Regularization Parameter Estimation for Feedforward Neural Networks

Ping Guo, Michael R. Lyu, *Senior Member, IEEE,* and C. L. Philip Chen, *Senior Member, IEEE*

***Abstract*—Under the framework of the Kullback–Leibler (KL) distance, we show that a particular case of Gaussian probability function for feedforward neural networks (NNs) 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.**

***Index Terms*—Regularization parameter estimation, small training data set, Tikhonov regularizer.**

I. Introduction

It is well known that the goal of training neural networks (NNs) 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 NN, 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. Controlling an appropriate complexity of the network can improve generalization. There are two main approaches for this purpose: 1) model selection and 2) regularization. Model selection for a feedforward NN 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 the addition of artificial noise to the inputs during training [6], [7].

The regulation 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 the Bayesian method [12]. Most work uses a validation set to select the regularization parameter [13]–[16]. This requires the splitting of 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 the 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 has been a lot of research work conducted on smoothing parameter estimation of kernel density function [21]–[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 modeled 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 KL 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

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

Manuscript received May 21, 2001; revised December 17, 2001. This work was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project CUHK4193/00E). This paper was recommended by Associate Editor N. Pal.

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

M. R. Lyu is with the Department of Computer Science and 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 Electrical and Computer Engineering, The University of Texas, San Antonio, TX 78249–0669 USA (email: pchen@utsa.edu).

Digital Object Identifier 10.1109/TSMCB.2003.808176

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

We define

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

$$J_2(h) \equiv \iint p_h(\mathbf{x},\mathbf{z}) \ln p_{h0}(\mathbf{x},\mathbf{z})\, d\mathbf{x}\, d\mathbf{z}$$

$$p_{h0}(\mathbf{x},\mathbf{z}) \equiv \frac{p_h(\mathbf{x},\mathbf{z})}{P_0(\mathbf{x})}. \quad (4)$$

$J_1(h,\Theta)$ 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, (1) becomes

$$J(h,\Theta) = J_1(h,\Theta) + J_2(h). \quad (5)$$

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

$$p_{h_x}(\mathbf{x}) = \frac{1}{N}\sum_{x_i\in D} K_{h_x}(\mathbf{x}-\mathbf{x}_i)$$

$$K_{h_x}(\mathbf{x}-\mathbf{x}_i) = \frac{1}{h_x^d} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_x}\right) \quad (6)$$

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

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

The kernel density function used the most is Gaussian kernel

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

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 the representative of which permits the shortest coding of the data, then the system should be optimized with optimal or *ideal* code length. The parameters h_x, h_z should be chosen with minimized KL distance function based on the given data set according to

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

where Θ^* is the learned NN parameter and $J(h,\Theta)$ is represented by (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

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

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 (7) and (10) into (3), integration can be approximated by summation, and we obtain

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

where

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{e}_x, \quad \mathbf{z}'_i = \mathbf{z}_i + \mathbf{e}_z. \quad (12)$$

$\mathbf{e}_x, \mathbf{e}_z$ 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, which we use in this paper, is the Taylor expansion approximation for an integral

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

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

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

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

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

$W = \{W_{z\,|\,y}, W_{y\,|\,x}\}$ 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, and $W_{z\,|\,y}$ 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

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

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

Considering that random noise is added to the input data only during training, Bishop [29] proven that in the ML estimation case, (11) can be reduced to the first-order Tikhonov regularizer [30] for feedforward NN 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 (13).

Let $f(\mathbf{x},\mathbf{z},w) = \|\mathbf{z} - g(\mathbf{x},W)\|^2$, $f(\mathbf{x},\mathbf{z},w)$ be 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

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

Equation (14) becomes

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

Notice that for any density function, the integration in the whole space should be equal to 1, i.e.

