In the kernel density function, $\mathbf{I}_d$ is a $d \times d$ dimensional identity matrix. In this paper, we use $\{d_x,\, d_z\}$ to represent the dimension of input $\mathbf{x}$ and output $\mathbf{z}$ vector, respectively.

According to the principle of minimum description length (MDL)[25],[26], the best model class for a set of observed data is the one whose representative permits the shortest coding of the data, then the system should be optimized with optimal or *ideal* codelength. The parameters $h_x,\, h_z$ should be chosen with minimized Kullback–Leibler distance function based on the given data set according to

$$\{h_x, h_z\} = \arg\min_h J(h, \Theta^*), \tag{9}$$

where Θ^* is the learned neural network parameter and $J(h, \Theta)$ is represented by Eq. (1).

In the following sections we will discuss the regularization problem with a finite training data set D.

III. Tikhonov Regularizer

When estimating network parameter by Maximum Likelihood (ML) learning, we minimize the function $J(h, \Theta)$ to find the network parameter Θ with a fixed parameter h. For a particular design, the conditional probability function can be written in the form

$$P(\mathbf{z}|\mathbf{x}, \Theta) = P(\mathbf{z}|f(\mathbf{x}, \Theta)) \tag{10}$$

where $f(\mathbf{x}, \Theta)$ is a function of input variable $\mathbf{x}$ and parameter Θ.

In the network parameter learning procedure, only J_1 is involved because J_2 does not contain the parameter Θ.

To evaluate the function J_1, one of the techniques is the well-known *Monte Carlo integration*[27],[28]. In the *Monte Carlo integration* approximation, when substituting Eqs. (7) and (10) into Eq. (3), integration can be approximated by summation, and we obtain

$$J_1(h,\Theta) = -\frac{1}{N'}\sum_{i=1}^{N'} \ln P(\mathbf{z}'_i|f(\mathbf{x}'_i,\Theta)), \tag{11}$$

where

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{e}_x, \qquad \mathbf{z}'_i = \mathbf{z}_i + \mathbf{e}_z. \tag{12}$$

$\mathbf{e}_x$, $\mathbf{e}_z$ are data points drawn from distribution $p_h(\mathbf{x},\mathbf{z})$. In this case, $J_1(h,\Theta)$ is equivalent to a negative likelihood function of the system.

In the *Monte Carlo integration* approximation, we need to generate a number of data sets, which is very computation-intensive.

Another method is the Taylor expansion approximation for an integral, which we use in this paper,

$$J_1(h,\Theta) = -\iint p_h(\mathbf{x},\mathbf{z}) \ln P(\mathbf{z}|f(\mathbf{x},\Theta)) d\mathbf{x}d\mathbf{z}. \tag{13}$$

When we consider one special case, $P(\mathbf{z}|f(\mathbf{x},\Theta)) = G(\mathbf{z}, g(\mathbf{x},W), \sigma^2\mathbf{I}_{d_z})$ is Gaussian density function,

$$G(\mathbf{z}, g(\mathbf{x},W), \sigma^2\mathbf{I}_{d_z}) = \frac{1}{(2\pi\sigma^2)^{d_z/2}} \exp[-\frac{1}{2\sigma^2}||\mathbf{z} - g(\mathbf{x},W)||^2]$$

$$\begin{aligned} J_1(h,\Theta) &= -\iint p_h(\mathbf{x},\mathbf{z}) \ln G(\mathbf{z}, g(\mathbf{x},W), \sigma^2\mathbf{I}_{d_z}) d\mathbf{x}d\mathbf{z} \\ &= \iint p_h(\mathbf{x},\mathbf{z})[\frac{1}{2\sigma^2}||\mathbf{z} - g(\mathbf{x},W)||^2] d\mathbf{x}d\mathbf{z} \\ &\quad + \frac{d_z}{2} \ln 2\pi\sigma^2 \end{aligned} \tag{14}$$

where $g(\mathbf{x},W)$ is a neural network mapping function. For example, in three-layer feedforward neural network with k hidden neurons case,

$$g(\mathbf{x},W) = S(W_{z|y} \cdot S(W_{y|x} \cdot \mathbf{x})). \tag{15}$$

$W = \{W_{z|y}, W_{y|x}\}$ is a network weight parameter vector, $W_{y|x}$ is a $d_x \times k$ matrix which connects the input space R_x and the hidden space R_y, $W_{z|y}$ is a $k \times d_z$ matrix which connects the hidden space R_y and the output space R_z. $S(\cdot)$ is a sigmoidal function,

$$S(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}. \tag{16}$$

Eq. (14) will result in the traditional sum-square-errors function in the maximum likelihood learning case at the limit of $h \to 0$, when we omit some factors irrelevant to the network weight parameter W.

Based on consideration of that random noise is added to the input data only during training, Bishop[29] proved that in ML estimation case, Eq. (11) can be reduced to the first order Tikhonov regularizer[30] for feedforward neural network with approximations.

On the other hand, addition of random noise to the input data is equivalent to smoothing in kernel density estimation, thus we can also obtain the same result directly from Eq. (13).

Let $f(\mathbf{x}, \mathbf{z}, w) = ||\mathbf{z} - g(\mathbf{x}, W)||^2$, $f(\mathbf{x}, \mathbf{z}, w)$ is a scale function of vector variable $\mathbf{x}$ and $\mathbf{z}$. When we expand $f(\mathbf{x}, \mathbf{z}, w)$ as a Taylor series in powers of $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_i$, $\Delta\mathbf{z} = \mathbf{z} - \mathbf{z}_i$ and denote $f'(\mathbf{x}_i, \mathbf{z}, w) = \nabla_x f(\mathbf{x}_i, \mathbf{z}, w)$. When taking only up to the second order term, then we obtain

$$\begin{aligned} f(\mathbf{x}, \mathbf{z}, w) \approx\ & f(\mathbf{x}_i, \mathbf{z}_i, w) + (f'_x)^T \Delta\mathbf{x} \\ & + \frac{1}{2}(\Delta\mathbf{x})^T f''_x \Delta\mathbf{x} + (\Delta\mathbf{x})^T f''_{x,z} \Delta\mathbf{z} \\ & + (f'_z)^T \Delta\mathbf{z} + \frac{1}{2}(\Delta\mathbf{z})^T f''_z \Delta\mathbf{z} \end{aligned} \tag{17}$$

Eq. (14) becomes

$$\begin{aligned}
J_1(h,\Theta) &= \iint p_h(\mathbf{x},\mathbf{z})[\frac{1}{2\sigma^2}f(\mathbf{x},\mathbf{z},w)]d\mathbf{x}d\mathbf{z} \\
&\quad +\frac{d_z}{2}\ln 2\pi\sigma^2 \\
&\approx \frac{1}{2N\sigma^2}\sum_{i=1}^{N}\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z}) \\
&\quad \times[f(\mathbf{x}_i,\mathbf{z}_i,w)+(f'_x)^T\Delta\mathbf{x}+\frac{1}{2}(\Delta\mathbf{x})^T f''_x\Delta\mathbf{x} \\
&\quad +(f'_z)^T\Delta\mathbf{z}+(\Delta\mathbf{x})^T f''_{x,z}\Delta\mathbf{z} \\
&\quad +\frac{1}{2}(\Delta\mathbf{z})^T f''_z\Delta\mathbf{z}]d\mathbf{x}d\mathbf{z}+\frac{d_z}{2}\ln 2\pi\sigma^2 \qquad (18)
\end{aligned}$$

