Geometrical Interpretation and Architecture Selection of MLP

Cheng Xiang, *Member, IEEE*, Shenqiang Q. Ding, and Tong Heng Lee, *Member, IEEE*

***Abstract*—A geometrical interpretation of the multilayer perceptron (MLP) is suggested in this paper. Some general guidelines for selecting the architecture of the MLP, i.e., the number of the hidden neurons and the hidden layers, are proposed based upon this interpretation and the controversial issue of whether four-layered MLP is superior to the three-layered MLP is also carefully examined.**

***Index Terms*—Architecture selection, geometrical interpretation, multilayer perceptron (MLP).**

I. Introduction

EVERY PRACTITIONER of the multilayer perceptron (MLP) faces the same architecture selection problem: how many hidden layers to use and how many neurons to choose for each hidden layer? Unfortunately, there is no foolproof recipe at the present time, and the designer has to make seemingly arbitrary choices regarding the number of hidden layers and neurons. The common practice is simply regarding the MLP as a sort of magic black box and choosing a sufficiently large number of neurons such that it can solve the practical problem in hand. Designing and training a neural network and making it to work seems more of an art than a science. Without a profound understanding of the design parameters, some people still feel uneasy to use the MLP even though the neural networks have already proven to be very effective in a wide spectrum of applications, in particular the function approximation and pattern recognition problems. Therefore, it is of great interest to gain deeper insight into the functioning of the hidden neurons and hidden layers and change the architecture design for MLP from the state of art into state of technology.

Traditionally, the main focus regarding the architecture selection of MLP has been centered upon the growing and pruning techniques [1]–[4]. Recently, a great deal of attention has also been drawn on applying evolutionary algorithms to evolve both the parameters and architectures of the artificial neural networks [5]–[8]. Such kinds of hybrid algorithms are commonly referred to in the literature as evolutionary artificial neural networks (EANNs) (for a detailed survey see [9]). One essential feature of EANN is the combination of the two distinct forms of adaptation, i.e., learning and evolution, which makes the hybrid systems adapt to the environment more efficiently. However, one major drawback of EANN is that its adaptation speed is usually very slow due to its nature of population and random search.

In all these approaches discussed above, any *a priori* information regarding the geometrical shape of the target function is generally not exploited to aid the architecture design of MLP. In contrast to them, it will be demonstrated in this paper that it is the geometrical information that will simplify the task of architecture selection significantly. We wish to suggest some general guidelines for selecting the architecture of the MLP, i.e., the number of hidden layers as well as the number of hidden neurons, provided that the basic geometrical shape of the target function is known in advance, or can be perceived from the training data. These guidelines will be based upon the geometrical interpretation of the weights, the biases, and the number of hidden neurons and layers, which will be given in the next section of the paper.

It will be shown that the architecture designed from these guidelines is usually very close to the minimal architecture needed for approximating the target function satisfactorily, and in many cases is the minimal architecture itself. As we know, searching for a minimal or subminimal structure of the MLP for a given target function is very critical not only for the obvious reason that the least amount of computation would be required by the minimal structured MLP, but also for a much deeper reason that the minimal structured MLP would provide the best generalization in most of the cases. It is well known that neural networks can easily fall into the trap of "over-fitting,"and supplying a minimal structure is a good medicine to alleviate this problem.

In the next section, the geometrical interpretation of the MLP will be presented. This interpretation will be first suggested for the case when the activation function of the hidden neuron is piecewise linear function and then is extended naturally to the case of sigmoid activation functions. Following this, a general guideline for selecting the number of hidden neurons for three-layered (with one hidden layer) MLP will be proposed based upon the geometrical interpretation. The effectiveness of this guideline will be illustrated by a number of simulation examples. Finally, we will turn our attention to the controversial issue of whether four-layered (with two hidden layers) MLP is superior to the three-layered MLP. With the aid of the geometrical interpretation and also through carefully examining various contradictory results reported in the literature, it will be demonstrated that in many cases four-layered MLP is slightly more efficient than three-layered MLP in terms of the minimal number of parameters required for approximating the target function,

Manuscript received March 27, 2003; revised December 3, 2003. This work was supported by the National University of Singapore Academic Research Fund R-263-000-224-112.

C. Xiang and T. H. Lee are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119260 (e-mail: elexc@nus.edu.sg; eleleeth@nus.edu.sg).

S. Q. Ding is with STMicroelectronics, Corporate R&D, Singapore 117684 (e-mail: shenqiang.ding@st.com).

Digital Object Identifier 10.1109/TNN.2004.836197

1045-9227/$20.00 © 2005 IEEE

and for a certain class of problems the four-layered MLP outperforms three-layered MLP significantly.

II. Geometrical Interpretation of MLP

Consider a three-layered $1-N-1$ MLP, with one input neuron, N hidden neurons and one output neuron. The activation function for the hidden neuron is the piecewise linear function described by

$$\varphi(v)=\begin{cases}1, & v\geq 0.5\\ v+0.5, & -0.5<v<0.5\\ 0, & v\leq -0.5\end{cases} \tag{1}$$

and plotted in Fig. 1.

Let the weights connecting the input neuron to the hidden neurons be denoted as $w_i^{(1)}(i=1,\ldots,N)$, the weights connecting the hidden neurons to the output neuron be $w_i^{(2)}$, the biases for the hidden neurons be $b_i^{(1)}$, and the bias for the output neuron be $b^{(2)}$. The activation function in the output neuron is the identity function such that the output y of the MLP with the input feeding into the network is

$$y(x)=\sum_{i=1}^{N} w_i^{(2)}\varphi\left(w_i^{(1)}x+b_i^{(1)}\right)+b^{(2)}. \tag{2}$$

It is evident that $y(x)$ is just superposition of N piecewise linear functions plus the bias. From (1) we know that each piecewise linear function in (2) is described by

$$w_i^{(2)}\varphi\left(w_i^{(1)}x+b_i^{(1)}\right)=\begin{cases}w_i^{(2)}, & w_i^{(1)}x+b_i^{(1)}\geq 0.5\\ w_i^{(2)}\left(w_i^{(1)}x+b_i^{(1)}+0.5\right), & -0.5<w_i^{(1)}x+b_i^{(1)}<0.5.\\ 0, & w_i^{(1)}x+b_i^{(1)}\leq -0.5\end{cases} \tag{3}$$

In the case of $w_i^{(1)}>0$, we have

$$w_i^{(2)}\varphi\left(w_i^{(1)}x+b_i^{(1)}\right)=\begin{cases}w_i^{(2)}, & x\geq \frac{0.5}{w_i^{(1)}}-\frac{b_i^{(1)}}{w_i^{(1)}}\\ w_i^{(2)}\left(w_i^{(1)}x+b_i^{(1)}+0.5\right), & \frac{-0.5}{w_i^{(1)}}-\frac{b_i^{(1)}}{w_i^{(1)}}<x<\frac{0.5}{w_i^{(1)}}-\frac{b_i^{(1)}}{w_i^{(1)}}\\ 0, & x\leq \frac{-0.5}{w_i^{(1)}}-\frac{b_i^{(1)}}{w_i^{(1)}}\end{cases} \tag{4}$$

whose graph is shown in Fig. 2.

This piecewise linear function has the same geometrical shape as that of (1), comprising two pieces of flat lines at the two ends and one piece of line segment in the middle. Any finite piece of line segment can be completely specified by its width (span in the horizontal axis), height (span in the vertical axis), and position (starting point, center, or ending point). And it is obvious from (4) and Fig. 2 that the width of the middle line segment is $1/w_i^{(1)}$, the height is $w_i^{(2)}$, the slope is, therefore, $w_i^{(1)}w_i^{(2)}$, and the starting and ending points are $((-0.5/w_i^{(1)})-(b_i^{(1)}/w_i^{(1)}),0)$ and $((0.5/w_i^{(1)})-(b_i^{(1)}/w_i^{(1)}),w_i^{(2)})$, respectively. Once this middle line segment is specified the whole piecewise line is then completely determined. From above discussion it is natural to suggest the following geometrical interpretation for the three-layered MLP with piecewise linear activation functions.

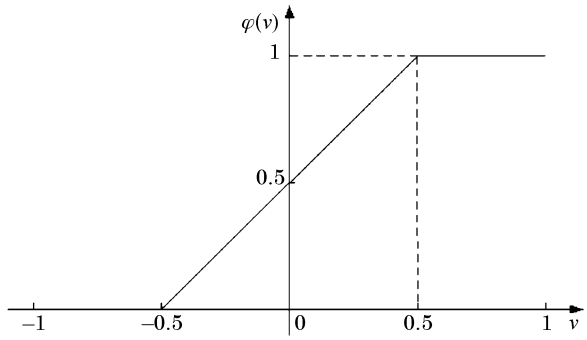


Fig. 1. Piecewise linear activation function.

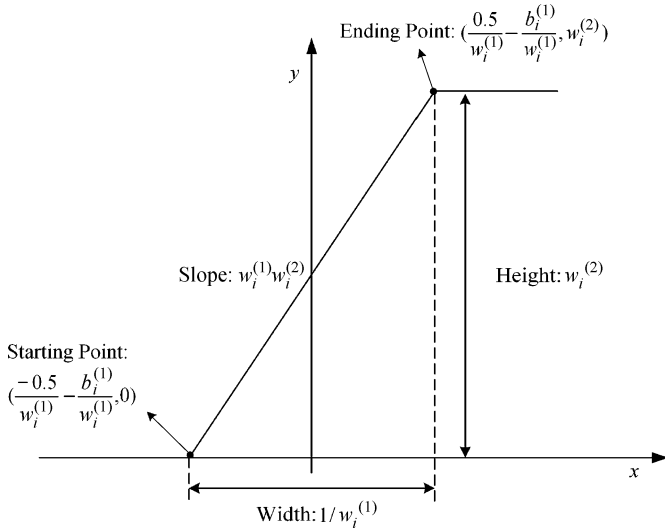


Fig. 2. Basic building block of MLP.

1) The number of hidden neurons corresponds to the number of piecewise lines that are available for approximating the target function. These piecewise lines act as the basic building blocks for constructing functions.
2) The weights connecting the input neuron to the hidden neurons $w_i^{(1)}$ completely determine the widths of the middle line segments of the basic building blocks. By adjusting these weights, the widths of the basic elements can be changed to arbitrary values.
3) The weights connecting the hidden neurons to the output neuron $w_i^{(2)}$ totally decide the heights of the middle line segments of the basic building blocks. The heights can be modified to any values by adjusting these weights.
4) The products of the weights $w_i^{(1)}w_i^{(2)}$ specify the slopes of the middle line segments of the basic building blocks.
5) The biases in the hidden neuron $b_i^{(1)}$ govern the positions of the middle line segments of the basic building blocks. By adjusting the values of these biases, the positions of the building blocks can be located arbitrarily.
6) The bias in the output neuron $b^{(2)}$ provides an offset term to the whole value of the function.

Using the fact that the widths, the heights, and the positions of the middle line segments of the basic building blocks can be adjusted arbitrarily, we are ready to state and prove Theorem 1 as follows.

Theorem 1: Let $f(x)$ be any piecewise linear function defined in any finite domain $-\infty < a \leq x \leq b < \infty$ there exists at least one three-layered MLP, denoted as $NN(x)$, with piecewise linear activation functions for the hidden neurons that can represent $f(x)$ exactly, i.e., $NN(x) = f(x)$ for all $x \in [a, b]$.

The proof of Theorem 1 is quite straightforward by directly constructing one MLP that can achieve the objective.

Proof: Let $f(x)$ be any piecewise linear function consisting of arbitrary number N of line segments. Each line segment is completely determined by its starting and ending points. Let us denote the two boundary points of the ith line segment as $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$, where $i = 1, \ldots, N$, $x_0 = a$, and $x_N = b$. The width and height of the ith line segment are then $x_i - x_{i-1}$ and $f(x_i) - f(x_{i-1})$, respectively.

Let us construct a three-layered MLP as follows. Let the number of the hidden neurons be N, the same as the number of the piecewise lines in $f(x)$. Each of the hidden neuron will then provide one piecewise line, whose width, height, and starting point can be arbitrarily adjusted by the weights and biases. One natural way of choosing the weights and biases is to make the middle line segment provided by the ith neuron match the ith line segment in $f(x)$. Therefore, the parameters of the MLP can be calculated as follows.

To match the width, set

$$\frac{1}{w_i^{(1)}} = x_i - x_{i-1}, \quad i = 1, \ldots, N \tag{5}$$

to match the height, set

$$w_i^{(2)} = f(x_i) - f(x_{i-1}), \quad i = 1, \ldots, N \tag{6}$$

to match the position, set

$$\frac{-0.5}{w_i^{(1)}} - \frac{b_i^{(1)}}{w_i^{(1)}} = x_{i-1}, \quad i = 1, \ldots, N \tag{7}$$

to match the exact value of $f(x)$, an offset term has to be provided as

$$b^{(2)} = f(x_0) = f(a). \tag{8}$$

The parameters of the three-layered MLP are completely determined by (5)–(8). Because of the special property of the activation function that the lines are all flat (with zero slope) except the middle segment, the contribution to the slope of the line segment in the interval of $[x_{i-1}, x_i]$ comes only from the middle line segment provided by the ith neuron. From (5) and (6), it is obvious that the slope of the each line segment of MLP matches that of $f(x)$. All we need to show now is that the output value of MLP at the starting point for each line segment matches that of $f(x)$, then the proof will be complete.

At the initial point $x = x_0$, all the contributions from the hidden neurons are zero, and the output value of the MLP is just the bias $b^{(2)}$

$$NN(x_0) = b^{(2)}. \tag{9}$$

At point $x = x_1$, which is the ending point of the line segment provided by the first neuron, the output value of the first neuron is $w_1^{(2)}$ while the output values of all other neurons are zero, therefore, we have

$$NN(x_1) = w_1^{(2)} + b^{(2)}. \tag{10}$$

Similar argument leads to

$$NN(x_i) = w_1^{(2)} + \ldots + w_i^{(2)} + b^{(2)}, \quad i = 1, \ldots, N. \tag{11}$$

From (6) and (8), it follows immediately that:

$$NN(x_i) = f(x_i), \quad i = 0, 1, \ldots, N. \tag{12}$$

This completes the proof of Theorem 1.

Comment 1: The weights and biases constructed by (5)–(8) are just one set of parameters that can make the MLP represent the given target function. There are other possible sets of parameters that can achieve the same objective. For instance, for purpose of simplicity we set $w_i^{(1)} > 0$ in all our discussions so far. Without this constraint there would be many other combinations of the building blocks that may construct the same piecewise linear function exactly considering the fact that the slope of each piecewise line is described by $w_i^{(1)} w_i^{(2)}$. This implies that the global minimum may not be unique in many cases.

Comment 2: In the proof given for Theorem 1, N hidden neurons are used to approximate the function consisting of N piecewise line segments, and the domain of the middle line segment for each basic building block does not overlap with each other. If some domains of the middle line segments overlap, then it is possible for $1 - N - 1$ MLP to approximate functions comprising more than N piecewise line segments. But then the slopes around these overlapping regions are related, and cannot be arbitrary. A couple of such examples are depicted in Fig. 3, where the solid line is the combination of two basic building blocks, which are plotted with dash-dotted and dashed lines, respectively.

Comment 3: Since any bounded continuous function can be approximated arbitrarily closely by piecewise linear function, Theorem 1 simply implies that any bounded continuous function can be approximated arbitrarily closely by MLP, which is the well-known universal approximation property of the MLP proven in [10]–[12]. Although the proof presented in this paper only considers the case of piecewise linear activation functions, the constructive and geometrical nature of this proof makes this elegant property of MLP much more transparent than other approaches.

Comment 4: The geometrical shape of the sigmoid activation function is very similar to the piecewise linear activation function, except the neighborhood of the two end points are all

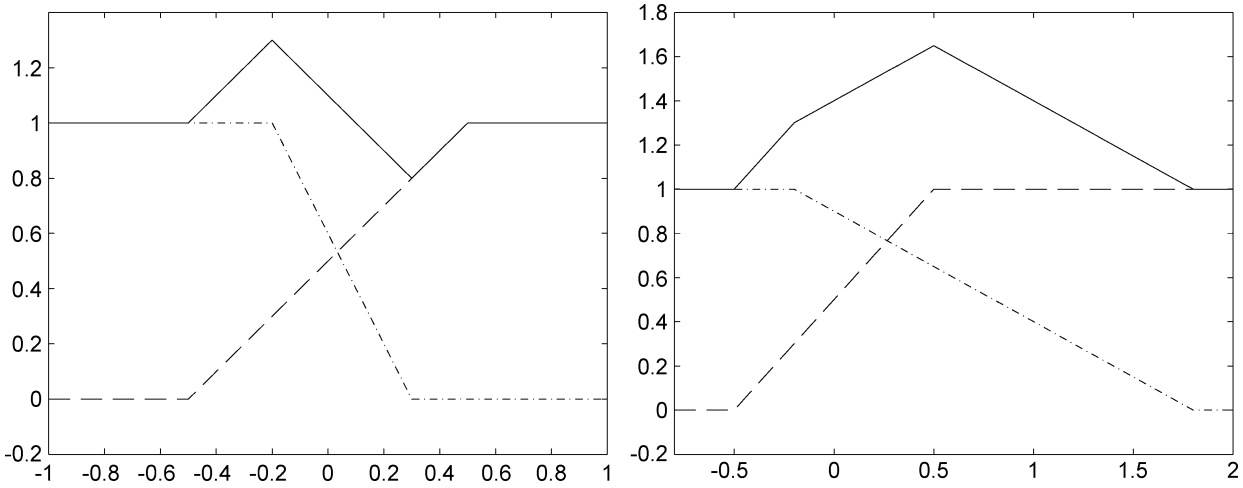


Fig. 3. Overlapping of basic building blocks.

smoothed out, as shown in Fig. 4. Therefore, the previous geometrical interpretation of the MLP can be applied very closely to the case when sigmoid activation functions are used. Further, since the sigmoid function smoothes out the nonsmooth end points, the MLP with sigmoid activation functions is more efficient to approximate smooth functions.