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

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

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

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

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

$$\begin{aligned} &\quad = h_x\{\|g'(\mathbf{x},W)\|^2 - \|[\mathbf{z}_i - g(\mathbf{x}_i,W)]g''(\mathbf{x}_i,W)\|\} \\ &\iint G(\mathbf{x},\mathbf{x}_i,h_x\mathbf{I}_{d_x})\, G(\mathbf{z},\mathbf{z}_i,h_z\mathbf{I}_{d_z}) \left[\frac{1}{2}(\Delta\mathbf{z})^T f''_z\Delta\mathbf{z}\right] d\mathbf{x}\, d\mathbf{z} \\ &\quad = \frac{h_z}{2}\,\mathrm{trace}[f''_z] = d_z h_z. \end{aligned} \tag{23}$$

With the above results, the integration becomes

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

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 (24) is irrelevant to the weight parameter, and can be neglected as well [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

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

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

In (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 field at a point, and assumed it to be 0. 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 ML 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 (9) with the minimized KL distance.

In implementation, we can give a fixed h_x value, run an optimizing algorithm such as back-propagation to obtain a series of network parameter Θ^*, then give another h_x value, and so on. 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, next 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 (1) with respect to $(\partial/\partial h_x)J(h, \Theta) = 0$, we can obtain the formula for estimating regularization parameter.

To find the minimization of (1) corresponding to h_x, we conduct the following derivation. Considering $J_1(h, \Theta)$ approximation, from (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 (4), when $J_2(h)$ is a continuous and differentiable function, the last term of (29) 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 proven 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 $p_h(\mathbf{x},\mathbf{z})$

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

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

The first term in (34) is

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

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

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

Then we have

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

■

With the above results, (30) reduces to

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

That is

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

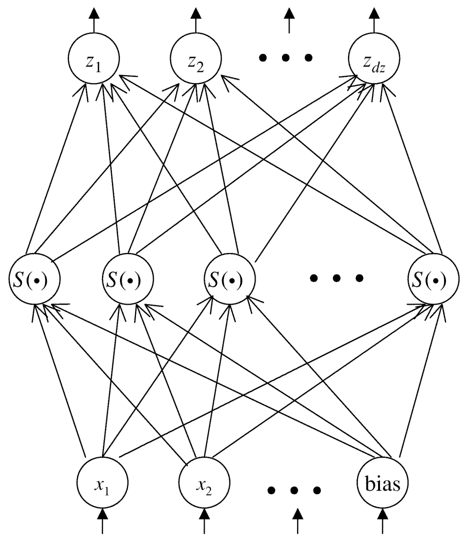

Fig. 1. Three-layer NN architecture schematic map.

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

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

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

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

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 (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