Notice that for any density function, the integration in the whole space should be equal to one, i.e.,

$$\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})d\mathbf{x}d\mathbf{z}=1 \qquad (19)$$

$$\begin{aligned}
&\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})f(\mathbf{x}_i,\mathbf{z}_i,w)d\mathbf{x}d\mathbf{z} \\
= \;& f(\mathbf{x}_i,\mathbf{z}_i,w)=||\mathbf{z}_i-g(\mathbf{x}_i,W)||^2 \qquad (20)
\end{aligned}$$

For Gaussian type function integrals[6], we can obtain

$$\begin{aligned}
&\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z}) \\
&\times[(f'_x)^T\Delta\mathbf{x}+(f'_z)^T\Delta\mathbf{z}]d\mathbf{x}d\mathbf{z}=0, \\
&\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z}) \\
&\times[(\Delta\mathbf{x})^T f''_{x,z}\Delta\mathbf{z}]d\mathbf{x}d\mathbf{z}=0. \qquad (21)
\end{aligned}$$

$$\begin{aligned} & \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\ & \times [\frac{1}{2} (\Delta \mathbf{x})^T f''_x \Delta \mathbf{x}] d\mathbf{x} d\mathbf{z} \\ = & \frac{h_x}{2} \text{trace}[f''_x] \\ = & h_x \{ ||g'(\mathbf{x}, W)||^2 - ||[\mathbf{z}_i - g(\mathbf{x}_i, W)] g''(\mathbf{x}_i, W)|| \} \end{aligned} \tag{22}$$

$$\begin{aligned} & \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\ & \times [\frac{1}{2} (\Delta \mathbf{z})^T f''_z \Delta \mathbf{z}] d\mathbf{x} d\mathbf{z} \\ & = \frac{h_z}{2} \text{trace}[f''_z] = d_z h_z \end{aligned} \tag{23}$$

With the above results, the integration becomes

$$\begin{aligned} J_1(h, \Theta) = & \iint p_h(\mathbf{x}, \mathbf{z}) [\frac{1}{2\sigma^2} f(\mathbf{x}, \mathbf{z}, w)] d\mathbf{x} d\mathbf{z} \\ & + \frac{d_z}{2} \ln 2\pi\sigma^2 \\ \approx & \frac{1}{2N\sigma^2} \sum_{i=1}^{N} \{ ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2 \\ & + h_x [||g'(\mathbf{x}, W)||^2 \\ & - ||(\mathbf{z}_i - g(\mathbf{x}_i, W)) g''(\mathbf{x}_i, W)||] \} \\ & + h_z \frac{d_z}{2\sigma^2} + \frac{d_z}{2} \ln 2\pi\sigma^2 \end{aligned} \tag{24}$$

Because the term $h_z d_z / 2\sigma^2$ in the above equation is not implicitly related to the network weight parameter W, we can omit this term in weight parameter learning. This also illustrates that smoothing on output cannot improve network generalization, thus we can let $h_z \to 0$ without loss of generality. The last term in the above equation is irrelevant to the weight parameter, and can be neglected too[6]. Now the equation becomes

$$\begin{aligned} J_1(h,\Theta) \approx & \frac{1}{2N\sigma^2}\sum_{i=1}^{N}\{||\mathbf{z}_i - g(\mathbf{x}_i,W)||^2 \\ & + h_x[||g'(\mathbf{x},W)||^2 \\ & -||(\mathbf{z}_i - g(\mathbf{x}_i,W))g''(\mathbf{x}_i,W)||]\} \end{aligned} \tag{25}$$

Rewrite the equation in the form

$$J_1 \approx J_s + h_x J_r \tag{26}$$

where

$$\begin{aligned} J_s &= \frac{1}{2N\sigma^2}\sum_{i=1}^{N}||\mathbf{z}_i - g(\mathbf{x}_i,W)||^2 \\ J_r &= \frac{1}{2N\sigma^2}\sum_{i=1}^{N}\{||g'(\mathbf{x}_i,W)||^2 \\ & -||(\mathbf{z}_i - g(\mathbf{x}_i,W))g''(\mathbf{x}_i,W)||\} \end{aligned} \tag{27}$$

In the above equation, J_s represents the traditional sum-square-error function, while J_r stands for a regularization term.

In Eq. (27), the second derivative term is the Hessian term. Reed[31] described it as an approximate measure of the difference between the average surrounding values and the precise value of the filed at a point, and assumed it to be zero. Bishop[29],[32] considered that when minimizing the cost function, the second term in J_r involving the second derivatives of the network function $g(\mathbf{x},W)$ vanishes to $\mathcal{O}(h_x)$. For sufficiently small values of the smooth parameter h_x, this leads to

$$\begin{aligned} J_1 &\approx J_s + h_x J_r \\ &= \frac{1}{2N\sigma^2}\sum_{i=1}^{N}\{||\mathbf{z}_i - g(\mathbf{x}_i,W)||^2 + h_x||g'(\mathbf{x}_i,W)||^2\} \end{aligned} \tag{28}$$

¿From the above we can easily see that under some approximation one special case $J(h, \Theta)$ function is reduced to the first order Tikhonov regularizer in the sense of maximum likelihood learning.

Furthermore, from the above results it is easy to see that the parameter h_x controls the degree of smoothness of the network mapping, just the same as the problem of controlling the degree of smoothing in a nonparametric estimation. The optimum value of h_x is problem-dependent. Using the traditional sum-square-error function cannot select this parameter completely with a given data set. Instead, it needs to use separated training and validation data sets, and to be optimized by the cross-validation method or another validation data set.

In the next section we develop a formula to estimate this regularization coefficient based on the training data set.

IV. Estimation of Regularization Parameter

When $h \neq 0$, according to the principle of MDL, the regularization coefficient h can be estimated according to Eq. (9) with the minimized KL distance.

In implementation, we can give a fixed h_x value, run optimizing algorithm such as back-propagation to obtain a series of network parameter Θ^*, then give another h_x value, so on and so forth. We choose h_x^* such that its corresponding value of $J(h_x^*, h_z, \Theta^*)$ is the smallest. This is an exhaustive search method which is computation-expensive, but it can give an exact solution for regularization parameter.

From practical implementation consideration, in the following we will derive the formula which is approximately the estimation regularization parameter based on training data in the network parameter learning processing.

For some problems, e.g., function mapping, in special cases we can assume that $P_0(x)$ is a uniformly distributed function and regard it as h independent. With this assumption, from Eq. (1) with respect to $\frac{\partial}{\partial h_x} J(h, \Theta) = 0$, we can obtain the formula for estimating regularization parameter.