Comment 5: When the input space is high dimensional, then each hidden neuron provides a piecewise hyperplane as the basic building block that consists of two flat hyperplanes and one piece of hyperplane in the middle. The position and width of the middle hyperplane can be adjusted by the weights connecting the input layer to the hidden layer and the biases in the hidden layer, while the height can be altered by the weights connecting the hidden layer to the output layer. A two-dimensional (2-D) example of such building blocks is shown in Fig. 5 where sigmoid activation functions are used.

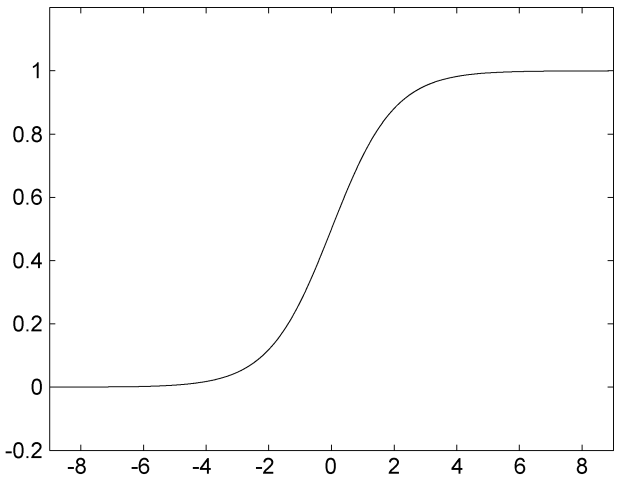


Fig. 4. Sigmoid activation function.

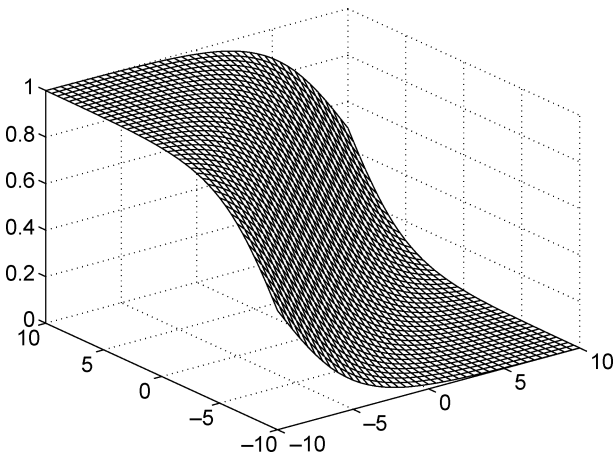


Fig. 5. 2-D basic building block.

III. Selection of Number of Hidden Neurons for Three-Layered MLP

Based upon previous discussion regarding the geometrical meaning of the number of hidden neurons, the weights and the biases, a simple guideline for choosing the number of hidden neurons for the three-layered MLP is proposed as follows.

Guideline One: Estimate the minimal number of line segments (or hyperplanes in high-dimensional cases) that can construct the basic geometrical shape of the target function, and use this number as the first trial for the number of hidden neurons of the three-layered MLP.

This guideline has been tested with extensive simulation studies. In all of the cases investigated, this minimal number of line segments is either very close to the minimal number of hidden neurons needed for satisfactory performance, or is the minimal number itself in many cases. Some of the simulation examples will be discussed below to illuminate the effectiveness of this guideline. All of the simulations have been conducted using the neural network toolbox of MATLAB. The activation function for the hidden neurons is hyperbolic tangent function (called "tansig" in MATLAB), and that for the output neurons is the identity function (called "purelin" in MATLAB) in most cases. Batch training is adopted and the Levenberg–Marquardt algorithm [13], [14] (called "trainlm" in MATLAB) is used as the learning algorithm. The Nguyen–Widrow method [15] is utilized in the toolbox to initialize the weights of the each layer of the MLPs.

Comment 6: The selection of the activation function and training algorithm is another interesting issue which has been

investigated by other papers [16]–[18]. We will not delve into this issue here and the choice of "tansig" and "trainlm" is simply made by trial and error studies.

Simulation One: The target function is described by

$$f(x) = x^3 + 0.3x^2 - 0.4x, \quad x \in [-1, 1]. \tag{13}$$

The training set consists of 21 points, which are chosen by uniformly partitioning the domain $[-1, 1]$ with grid size of 0.1. And the test set comprises 100 points uniformly randomly sampled from the same domain. Following Guideline One, the least number of line segments to construct the basic geometrical shape of $f(x)$ is obviously three, therefore, 1-3-1 MLP is tried first. It turns out that 1-3-1 MLP is indeed the minimal sized MLP to approximate $f(x)$ satisfactorily. After only 12 epochs, the mean square error (MSE) of the training set decreases to 2.09×10^{-6}, and the test error (MSE) is 1.27×10^{-6}. The result is shown in Fig. 6, where the dotted line is the target function, and the dash-dotted line is the output of the MLP, which almost coincide with each other exactly.

Comment 7: It is obvious that such good approximation result cannot be achieved using three pieces of pure line segments. The smoothing property of the sigmoid function plays an important role in smoothing out the edges.

Simulation Two: Assume that the training data for the target function in Simulation One are corrupted by noises uniformly distributed in $[-0.05, 0.05]$ while the test set remains intact. Both 1-3-1 and 1-50-1 MLPs are used to learn the same set of training data and the results are shown in Table I and plotted in Fig. 7.

Comment 8: The purpose of this simulation example is to show the necessity of searching for minimal architecture. It is evident that 1-3-1 MLP has the best generalization capability, which approximates the ideal target function closely even though the training data is corrupted. In contrast to this, the 1-50-1 MLP falls badly into the trap of "over-fitting."

Comment 9: Another popular approach to deal with the "over-fitting" problem is the regularization methods [2], [19]–[22], which minimize not only the approximation errors but also the sizes of the weights. The mechanism of this regularization scheme can now be readily elucidated from the geometrical point of view as follows. Since the slope of each building block is roughly proportional to $w_i^{(1)} w_i^{(2)}$ as discussed before, the smaller the weights, the gentler the slope of each building block and, hence, the smoother the shape of the overall function. It is important to note that the biases are not related to the slopes of the building blocks and, hence, should not be included in the penalty terms for regularization, which was in fact not recognized in the early development of the regularization methods [2], [19], but only rectified later in [20]. The reader is referred to [23] for further details on this subject.

Simulation Three: We intend to approximate a more complicated function specified as

$$f(x) = 0.5sin(\pi x)^3 - \frac{2}{x^3 + 2} - 0.1cos(4\pi x) + |x|, \quad x \in [-1, 1.6]. \tag{14}$$

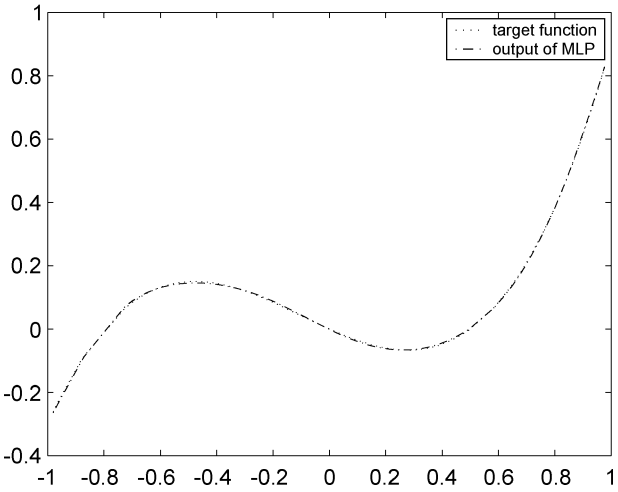


Fig. 6. Simulation One: a simple one-dimensional (1-D) example.

TABLE I
SIGNIFICANTLY DIFFERENT PERFORMANCES OF 1-3-1 AND 1-50-1 MLPS

MLPs	Epochs	Training error(MSE)	Test error(MSE)
1-3-1	100	5.65×10^{-4}	3.61×10^{-4}
1-50-1	2	1.31×10^{-10}	0.29

The training set contains 131 points, which are chosen by uniformly dividing the domain $[-1, 1.6]$ with grid size of 0.02. The test set includes 200 points randomly selected within the same domain. It is observed that at least nine line segments are needed to construct the basic shape of the target function and, hence, 1-9-1 is decided to be the first trial. After 223 epochs, the mean square training error and test error are 9.99×10^{-6} and 8.87×10^{-6}, respectively, and the bound of test error is 0.01. The approximation is almost perfect as shown in Fig. 8.

Comment 10: Smaller sized MLP such as 1-8-1 and 1-7-1 are also tested to solve this problem. Both of them are able to provide good approximations except in the small neighborhood around $x = 0$ where the error bound is bigger than 0.01 (but smaller than 0.04). The reader is referred back to Comment 2 for understanding the possibility that the minimal number of the hidden neurons (building blocks) may be smaller than the number of line segments for a given target function. In this example, if we consider approximation with error bound of 0.04 as satisfactory, then the minimal structure would be 1-7-1 instead of 1-9-1.

Simulation Four: Let us consider a simple 2-D example, a Gaussian function described by

$$f(x, y) = \frac{5}{2\pi} e^{\left(-\frac{x^2+y^2}{2}\right)}, \quad x, y \in [-4, 4]. \tag{15}$$

The training set comprises 289 points, which are chosen by uniformly partitioning the domain $[-4, 4]$ with grid size of 0.5. The test set composes of 1000 points randomly sampled from the same domain. It is apparent that at least 3 piecewise planes are needed to construct the basic geometrical shape of the Gaussian function: a hill surrounded by flat plane. Therefore, from our guideline a 2-3-1 MLP is first tried to approximate this function. After 1000 epochs, the training error (MSE) decreases to 8.58×10^{-5}, and the test error (MSE) is 8.56×10^{-5}.

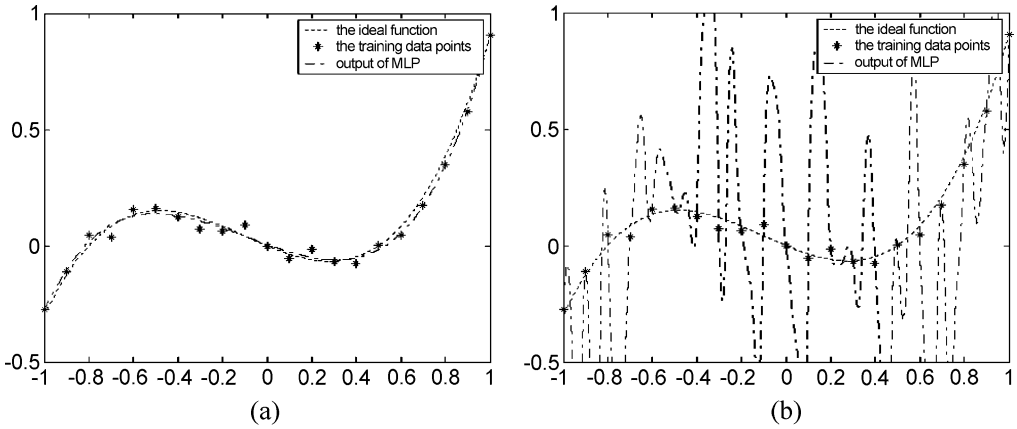


Fig. 7. Simulation Two: the noisy 1-D example. (a) Approximation by 1-3-1 MLP. (b) Approximation by 1-50-1 MLP.

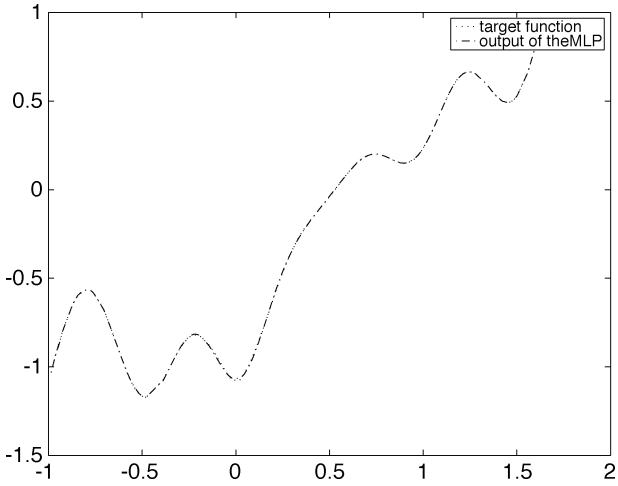


Fig. 8. Simulation Three: a complicated 1-D example.

The result is reasonably good as shown in Fig. 9, if we consider the error bound of about 0.07 to be acceptable.

Comment 11: It is worth noting that the activation function used for the output neuron in Simulation Four is not the identity function, but the logistic function (called "logsig" in MATLAB). Since the sigmoid function has the property of flattening things outside of its focused domain, it is possible to approximate a function within a certain region while keeping other areas flat, which is very suitable for the type of Gaussian hill problem. Without this flattening property, it would be difficult to improve the approximation at one point without worsening other parts. That is why the size of the three-layered MLP has to be increased to around 2-20-1 to achieve similar error bound if the identity activation function is used in the output neuron.

Simulation Five: We consider a more complicated 2-D example as follows:

$$f(x,y) = 0.1x^2 - 0.05y^2 + sin(0.16x^2 + 0.16y^2), \quad x, y \in [-4.5, 4.5]. \quad (16)$$

The training set composes of 100 points, by uniformly partitioning the domain $[-4.5, 4.5]$ with grid size of 1.0. The test set contains 1000 points randomly chosen from the same domain. In order to apply our guideline, we have to estimate the least number of piecewise planes to construct the basic shape of this target function. It appears that at least three pieces of planes are needed to construct the valley in the middle, six pieces of planes to approximate the downhills outside the valley, and additional four pieces of planes to approximate the little uphills at the four corners, which are shown in Fig. 10. The total number of piecewise planes is then estimated to be 13, hence, a 2-13-1 MLP is first tried to approximate this function. After 5000 epochs, both the training error (MSE) and test error (MSE) decrease to 0.0009 and 0.0018, respectively. The approximation result is quite well with error bound of 0.15, as shown in Fig. 11.

It is observed that the local minima problem is quite severe for this simulation example. Approximately only one out of ten trials with different initial weights may achieve error bound of 0.15.

To alleviate this local minima problem, as well as to further decrease the error bound of the test set, EANNs are applied to this example. One of the popular EANN systems, EPNET [24], [25], is adopted to solve the approximation problem for the function (16) with the same training and test sets mentioned before. Here, the EPNET is simplified by removing the connection removal and addition operators, due to the fact that only fully-connected three-layered MLPs are used. The flowchart is given in Fig. 12, which is a simplified version of the flowchart in [25].

The reader is referred to [24], [25] for detailed description of the EPNET algorithm. The following remarks are in order as follows to explain some of the blocks in the flowchart:

1) "MBP training" refers to training with modified back-propagation algorithm, which is chosen to be the Levenberg–Marquardt algorithm (trainlm) in this paper.
2) "MRS" refers to the modified random search algorithm, and the reader is referred to [26] for further details.
3) "Selection" is done by randomly choosing one individual out of the population with probabilities associated with the performance ranks, where the higher probabilities are assigned to the individuals with worse performances. This is in order to improve the performance of the whole population rather than improving a single MLP as suggested in [24], [25].
4) "Successful" means the validation error bound has been reduced substantially, for instance, by at least 10%

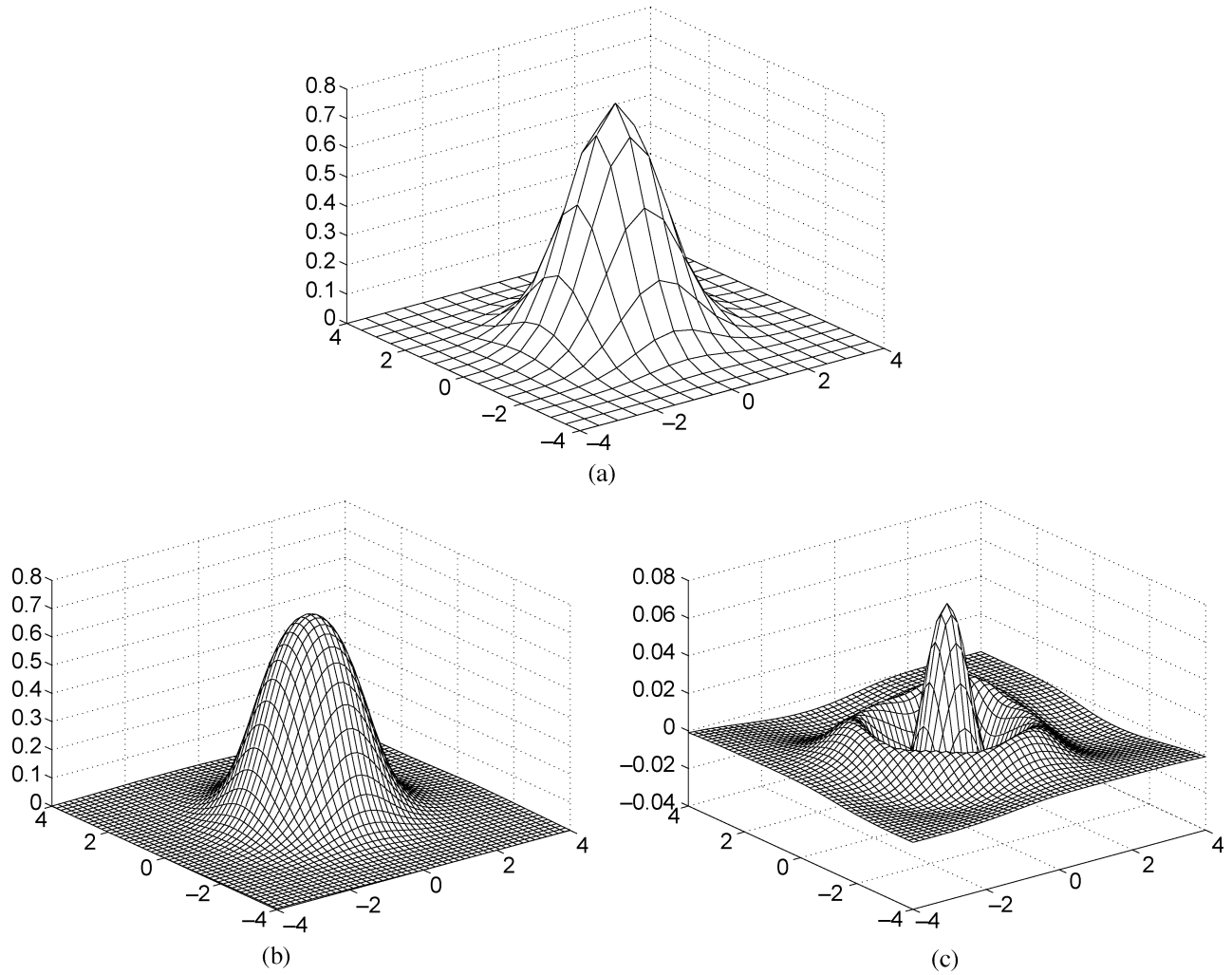


Fig. 9. Simulation Four: approximation of Gaussian function. (a) Training data. (b) Output of the neural network. (c) Approximation error.