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

Applying SDA and considering small h, we obtain

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

With this approximation, (46) is reduced to

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

The second term in (43) is reduced to

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

Then, (43) becomes

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

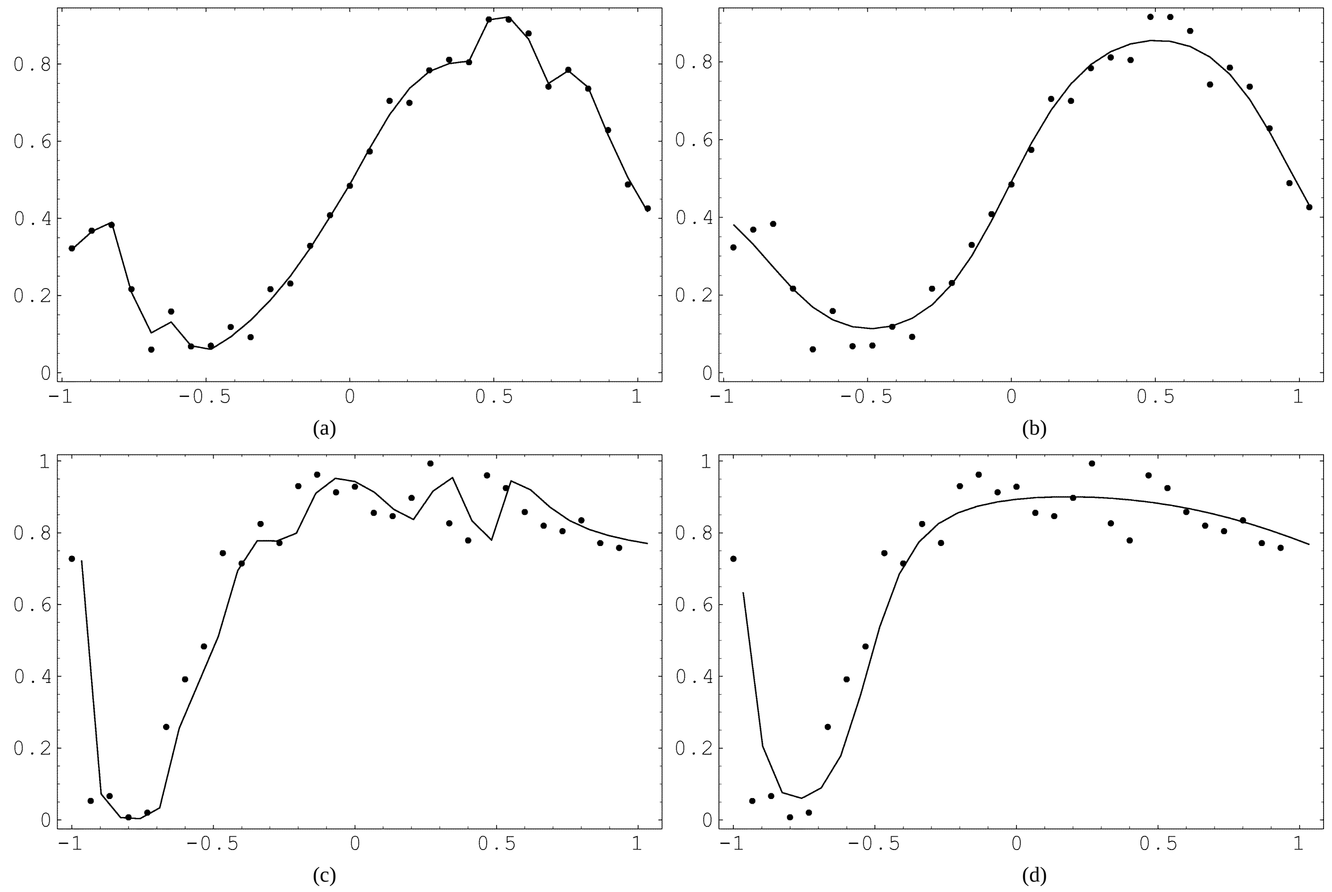

Fig. 2. The NN input-output. Dots are training samples, while solid line is network output. (a) and (b) Sine function approximation problem. After training is stopped, dynamically-estimated $h_x = 2.87 \times 10^{-4}$. (c) and (d) Exponential function approximation problem. After the training is stopped,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

Notice that in ML 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 (48) and (49), in the SDA case, from (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 [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 nonzero value of h_x, and use this value to evaluate W by the well-known back-propagation algorithm [36], then periodically reestimate the value of h_x by (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 the 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 the 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, (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.

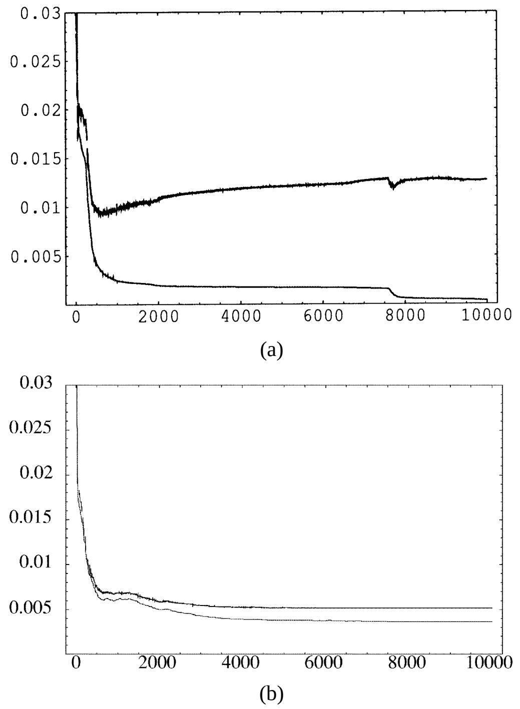

Fig. 3. Training epoch for the exponential function approximation problem. The upper line represents validation error, while the lower line depicts training error. Without regularization, training error is small while validation error is large. With regularization, validation error is reduced and training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