To find the minimization of Eq. (1) corresponding to h_x, we conduct the following derivation. Considering $J_1(h, \Theta)$ approximation, from Eq. (5) we obtain,

$$\begin{aligned}\frac{\partial}{\partial h_x}J(h,\Theta) &= \frac{\partial}{\partial h_x}J_1(h,\Theta)+\frac{\partial}{\partial h_x}J_2(h)\\ &\approx J_r+\frac{\partial}{\partial h_x}J_2(h).\end{aligned} \tag{29}$$

¿From Eq. (4), when $J_2(h)$ is a continuous and differentiable function, the last term of the above equation becomes

$$\frac{\partial}{\partial h_x}J_2(h)=\iint \frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}[1+\ln p_h(\mathbf{x},\mathbf{z})]d\mathbf{x}d\mathbf{z} \tag{30}$$

Note it can be proved that

$$\iint \frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}d\mathbf{x}d\mathbf{z}=0. \tag{31}$$

Proof: Because the joint kernel density $p_h(\mathbf{x},\mathbf{z})$ in this work is designed as Gaussian kernel function,

$$p_h(\mathbf{x},\mathbf{z})=\frac{1}{N}\sum_{i=1}^{N}G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z}). \tag{32}$$

We can compute the partial derivative of $p_h(\mathbf{x},\mathbf{z})$,

$$\begin{aligned}&\frac{\partial}{\partial h_x}p_h(\mathbf{x},\mathbf{z})=-\frac{d_x}{2h_x}p_h(\mathbf{x},\mathbf{z})\\ &+\frac{1}{2Nh_x^2}[\sum_{i=1}^{N}G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})||\mathbf{x}-\mathbf{x}_i||^2]\end{aligned} \tag{33}$$

$$\begin{aligned}&\iint \frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}d\mathbf{x}d\mathbf{z}=-\frac{d_x}{2h_x}\iint p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z}\\ &+\frac{1}{2Nh_x^2}\iint\sum_{i=1}^{N}G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})\\ &\times||\mathbf{x}-\mathbf{x}_i||^2d\mathbf{x}d\mathbf{z}\end{aligned} \tag{34}$$

The first term in the above equation is

$$\begin{aligned}
&-\frac{d_x}{2h_x}\iint p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} = \\
&-\frac{d_x}{2Nh_x}\sum_{i=1}^{N}\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})d\mathbf{x}d\mathbf{z} \\
&= -\frac{d_x}{2h_x}. \qquad (35)
\end{aligned}$$

As the second term is also Gaussian type integration, it can be evaluated to

$$\begin{aligned}
& \frac{1}{2Nh_x^2}\iint\sum_{i=1}^{N}G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z}) \\
& \times||\mathbf{x}-\mathbf{x}_i||^2 d\mathbf{x}d\mathbf{z} \\
= & \frac{d_x}{2h_x}. \qquad (36)
\end{aligned}$$

Then we have

$$\iint\frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}d\mathbf{x}d\mathbf{z} = -\frac{d_x}{2h_x}+\frac{d_x}{2h_x} = 0. \qquad (37)$$

■

With the above results, Eq. (30) reduces to

$$\frac{\partial}{\partial h_x}J_2(h) = \iint\frac{\partial p_h(\mathbf{x},\mathbf{z})}{\partial h_x}\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} \qquad (38)$$

That is,

$$\begin{aligned}
\frac{\partial}{\partial h_x}J_2(h) = & -\frac{d_x}{2h_x}\iint p_h(\mathbf{x},\mathbf{z})\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} \qquad (39)\\
& -\frac{1}{2Nh_x^2}\sum_{i=1}^{N}\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x}) \\
& \times G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z})||\mathbf{x}-\mathbf{x}_i||^2\ln p_h(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z}
\end{aligned}$$

For parameter optimization, the δ learning rule with learning factor being one becomes[33]

$$\delta h_x = -\frac{\partial J(h,\Theta)}{\partial h_x}. \qquad (40)$$

When minimizing $J(h, \Theta)$ with respect to h_x, the following gradient descent equation can be obtained

$$\delta h_x = -J_r + \frac{d_x}{2h_x} E_a(h), \tag{41}$$

or let $\delta h_x = 0$, we get

$$h_x = \frac{d_x E_a(h)}{2J_r} \tag{42}$$

where

$$\begin{aligned} E_a(h) &= \iint p_h(\mathbf{x}, \mathbf{z}) \ln p_h(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z} \\ &\quad - \frac{1}{N d_x h_x} \sum_{i=1}^{N} \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\ &\quad \times ||\mathbf{x} - \mathbf{x}_i||^2 \ln p_h(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z}. \end{aligned} \tag{43}$$

This is a formula for estimating regularization parameter based on training data. It can be used to optimize h_x iteratively. The integration in the above equation can be evaluated by *Monte Carlo integration*.

In practical implementation, especially for the small training data set case, we can use sparse data approximation (SDA) in Eq. (43). That is, if data i is not correlated with data j for sparse data distribution, we can consider integration at $\mathbf{x}$ around $\mathbf{x}_i$, $\mathbf{z}$ around $\mathbf{z}_i$ only, and ignore other data. With this approximation, now let us evaluate the integration in $E_a(h)$, in which the first term is

$$\begin{aligned} &\iint p_h(\mathbf{x}, \mathbf{z}) \ln p_h(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z} \\ = &\frac{1}{N} \sum_{i=1}^{N} \{ \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\ &\times \ln \sum_{j=1}^{N} G(\mathbf{x}, \mathbf{x}_j, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_j, h_z \mathbf{I}_{d_z}) d\mathbf{x} d\mathbf{z} \} \\ &- \ln N \end{aligned} \tag{44}$$

Applying sparse data approximation and considering small h, we obtain

$$\begin{aligned}
& G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\
& \times \ln \sum_{j=1}^{N} G(\mathbf{x}, \mathbf{x}_j, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_j, h_z \mathbf{I}_{d_z}) \\
\approx\ & G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\
& \times \ln\{G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z})\} \\
=\ & G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\
& \times\{-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h_x} - \frac{||\mathbf{z}-\mathbf{z}_i||^2}{2h_z} \\
& -\frac{d_x}{2}\ln(2\pi h_x) - \frac{d_z}{2}\ln(2\pi h_z)\} \qquad (45)
\end{aligned}$$

With above approximation, Eq. (44) is reduced to

$$\begin{aligned}
& \iint p_h(\mathbf{x}, \mathbf{z}) \ln p_h(\mathbf{x}, \mathbf{z}) d\mathbf{x} d\mathbf{z} \qquad (46) \\
\approx\ & -\frac{d_x}{2}[1+\ln(2\pi h_x)] - \frac{d_z}{2}[1+\ln(2\pi h_z)] - \ln N.
\end{aligned}$$

The second term in Eq. (43) is reduced to

$$\begin{aligned}
& \frac{1}{N d_x h_x}[\sum_{i=1}^{N} \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\
& \times||\mathbf{x}-\mathbf{x}_i||^2 \ln p_h(\mathbf{x}, \mathbf{z})] d\mathbf{x} d\mathbf{z} \\
\approx\ & \frac{1}{N d_x h_x} \sum_{i=1}^{N} \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\
& \times||\mathbf{x}-\mathbf{x}_i||^2[-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h_x} - \frac{||\mathbf{z}-\mathbf{z}_i||^2}{2h_z} \\
& -\frac{d_x}{2}\ln(2\pi h_x) - \frac{d_z}{2}\ln(2\pi h_z)] d\mathbf{x} d\mathbf{z} - \ln N \\
=\ & -d_x - d_x(d_x-1)^2 - \frac{d_x}{2}[1+\ln(2\pi h_x)] \\
& -\frac{d_z}{2}[1+\ln(2\pi h_z)] - \ln N \qquad (47)
\end{aligned}$$