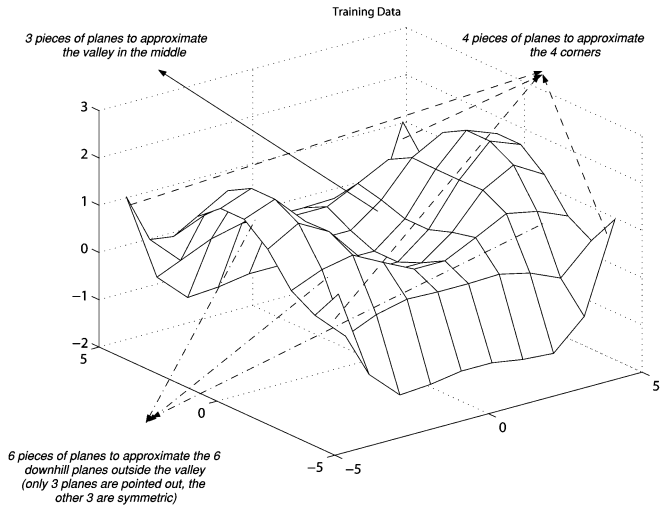


Fig. 10. Estimation of piecewise planes to construct the basic geometrical shape of the target function (16).

in our simulations. The validation set contains 1000 random samples uniformly distributed in the domain of $[-4.5, 4.5] \times [-4.5, 4.5]$.

5) "The performance goal" is set as 0.1 for the validation error bound. Once the goal is met, the evolutionary process will stop, and the best candidate (with the lowest error bound) will be selected to approximate the target function.

The size of the population is 10, and the initialization of the population can be done in different ways. Since 2-13-1 has already been estimated by Guideline One to be good candidate for the structure of MLP, it is natural to initialize the population with the same structures of 2-13-1. It is shown in Table II that after only 69 generations one of the MLPs achieves the performance goal of error bound of 0.1. If the population for the first generation is chosen without this guideline, for instance, initialized with 2-5-1 MLPs, or 2-20-1 MLPs, or a set of different structured MLPs in which the numbers of hidden neurons are randomly selected in the range of [5,30], as suggested in [24], the convergence speed is usually much slower as shown in Table II.

Comment 12: The number of generations needed to achieve the performance goal, and the structures of the best candidate may differ with different experiments, and the results reported in Table II is from one set of experiments out of five. It is interesting to note that the final structure of the best candidate

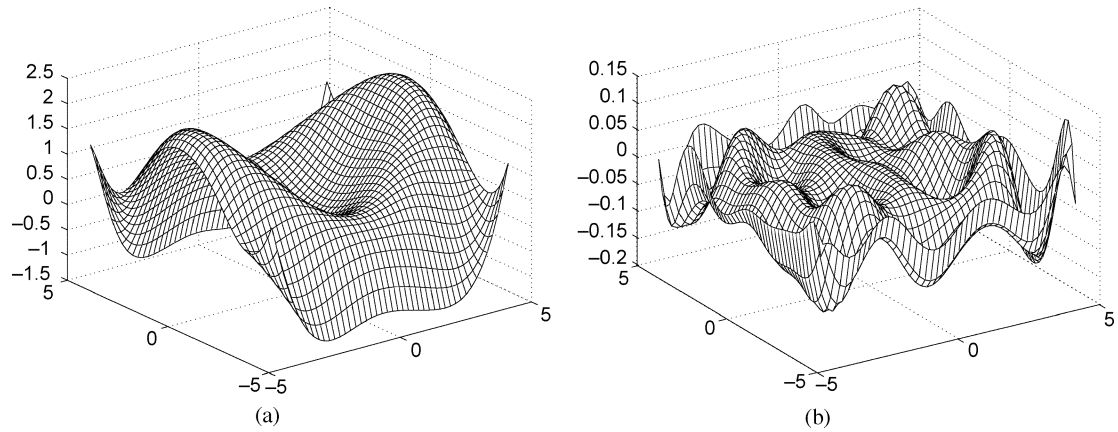


Fig. 11. Simulation Five: a more complicated 2-D example. (a) Output of the neural network. (b) Approximation error.

TABLE II
PERFORMANCE COMPARISON OF EPNET WITH DIFFERENT INITIAL POPULATIONS

Structures of the initial population	Generations needed to succeed	Error bounds of the best MLP	Structures of the best MLP
2-13-1	69	0.0946	2-15-1
2-5-1	365	0.0988	2-17-1
2-20-1	216	0.0944	2-16-1
Mixed structures	229	0.0913	2-15-1

usually converges to a narrow range from 2-15-1 to 2-17-1 regardless of the structures of the initial population, which is indeed not far from our initial estimation of 2-13-1. Therefore, it is not surprising that the EPNET with initial population of 2-13-1 MLPs always converges faster than other approaches although the number of generations to evolve varies with different sets of simulation studies.

Comment 13: It also has to be stressed that the performance goal of 0.1 error bound can be hardly achieved by training a 2-15-1, or 2-16-1 MLP solely with standard BP or modified BP due to the local minima problem. The combination of evolutionary algorithm and neural networks (EANN) indeed proves to be more efficient as seen from our simulation studies, and our proposed guideline can be used to generate the initial population of the EANNs, which can speed up the evolution process significantly.

Comment 14: It is noticed that the difficulty in estimating the least number of hyperplane pieces to construct the basic geometrical shape of the target function increases with the complexity of the target function. In particular, when the dimension is much higher than 2 as in many cases of pattern recognition problems, it is almost impossible to determine the basic geometrical shape of the target function. Hence, Guideline One can be hardly applied to very high-dimensional problems unless *a priori* information regarding the geometrical shapes of the target functions are known through other means. Either pruning and growing techniques [1]–[4] or EANNs [5]–[9], [24], [25] are then recommended to deal with such problems where little geometrical information is available.

IV. ADVANTAGES OFFERED BY FOUR-LAYERED MLP

The question of whether adding another hidden layer to the three-layered MLP is more effective has remained a controversial issue in the literature. While some results [27]–[29] have suggested that four-layered MLP is superior to three-layered MLP from various points of views, other result [30] has shown that four-layered networks are more prone to fall into bad local minima, but that three- and four-layered MLPs perform similarly in all other respects. In this section, we will try to clarify the issues raised in the literature, and provide a few guidelines regarding the choice of one or two hidden layers by applying the geometrical interpretations in Section II.

One straightforward interpretation of four-layered MLP is simply regarding it as a linear combination of multiple three-layered MLPs by observing that the final output of the four layered MLP is nothing but linear combination of the outputs of the hidden neurons in the second hidden layer, which themselves are simply the outputs of three-layered MLPs. Thus, the task of approximating a target function is essentially decomposed into tasks of approximating subfunctions with these three-layered MLPs. Since all of them share the same hidden neurons but with different output neurons, these three-layered MLPs share the same weights connecting the input layers to the first hidden layers, but with different weights connecting the first hidden layers to the "output" neurons (the neurons in the second hidden layer of the four-layered MLP). According to the geometrical interpretation discussed before, it is apparent that the corresponding basic building blocks of these three-layered MLPs share the same widths and positions, but with different heights.

One obvious advantage gained by decomposing the target function into several subfunctions is that the total number of the parameters of the four-layered MLP may be smaller than that of three-layered MLP. Because the number of the hidden neurons in the first hidden layer can be decreased substantially if the target function is decomposed into subfunctions which possess simpler geometrical shapes and, hence, need less number of the building blocks to construct.

Simulation Six: Consider the approximation problem in Simulation Three with the same training and the test sets. Several four-layered MLPs are tested and it is found that 1-3-3-1 MLP with 22 parameters can achieve similar performance as that of 1-9-1 MLP consisting of 28 parameters. After 447 epochs, the training and test errors (MSE) decrease to 9.97×10^{-5} and

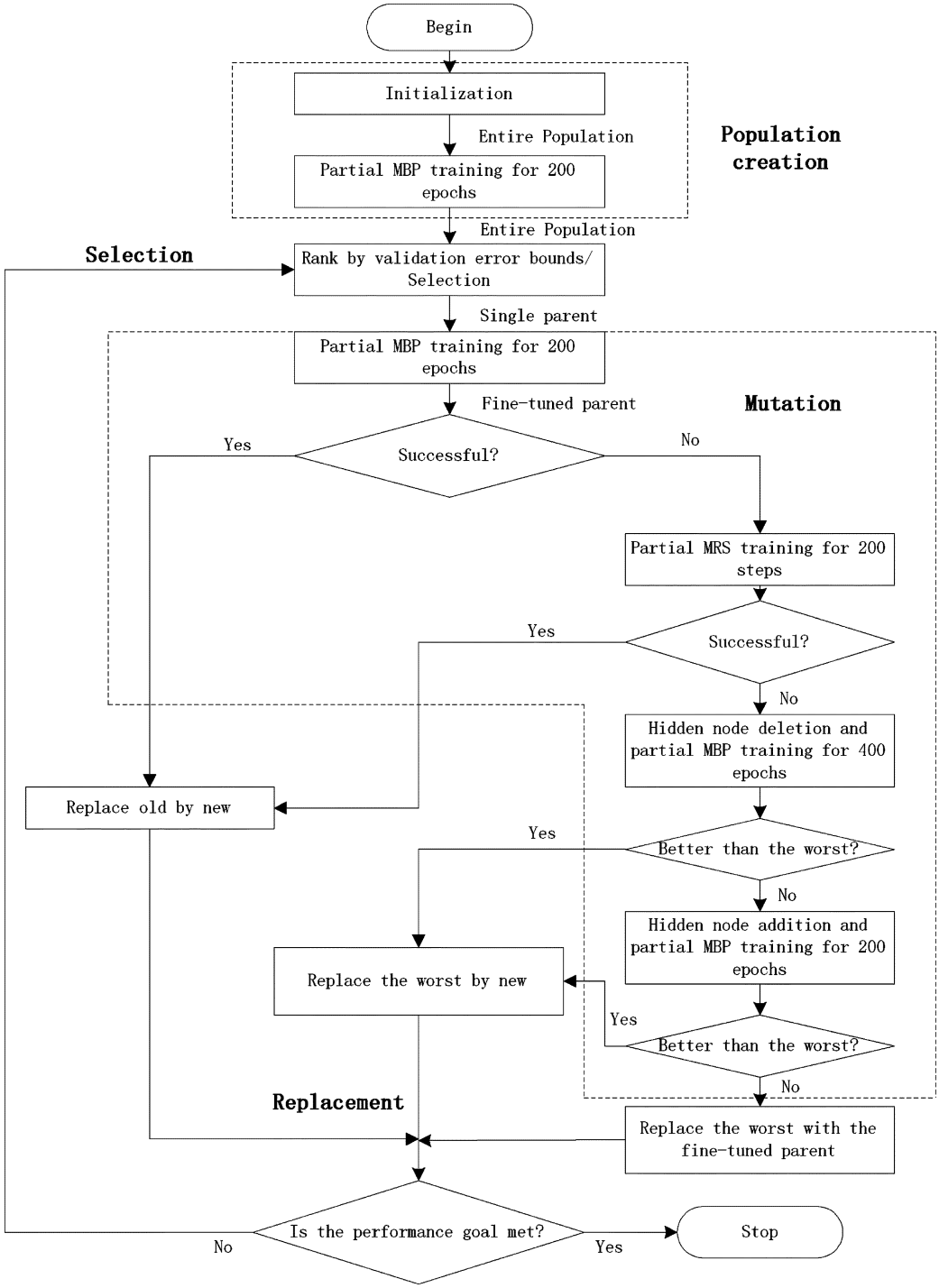


Fig. 12. Flowchart of the simplified EPNET.

9.72×10^{-5}, and the error bound of the test set is about 0.01. Due to local minima problem, it is hard to get a good result by only one trial, and the success rate is about one out of ten.

Simulation Seven: We also revisit the 2-D problem in Simulation Five with the same training and test data sets. A 2-4-5-1 MLP is searched out to approximate the function satisfactorily. The total number of the parameters of this four-layered MLP is 43, while the total number of the parameters for the former 2-13-1 network is 53. After 1241 epochs, the training and test errors (MSE) reduce to 9.98×10^{-5} and 1.09×10^{-4}, respectively, and the test error bound is about 0.05.

From above two simulation examples, it is clear that four-layered MLP is more efficient than three-layered MLP in terms of the minimal number of parameters needed to achieve similar performance. However, the difference between the minimal numbers of the parameters usually is not very large, and the three-layered MLP may be more appealing considering the fact that four-layered MLP may be more prone to local minima

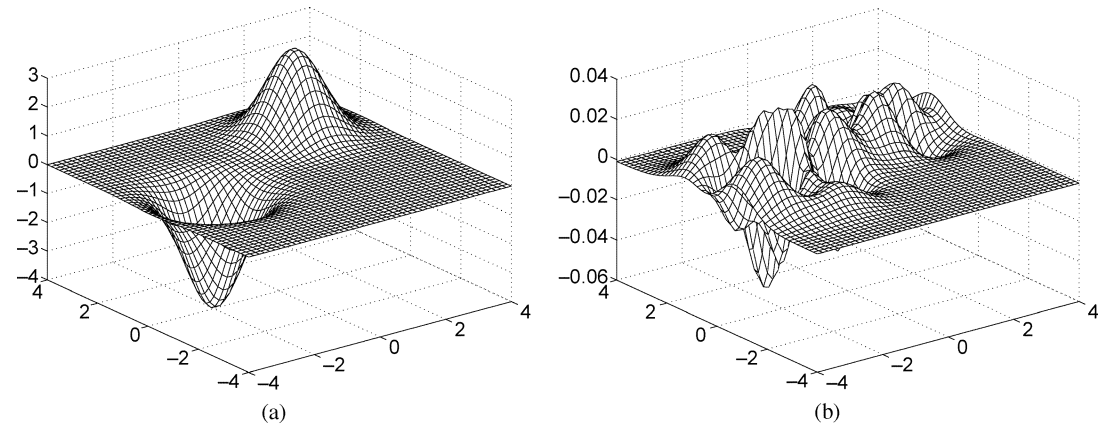


Fig. 13. Simulation Eight: approximating hill and valley with a 2-4-2-1 MLP. (a) Output of the neural network. (b) Approximation error.

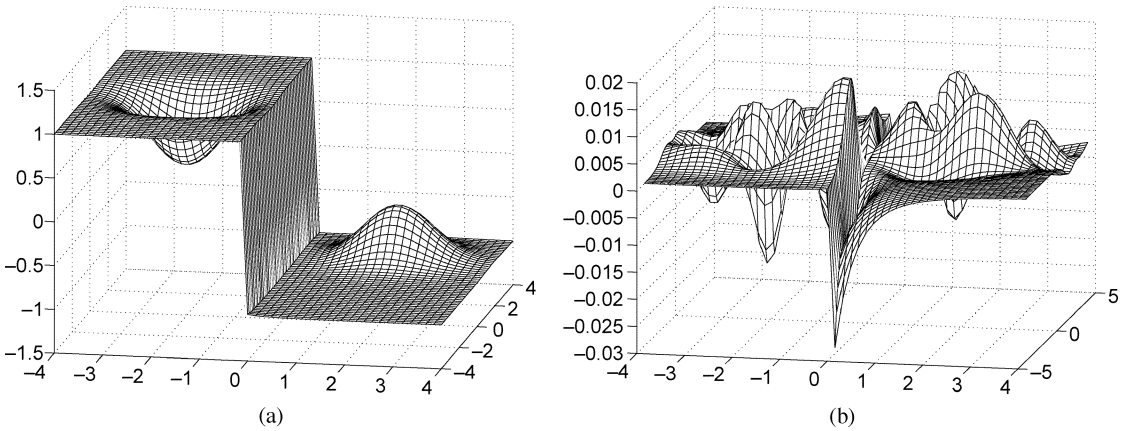


Fig. 14. Simulation Nine: approximating hill and valley with a cliff by a 2-5-1 MLP. (a) Output of the neural network. (b) Approximation error.

traps because of its more complicate structure as pointed out in [30]. But there are certain situations that four-layered MLP is distinctively better than three-layered MLP as illustrated below.

Simulation Eight: Consider an example [31] made of a Gaussian hill and a Gaussian valley as follows:

$$f(x,y) = 3e^{\left(-(x-2)^2-(y-2)^2\right)} - 4e^{\left(-(x+2)^2-y^2\right)}, \quad x,y \in [-4,4]. \quad (17)$$

The training set consists of 6561 points, which are sampled by uniformly partitioning the domain $x, y \in [-4,4]$ with grid size of 0.1. The test set comprises 2500 points randomly chosen from the same domain. A 2-4-2-1 network is used to approximate it quite well as shown in Fig. 13. The training error (MSE) reduces to 9.97×10^{-5} after 102 epochs, the test error (MSE) is 8.92×10^{-5} and the error bound is about 0.05. However, if three-layered MLP is used, then the minimal size has to be around 2-30-1 to achieve similar performance. The total number of parameters of 2-4-2-1 is only 25, while that of 2-30-1 is 121, which is much higher. Why does four-layered MLP outperform three-layered MLP so dramatically for this problem? Before we reveal the answer to this question, let us consider another related hill and valley example.