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

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

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:

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

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

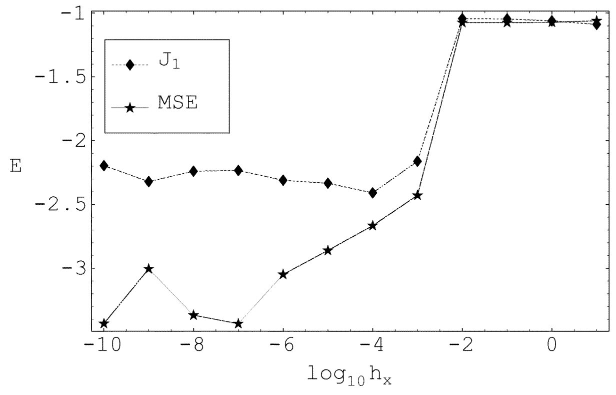

Fig. 4. Training mean square error (MSE) on the training data set and J_1 on the validation data set, plotted versus the smooth parameter h_x. The network was trained by 30 samples which are drawn from the exponential function. We use a validation data set with 30 data points to calculate J_1 value again after the training is stopped. For each h_x value, the network was trained until the total error J_1 [(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.

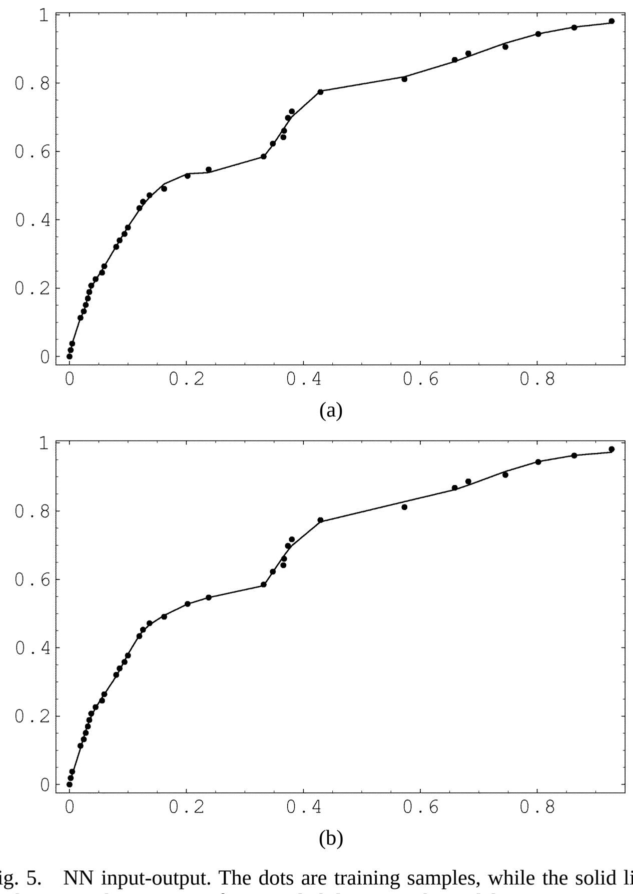

Fig. 5. NN input-output. The dots are training samples, while the solid line is the network output. Software reliability growth model approximation is applied to data set sys1. After training is stopped, dynamically estimated $h_x = 1.17 \times 10^{-8}$. Because the noise is very small, the difference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

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 (52) and α/β in (59), where their difference is only the constant