Then Eq. (43) becomes

$$\begin{aligned}
E_a(h) &= \iint p_h(\mathbf{x},\mathbf{z}) \ln p_h(\mathbf{x},\mathbf{z}) d\mathbf{x} d\mathbf{z} \\
&- \frac{1}{N d_x h_x} \sum_{i=1}^{N} \iint G(\mathbf{x}, \mathbf{x}_i, h_x \mathbf{I}_{d_x}) G(\mathbf{z}, \mathbf{z}_i, h_z \mathbf{I}_{d_z}) \\
&\quad \times ||\mathbf{x} - \mathbf{x}_i||^2 \ln p_h(\mathbf{x},\mathbf{z}) d\mathbf{x} d\mathbf{z} \\
&\approx -\frac{d_x}{2}[1 + \ln(2\pi h_x)] - \frac{d_z}{2}[1 + \ln(2\pi h_z)] - \ln N \\
&\quad -[-d_x - d_x(d_x - 1)^2 - \frac{d_x}{2}[1 + \ln(2\pi h_x)] \\
&\quad -\frac{d_z}{2}[1 + \ln(2\pi h_z)] - \ln N] \\
&= d_x[1 + (d_x - 1)^2]
\end{aligned} \tag{48}$$

Notice that in maximum likelihood estimation,

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2 \tag{49}$$

¿From the above discussion, with Eqs. (48) and (49), in sparse data approximation case, from Eq. (42) we can obtain the following equation for rough estimation of h_x:

$$h_x \approx d_x^2 [1 + (d_x - 1)^2] \frac{\sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2}{\sum_{i=1}^{N} ||g'(x_i, W)||^2} \tag{50}$$

This is an approximate estimation of h_x by using the sum-square-error and penalty term, which is quite different from the equation obtained in Ref.[24]. In implementation, we need to find h_x and weight W by some adaptive learning algorithms. For example, we can first make some initial guess for a small non-zero value of h_x, and use this value to evaluate W by the well-known back-propagation algorithm[36], then periodically re-estimate the value of h_x by Eq. (50) in training processing. The advantage of this result is that only applying training data can be sufficient in estimating regularization coefficients, and h_x can be optimized on-line with minimized generalization error.

V. Discussion

In fact, the equation with regularization resulting from KL distance for feedforward networks is not completely equivalent to Tikhonov regularizer. Moreover, the starting point of deriving the regularization parameter estimation equation is different from the Mackey's Bayesian evidence or MAP for hyper-parameters[12],[35]. For example, Mackey assumes the *prior* distribution of weight is Gaussian with hyper-parameter as the regularization parameter, and the penalty term is in the weight decay form. While we use nonparametric kernel density distribution, a particular approximation is equivalent to Tikhonov regularizer. The penalty term is the first derivation of sum-square-errors of a network mapping function. This form is reduced to weight decay when the mapping function is in a generalized linear network, $g_j(\mathbf{x}, W) = \sum_{l=1}^{d_x} w_{j,l} x_l$. Therefore,

$$\sum_{i=1}^{N} ||g'(\mathbf{x}_i, W)||^2 = N \sum_{j=1}^{M} w_j^2 \tag{51}$$

where M represents the number of network weight parameters and w_j is an element of the matrix W in a vector expression.

With the generalized linear network assumption, Eq. (50) becomes

$$h_x \approx d_x^2 [1 + (d_x - 1)^2] \frac{\sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2}{N \sum_{j=1}^{M} w_j^2} \tag{52}$$

Now let us see the similarity of MAP approximation with our result in estimating the regularization parameter.

The cost function in Mackey's Bayesian inference is[12],[35]

$$S(w) = \frac{\beta}{2} \sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, W)||^2 + \frac{\alpha}{2} \sum_{j=1}^{M} w_j^2 \tag{53}$$

In minimizing this cost function to find the network weight parameter W, the effective value of the regularization parameter depends only on the ratio α/β, since an overall multiplicative factor is unimportant. This means h_x should be equivalent to α/β under some approximations.

In Mackey's results[12],[35], a very rough approximation condition is $\gamma = M$ and $N \gg M$.

$$\gamma \equiv \sum_{j=1}^{M} \frac{\lambda_j}{\lambda_j + \alpha} \tag{54}$$

where $\{\lambda_j\}$ denotes the eigenvalues of $\mathbf{H}$, the Hessian of unregularized cost function,

$$\mathbf{H} = \beta \nabla_w^2 E_D, \qquad E_D = \frac{1}{2} \sum_{i=1}^{N} ||\mathbf{z}_i - g(\mathbf{x}_i, w)||^2 \tag{55}$$

The matrix $\mathbf{A}$ is related to parameter α in the following form,

$$\mathbf{A} = \mathbf{H} + \alpha \mathbf{I}. \tag{56}$$

In order to compare with Mackey's formula, we rewrite the parameters α and β from[12],[35] in the following:

$$\beta = N/2E_D = N/\sum_{i=1}^{N} \{\mathbf{z}_i - g(\mathbf{x}_i, w)\}^2 \tag{57}$$

$$\alpha = M/2E_W = \frac{M}{\sum_{j=1}^{M} w_j^2} \tag{58}$$

Consequently,

$$\frac{\alpha}{\beta} = M \frac{\sum_{i=1}^{N} \{\mathbf{z}_i - g(\mathbf{x}_i, w)\}^2}{N \sum_{j=1}^{M} w_j^2} \tag{59}$$

Here we can clearly note the similarity between h_x in Eq. (52) and α/β in Eq.(59), where their difference is only the constant coefficient. In h_x estimation, the constant coefficient is dependent on the dimension of input space, while in α/β estimation, the constant coefficient is the dimension of weight parameter vector. This can be explained by the fact that Mackey's result is obtained in parameter space approximation, while our result is in data space approximation. Compared to the approximation condition, our approximation is based on the sparse data set, which is a reasonable approximation for the small-number training data set case. While in Mackey's approximation, it requires $N \gg M$. In the following function mapping experiments, we design that $N = 30$, $d_x = d_z = 1$, the hidden neuron number is $k = 15$, and $M = (d_x + 1) \times k + k \times d_z = 45$. Because the experimental condition does not satisfy Mackey's very rough approximation condition $N \gg M$, it cannot be successful in estimating regularization parameter on-line with Eq. (59). In fact, the condition $N \gg M$ means that training sample number should be large enough compared to network complexity. If we have enough training samples, the generalization is also improved without regularization[6].

As we know, there is no free lunch for the optimization problem. To get the best regularization parameter value, the parameter numerical evaluation involves computation of Hessian matrix and log determinant of $\mathbf{A}^{-1}$, as well as eigenvalues of Hessian in Mackey's Bayesian inference. While in our approximation, it involves integration in data space. To save computational cost and on-line optimizing regularization parameter, a rough approximation is needed, but in this case the parameter value may not be the best one, and generalization error may not be the smallest with approximations.

VI. Experiments

Several experiments have been done with dynamically adjusting regularization parameter h_x. The network structure used in the experiments is shown in Figure 1.