Simulation Nine: It is still a hill and valley problem as described below and shown in Fig. 14, with training and test data set constructed in the same way as that in Simulation Eight

$$f(x,y) = \begin{cases} 0.6e^{\left(-(x-2)^2-(y-2)^2\right)} - 0.8e^{\left(-(x+2)^2-y^2\right)} - 1 \\ \qquad\qquad x \in [0,4], \quad y \in [-4,4] \\ 0.6e^{\left(-(x-2)^2-(y-2)^2\right)} - 0.8e^{\left(-(x+2)^2-y^2\right)} + 1 \\ \qquad\qquad x \in [-4,0), \quad y \in [-4,4] \end{cases}. \quad (18)$$

At first glance of the geometrical shape of this function, it appears more complicated than that of the previous example because of the sharp discontinuity, i.e., a cliff, presented along the line $x = 0$, and a larger sized MLP would be expected to approximate it satisfactorily. However, a stunningly simple 2-5-1 three-layered MLP with hyperbolic tangent function as the activation function for the output neuron can approximate it astonishingly well with training error (MSE) of 1.84×10^{-5} and test error (MSE) of 1.85×10^{-5} after only 200 epochs. And the test error bound is even less than 0.03, as shown in Fig. 14.

After careful analysis of these two examples, it is finally realized that the essential difference between these two examples is the location of the flat areas. The flat regions in Simulation Eight lie in the middle, while those in Simulation Nine are located on the top as well as at the bottom. It is noticed previously in the Gaussian function example (Simulation Four) that the sigmoid

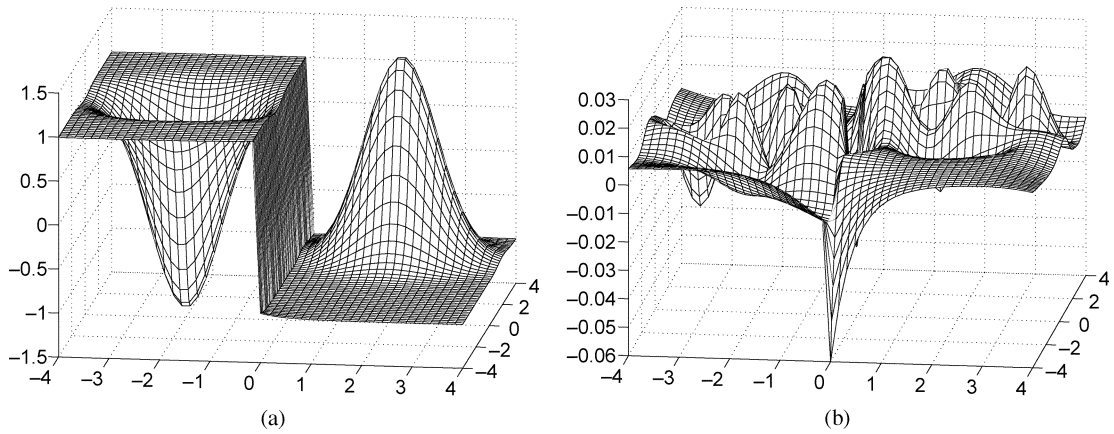


Fig. 15. Simulation Ten: the modified example of hill and valley with a cliff. (a) Output of the neural network. (b) Approximation error.

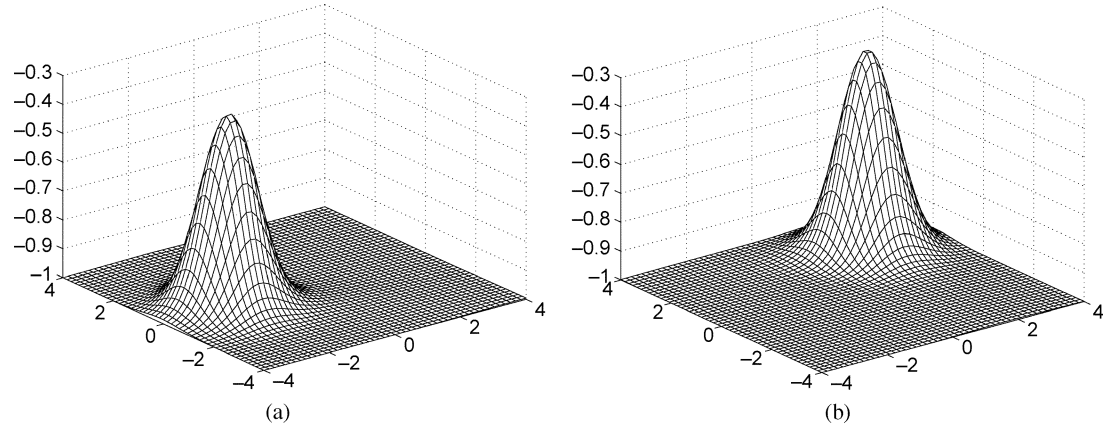


Fig. 16. Outputs of the neurons in the second hidden layer for the 2-4-2-1 MLP. (a) Output of the first hidden neuron. (b) Output of the second hidden neuron.

function has the nice property of flattening things outside its focused domain, but the flat levels must be located either on the top or at the bottom, dictated by its geometrical shape. Therefore, it is much easier to approximate the function in Simulation Nine with three-layered MLP than the function in Simulation Eight. To further verify this explanation, we increase the height of the hill as well as the depth of the valley in Simulation Nine such that they are higher or lower than the two flat planes, then it becomes very difficult to approximate with three-layered MLP, as demonstrated in the following simulation.

Simulation Ten: The target function in Simulation Nine is modified as follows:

$$f(x,y)=\begin{cases} 2.3e^{\left(-(x-2)^2-(y-2)^2\right)}-2.4e^{\left(-(x+2)^2-y^2\right)}-1 \\ \qquad\qquad x\in[0,4],\quad y\in[-4,4] \\ 2.3e^{\left(-(x-2)^2-(y-2)^2\right)}-2.4e^{\left(-(x+2)^2-y^2\right)}+1 \\ \qquad\qquad x\in[4,0),\quad y\in[4,4] \end{cases}. \tag{19}$$

The difference between this example and that of Simulation Nine is that the two flat planes are no longer present at the top or the bottom any more. The sampling points of training and test sets remain the same as those in Simulation Nine. The number of hidden neurons has to be increased from 5 to around 35 for the three-layered MLP, while a simple 2-5-2-1 MLP can approximate it quite well if four-layered MLP is used. After 1000 epochs, the training error (MSE) goes to 6.33×10^{-5}, the MSE and error bound of test set are 6.42×10^{-5} and 0.06, respectively. The result is shown in Fig. 15.

From above discussion the reason why a simple 2-4-2-1 four layered MLP can approximate the hill and valley very well in Simulation Eight should be also clear now. As we mentioned before, the four-layered MLP has the capability of decomposing the task of approximating one target function into tasks of approximating subfunctions. If the target function with flat regions in the middle as in the case of Simulation Eight and Ten can be decomposed into linear combination of subfunctions with flat areas on the top or at the bottom, then this target function can be approximated satisfactorily by a four-layered MLP because each of the subfunction can be well approximated by a three-layered MLP now. To validate this explanation, the outputs of the hidden neurons in the second hidden layer of the 2-4-2-1 network in Simulation Eight are plotted out in Fig. 16, which are interestingly in the shape of a hill with flat areas around. It is apparent that these two subfunctions which are constructed by three-layered MLPs can easily combine into a shape consisting of a hill and a valley by subtraction.

Comment 15: The way of decomposing the target function by the four-layered MLP is not unique and largely depends upon the initialization of the weights. For instance, the shapes of the outputs of the hidden neurons are totally different from those of Fig. 16, as shown in Fig. 17, when different initial weights

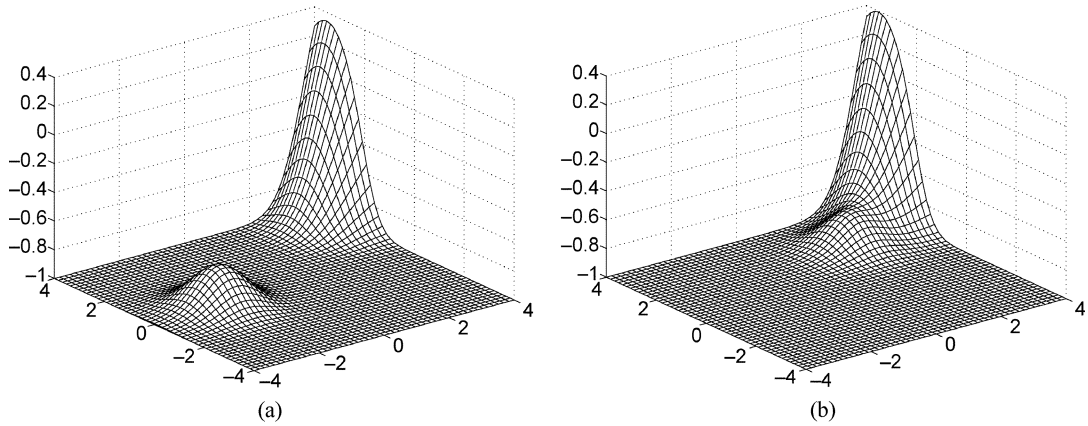


Fig. 17. Outputs of the neurons in the second hidden layer with different initialization. (a) Output of the first hidden neuron. (b) Output of the second hidden neuron.

are used. However, both of them share the common feature that the flat areas are all located at the bottom, which can be easily approximated by three-layered MLPs.

In summary, we have following two guidelines regarding the choice of one or two hidden layers to use.

Guideline Two: Four-layered MLP may be considered for purpose of decreasing the total number of the parameters. However, it may increase the risk of falling into local minima in the mean time.

Guideline Three: If there are flat surfaces located in the middle of the graph of the target function, then four-layered MLP should be used instead of three-layered MLP.

Comment 16: The Gaussian hill and valley example is the most well known example [31] to show the advantage of using two hidden layers over using one hidden layer. However, very little explanation has been provided except Chester suggested an interpretation in [27], which was not well founded.

Comment 17: Sontag proved in [28] that a certain class of "inverse" problems in general can be solved by functions computable by four-layered MLPs, but not by the functions computable by three-layered MLPs. However, the precise meaning of "computable" defined in [28] is exact representation, not approximation. Therefore, his result does not imply the existence of functions that can be approximated only by four layered MLPs, but not by three-layered MLPs, which is still consistent with the universal approximation theorem.

V. Conclusion

A geometrical interpretation of MLPs is suggested in this paper, on the basis of the special geometrical shape of the activation function. Basically, the hidden layer of the three-layered MLP provides the basic building blocks with shapes very close to the piecewise lines (or piecewise hyperplanes in high-dimensional cases). The widths, heights and positions of these building blocks can be arbitrarily adjusted by the weights and biases.

The four-layered MLP is interpreted simply as linear combination of multiple three-layered MLPs that share the same hidden neurons but with different output neurons. The number of the neurons in the second hidden layer is then the number of these three-layered MLPs which construct corresponding subfunctions that would combine into an approximation of the target function.

Based upon this interpretation, three guidelines for selecting the architecture of the MLP are then proposed. It is demonstrated by extensive simulation studies that these guidelines are very effective for searching the minimal structure of the MLP, which is crucial in many application problems. For easy reference, these guidelines are summarized here again as follows.

Guideline One: Choose the first trial for the number of the hidden neurons of the three-layered MLP as the minimal number of line segments (or hyperplanes in high-dimensional cases) that can approximate the basic geometrical shape of the target function which is given *a priori* or may be perceived from the training data. This number can also be used to generate the initial population for EANN or the starting point for growing and pruning the neural networks, which may speed up the learning process substantially.

Guideline Two: Four-layered MLP may be considered for purpose of decreasing the total number of the parameters.

Guideline Three: If there are flat surfaces located in the middle of the graph of the target function, then four-layered MLP should be used instead of three-layered MLP.

The suggested geometrical interpretation is not only useful to guide the design of MLP, but also sheds light on some of the beautiful but somewhat mystic properties of the MLP. For instance, the universal approximation property can now be readily understood from the viewpoint of piecewise linear approximation as proven in Theorem 1. And also it does not escape our notice that this geometrical interpretation may provide a light to illuminate the advantage of MLP over other conventional linear regression methods, shown by Barron [32], [33], that the MLP may be free of the "curse of dimensionality," since the number of the neurons of MLP needed for approximating a target function depends only upon the basic geometrical shape of the target function, not on the dimension of the input space.

While the geometrical interpretation is still valid with the dimension of the input space increasing, the guidelines can be hardly applied because the basic geometrical shapes of high-dimensional target functions are very difficult to determine. Consequently, how to extract the geometrical information of a high-dimensional target function from the training data available would be a very interesting and challenging problem.

ACKNOWLEDGMENT

The authors would like to thank K. S. Narendra for suggesting the architecture selection problem of the neural networks. The authors also wish to thank the anonymous reviewers for their valuable comments to improve the quality of this paper.

REFERENCES

[1] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," *Adv. Neural Inform. Process. Syst.*, vol. 2, pp. 598–605, 1990.
[2] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination with application to forecasting," *Adv. Neural Inform. Process. Syst.*, vol. 3, pp. 875–882, 1991.
[3] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 1, San Francisco, CA, 1992, pp. 293–299.
[4] D. R. Hush, "Learning from examples: from theory to practice," in *Proc. Tutorial #4, 1997 Int. Conf. Neural Networks*, Houston, TX, Jun. 1997.
[5] E. Alpaydim, "GAL: networks that grow when they learn and shrink when they forget," *Int. J. Patt. Recognit. Artif. Intell.*, vol. 8, no. 1, pp. 391–414, 1994.
[6] T. Jasic and H. Poh, "Artificial and real world mapping problems," in *Lecture Notes in Computer Science*. New York: Springer-Verlag, 1995, vol. 930, pp. 239–245.
[7] M. Sarkar and B. Yegnanarayana, "Evolutionary programming-based probabilistic neural networks construction technique," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'97)*, pp. 456–461.
[8] P. A. Castillo, J. Carpio, J. J. Merelo, V. Rivas, G. Romero, and A. Prieto, "Evolving multilayer perceptrons," *Neural Process. Lett.*, vol. 12, no. 2, pp. 115–127, 2000.
[9] X. Yao, "Evolutionary artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[10] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, pp. 359–366, 1989.
[11] G. Cybenko, "Approximation by superpositions of sigmoidal function," *Math., Control, Signals, Syst.*, vol. 2, pp. 303–314, 1989.
[12] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Netw.*, vol. 2, pp. 183–192, 1989.
[13] D. W. Marquardt, "Nonlinear modeling," *J. Soc. Ind. Appl. Math.*, vol. 11, pp. 431–441, 1963.
[14] J. J. Mor, "The Levenberg-Marquardt algorithm: implementation and theory, in numerical analysis," in *Lecture Notes in Mathematics*, G. A. Watson, Ed. Berlin, Germany: Springer-Verlag, 1977, vol. 630, pp. 105–116.
[15] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural network by choosing initial values of the adaptive weights," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'90)*, vol. 3, 1990, pp. 21–26.
[16] D. R. Hush and J. M. Salas, "Improving the learning rate of back-propagation with the gradient reuse algorithm," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 1, San Diego, CA, 1988, pp. 441–447.
[17] A. Mennon, K. Mehrotra, C. K. Mohan, and S. RanKa, "Characterization of a class of sigmoid functions with applications to neural networks," *Neural Netw.*, vol. 9, pp. 819–835, 1996.
[18] S. Amari, "Natural gradient works efficiently in learning," *Neural Computat.*, vol. 10, no. 2, pp. 251–276, 1998.
[19] D. Plaut, S. Nowlan, and G. Hinton, "Experiments on learning by back-propagation," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-CS-86-126, 1986.
[20] J. E. Moody and T. Rögnvaldsson, "Smoothing regularizers for projective basis function networks," *Adv. Neural Inform. Process. Syst.*, vol. 9, pp. 585–591, 1997.
[21] D. J. C. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, pp. 415–447, 1992.
[22] ——, "A practical Bayesian framework for back-propagation networks," *Neural Computat.*, vol. 4, pp. 448–472, 1992.
[23] S. Q. Ding and C. Xiang *et al.*, "Overfitting problem: a new perspective from the geometrical interpretation of MLP," in *Design and Application of Hybrid Intelligent Systems*, A. Abraham *et al.*, Eds. Amsterdam, The Netherlands: IOS Press, 2003, pp. 50–57.
[24] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.
[25] G. A. Riessen, G. J. Williams, and X. Yao, "PEPNet: parallel evolutionary programming for constructing artificial neural networks," in *Proc. 6th Int. Conf. Evolutionary Programming*, 1997, pp. 35–45.
[26] F. Solis and R. B. Wets, "Minimization by random search techniques," *Math. Oper. Res.*, vol. 6, no. 1, pp. 19–30, 1981.
[27] D. L. Chester, "Why two hidden layers are better than one," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'90)*, vol. 1, 1990, pp. 265–268.
[28] E. D. Sontag, "Feedback stabilization using two-hidden-layer nets," *IEEE Trans. Neural Netw.*, vol. 3, no. 6, pp. 981–990, Nov. 1992.
[29] S. Tamura and M. Tateishi, "Capabilities of a four-layered feed-forward neural network: four layers versus three," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 251–255, Mar. 1997.
[30] J. D. Villiers and E. Barnard, "Backpropagation neural nets with one and two hidden layers," *IEEE Trans. Neural Netw.*, vol. 4, no. 1, pp. 136–141, Jan. 1992.
[31] Newsgroup: comp.ai.neural-nets FAQ (2002). [Online]. Available: ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hl
[32] A. R. Barron, "Neural net approximation," in *Proc. 7th Yale Workshop Adaptive and Learning Systems*, New Haven, CT, 1992, pp. 69–72.
[33] ——, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 930–945, May 1993.

Cheng Xiang (M'01) received the B.S. degree in mechanical engineering from Fudan University, China, in 1991, the M.S. degree in mechanical engineering from the Institute of Mechanics, Chinese Academy of Sciences, in 1994, and the M.S. and Ph.D. degrees in electrical engineering from Yale University, New Haven, CT, in 1995 and 2000, respectively.