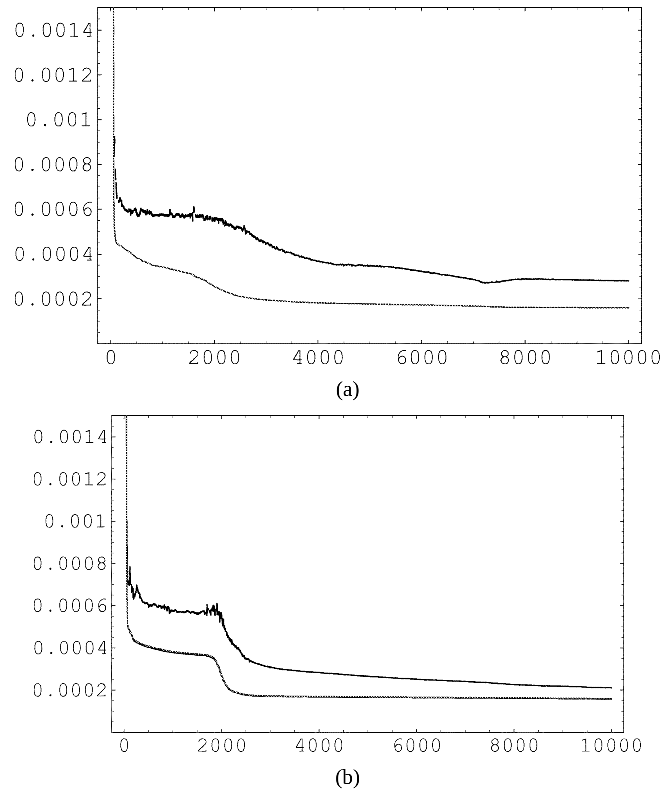

Fig. 6. Training epoch for the software reliability growth model data set sys1. The upper line represents validation error, while the lower line depicts training error. Without regularization, training error is small while validation error is a bit large. With regularization, validation error is reduced.

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 (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 Fig. 1.