In the implementation, we train the three-layer neural network by back-propagation algorithm. The regularization term used in training processing is Eq. (51) with regularization parameter h_x. At the beginning of the training processing a small value of h_x is initialized, then it is periodically re-estimated by Eq. (52). The training processing is stopped until the total error J_1 is minimized, measured by either successive error difference being less than 10^{-8} or over 10^4 training epoch being passed. Followings are the pseudo-code for the algorithm described above.

```
/* Initializing weight parameters W and h_x
/* with small random values.
/* Set the BP learning factor Mu and an integer value Icf
/* for periodically re-estimating h_x.

For t = 1 to 10^(4),
   Net_output = S(W_z|y S(W_y|x X)),
   Net_error = ||Target_Z - Net_output||^2,
   Reg_term = N*Sum(w_i^2),
   Js(t) = Net_error/(2N),
   W(t) = W(t-1) - Mu* Grad_w[J1(t-1)],
   J1(t) = Js(t) + h_x* Reg_term/(2N),
   If t MOD Icf == 0,
       h_x = Net_error/Reg_term,
   Else Continue.
   If |J1(t)-J1(t-1)|<10^(-8),
       Goto End,
   Else Continue.
Next t
End
```

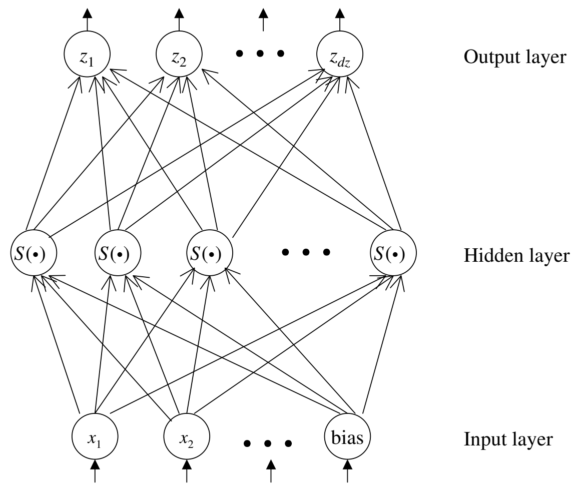


Fig. 1. The three-layer neural network architecture schematic map.

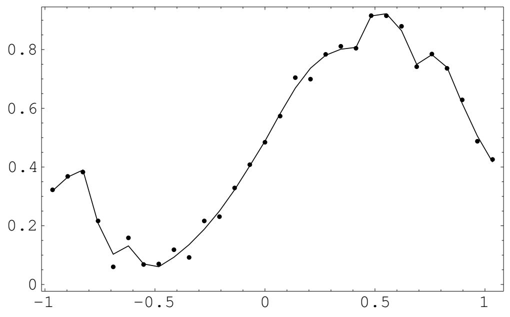
Some results are drawn in Figures 2–8. The results show that the optimal regularization parameter h_x can be found by seeking the minimum of $J(h, \Theta)$ with the training data set only. We also apply the minimal generalization error method to validate the experimental results, and the same order of h_x has been obtained (see Figure 4). This confirms that the new parameter estimation formula is a good approximation. Unlike early stopping strategy, this new regularization parameter formula can work for overtrained network and does not need another validation set to guard when the training should stop.

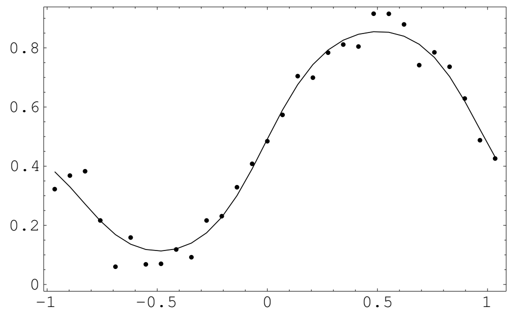
The function mapping problem was considered in the experiments, and the sine and exponential functions were applied. In order to represent sufficient network complexity, we used 15 hidden neurons in a three-layer network. Only 30 training samples were generated with Gaussian noise added to the output. With this kind of network architecture, if without regularization, the phenomenon of over-fitting to noise can be observed as shown in Figure 2. In Figures 2 and 3, it is shown that with regularization, the network output is smoothed and generalization performance is improved. Figure 4 shows that the minimal J_1 value indicates h_x value around 10^{-4}.

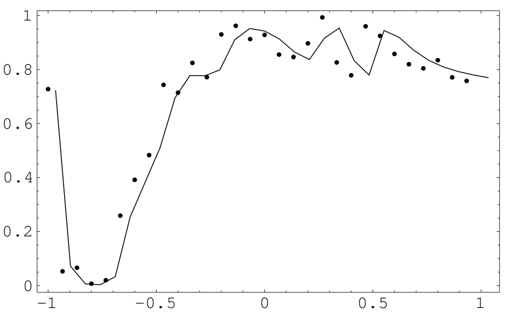
Real-world data sets are used in the experiments too. The data sets are software failure data sys1 and sys3, which are contained in the attached Compact Disk of the *Handbook of software Reliability Engineering* [34]. The sys1 data set contains 54 data points. In order to validate the parameter estimation results, we partition the sys1 data into two parts: a training set and a validation set. The training set consists of 37 samples which are randomly drawn from the original data set. The remaining 17 samples comprise the validation set. The data sets are normalized to the range of values [0,1]. Normalization is a standard procedure for data preprocessing. In the software reliability investigation problem, the network input is successive normalized failure occurrence times, and the network output is the accumulated failure numbers. During the training phase, each input sample x_t at time t is associated with the corresponding output value z_t at the same time t. The experimental results are shown in Figures 5–7. From Figure 6, it can be observed that with regularization, the validation error is less than that without regularization. Figure 7 shows that the minimal J_1 value indicates h_x in the range of 10^{-8} to 10^{-10}, while dynamically-estimated h_x value is 1.17×10^{-8}.

Another data set is sys3, which contains 278 data points. In the experiment, the number of training data is about 2/3 of the total data number. That is, it consists of 186 randomly-drawn samples from the original data set. The remaining 92 samples form the validation set. Because this data set is a bit large and the noise is small, it makes no obvious difference in the obtained results with respect to dynamical regularization. The trained network output is shown in Figure 8.

Experiments have been done for the comparison of regularization parameter estimation formula Eq. (59) and Eq. (52) performance. From the results we observe that the estimator is problem-dependent, and it is hard to say that one estimator is better in all cases. For the case when $N > M$ or $N \sim M$, MAP-approximation-based regularization parameter estimation formula performance is good, sometimes better than SDA-based formula. However, when we use many of hidden neurons, for the case $N < M$, MAP-approximation-based formula performance becomes poor.


(a) Without regularization


(b) With regularization


(c) Without regularization

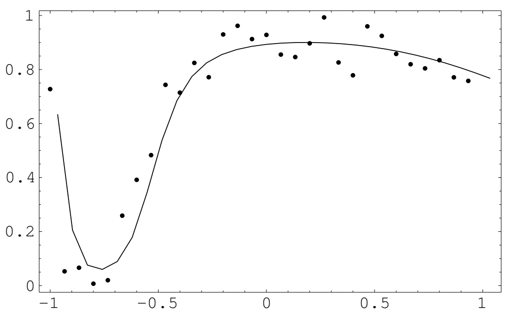
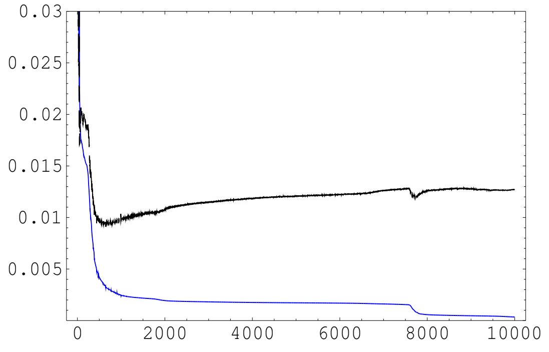
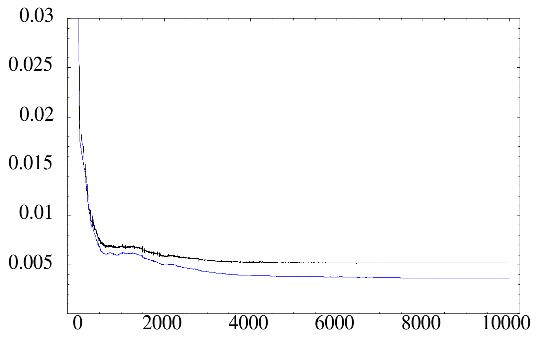