From 2000 to 2001, he was a Financial Engineer at Fannie Mae, Washington, DC. Currently, he is an Assistant Professor in the Department of Electrical and Computer Engineering, National University of Singapore. His research interests include computational intelligence, adaptive systems, and pattern recognition.

Shenqiang Q. Ding received the B.Eng. degree in automation from University of Science and Technology of China (USTC), Hefei, China, in 2002 and the M.Eng degree in electrical and computer engineering from the National University of Singapore, Singapore, in 2004.

He is currently a System Engineer in STMicroelectronics, Corporate R&D, Singapore. His research interests include computational intelligence, nano device modeling, and chaotic time series prediction.

Tong Heng Lee (M'00) received the B.A. degree (First Class Honors) in engineering tripos from Cambridge University, Cambridge, U.K., in 1980 and the Ph.D. degree from Yale University, New Haven, CT, in 1987.

He is a Professor in the Department of Electrical and Computer Engineering at the National University of Singapore, Singapore. He is also currently Head of the Drives, Power, and Control Systems Group in this Department, and Vice-President and Director of the Office of Research at the University. His research interests are in the areas of adaptive systems, knowledge-based control, intelligent mechatronics, and computational intelligence. He currently holds Associate Editor appointments in *Automatica*, *Control Engineering Practice*, the *International Journal of Systems Science*, and *Mechatronics*. He has also coauthored three research monographs, and holds four patents (two of which are in the technology area of adaptive systems, and the other two are in the area of intelligent mechatronics).

Dr. Lee is a recipient of the Cambridge University Charles Baker Prize in Engineering. He is Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS B.

Estimating the Number of Hidden Neurons in a Feedforward Network Using the Singular Value Decomposition

E. J. Teoh, K. C. Tan, and C. Xiang

***Abstract*—In this letter, we attempt to quantify the significance of increasing the number of neurons in the hidden layer of a feedforward neural network architecture using the singular value decomposition (SVD). Through this, we extend some well-known properties of the SVD in evaluating the generalizability of single hidden layer feedforward networks (SLFNs) with respect to the number of hidden layer neurons. The generalization capability of the SLFN is measured by the degree of linear independency of the patterns in hidden layer space, which can be indirectly quantified from the singular values obtained from the SVD, in a postlearning step. A pruning/growing technique based on these singular values is then used to estimate the necessary number of neurons in the hidden layer. More importantly, we describe in detail properties of the SVD in determining the structure of a neural network particularly with respect to the robustness of the selected model.**

I. Introduction

The ability of a neural network to generalize well on unseen data depends on a variety of factors, most important of which is the matching of the network complexity with the degree of freedom or information that is inherent in the training data. Matching of this information with complexity is critical for networks that are able to generalize well. A measure of the complexity of a structure is its number of free, or adjustable parameters, which for a feedforward neural network is the number of synaptic weights. Clearly, the capacity of a feedforward neural network in learning the samples of the training set is proportional to its complexity [1], [2]. For a simple single hidden layer feedforward network (SLFN) architecture, the number of weights in the structure is a function of the number of hidden layer neurons; each hidden neuron added increases the number of connections by a factor of $(M+1)+C$ where M and C are the number of input and output nodes, respectively, (the additional dimension accounts for the bias for the hidden layer neurons). See [3] for an overview of the complexity of learning in neural networks.

The objective here is to determine a parsimonious model that provides a reliable performance for the given (training) data, with the smallest structural complexity—as the model complexity increases, with all other factors (such as the training data) held constant, the performance of this network improves up to a certain limit in the complexity, after which the model performance on the unseen testing data then deteriorates. Clearly, there is a tradeoff in the approximation obtained in the training set and the generalization of the unseen testing data. The aim of structural selection is to find this compromise value. If the model complexity is below it, underfitting occurs; conversely overfitting occurs when the model complexity is above this compromise value. Determining this value is difficult, depending not only on our definition of an "optimal" value but also on the amount and quality of data. A computationally simple approach would thus be best, as will be put forward in this letter.

Manuscript received October 12, 2005; revised April 20, 2006.

The authors are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117576, Singapore (e-mail: eletankc@nus.edu.sg).

Color versions of Figs. 1–8 are available online at http://ieeexplore.ieee.org.
Digital Object Identifier 10.1109/TNN.2006.880582

II. Preliminaries

A. Related Work

Previous work on estimating the number of hidden layer neurons have often focused on the learning capabilities of the SLFN on a training set without accounting for the possibility of the network being overtrained [4]–[7], [1], [2]. Projection onto increasingly higher dimensional (hidden layer) space is achieved by implementing increasingly larger number of neurons in the hidden layer. Cover's seminal work using concepts from combinatorial geometry rigorously showed that patterns projected onto higher dimensions are more likely to be linearly separable [8]. This work was followed by Huang [4] and Sartori and Antsaklis [5], demonstrating that a simple SLFN can achieve perfect accuracy on an N-size problem (there are N samples in the training set) when N hidden neurons are used—this creates N hyperplanes that partition the N samples into N distinct regions. This approach however, was achieved using a simple matrix inverse without any use of any iterative training algorithms.

The use of the singular value decomposition (SVD) in neural networks have been briefly highlighted by [9]–[11]. [12] used the SVD to demonstrate that the effective ranks of the hidden unit activations and the hidden layer weights are equal to each other as the solution converges thus indicating that the learning algorithm (the backpropagation using gradient descent was used) can be stopped. On the other hand, geometrical methods such as in [13] have been used to great effect in providing greater insight into the nature of the problem; however, such methods are only applicable to problems in which the dimensionality of the problem is in either two–dimensional (2-D) or three-dimensional (3-D) for the problem is thus visualizable.

Traditionally, the simplest possible approach in network pruning would be to first train the network and to subsequently remove weights that are of small magnitudes—clearly this method, though simple, has obvious flaws. Other more sophisticated pruning techniques such as *optimal brain damage* (OBD) [14] and *optimal brain surgeon* (OBS) [15] make use of the Taylor approximation of the error functional, focusing on the second-order Hessian matrix in determining the sensitivities of the weights of the trained network to remove redundant weights.

B. Notations

For the purpose of clarity and uniformity, we define some symbols and abbreviations that will be used in this letter. Consider an SLFN with n hidden neurons trained on a set of N training patterns, X of dimension $M+1$ (an added dimension is included to account for the bias input). The nonlinearity of the hidden layer activation functions is $h(\cdot)$. The output activations of these hidden neurons are represented by the matrix H, where $H = h(XW)$ with $X \in \mathbb{R}^{N\times(M+1)}, W \in \mathbb{R}^{(M+1)\times n}$ and $H \in \mathbb{R}^{N\times n}$. Typically, we have an overdetermined system where $N \geq M$ (H is *tall and thin*). The discussion that follows throughout this letter will center about the singular value decomposition (SVD) of the hidden layer activation matrix H.

III. The Singular Value Decomposition (SVD)

In this section, we motivate the use of the SVD as a tool to estimate the necessary number of neurons to be used in training an SLFN. Given a real matrix $H \in \mathbb{R}^{N\times n}$, applying the SVD results in the orthogonal transformation. $U \in \mathbb{R}^{N\times N}$ is known as the left singular vectors of H and $V \in \mathbb{R}^{n\times n}$ is known as the right singular vectors of H. Both U and V are orthonormal. $\Sigma \in \mathbb{R}^{N\times n} = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$ where $(\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0 = \sigma_{n+1} = \cdots = \sigma_N)$ is a diagonal matrix with unique, nonnegative entries ordered in decreasing magnitude. This

1045-9227/$20.00 © 2006 IEEE

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 10, 2009 at 19:35 from IEEE Xplore. Restrictions apply.

decoupling technique of the SVD allows the expression of the original matrix as a sum of the first n columns of u and v^T, weighted by the singular values, i.e., $H = \sum_{i=1}^{n} \sigma_i u_i v_i^T$. The matrix $M_i = u_i v_i^T$ is known as the ith mode of H, where $H = \sigma_1 M_1 + \sigma_2 M_2 + \cdots + \sigma_n M_n$. The rank of H is determined by observing that the n largest singular values are nonzero, whereas the $N - n$ smallest singular values are zero.

Supposing $\tilde{H} = H + E$, let the singular values of H and $\tilde{H}$ be $(\sigma_1, \ldots, \sigma_k)$ and $(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_k)$, respectively. Likewise, we let U_1, U_2, V_1, V_2 and $\tilde{U}_1, \tilde{U}_2, \tilde{V}_1, \tilde{V}_2$ be the orthonormal bases of H and $\tilde{H}$, respectively. From Schmidt's *subspace theorem* [16], we have (with $\|\cdot\|_F$ denoting the Frobenius norm)

$$\tilde{\sigma}_{k+1}^2 + \cdots + \tilde{\sigma}_n^2 \leq \|E\|_F^2. \tag{1}$$

With this, the SVD of $\tilde{H}$ reveals the rank in a manner that its $n - k$ smallest singular values are bounded by the Frobenius norm of E [16]. This also allows us to conclude that the spaces spanned by $\tilde{U}_1, \tilde{U}_2, \tilde{V}_1, \tilde{V}_2$ are approximations to the original space spanned by U_1, U_2, V_1, V_2 to about $\sigma_k^{-1}\|E\|$. Hence, if σ_k is reasonably large as compared to E, the SVD provides a good approximation to the desired subspaces [16]. The use of the SVD, specifically its spectral properties through its singular values (the diagonals of Σ), provides us with an indication of the degree of linear independency of the matrix H.

Algorithms for computing the SVD can be found in [17], where its derivations, concepts, and applications are discussed further in depth. This letter will focus on the use of the SVD as a robust tool in estimating the number of hidden layer neurons without focusing on the algorithmic details of the SVD, which can be found in the literature highlighted previously.

IV. Estimating the Number of Hidden Layer Neurons

A. The Construction of Hyperplanes in Hidden Layer Space

Every neuron in the hidden layer constructs a hyperplane in the input feature space [4]. Quite clearly, and in a geometrical sense, the rank of H denotes the space in which the columns of H occupy, representing the number of (unique) separating hyperplanes of the system. In the case where $n = N - 1$ hidden neurons are used (with a suitable nonlinear activation function) such that H is full-ranked, this satisfies the rank requirement of [4], [5], and [1], giving rise to $N - 1$ separating hyperplanes for the N training samples. Using the simple matrix inverse [4], [5], [1], we are guaranteed perfect reconstruction for the training set.[1] Intuitively, if H is of higher rank, H better fits the training data at the expense of generalization when $\lim_{n \to N} \text{rank}(H)$. This full-rank condition also ensures that the patterns projected onto the hidden layer space are linearly independent; accordingly, this is also known as ϕ-*general position* as described in [8]. h is equivalent to ϕ in the context of this letter.

B. Actual Rank (K) Versus Numerical Rank (N): H_k Versus H_n

The actual rank (say k) of H may be different from its numerical rank (say n), where $k \leq n$. Such a situation usually arises when the original matrix X is "contaminated" by E, resulting in a matrix $\tilde{H}$, such that $\tilde{H} = X + E$ [16], with $\text{rank}(H) = k$ and $\text{rank}(\tilde{H}) = n$. This "contamination" we commonly refer to as *noise* obstructs the characterization of the true properties of the system or problem given the observed training samples. As will be highlighted subsequently, this "noise" (when taken in our context) corresponds to the marginal role that is played by additional hidden layer neurons that tend to fit the features of the training samples which are not truly representative of the intrinsic underlying distribution of observations. Aside from noise, this "contamination" can also consist of environment and measurement errors or features that is only present in the training but not the testing set.

Knowledge of the rank of H is often the initial stage in determining further information that can be derived from the column and null spaces of H. The use of the SVD as a rank-revealing decomposition allows us to obtain approximations to the subspaces spanned by H_n and H_k. Thus, even if the transformed patterns produced in the hidden layer space are in general position, some of these patterns may be degenerate, in the sense that certain projected patterns can "almost" be represented as a linear combination of other projected patterns. However, is this additional hidden neuron contributing to the actual separability of the samples, or is it merely compensating for the noise that is present in the observations? Clearly, there is a limit, for which introducing additional hidden neurons will tend to overfit the training data. The *numerical rank* of H is defined as the number of linearly independent columns of H. Equivalently, the rank of a matrix H is also the minimum n which satisfies $\sigma_n > \sigma_{n+1} = 0$ with $\sigma_i (i = \{1, 2, \ldots, N\})$ being the singular values of H. However, a value k which describes the *actual rank* of the matrix H is more useful for us in estimating the appropriate number of hidden neurons to be used in the SLFN for a given problem.

Often, the presence of degeneracy in H is subtle and not readily apparent - this is to say that even if H is full-ranked, it may be numerically rank-deficient or unstable in that it has nonzero singular values that are close to zero. Under many practical circumstances, the presence of noise and error will give rise to a matrix $\tilde{H}$ that from a mathematical perspective is full-ranked and linearly independent, yet is *almost* linearly dependent. The SVD allows us to estimate the number of columns of H (corresponding to the number of hidden layer neurons that should be used for the problem) that are most linearly independent.

This suggests that the column space of H_k spans a region that is almost similar to that spanned by the column space of H_n. Conceptually, we may arrive at the conclusion that H_n is the result of a perturbation of a rank-k matrix H_k. Thus we use a more general concept of the rank of a matrix by specifying a tolerance below which it is of full rank. In the context of representing the input patterns in hidden layer space, we can think of additional hidden layer neurons as causing degeneracy in this hidden layer space, for increasing the number of hidden layer neurons is similar to introducing noise into the system, thus perturbing the hidden layer space (which could have been reasonably represented using k hidden neurons instead) such that the hidden layer space is now being represented by n hidden neurons which are of marginal benefit.

Theorem 1: Define the *numerical ϵ-rank* k_ϵ of the matrix H with respect to some tolerance ϵ [18] by

$$k_\epsilon = k_\epsilon(H, \epsilon) \equiv \min_{\|E\|_2 \leq \epsilon} \{\text{rank}(H + E)\}. \tag{2}$$

Theorem 1 states that if there is a "gap" between the k_ϵth and the $k_{\epsilon+1}$th singular values of size ϵ (i.e., $\sigma_{k_\epsilon} > \epsilon > \sigma_{k_\epsilon+1}$), then H has actual rank $(\epsilon - \text{rank})$ k_ϵ. The larger this gap ϵ is, the more "robust" the matrix H' is to perturbation (the more "confident" we are that the matrix H can be truncated at k)—Hansen [18] further elucidates that the ϵ-rank of a matrix H is equal to the number of columns of H that are guaranteed to be linearly independent for any perturbation of H with norm less than or equal to the tolerance ϵ, i.e., (robust to perturbations of E such that $\|E\|_2 \leq \epsilon$). The sensitivity to the errors E is now influenced by the ratio of σ_1/σ_k instead of the usual conditional number of H, $\text{cond}(H) = \sigma_1/\sigma_n$. Golub *et al.* [19], from which the following definition originates, provides an excellent treatment of the

[1] A reviewer suggested that the validity of [4] and [5] is based on the condition that the same value may be chosen for all the hidden biases; however, in many practical applications, the hidden biases may be adjusted with different values. In fact, as pointed out by [1], N hidden neurons are actually used in [4] and [5].

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 10, 2009 at 19:35 from IEEE Xplore. Restrictions apply.

concept of numerical rank. To avoid possible problems when ϵ is itself perturbed, the definition of actual rank is refined by introducing δ as an upper bound for ϵ for which the numerical rank remains at least equal to k.

Theorem 2: The matrix H has a numerical rank of (δ, ϵ, r) with respect to the norm $\|\cdot\|$ if δ, ϵ and r satisfies the following:

i) $k = \inf\{\text{rank}(\tilde{H}) : \|H - \tilde{H}\| \leq \epsilon\}$;

ii) $\epsilon < \delta \leq \sup\{\eta : \|H - \tilde{H}\| \leq \eta \Rightarrow \text{rank}(\tilde{H}) \geq k\}$. σ_k provides an upper bound for δ while σ_{k+1} provides a lower bound for ϵ, i.e., $\delta \leq \sigma_r$ and $\epsilon \geq \sigma_{k+1}$. This is the best ratio that can be obtained for ϵ/γ is σ_{k+1}/σ_k. Note that the norms used previously are either the L_2 or Frobenius norms.

The previous definitions are equivalent to saying that the matrix H is linearly-independent when perturbed by E up to a threshold determined by $\|E\|_2 \leq \epsilon$. This result also means that the singular values of H satisfy $\sigma_{k_\epsilon} > \epsilon > \sigma_{k_\epsilon+1}$. This concept will be used in Section V as a robustness measure of the estimate that is made of the number of hidden layer neurons. As described in [18] and originally in [19], a *well-determined gap* between the singular values of σ_{k_ϵ} and $\sigma_{k_\epsilon+1}$, represented by ϵ should exist in order for the previous definition to make much sense; k_ϵ should be, in other words, well-defined for small perturbations of the threshold ϵ and the singular values. Alternatively, the numerical ϵ-rank is the smallest integer k for which $\sum_{j=k+1}^{n} \sigma_j^2 \leq \epsilon^2$ (again, Schmidt's *subspace theorem*).

This result suggests that as more neurons are added to the hidden layer, the contributory effect of each additional hidden neuron decreases after a certain threshold. In economic theory, this could be seen as *diminishing marginal returns*. From a geometric point of view, additional hyperplanes constructed by these newly introduce hidden neurons are not *that* unique, or different as compared to existing hyperplanes (these new hyperplanes may be almost parallel to existing ones). The "significance" of these new hyperplanes can be quantified by examining the singular values of the matrix H as more hidden layer neurons are added.

These singular values indicate the degree of mutual correlation between features in the hidden layer space with column degeneracy resulting when these hidden space features are highly correlated, which in turn leads to the conclusion that these additional neurons are redundant. While the singular values do not provide information on which of these features are correlated, the presence of small singular values would indicate that these additional hidden neurons can be removed without affecting the performance of the SLFN significantly. However, it is the relative and not the absolute magnitude of the singular values that interests us - this is described in further detail in Section V.