In the implementation, we train the three-layer NN by back-propagation algorithm. The regularization term used in training processing is (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 reestimated by (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 pseudocode 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
```

Some results are drawn in Figs. 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 Fig. 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 Fig. 2. In Figs. 2 and 3, it is shown that with regularization, the network output is smoothed and generalization performance is improved. Fig. 4 shows that the minimal J_1 value indicates h_x value around 10^{-4}.

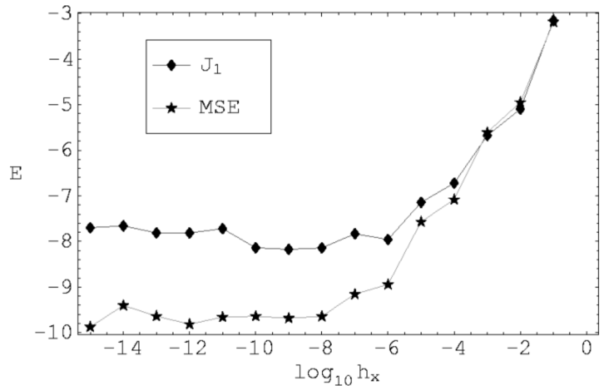

Fig. 7. Training mean square error (MSE) on the training data set and J_1 on the validation data set, plotted versus the smooth parameter h_x. The network was trained by 37 samples which are drawn from the sys1 data set. We use a validation data set with 17 data points to calculate J_1 value again after 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.

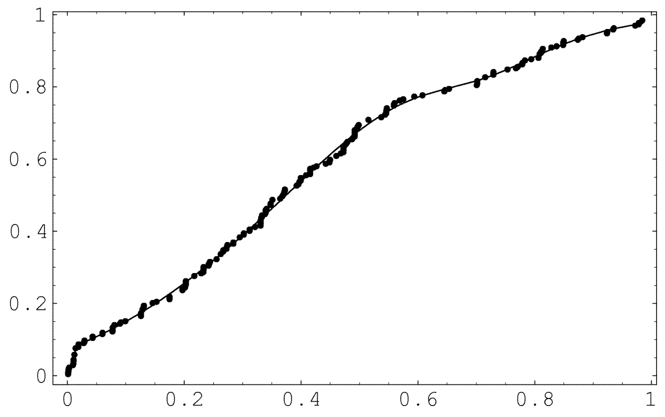

Fig. 8. NN input-output. The dots are training samples, while solid line is the network output. For software reliability growth model data set sys3, regularization does not make a significant difference.

Real-world data sets are used in the experiments as well. 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: 1) a training set and 2) 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 Figs. 5–7. From Fig. 6, it can be observed that with regularization, the validation error is less than that without regularization. Fig. 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 Fig. 8.

Experiments have been done for the comparison of regularization parameter estimation formula (59) and (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.

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 NNs in the ML learning case, where the regularization parameter is the smoothing parameter h_x in the kernel density function. Under the framework of KL distance, we derive the formula for approximately estimating regularization 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.

Acknowledgment

The authors would like to thank the anonymous referees for their constructive comments on this paper.

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: Morgan Kaufmann, 1990, vol. 2, pp. 598–605.

[2] L. K. Hansen and C. E. Rasmussen, "Pruning from adaptive regularization," *Neural Comput.*, vol. 6, no. 6, pp. 1222–1231, 1994.

[3] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Comput.*, vol. 7, pp. 219–269, 1995.

[4] L. Wu and J. Moody, "A smoothing regularizer for feedforward and recurrent neural networks," *Neural Comput.*, vol. 8, no. 3, pp. 463–491, 1996.

[5] G. E. Hinton, "Learning translation invariant recognition in massively parallel networks," in *Proc. PARLE Conf. Parallel Architectures and Languages Europe*, A. J. Nijman, J. W. de Bakker, and P. C. Treleaven, Eds. Berlin, Germany, 1987, pp. 1–13.

[6] C. M. Bishop, *Neural Networks for Pattern Recognition*. London, U.K.: Oxford Univ. Press, 1995.

[7] Y. Grandvalet and S. Canu, "Noise injection: Theoretical prospects," *Neural Comput.*, vol. 9, no. 5, pp. 1093–1108, 1997.

[8] A. M. Thompson, J. C. Brown, J. W. Kay, and D. M. Titterington, "A study of methods of choosing the smoothing paprameter in image restoration by regularization," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, no. 4, pp. 326–339, 1991.

[9] P. R. Johnston and R. M. Gulrajani, "A new method for regularization parameter determination in the inverse problem of electrocardiography," *IEEE Trans. Biomed. Eng.*, vol. 44, pp. 19–39, Jan. 1997.

[10] G. Wahba, *Spline Models for Observational Data, CBMS-NSF Regional Conference Series in Applied Mathematics*. Philadelphia, PA: SIAM, 1990, vol. 59.

[11] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*. London, U.K.: Chapman & Hall, 1993.

[12] D. J. C. MacKay, "Bayesian interpolation," *Neural Comput.*, 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," *Proc. IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing*, vol. VI, pp. 62–71, 1996.

[14] L. N. Andersen, J. Larsen, L. K. Hansen, and M. Hintz-Madsen, "Adaptive regularization of neural classifiers," *Proc. IEEE Workshop Neural Networks for Signal Processing*, vol. VII, pp. 24–33, 1997.

[15] D. Chen and M. T. Hagan, "Optimal use of regularization and cross-validation in neural network modeling," in *Proc. Int. Joint Conf. Neural Networks*, vol. 2, 1999, pp. 1275–1280.

[16] K. hagiwara and K. Kuno, "Regularization learning and early stoping in linear networks," *Proc. IEEE-INNS-ENNS Int. Joint Conf. Neural Networks*, vol. 4, pp. 511–516, 2000.

[17] I. Rivals and L. Personnaz, "On cross-validation for model selection," *Neural Comput.*, vol. 11, pp. 863–870, 1999.

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

[19] L. Devroye, *A Course in Density Estimation*. Cambridge, MA: Birkhaüser, 1987.