(d) With regularization

Fig. 2. The neural network input-output. Dots are training samples, while solid line is network output. (a, b) are for the sine function approximation problem. After training is stopped, dynamically-estimated $h_x = 2.87 \times 10^{-4}$. (c, d) are for the exponential function approximation problem. After the training is stopped, dynamically-estimated $h_x = 1.27 \times 10^{-4}$.



(a) Without regularization



(b) With regularization

Fig. 3. Training epoch for the exponential function approximation problem. Upper line represents validation error, while lower line depicts training error. Without regularization, training error is small while validation error is large. With regularization, validation error is reduced and training error is increased a little, illustrating that over-fitting does not occur.

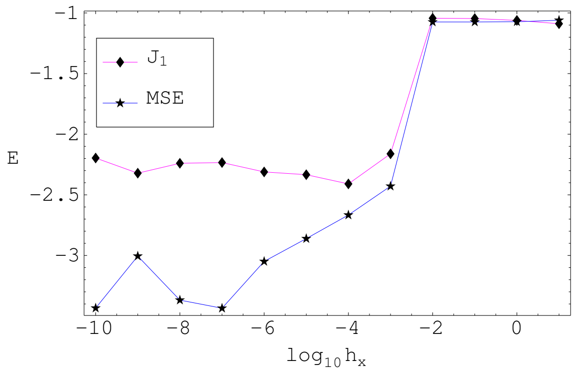


Fig. 4. The training mean square error (MSE) on the training data set and J_1 on the validation data set, plotted versus the smooth parameter h_x. The network was trained by 30 samples which are drawn from the exponential function. We use a validation data set with 30 data points to calculate J_1 value again after the training is stopped. For each h_x value, the network was trained until the total error J_1 (Eq. (28)) was minimized, measured by successive error difference being less than 10^{-8} or over 10^4 epoch being passed. The minimal J_1 indicates an optimal $\log_{10} h_x \approx -4$. Dynamically-estimated h_x value is 1.27×10^{-4} in this case.

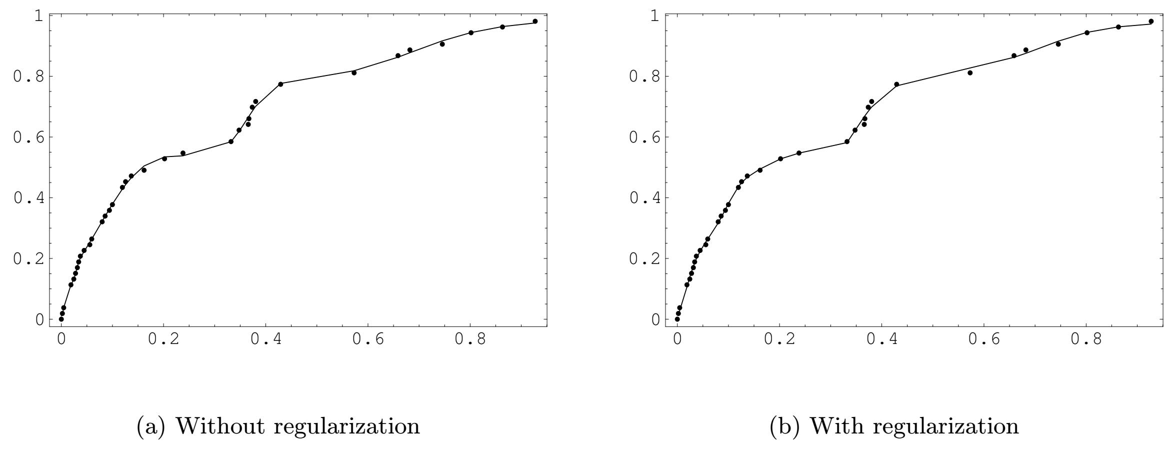


Fig. 5. The neural network input-output. Dots are training samples, while solid line is the network output. Software reliability growth model approximation is applied to data set sys1. After training is stopped, dynamically-estimated $h_x = 1.17 \times 10^{-8}$. Because the noise is very small, the difference with and without regularization is not obvious.

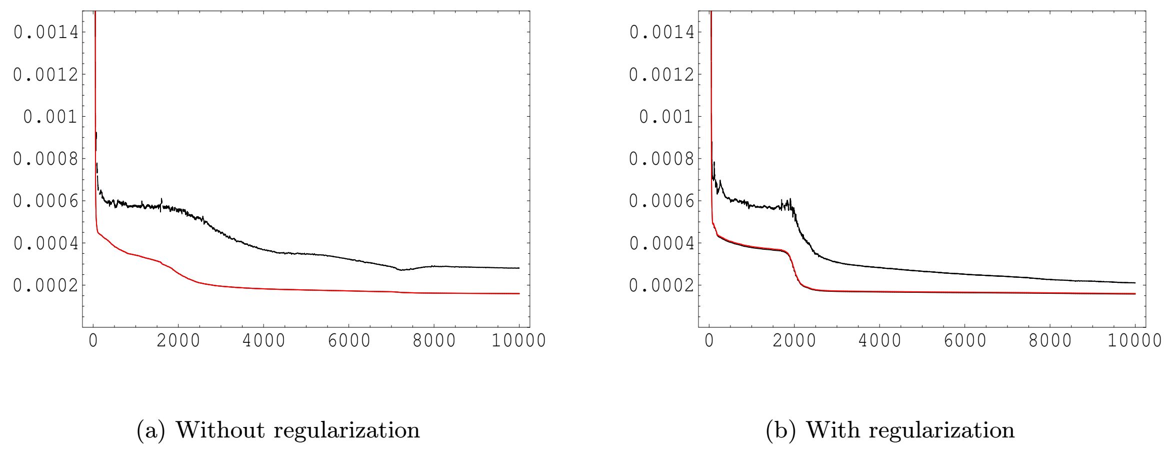


Fig. 6. Training epoch for the software reliability growth model data set sys1. Upper line represents validation error, while lower line depicts training error. Without regularization, training error is small while validation error is a bit large. With regularization, validation error is reduced.