V. Pruning/Growing Technique Based on the SVD

In this section, we define the "small" singular values purely in a relative sense. Here we describe some possible methods in selecting the threshold γ. These methods are largely heuristical in nature, as is common in the determination of model order in system identification problems [20].

A. Determining the Threshold

Suppose $H_n \in \mathbb{R}^{N \times n}$ represents the hidden layer output activation matrix of an SLFN with n neurons in the hidden layer. Let $\sigma_i(H_n)$ denote the ith singular value of the SLFN with n hidden neurons. A way to measure the contribution of the i singular value to say, the separability of classes, is to relate its value to other singular values in Σ.

Using the singular values obtained from the SVD, the *condition number* of any matrix H by definition is given by $\kappa(H_n) = \sigma_1(H_n)/\sigma_n(H_n) = \|H_n\| \cdot \|H_n^\dagger\|$, where σ_i is the ith singular value of Σ. The condition number is a measure of how far the matrix H_n is from a matrix H_k of lesser rank $(k < n)$ [21]. Adopting a similar approach, we can limit the number of hidden neurons to k, where we attempt to find a value of k which satisfies the following: $\min_k\{\sigma_{k+1}(H_n)/\sigma_1(H_n)\} \leq \gamma$, with γ as some threshold.

Alternatively, we could also take into account the sensitivity of the singular values $(\epsilon = \sigma_k - \sigma_{k+1})$, as an indication of the "robustness" in the selection of a number $k(< n)$. This sensitivity value is also used to determine the numerical $\epsilon -$ rank r_ϵ [18] of the matrix H (this was explained in Section IV)

$$r_\epsilon = r_\epsilon(H, \epsilon) \equiv \min_{\|E\|_2 \leq \epsilon} \{\text{rank}(H + E)\}. \tag{3}$$

Since the L_2-norm of H is equivalent to the sum-of-squares of the singular values of H (U and V are orthogonal and their norms are unity), another approach would be to find k such that selection of the k largest singular values retain the "spectral energy" of H, i.e., $\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2 \geq \gamma$ with γ being some prespecified threshold.

If ϵ is large, we can assume that the matrix H is relatively robust to perturbations (such as rounding, measurement or observation errors); conversely, if ϵ is selected to be small (but not too small such that the numerical $\epsilon -$ rank k_ϵ does not make sense [18], external noise that is introduced to the system may cause the matrix H to be rank-degenerate. Sometimes, there is no clear value for k where $\sigma_k - \sigma_{k+1}$ is obvious. If the SLFN has been well-trained and has converged (there is little change in its weight values), in some problems, the decay of the singular values is gradual and not very distinct, and hence we cannot conclude confidently that the numerical rank of matrix H is less than its actual rank.

In what follows, we use both σ_i and $\epsilon_i (i = \{1, \ldots, n\})$ to estimate the necessary number (k) of hidden layer neurons to implement. Recall that $\epsilon_k = \sigma_k - \sigma_{k+1}$. We associate the ϵ_i with σ_i. Ideally, we want to preserve all $i = \{1, 2, \ldots, k\}$ such that σ_i is large and ϵ_k is large. Simply, we want to retain singular values that contribute to the overall spectral energy (this corresponds to large $\sigma_i, i = \{1, 2, \ldots, k\}$) and yet remain sufficiently robust (this corresponds to a large ϵ_k).

Let $\hat{\sigma}_i$ denote the normalized or scaled singular value, i.e., $\hat{\sigma}_i = \sigma_i / \sum_{j=1}^{n} \sigma_j$. The simple measure (ρ_i) that we use is a λ-weighted (or regularized) linear combination of σ_i and ϵ_i, i.e., $\rho_i = \lambda\sigma_i + (1-\lambda)\epsilon_i$. Since $\epsilon_i = \sigma_i - \sigma_{i+1}$, we have $\rho_i = \lambda\sigma_i + (1-\lambda)(\sigma_i - \sigma_{i+1}) = \sigma_i + (\lambda - 1)\sigma_{i+1}$, where $\lambda \in [0, 1]$.

To summarize the possible measures or criteria that can be used in determining the appropriate or necessary value for $k = R(H, \gamma)$, we have the following:

1) $\min_k\{(\sigma_{k+1}/\sigma_1) \leq \gamma\}$ or $\min_k\{(\sigma_k/\sigma_1) < \gamma\}$;
2) $\min_k\{\epsilon_k \geq \gamma\}$;
3) $\min_k\{(\sum_{i=1}^{k} \sigma_i / \sum_{i=1}^{n} \sigma_i) \geq \gamma\}$ or $\min_k\{(\sum_{i=k+1}^{n} \sigma_i / \sum_{i=1}^{n} \sigma_i) < \gamma\}$;
4) $\min_k\{(\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2) \geq \gamma\}$ or $\min_k\{(\sum_{i=k+1}^{n} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2) < \gamma\}$;
5) $\min_k\{[\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2]^{(1/2)} \geq \gamma\}$;
6) $\min_k\{(\lambda\sigma_k + (1-\lambda)\epsilon_k) \leq \gamma\}$;
7) $\min_k\{(\lambda\hat{\sigma}_k + (1-\lambda)\hat{\epsilon}_k) \leq \gamma\}$.

Criteria 1)–5) have been previously considered in [22] and [17]. Here, in criteria 6) and 7), we propose using both the singular values σ and its associated sensitivity ϵ in determining the actual rank k of the matrix H_n. Presently, the threshold value γ is chosen heuristically and is largely problem-dependent—there is a lack of available theoretical or analytical expressions—however, [22] provides some statistical techniques in constructing suitable threshold values for γ, particularly for rank determination in signal processing problems. By varying the threshold γ we are able to "control" the confidence that we have in the selected model complexity of the network—a higher γ would reflect a greater level of confidence that $k = n$. Further work, using

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 10, 2009 at 19:35 from IEEE Xplore. Restrictions apply.

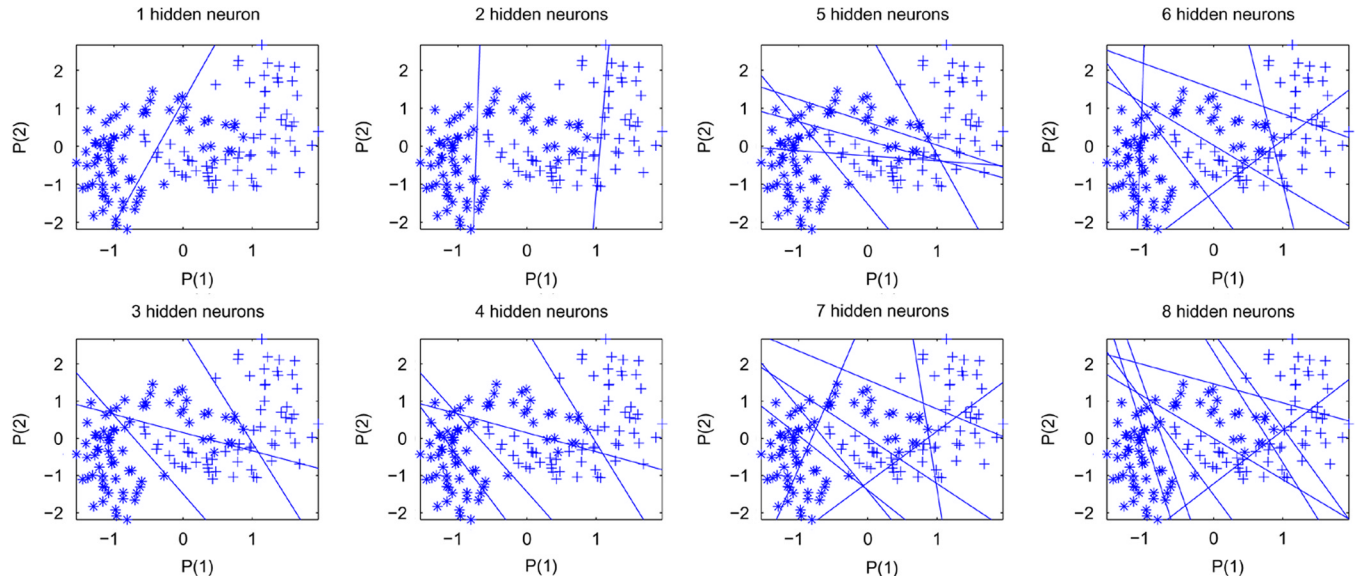


Fig. 1. *Banana*: 1–8 hidden neurons.

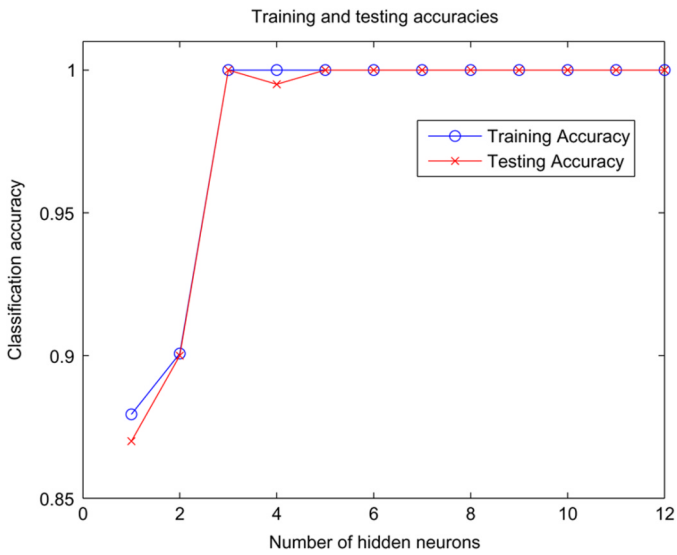


Fig. 2. *Banana*: Train/Test accuracies.

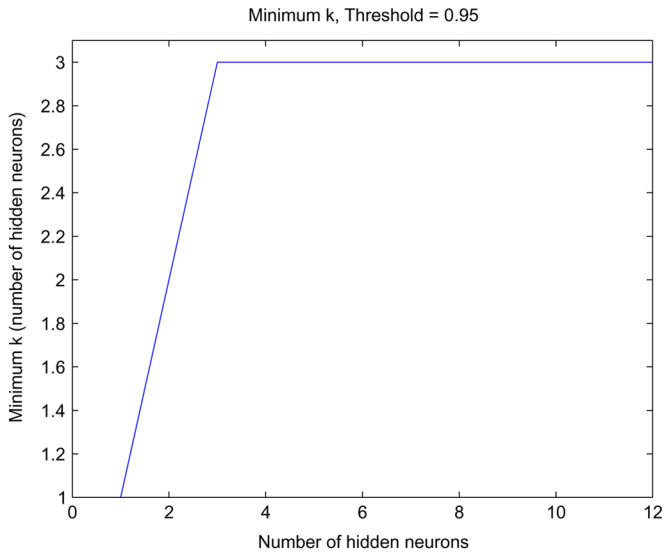


Fig. 3. *Banana*: Criterion 4).

more advanced measures of structural complexity such as the Akaike or Bayesian information criterion is possible.

Typically, we train the feedforward network on the training set using a tuning-based learning algorithm such as the well-known back-propagation with steepest-gradient descent (scaled-conjugate variant) algorithm, using a sufficiently large number of hidden layer neurons (in our simulations, 20 hidden layer neurons were often used as a starting value). A sufficiently large number of hidden neurons is required because too few hidden neurons often give a large first singular value, which, together with a high threshold, provide an incorrect indication of the rank of the output activations of the hidden layer neurons (H). The network is then trained to convergence, until a minimum in the training error is obtained where there is little or no change in the error (we used 5000–10000 epochs).

Pruning as well as growing of the network is applied *after* the learning phase after all observations (training samples) have been presented. The SVD is then applied on the output activations of the hidden layer neurons (H), and the network subsequently pruned after an appropriate threshold (γ) and selection criteria (using any criteria 1–7) is chosen, thus obtaining an estimated rank of H as k. After the rank k is obtained, we retrain the network using the same learning algorithm, but with the number of hidden layer neurons in the network set to k. Our results demonstrate improved generalization performance on the testing set with insignificant change in the training error.

Conversely, in developing a growing method using the SVD, we incorporate additional neurons into the hidden layer only *if the decay of singular values of* H *does not show a distinct gap*. In this case, geometrically, the construction of the hyperplanes are more or less distinct from one another, and perhaps the solution to the problem would benefit from the incorporation of additional neurons into the hidden layer. The notion of a "distinct" gap can be approached from the same perspective as that of the pruning method used previously, where $\epsilon = \sigma_k - \sigma_{k+1}$ is

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 10, 2009 at 19:35 from IEEE Xplore. Restrictions apply.

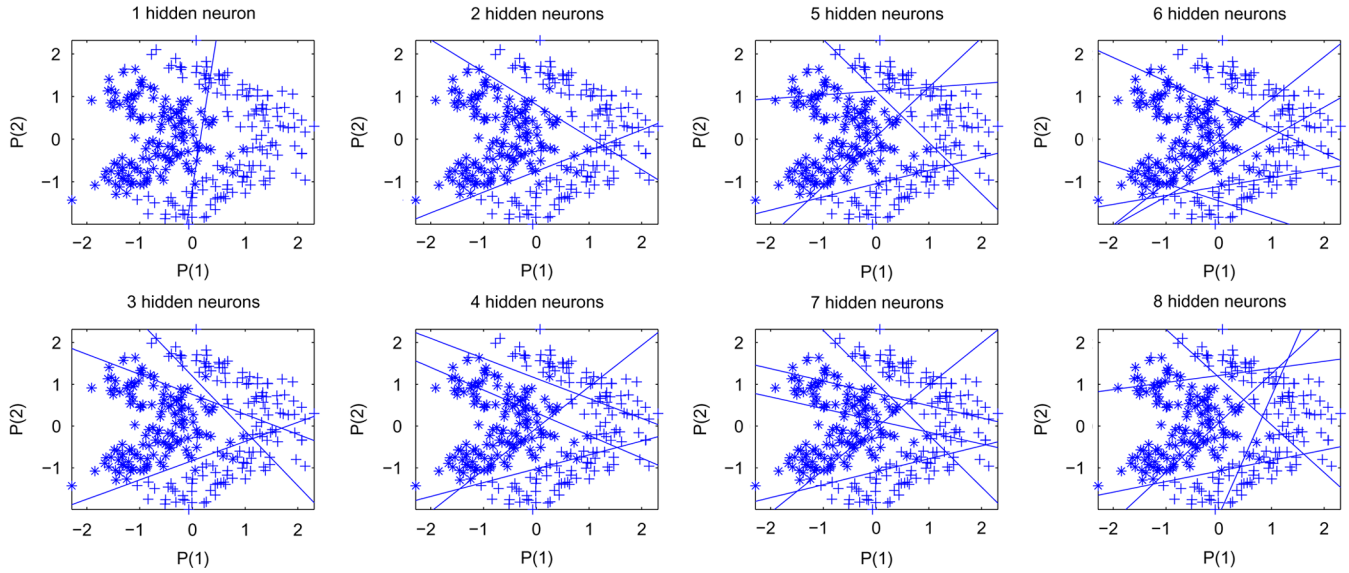


Fig. 4. *Lithuanian*: 1–8 hidden neurons.

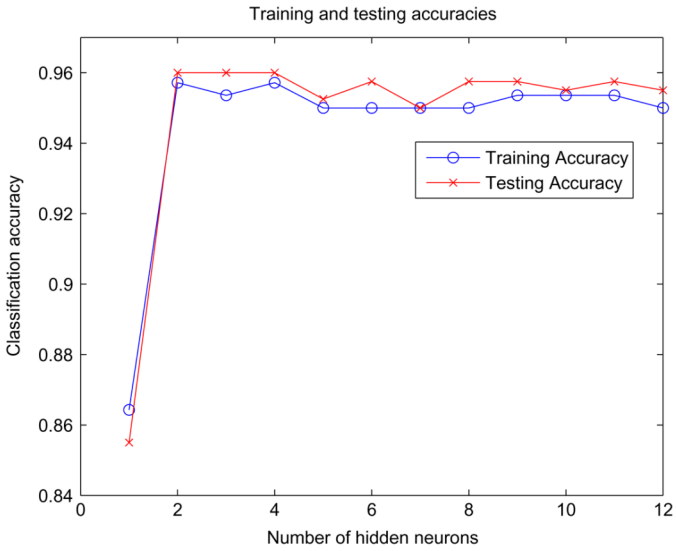


Fig. 5. *Lithuanian*: Train/Test accuracies.

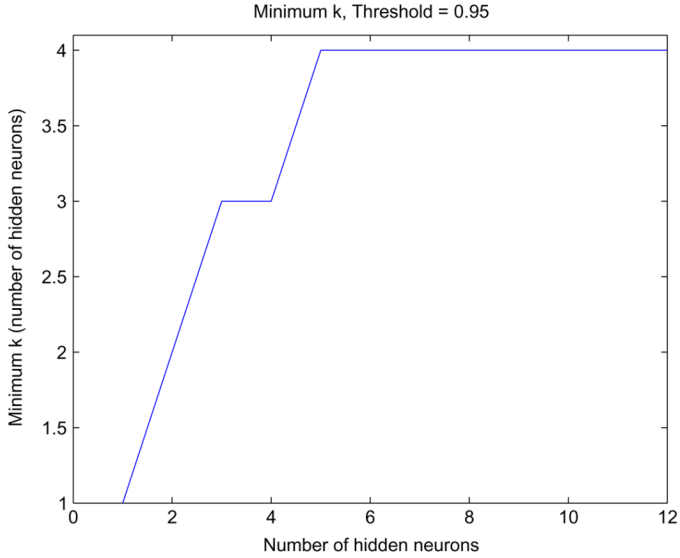


Fig. 6. *Lithuanian*: Criterion 4).

the measure of robustness used. Unlike OBD and OBS, this approach is significantly faster—we base our criteria on the geometric nature of the hyperplanes in hidden layer space, and hence do not make use of the error functional in determining the number of redundant neurons.