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

[21] C. O. Wu, "A cross-validation bandwidth choice for kernel density estimates with selection biased data," *J. Multivariate Anal.*, vol. 61, pp. 38–60, 1997.

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

[23] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. New York: Academic, 1990.

[24] L. Xu, "Bayesian Ying-Yang system and theory as a unified statistical learning approach (VII): Data smoothing," in *Proc. Int. Conf. Neural Information Processing*, vol. 1, Kitakyushu, Japan, 1998, pp. 243–248.

[25] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[26] A. Barron, J. Rissanen, and B. Yu, "The minimum description length prnciple in coding and modeling," *IEEE Trans. Inform. Theory*, vol. 44, pp. 2743–2760, Oct. 1998.

[27] G. S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*. Berling, Germany: Springer-Verlag, 1996.

[28] J. E. Gentle, *Random Number Generation and Monte Carlo Methods*. Berlin, Germany: Springer-Verlag, 1998.

[29] C. M. Bishop, "Training with noise is equivalent to Tikhonov regularization," *Neural Comput.*, vol. 7, no. 1, pp. 108–116, 1995.

[30] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-Posed Problems*. Washington, DC: Winston, 1977.

[31] R. Reed, R. J. Marks II, and S. 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," Aston University, Birmingham, U.K., Tech. Rep. NCRG/95/022, 1995.

[33] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Netw.*, vol. 1, pp. 295–307, 1988.

[34] *Handbook of Software Reliability Engineering*, McGraw-Hill, New York, 1996.

[35] D. J. C. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Comput.*, 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*. Cambridge, MA: MIT Press, 1986, vol. 1, pp. 318–362.

Ping Guo received the M.S. degree in physics from Peking University, Beijing, China, and the Ph.D. degree in computer science from the Chinese University of Hong Kong, Shatin, NT, Hong Kong.

He is currently a Professor with the Computer Science Department, Beijing Normal University, Beijing, China. From 1993 to 1994, he was with the Department of Computer Science and Engineering, Wright State University, Dayton, OH, as a Visiting Faculty. From May 2000 to August 2000, he was with the National Laboratory of Pattern Recognition at Chinese Academy of Sciences, Beijing, as a Guest Researcher. His current research interests include neural network, image process, software reliability engineering, optical computing, and spectra analysis.

Michael R. Lyu (S'84–M'88–SM'97) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1981, the M.S. degree in computer engineering from the University of California, Santa Barbara, in 1985, and the Ph.D. degree in computer science from University of California, Los Angeles, in 1988.

He is currently a Professor, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong. He was with the Jet Propulsion Laboratory, Pasadena, CA, as Technical Staff Member from 1988 to 1990. From 1990 to 1992, he was with the Department of Electrical and Computer Engineering, The University of Iowa, Iowa City, as an Assistant Professor. From 1992 to 1995, he was Member of Technical Staff in the applied research area of Bell Communications Research (Bellcore), Morristown, NJ. From 1995 to 1997, he was Research Member of Technical Staff at Bell Laboratories, Murray Hill, NJ, which was first part of AT&T and later became part of Lucent Technologies. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, wireless communication networks, Web technologies, digital libraries, 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 was the editor of two book volumes: *Software Fault Tolerance* (New York: Wiley, 1995) and *The Handbook of Software Reliability Engineering* (Piscataway, NJ: IEEE and New York: McGraw-Hill, 1996). He has been frequently invited as a keynote or tutorial speaker to conferences and workshops in U.S., Europe, and Asia. He is an Associate Editor of the *Journal of Information Science and Engineering.*

Dr. Lyu 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'01, and the WWW10 Program Co-Chair. He is an Associate Editor of IEEE TRANSACTIONS ON RELIABILITY, and IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING.

C. L. Philip Chen (S'88–M'88–SM'94) received the M.S. degree from the University of Michigan, Ann Arbor, in 1985, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1988.

From 1988 to 1989, he was a Visiting Assistant Professor, School of Engineering and Technology, Purdue University-Indianapolis, Indianapolis, IN. From September 1989 to August 2002, he was with the Computer Science and Engineering Department, Wright State University, Dayton, OH, as an Assistant Professor, Associate Professor, and Full Professor. Currently, he is with the Department of Electrical and Computer Engineering, The University of Texas at San Antonio, as a Full Professor.

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