VII. Conclusion

In this paper, we show that one particular case of the system entropy with Gaussian probability density reduces into the first order Tikhonov regularizer for feedforward neural networks in the maximum likelihood learning case, where the regularization parameter is the smoothing parameter h_x in the kernel density function. Under the framework of Kullback-Leibler distance, we derive the formula for approximately estimating regulariza-

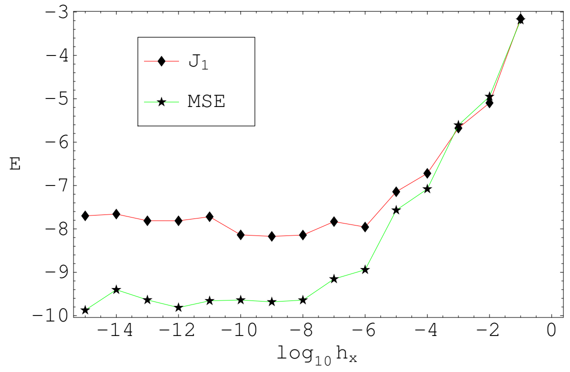


Fig. 7. The training mean square error (MSE) on the training data set and J_1 on the validation data set, plotted versus the smooth parameter h_x. The network was trained by 37 samples which are drawn from the sys1 data set. We use a validation data set with 17 data points to calculate J_1 value again after training is stopped. For each h_x value, the network was trained until the total error J_1 was minimized, measured by over 10^4 epoch being passed. The minimal J_1 indicates an optimal value around $\log_{10} h_x \approx -9$. Dynamically-estimated h_x value is 1.17×10^{-8} in this case.

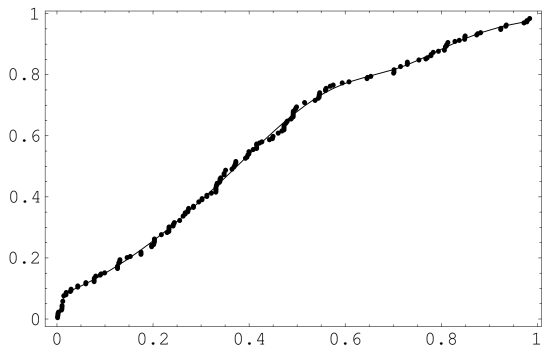


Fig. 8. The neural network input-output. Dots are training samples, while solid line is the network output. For software reliability growth model data set sys3, regularization does not make a significant difference.

tion parameter using training data. Experiments show that our estimated regularization parameter is in the same order as that estimated by validation method. However, our method requires much less computational resource than the validation search method.

Acknowledgement

The authors would like to thank the anonymous referees for their constructive comments on this paper. This research work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project No. CUHK4193/00E).

References

[1] Y. Le Cun, J.S. Denker and S.A. Solla, “Optimal Brain Damage,” in *Advanced in Neural Information Processing Systems*, D. S. Touretzky, Ed., San Mateo, CA, 1990, vol. 2, pp. 598–605, Morgan Kaufmann Publisher.

[2] Lars K. Hansen and Carl E. Rasmussen, “Pruning from Adaptive Regularization,” *Neural Computation*, vol. 6, no. 6, pp. 1222–1231, 1994.

[3] F. Girosi, M. Jones and T. Poggio, “Regularization Theory and Neural Networks Architectures,” *Neural Computation*, vol. 7, pp. 219–269, 1995.

[4] Lizhong Wu and John Moody, “A Smoothing Regularizer for Feedforward and Recurrent Neural Networks,” *Neural Computation*, vol. 8, no. 3, pp. 463–491, 1996.

[5] G. E. Hinton, “Learning Translation Invariant Recognition in Massively Parallel Networks,” in *Proceedings PARLE Conference on Parallel Architectures and Languages Europe*, A. J. Nijman J.W. de Bakker and P. C. Treleaven, Eds., Berlin, 1987, pp. 1–13, Springer-Verlag.

[6] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.

[7] Yves Grandvalet and Stephane Canu, “Noise Injection: Theoretical Prospects,” *Neural Computation*, vol. 9, no. 5, pp. 1093–1108, 1997.

[8] Alan M. Thompson, John C. Brown, Jim W. Kay and D. Michael Titterington, “A Study of Methods of Choosing the Smoothing Papprameter in Image Restoration by Regularization,” *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 326–339, 1991.

[9] Peter R. Johnston and Ramesh M. Gulrajani, “A New Method for Regularization Parameter Determination in the Inverse Problem of Electrocardiography,” *IEEE Trans. on Biomedical Engineering*, vol. 44, no. 1, pp. 19–39, January 1997.

[10] G. Wahba, *Spline Models for Observational Data*, vol. 59 of *CBMS-NSF regional conference series in applied mathematics*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[11] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*, Chaoman and Hall, London, 1993.

[12] D. J. C. MacKay, “Bayesian Interpolation,” *Neural Computation*, vol. 4, no. 3, pp. 415–447, 1992.

[13] J. Larsen, L.K. Hansen, C. Svarer and M. Ohlsson, “Design and Regularization of Neural Networks: the Optimal Use of a Validation Set,” in *Proceedings of the 1996 IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing*, S. Usui, Y. Tohkura, S. Katagiri and E. Wilson, Ed., 1996, vol. VI, pp. 62–71.

[14] L. Nonboe Andersen, J. Larsen, L.K. Hansen and M. Hintz-Madsen, “Adaptive Regularization of Neural Classifiers,” in *Proceedings of the 1997 IEEE Workshop on Neural Networks for Signal Processing*, J. Principe, L. Gile, N. Morgan and E. Wilson, Ed., 1997, vol. VII, pp. 24–33.

[15] Dingding Chen and M. T. Hagan, “Optimal Use of Regularization and Cross-validation in Neural Network Modeling,” in *Proceedings of the 1999 International Joint Conference on Neural Networks*, 1999, vol. 2, pp. 1275–1280.

[16] Katsuyuki hagiwara and Kazuhiro Kuno, “Regularization Learning and Early Stoping in Linear Networks,” in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, S-I, Amari, C.L. Giles, M. Gori and V. Piuri, Ed., 2000, vol. 4, pp. 511–516.

[17] Isabelle Rivals and Leon Personnaz, “On Cross Validation for Model Selection,” *Neural Computation*, vol. 11, pp. 863–870, 1999.

[18] S. Kullback, *Information Theory and Statistics*, Wiley, New York, 1959.

[19] L. Devroye, *A Course in Density Estimation*, Birhhauser Publisher, Boston, 1987.

[20] D. Bosq, *Nonparametric Statistics for Stochastic Processes: Estimation and Prediction*, Springer-Verlag Inc., New York, 1996.

[21] C. O. Wu,"A Cross-Validation Bandwidth Choice for Kernel Density Estimates with Selection Biased Data", *Journal of Multivariate Analysis* vol. 61, pp. 38–60, 1997.

[22] C. Gu, "Model indexing and smoothing parameter selection in nonparametric function estimation (with discussion)", *Statistica Sinica*, vol. 8, No. 3, pp. 607–646, 1998.

[23] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, Boston, second edition, 1990.

[24] Lei Xu, "Bayesian Ying-Yang System and Theory as A Unified Statistical Learning Approach (VII): Data Smoothing," in *Proceedings of Intentional Conference on Neural Information Processing*, Kitakyushu, Japan, 1998, 1, pp. 243–248.

[25] J. Rissanen, "Modeling by Shortest Data Description," *Automatica*, vol. 14, pp. 465–471, 1978.

[26] Andrew Barron, Jorma Rissanen and Bin Yu, "The Minimum Description Length Prnciple in Coding and Modeling," *IEEE Trans. on Information Theory*, vol. 44, no. 6, pp. 2743–2760, October 1998.

[27] George S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, Springer-Verlag, New York, 1996.