VI. Simulation Results and Discussion

We investigate the performance of the pruning/growing method using the singular values and the associated sensitivity values from performing the SVD on H (in a postlearning step), on two types of classification data sets: 1) two-toy data sets, and 2) well-known data sets. The use of the toy data sets are studied as these are of low-dimension (2-D) and are illustrative examples that would thus offer better visual insight of the problem and its proposed solution (particularly in the construction of hyperplanes in hidden layer space), while the real-life data sets provide some justification and indications for use in practical circumstances.[2] The SLFN for each data set was trained for 20 runs for 5000 epochs (to ensure that it is sufficiently well trained). The average training and testing accuracies were then obtained. The back-propagation algorithm was implemented using a batch scaled-conjugate gradient (SCG) variant [23]. For the real-life classification data sets, we use the proposed criteria 6) and 7) from Section V. However, for the toy data sets, we use criterion 4) with a threshold of $\gamma = 0.95$ to demonstrate our ideas. Note that the classification accuracies are measured using a normalized scale $\in [0, 1]$.

A. Toy Data Sets

Banana Data Set: See Figs. 1–3. We consider an artificially created data set (*banana* data set) consisting of two classes and 200 samples of which 140 samples were used as the training set and the remaining for

[2]These data sets were obtained from the University of California at Irvine (UCI) Machine Learning database at http://www.ics.uci.edu/\~mlearn/ML-Repository.html

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 10, 2009 at 19:35 from IEEE Xplore. Restrictions apply.

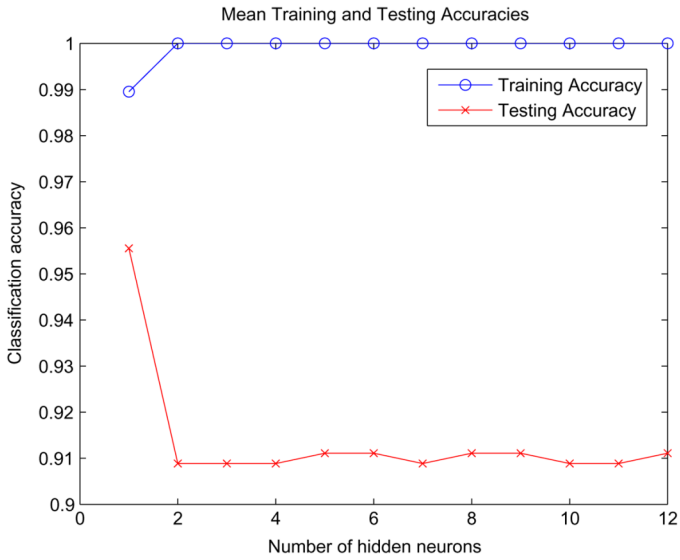


Fig. 7. Iris: Classification accuracies [two neurons, criteriom 7)].

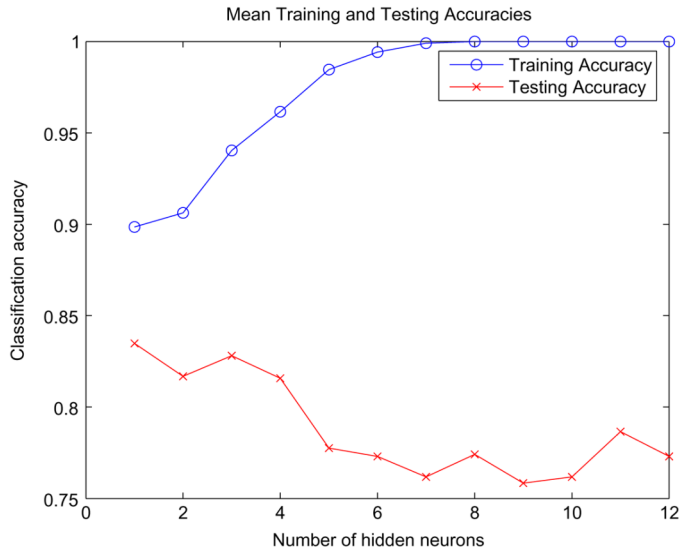


Fig. 8. Heart: Classification accuracies [three neurons, criterion 7)].

the testing set. Geometrically, the use of 2–3 hyperplanes constructed from 2–3 corresponding hidden neurons would be sufficient (see Fig. 1). Expectably, the first three singular values are dominant based on criterion 4), as shown in Fig. 3. Setting the threshold $\gamma = 0.95$, we obtain $k = 3$.

Lithuanian Data Set: See Figs. 4–6. We consider an artificially created data set (*lithuanian* data set) consisting of two classes and 200 samples of which 140 samples were used as the training set and the remaining for the testing set. Geometrically, the use of 3–4 hyperplanes constructed from 3–4 corresponding hidden neurons would be sufficient (see Fig. 4). Expectably, the first four singular values are dominant based on criterion 4), as shown in Fig. 6. Setting the threshold $\gamma = 0.95$, we obtain $k = 4$.

B. Real-Life Classification Data Sets

Figs. 7 and 8 show the mean training and testing accuracies and the suggested number of hidden neurons using criterion 7) of Section V on two data sets - the Iris and Heart data ses from the UCI machine learning repository.

C. Discussion

From the simulation results obtained, we see that increasing the number of hidden layer neurons after a certain threshold of neurons has been reached has a marginal effect on the resulting performance of the classifier, mostly on the training set. Generally, using more hidden neurons than is necessary has a detrimental effect on the classifier's performance on the testing set, as the capacity of this more complex SLFN will exploit the additional free parameters of the network in overfitting the samples in the training set. This can be observed from the increased classification accuracy on the training set at the expense of classification accuracy on the testing set.

For the problems described in Section V, instead of using the singular values or the sensitivity of the singular values in isolation, we use a λ-weighted (or regularized) linear combination of σ_i and ϵ_i. These are criteria 6) and 7) of Section V used in displaying the decay of singular values shown in the right columns of the figures in Section VI. For our simulations, we take $\lambda = 0.5$. A possible approach is to manually determine the actual rank k from a visual observation of the proposed criteria, which is graphically similar to that obtained from these right columns of the aforementioned figures.

It has to be noted that there is no ideal threshold for every problem, and unless prior information is available of the problem, the fine-tuning of the threshold involves a rather qualitative than analytical approach, and is to be expected from system identification problems [20]. Visual inspection of the singular values and their associated sensitivities provide an interesting (though manual) approach in determining the appropriate number of hidden neurons to be used for the given problem. This decay shows a range of hidden neurons that should be used in solving a given problem, such that the hyperplanes constructed from these neurons are as "unique" as possible—redundant *number* of neurons are identified (though not exact knowledge of the identities of the redundant neurons). Although *subset selection* [17] is a possible technique where the k most linearly independent columns of $H \in \Re^{N \times n}$ are picked, such an approach is quite complex, requiring the construction of a permutation matrix, and is hence much slower than retraining the network using the reduced number k of hidden neurons.

We thus argue that exact identification of redundant neurons is not feasible nor useful because the training algorithm constructs the separating hyperplanes (mapping features from input space to hidden layer space) based on the number of free weights available, fitting the distribution of training samples. The training algorithm would, to the best of its ability, construct these hyperplanes such that they are as linearly independent as possible, as a function of the number of hidden neurons. Note that the addition of every new hyperplane alters the positioning of previous hyperplanes as the training algorithm attempts to place this new set of hyperplanes to maximize the linear independency of the set of hyperplanes while improving the class separation. Retraining the network using the same training algorithm would result in better accuracy and is more efficient from a computational perspective as the training algorithm attempts to maximize the linear independency of the reduced set (k instead of n) of hidden layer neurons. This happens until a certain number of hidden layer neurons is used, after which additional hidden neurons introduced cannot be constructed such that they are linearly independent (for an illustrative example, refer to the hyperplanes constructed in the toy data sets—when increasingly larger number of hidden neurons are used, the hyperplanes tend to lie closely parallel to each other), and this manifests itself as the decay in the singular values of the matrix H.

With this said, a technique that can be used is to first train the SLFN on the given set of training data using a reasonably large number of hidden layer neurons. The SVD is then applied on the matrix of hidden layer activations (H), in a postlearning step. Upon visualization of the

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 10, 2009 at 19:35 from IEEE Xplore. Restrictions apply.

decay of singular values using the various criterion specified in Section V, we estimate the appropriate number of hidden layer neurons to be used for that particular problem; subsequently, the training algorithm is reapplied to the same set of training data using the reduced number (k instead of n) of hidden layer neurons.

Although most systems use nodes based on dot products and sigmoids, many other types of units or layers can be used. A common alternative is the radial basis function (RBF) network. In such networks, the SVD, when applied to the hidden layer output activation matrix (H) of a well-trained RBF network would provide an estimate of the number of basis functions to be implemented in the hidden layer. This is conceptually similar to the SLFN case for the MLP as discussed in this letter. In a similar vein, this approach can also be used in the context of estimating the number of hidden neurons (interneurons) in a recurrent neural network setting.

VII. Conclusion

This letter has presented an empirical approach in estimating the necessary number of hidden layer neurons for an SLFN given a set of training data, through the use of the singular values via the SVD in a postlearning step. These singular values provide a quantification of the degree of degeneracy, or equivalently, the level of linear independency of the training patterns in hidden layer space. The weights learned during the training stage of the SLFN project the features from input space onto a hidden layer space. However, while increasing the number of hidden layer neurons increases the capacity and thus the complexity of the system on the *training set*, the generalizability of the resulting SLFN is compromised as overfitting occurs. We believe the use of the SVD can be extended to the estimation of hidden neurons in feedforward neural networks with multiple hidden layers, as well as in recurrent neural networks. Another possible direction for future work could be a study into the use of the proposed approach on networks that have been trained using constructive learning algorithms that are nontuning based, such as the extreme learning machine (ELM) paradigm [24].

The main contribution of this letter is to present a simple framework for the use of the singular values as well as their corresponding sensitivities of the hidden layer output activation matrix H, in providing an indication of the necessary or appropriate number of hidden layer neurons (as well as a robustness measure of this estimate) that should be used in an SLFN for a given training set. While the current approach involves a rather qualitative and manual approach in determining the thresholds and number of hidden layer neurons, the use of the rank-revealing SVD allows us to gain a better practical and empirical insight into the geometry of hidden layer space, which would otherwise be difficult, if not outright impossible given that many of today's practical problems involve features that are of high-dimension and models that are of high-complexity.

Acknowledgment

The authors would like to thank the reviewers for their valuable suggestions and insightful comments in improving the overall content and clarity of the original letter.

References

[1] G. Huang and H. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE Trans. Neural Netw.*, vol. 9, no. 1, pp. 224–229, Jan. 1998.

[2] G. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 274–281, Mar. 2003.

[3] M. Anthony, "Probabilistic analysis of learning in artificial neural networks: The PAC model and its variants" London, U.K., Tech. Rep. NC-TR-94-3, 1994.

[4] S. Huang and Y. Huang, "Bounds on number of hidden neurons of multilayer perceptrons in classification and recognition," in *IEEE Int. Symp. Circuits Syst.*, 1990, vol. 4, pp. 2500–2503.

[5] M. Sartori and P. Antsaklis, "A simple method to derive bounds on the size and to train multi-layer neural networks," *IEEE Trans. Neural Netw.*, vol. 2, no. 4, pp. 467–471, Jul. 1991.

[6] S. Tamura, "Capabilities of a three-layer feedforward neural network," in *Proc. Int. Joint Conf. Neural Netw.*, 1991, pp. 2757–2762.

[7] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: Four layers versus three," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 251–255, Mar. 1997.

[8] T. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 326–34, 1965.

[9] M. Hayashi, "A fast algorithm for the hidden units in a multilayer perceptron," in *Proc. Int. Joint Conf. Neural Netw.*, 1993, vol. 1, pp. 339–342.

[10] S. Tamura, M. Tateishi, M. Matsumoto, and S. Akita, "Determination of the number of redundant hidden units in a three-layered feedforward neural network," in *Proc. Int. Joint Conf. Neural Netw.*, 1993, vol. 1, pp. 335–338.

[11] D. Psichogios and L. Ungar, "SVD-NET: An algorithm that automatically selects network structure," *IEEE Trans. Neural Netw.*, vol. 5, no. 3, pp. 513–515, May 1994.

[12] A. Weigend and D. Rumelhart, "The effective dimension of the space of hidden units," in *IEEE Int. Joint Conf. Neural Netw.*, 1991, pp. 2069–2074.

[13] C. Xiang, S. Ding, and T. Lee, "Geometrical interpretation and architecture selection of MLP," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 84–96, Jan. 2005.

[14] Y. Lecun, J. Denker, and S. Solla, "Optimal brain damage," *Adv. Neural Inform. Process. Syst.*, vol. 2, pp. 598–605, 1990.

[15] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. IEEE Int. Conf. Neural Netw.*, San Francisco, CA, 1992, vol. 1, pp. 293–299.

[16] G. Stewart, Determining Rank in the Presence of Error (TR-2972) Dept. Comp. Sci., Univ. Maryland, College Park, MD, Tech. Rep. (TR-92-108) Inst. Advanced Computer Studies, Oct. 1992.

[17] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: John Hopkins Univ. Press, 1999.

[18] P. Hansen, *Rank-Deficient and Discrete ILL-Posed Problems: Numerical Aspects of Linear Inversion*. Philadelphia, PA: SIAM, 1998.

[19] G. Golub, V. Klema, and G. Stewart, Rank Degeneracy and the Least Squares Problem Comp. Sci. Dept., School of Humanities and Sciences, Stanford Univ., Stanford, CA, Tech. Rep. (STAN-CS-76-559), 1976.

[20] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[21] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Trans. Autom. Control*, vol. AC-25, no. 2, pp. 164–176, Apr. 1980.

[22] K. Konstantinides and K. Yao, "Statistical analysis of effective singular values in matrix rank determination," *IEEE Trans. Acoust., Speech Signal Process.*, vol. 36, no. 5, pp. 757–763, May 1988.

[23] M. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Netw.*, vol. 6, pp. 525–533, 1993.

[24] G. Huang, Q. Zhu, and C. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2004, vol. 2, pp. 985–990.

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 10, 2009 at 19:35 from IEEE Xplore. Restrictions apply.

Overfitting problem: a new perspective from the geometrical interpretation of MLP

S.Q. DING and C. XIANG
Department of Electrical and Computer Engineering
National University of Singapore
Singapore, 119260

Abstract. A geometrical interpretation of the multilayer perceptron (MLP) is suggested in this paper. Basically, the hidden neurons are considered as the building-blocks for constructing the function with the corresponding weights and biases determining their geometrical shapes and positions. A guideline for architecture selection of MLP is then proposed based upon this interpretation and various prevalent approaches of dealing with the over-fitting problem are also reviewed from this new geometrical interpretation. In particular, the popular regularization methods are studied in detail. Not only the reason why regularization methods are effective to alleviate the over-fitting can be simply explained by the geometrical interpretation, but also a potential problem with regularization is predicted and verified.

1. Introduction

Multilayer perceptron (MLP) has already proven to be very effective in a wide spectrum of applications, in particular the function approximation and pattern recognition problems. Like other nonlinear estimation methods MLP also suffers from over-fitting. The best way to solve the over-fitting problem is to provide a sufficiently large pool of training data. But in most of the practical problems, the number of training data is limited and hence other methods such as model selection, early stopping, weight decay, and Bayesian regularization etc. are more feasible when a fixed amount of training data is given.

Model selection mainly focuses on the size of the neural network, i.e. the number of weights, while most other approaches are related to the size of the weights, directly or indirectly. They are actually the two aspects of the complexity of the networks. Therefore it is of great interest to gain deeper insight into the functioning of the size of the network and the size of weights in the context of the over-fitting problem.

In the section that follows, the geometrical interpretation of the MLP will be presented. This interpretation will be first suggested for the case when the activation function of the hidden neurons is piecewise-linear function and then extended naturally to the case of sigmoid activation functions.

Based on this geometrical interpretation, how the number and the size of the weights influence the over-fitting problem will then be clearly elucidated in section three. A guideline for architecture selection of MLP is also proposed and various prevalent approaches of dealing with the over-fitting problem are examined from the point of view of the new geometrical interpretation. In particular, the popular regularization training algorithms are studied in details. Not only the reason why regularization methods are very efficient to overcome the over-fitting can be simply explained by the geometrical interpretation, but also a potential problem with regularization is predicted and demonstrated.

2. Geometrical Interpretation of MLP

Consider a three-layered MLP 1-N-1, with one input, N hidden neurons and one output. The activation function for the hidden neuron is the piecewise-linear function described by

$$\varphi(v) = \begin{cases} 1, & v \geq 0.5 \\ v + 0.5, & -0.5 < v < 0.5 \\ 0, & v \leq -0.5 \end{cases} \tag{1}$$

Let the weights connecting the input neuron to the hidden neurons be denoted as $w_i^{(1)}$ (i=1,…,N), the weights connecting the hidden neurons to the output neuron be $w_i^{(2)}$, the biases for the hidden neurons be $b_i^{(1)}$, and the bias for the output neuron be $b^{(2)}$. The activation function in the output neuron is the identity function such that the output y of the MLP with the input x feeding into the network is

$$y(x) = \sum_{i=1}^{N} w_i^{(2)} \varphi(w_i^{(1)} x + b_i^{(1)}) + b^{(2)} \tag{2}$$

It is evident that y(x) is just superposition of N piecewise-linear functions plus the bias. From (1) we know that each piecewise-linear function in (2) is described by

$$w_i^{(2)} \varphi(w_i^{(1)} x + b_i^{(1)}) = \begin{cases} w_i^{(2)}, & w_i^{(1)} x + b_i^{(1)} \geq 0.5 \\ w_i^{(2)}(w_i^{(1)} x + b_i^{(1)} + 0.5), & -0.5 < w_i^{(1)} x + b_i^{(1)} < 0.5 \\ 0, & w_i^{(1)} x + b_i^{(1)} \leq -0.5 \end{cases} \tag{3}$$

In the case of $w_i^{(1)} > 0$, we have

$$w_i^{(2)} \varphi(w_i^{(1)} x + b_i^{(1)}) = \begin{cases} w_i^{(2)}, & x \geq \dfrac{0.5}{w_i^{(1)}} - \dfrac{b_i^{(1)}}{w_i^{(1)}} \\ w_i^{(2)}(w_i^{(1)} x + b_i^{(1)} + 0.5), & \dfrac{-0.5}{w_i^{(1)}} - \dfrac{b_i^{(1)}}{w_i^{(1)}} < x < \dfrac{0.5}{w_i^{(1)}} - \dfrac{b_i^{(1)}}{w_i^{(1)}} \\ 0, & x \leq \dfrac{-0.5}{w_i^{(1)}} - \dfrac{b_i^{(1)}}{w_i^{(1)}} \end{cases} \tag{4}$$

The graph for this weighted piecewise linear function is plotted in Figure 1.

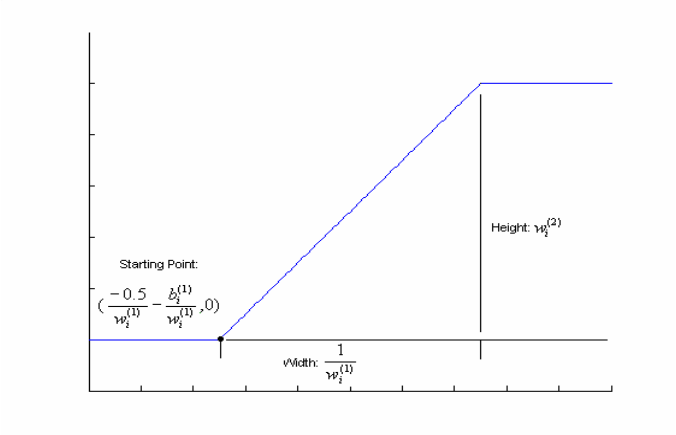


Figure 1: Piecewise linear function

This piece-wise linear function has the same geometrical shape as that of (1), consisting of two pieces of flat lines at the two ends and one piece of line segment in the middle. Any finite piece of line segment can be completely specified by its width (span in the horizontal axis), height (span in the vertical axis), and position (starting point, center, or ending point). And it is obvious from equation (4) and Figure 1 that the width of the middle line segment is $\frac{1}{w_i^{(1)}}$, the height is $w_i^{(2)}$, the slope is $w_i^{(1)} w_i^{(2)}$ and the starting point is $(\frac{-0.5}{w_i^{(1)}} - \frac{b_i^{(1)}}{w_i^{(1)}}, 0)$. Once this middle line segment is specified the whole piecewise line is then completely determined. From above discussion it is natural to suggest following geometrical interpretation for the three-layered MLP with piecewise-linear activation functions.

(1) The number of hidden neurons corresponds to the number of piecewise lines that are available for approximating the target function. These piecewise lines act as the basic building-blocks for constructing functions. The slope and position of each piecewise line are completely determined by the weights and biases.

(2) The weights connecting the input neuron to the hidden neurons completely determine the widths of the middle line segments of those basic building-blocks. By adjusting these weights, the widths of the basic elements can be changed to arbitrary values.

(3) The weights connecting the hidden neurons to the output neuron completely decide the heights of the middle line segments of those basic building-blocks. The heights can be modified to any values by adjusting these weights.

(4) The biases in the hidden neuron govern the positions of the middle line segments of those basic building-blocks. By adjusting the value of these biases, the positions of the building-blocks can be located arbitrarily.

(5) The bias in the output neuron provides an offset term to the final value of the function.

Using the fact that the widths, the heights, and the positions of the middle line segments of the basic building-blocks can be adjusted arbitrarily, we are ready to state Theorem 1 as follows.

Theorem 1: Let f(x) be any piecewise linear function defined in any finite domain, $-\infty < a \le x \le b < \infty$, there exists at least one three-layered MLP, denoted as NN(x), with piecewise linear activation functions for the hidden neurons that can represent f(x) exactly, i.e., NN(x)=f(x) for all $x \in [a, b]$.

The proof of theorem 1 is quite straightforward by directly constructing one MLP that can achieve the objective. Due to space limitation, the proof is not included, and the reader is referred to [1] for further details.

Since any bounded continuous function can be approximated arbitrarily closely by piecewise linear function, Theorem 1 simply implies that any bounded continuous function can be approximated arbitrarily closely by MLP, which is the well-known universal approximation property of the MLP.

The geometrical shape of the sigmoid activation function is very similar to the piecewise-linear activation function, except the neighborhood of the two end points are all smoothened out as shown in Figure 2. Therefore the previous geometrical interpretation of the MLP can be applied very closely to the case when sigmoid activation functions are used. Further, since the sigmoid function smoothen out the non-smooth end points, the MLP with sigmoid activation functions is more efficient to approximate smooth functions.

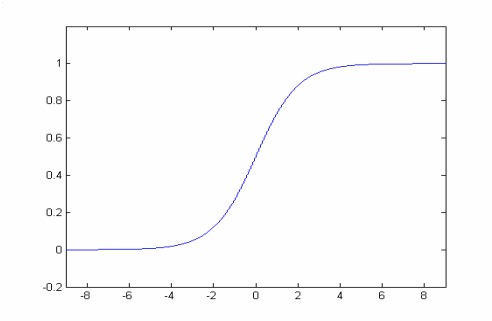


Figure 2: Sigmoid activation function

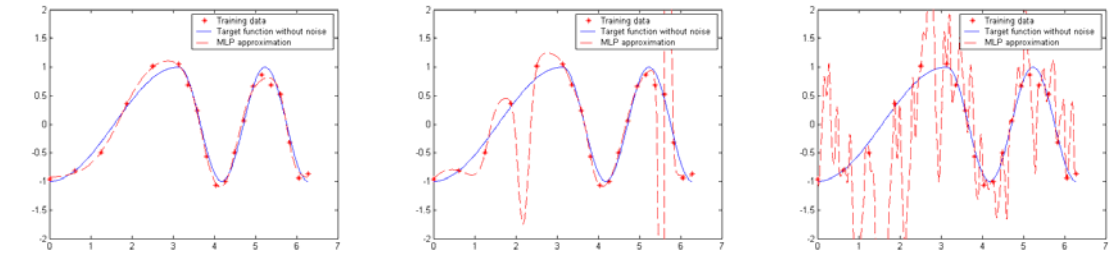
3. Over-fitting problem: overview and comparative study of available methods:

Applying the geometrical interpretation, a brief overview of over-fitting and some popular approaches to improve generalization will be discussed in this section.

An example of over-fitting problem [2] is illustrated in Figure 3, which is a function approximation with a three-layered (one hidden layered) MLP. The training dataset is created by

$$y = \begin{cases} -\cos(x) + v & 0 \le x < \pi \\ \cos(3(x-\pi)) + v & \pi \le x \le 2\pi \end{cases} \tag{5}$$

And the noise v is uniformly distributed within [–0.25,0.25]. The MLP is trained with Levenberg-Marquardt algorithm using Neural Network toolbox of MATLAB. With 4 hidden neurons, the approximation is fairly good. When the number of hidden neurons increases, significant over-fitting and poor generalization are observed. The output of the MLP fits the training data perfectly when number of the hidden neurons reaches 100, but the interpolation between the training points is extremely poor.



1-4-1 1-10-1 1-100-1

Figure 3: Example of over-fitting problem

From above example, it is obvious that the degree of over-fitting increases with the size of the neural network. However, Bartlett made a surprising observation that for valid generalization, the size of the weights is more important that the size of the network [3], which appears hardly to be true at first glance. But with the aid of the geometrical interpretation, this astonishing observation can be plainly explained as follows. Since the slope of each building-block is roughly proportional to $w_i^{(1)} w_i^{(2)}$, the smaller the weights, the gentler the slope of each building-block and hence the smoother the shape of the overall

function.

In fact, most of the prevalent methods to prevent over-fitting are concerned either with the size of the network or the size of the weights, which will be examined from this new perspective of the geometrical interpretation.

3.1 Model selection

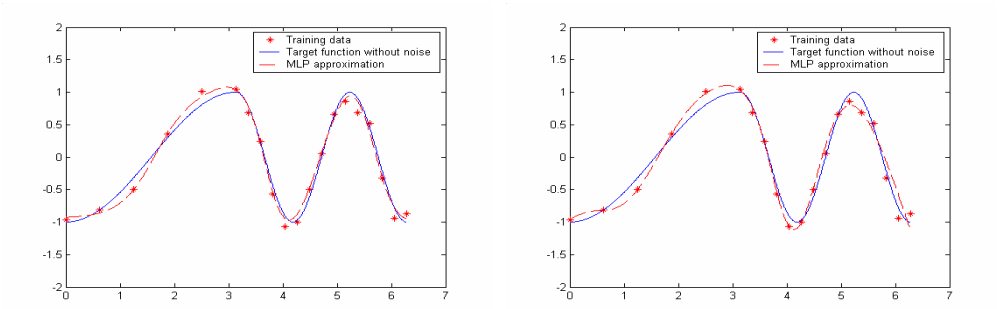
This approach focuses on the size of the network. Generally, a simple network will give good generalization performance. Normally, the model selection procedure is based on cross-validation to choose the optimal size using either pruning or growing techniques, which is usually time-consuming. Instead, based upon the geometric interpretation, a much simpler guideline is proposed as follows.

Guideline for Model Selection: Estimate the minimal number of line segments (or hyper-planes in high dimensional cases) that can construct the basic geometrical shape of the target function which is perceived from the training data, and use this number as the first trial for the number of hidden neurons of the three-layered MLP.

Following this guideline, obviously 4 hidden neurons are needed to approximate the function given in the example, and indeed 1-4-1 network gives very good generalization as seen from Figure 3. We have tested this guideline with extensive simulation studies. In all of the cases studied, the estimated number of the hidden neurons is either very close to the minimal number of hidden neurons needed for satisfactory performance, or is the minimal number itself in many cases. The reader is referred to [1] for more details.

3.2 Early stopping

Early stopping is another popular method to overcome the over-fitting problem in the training progress [4]. The main idea is to stop training when the validation error goes up. Figure 4 shows the results of using early stopping method in the former example, in which no significant over-fitting is observed even when the number of hidden units reaches 100. To apply early stopping successfully, it is critical to choose very small random initial values for the weights (chosen within [-0.1, 0.1] randomly in this example) and use a slow learning rate, which essentially prevents the weights from evolving into large values. And confining the size of the weights to be small is a good remedy to alleviate the over-fitting problem as discussed before.



1-4-1 1-100 –1

Figure 4: Early stopping for overcoming over-fitting problem

3.3 Regularization methods

Conventionally, the training progress is to minimize the cost function $F = E_D$, where E_D

is the summation of the squared errors. Regularization methods add a penalty term to the cost function. Usually, the penalty term is a function of the weights, which is called complexity penalty. Then the cost function becomes $F = E_D + \lambda E_w$, where E_w is the complexity penalty and λ is called regularization parameter.

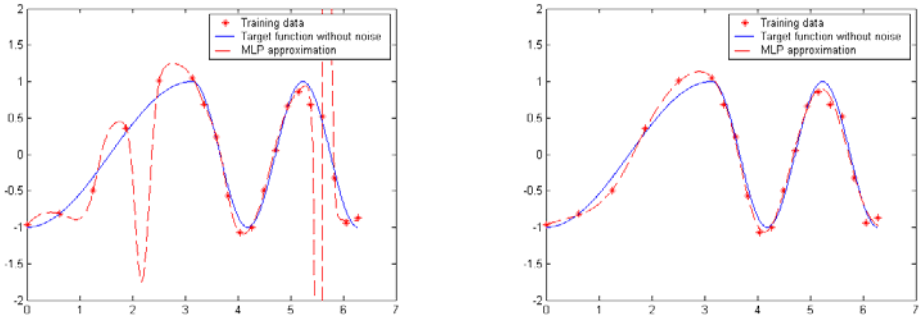
Weight decay [5] is the simplest one of the regularization approaches, where E_w is the summation of all the squared parameters including both weights and biases, and weight elimination [6] is actually a normalized version of weight decay. Both of them work effectively in some applications, but they do not work well all the times because they ignore the difference between the weights and the biases, as well as the interaction between the weights in different layers. For instance, from the geometrical interpretation, the biases are only related to the positions of the basic building-blocks, not the shapes, and hence should not be included in the penalty term.

A more recent regularization method proposed by Moody and Rögnvaldsson [7] work much better than the standard weight decay and weight elimination. In their approach, for the case of one-dimensional map, the complexity penalty E_w for the first order local smoothing regularizer can be reduced to

$$E_w = \sum_{i=1}^{N} (w_i^{(2)} w_i^{(1)})^2 \tag{6}$$

which is actually minimizing the slopes of the basic building-blocks from the point of view of geometrical interpretation. Therefore its superior performance can be simply attributed to its capability to distinguish the different roles played by the weights and the biases.

The choice of the regularization parameter λ also affects the performance of the generalization significantly. MacKay's Bayesian approach [8,9] to choose the regularization parameters is the most popular one. Using Bayesian regularization, 1-10-1 MLP may achieve good generalization result for the former example while it fails previously without regularization as shown in Figure 5. It is worth noting that the Bayesian regularization may break down if the number of data pairs N is small relative to the number of the free parameters k as pointed out by MacKay. But the reason and how large N/k must be for reliable approximation is still an open question [9]. Furthermore, this breaking down may also depend upon the initialization of the parameters as observed from our simulation studies.



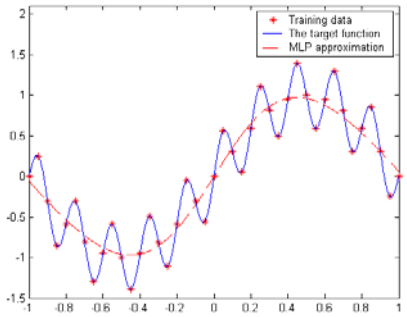
1-10-1 without Bayesian regularization 1-10-1 with Bayesian regularization

Figure 5: Bayesian regularization for overcoming over-fitting problem

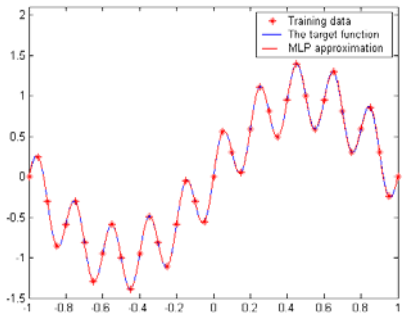
The regularization methods limit the size of the weights, which in turn restrict the slopes of the building-blocks to be small and hence results in smooth approximation. However, the strength of this approach is also its weakness. Based upon the geometrical interpretation, the MLP may have difficulty in approximating functions with significant high frequency components because the slopes of the building blocks are confined to be small. To verify this prediction, a simulation example is constructed as follows. A training dataset is created which contains 41 points according to the function:

$$y = \sin(\pi x) + 0.2\sin(10\pi x) \tag{7}$$

A MLP with 21 hidden neurons (which follows the previous model selection guideline) is used to approximate this function, and the initial weights are chosen randomly within [-1,1]. The results with and without Bayesian regularization are shown in Figure 6, where unexpected smooth solution can be seen when Bayesian regularization is used. Very interestingly, Bayesian regularization indeed acts as a low-pass filter, and fails to capture the high frequency component. Fortunately most of the high frequency signals result from noises in reality, and the Bayesian regularization may give the desired approximation by effectively filtering out the noise. But if the high frequency signals are useful signals instead of noise, then regularization approach may not be the right choice, and model selection method may be more proper.



1-21-1 with Bayesian regularization



1-21-1 without Bayesian regularization

Figure 6: A simple example where Bayesian regularization fails

4. Conclusion

Over-fitting is a critical issue for neural network applications. In order to gain deeper insights in the functioning of the size of the network, as well as the size of the weights, a geometrical interpretation of the MLPs is first suggested in this paper on the basis of the geometrical shape of the activation function. Based upon this interpretation, the size of the weights directly decides the shape of the basic building-blocks, the smaller the weights, the smoother the building-blocks. And the reason behind Bartlett's well-known observation that "for valid generalization, the size of the weights is more important than the size of the network" is now crystal clear from the viewpoint of this geometrical interpretation.

Various methods of preventing over-fitting are reviewed from this new perspective, and all of them can be elegantly explained by the suggested geometrical interpretation. A simple guideline for model selection is also suggested and applied successfully to the given example.

Regularization has emerged as the most popular approach to overcome over-fitting since no specific techniques are needed to select an optimal architecture and the available data can be fully used. However, a potential problem with the regularization method that it may fails to capture the high frequency characteristics of the function, is illuminated by the geometrical interpretation.

Acknowledgment

The research reported here was supported by NUS Academic Research Fund R-263-000-224-112.

References

[1] C. Xiang, S.Q. Ding and T.H. Lee. “Geometrical interpretation and architecture selection of MLP”. *IEEE Transactions on Neural Networks.* Submitted, 2003.

[2] R. Caruana, S. Lawrence and C.L. Giles. “Overfitting in neural networks: backpropagation, conjugate gradient, and early Stopping”. *Advance in Neural Information Processing Systems*, vol. 13, pp. 402-408, 2001.

[3] P. L. Bartlett. “For valid generalization, the size of the weights is more important than the size of the network”. *Advances in Neural Information Processing Systems*, vol. 9, pp. 134-140, 1997.

[4] W.S. Sarle. “Stopped training and other remedies for overfitting”. *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, pp. 352-360, 1995.

[5] D. Plaut, S. Nowlan, and G. Hinton. “Experiments on learning by backpropagation”. *Technical Report CMU-CS-86-126*, Carnegie Mellon University, Pittsburg, PA,1986.

[6] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman, “Generalization by weight-elimination with application to forecasting”. *Advances in Neural Information Processing Systems*, vol. 3, pp. 875-882, 1991.

[7] J.E. Moody and T. Rögnvaldsson, “Smoothing regularizers for projective basis function networks”. *Advances in Neural Information Processing Systems*, vol. 9, pp. 585-591,1997.

[8] D.J.C. MacKay. “Bayesian interpolation”. *Neural Computation*. Vol. 4, pp. 415-447, 1992

[9] D.J.C. MacKay. “A practical Bayesian framework for back-propagation networks”. *Neural Computation*. Vol. 4, pp. 448-472, 1992.