[28] James E. Gentle, *Random Number Generation and Monte Carlo Methods*, Springer, New York, 1998.

[29] C. M. Bishop, "Training with Noise is Equivalent to Tikhonov Regularization," *Neural Computation*, vol. 7, no. 1, pp. 108–116, 1995.

[30] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-posed Problems*, V. H. Winston and Sons, Washington D.C., 1977.

[31] Russell Reed, Robert J. Marks II, and Seho Oh, "Simiarities of Error Regularization, Sigmoid Gain Scaling, Target Smoothing, and Training with Jitter," *IEEE Trans. Neural Networks*, vol. 7, no. 3, pp. 529–538, 1995.

[32] C. M. Bishop, "Regularization and Complexity Control in Feed-forward Networks," Technical Report NCRG/95/022, Aston University, Birmingham, UK, 1995.

[33] R. A. Jacobs, "Increased Rates of Convergence through Learning Rate Adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[34] Michael R. Lyu, *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, McGraw Hill, 1996.

[35] D. J. C. MacKay, "A Practical Bayesian Framework for Backpropagation Networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[36] D.E. Rumelhurt, G.E. Hinton and R.J. Williams, "Learning internal representations by error propagating," in *Parallel Distributed Processing*, MIT Press (Cambridge), vol. 1, pp. 318–362, 1986.

photo_guo.jpg

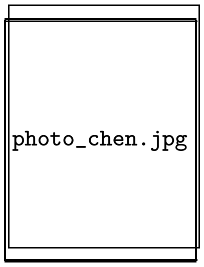
Ping Guo is currently a Professor at the Computer Science Department of the Beijing Normal University. From 1993 to 1994 he was with the Department of Computer Science & Engineering at the Wright State University as a visiting faculty. From May 2000 to August 2000 he was with the National Laboratory of Pattern Recognition at Chinese Academy of Sciences as a guest researcher. He received his M.S. degree in physics from Peking University, his Ph.D degree in Computer Science from the Chinese University of Hong Kong. His current research interests include neural network, image process, software reliability engineering, optical computing and spectra analysis.

photo_lyu.jpg

Michael R. Lyu is currently a Professor at the Computer Science and Engineering department of the Chinese University of Hong Kong. He worked at the Jet Propulsion Laboratory as a Technical Staff Member from 1988 to 1990. From 1990 to 1992 he was with the Electrical and Computer Engineering Department at the University of Iowa as an Assistant Professor. From 1992 to 1995, he was a Member of the Technical Staff in the Applied Research Area of the Bell Communications Research (Bellcore). From 1995 to 1997 he was a research Member of the Technical Staff at Bell Labs., which was first part of AT&T and later became part of Lucent Technologies.

Dr. Lyu's research interests include software reliability engineering, distributed systems, fault-tolerant computing, wireless communication networks, Web technologies, digital library, and E-commerce systems. He has published over 120 refereed journal and conference papers in these areas. He has participated in more than 30 industrial projects, and helped to develop many commercial systems and software tools. He has been frequently invited as a keynote or tutorial speaker to conferences and workshops in U.S., Europe, and Asia. He initiated the first International Symposium on Software Reliability Engineering (ISSRE) in 1990. He was the program chair for ISSRE'96, and has served in program committees for many conferences, including ISSRE, SRDS, HASE, ICECCS, ISIT, FTCS, ICDSN, EUROMICRO, APSEC, PRDC, PSAM and ICCCN. He is the General Chair for ISSRE'2001, and the WWW10 Program Co-Chair. He is the editor for two book volumes: Software Fault Tolerance, published by Wiley in 1995 and the Handbook of Software Reliability Engineering, published by IEEE and McGraw-Hill in 1996. He is an associated editor of IEEE Transactions on Reliability, IEEE Transactions on Knowledge and Data Engineering, and Journal of Information Science and Engineering.

Dr. Lyu received his B.S. in Electrical Engineering from National Taiwan University in 1981, his M.S. in Computer Engineering from University of California, Santa Barbara, in 1985, and his Ph.D. in Computer Science from University of California, Los Angeles, in 1988.



C.L. Philip Chen received his M.S. degree from the University of Michigan, Ann Arbor, Michigan, in 1985, and a Ph.D. degree from Purdue University, West Lafayette, Indiana, in Dec. 1988. In 1988-1989, he was a visiting Assistant Professor at the School of Engineering and Technology, Purdue University, Indianapolis, Indiana. Since September 1989, he has been at the Computer Science and Engineering Department, Wright State University, Dayton, Ohio, where he is currently a Professor.

He was a Conference Co-Chairman of the International Conference on Artificial Neural Networks in Engineering (ANNIE), 1995 and 1996, a Tutorial Chairman of Int'l Conference on Neural Networks, 1994, a Conference Co-Chairman of the Adaptive Distributed Parallel Computing, 1996, a Technical Committee of ANNIE, 1994-2002, a Program Committee of the IEEE Int'l Conference on Robotics and Automation, 1996 and 2001, and Int'l Conf. on IEEE/JRS Intelligent Robotics and Systems (IROS), 1998-2002.

List of Figures

1 The three-layer neural network architecture schematic map. 119

2 The neural network input-output. Dots are training samples, while solid line is network output. (a, b) are for the sine function approximation problem. After training is stopped, dynamically-estimated $h_x = 2.87 \times 10^{-4}$. (c, d) are for the exponential function approximation problem. After the training is stopped, dynamically-estimated $h_x = 1.27 \times 10^{-4}$. 121

3 Training epoch for the exponential function approximation problem. Upper line represents validation error, while lower line depicts training error. Without regularization, training error is small while validation error is large. With regularization, validation error is reduced and training error is increased a little, illustrating that over-fitting does not occur. 122

4 The training mean square error (MSE) on the training data set and J_1 on the validation data set, plotted versus the smooth parameter h_x. The network was trained by 30 samples which are drawn from the exponential function. We use a validation data set with 30 data points to calculate J_1 value again after the training is stopped. For each h_x value, the network was trained until the total error J_1 (Eq. (28)) was minimized, measured by successive error difference being less than 10^{-8} or over 10^4 epoch being passed. The minimal J_1 indicates an optimal $\log_{10} h_x \approx -4$. Dynamically-estimated h_x value is 1.27×10^{-4} in this case. 122

5 The neural network input-output. Dots are training samples, while solid line is the network output. Software reliability growth model approximation is applied to data set sys1. After training is stopped, dynamically-estimated $h_x = 1.17 \times 10^{-8}$. Because the noise is very small, the difference with and without regularization is not obvious. 123

6 Training epoch for the software reliability growth model data set sys1. Upper line represents validation error, while lower line depicts training error. Without regularization, training error is small while validation error is a bit large. With regularization, validation error is reduced. 123

7 The training mean square error (MSE) on the training data set and J_1 on the validation data set, plotted versus the smooth parameter h_x. The network was trained by 37 samples which are drawn from the sys1 data set. We use a validation data set with 17 data points to calculate J_1 value again after training is stopped. For each h_x value, the network was trained until the total error J_1 was minimized, measured by over 10^4 epoch being passed. The minimal J_1 indicates an optimal value around $\log_{10} h_x \approx -9$. Dynamically-estimated h_x value is 1.17×10^{-8} in this case. 124

8 The neural network input-output. Dots are training samples, while solid line is the network output. For software reliability growth model data set sys3, regularization does not make a significant difference. 124