STRUCTURE ANALYSIS OF NEURAL NETWORKS

DING SHENQIANG

NATIONAL UNIVERSITY OF SINGAPORE

2004

STRUCTURE ANALYSIS OF NEURAL NETWORKS **DING SHENQIANG** **2004**

STRUCTURE ANANLYSIS OF NEURAL NETWORKS

DING SHENQIANG
(B. Eng, University Of Science and Technology of China)

A THESIS SUBMITTED
FOR THE DEGREE OF MASTER OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

2004

Acknowledgements

I would like to express my most sincere appreciation to my supervisor, Dr. Xiang Cheng, for his good guidance, support and encouragement. His stimulating advice benefits me in overcoming obstacles on my research path.

I am also grateful to the Center for Intelligent Control (CIC), as well as the Control and Simulation Lab, Department of Electrical and Computer Engineering, National University of Singapore, which provides the research facilities to conduct the research work. I also wish to acknowledge National University of Singapore (NUS) for the financial support provided throughout my research work.

Thanks to many of my friends in Control and Simulation Lab, who have made contributions in various ways to my research and life here in the past two years.

Finally, special thanks to my wife Sun Yu, for her love and patience.

Table of Contents

Summary

This work seeks to conduct structure analysis of artificial neural networks, especially feedforward neural networks such as multilayer perceptrons (MLP) and radial basis function networks (RBFN).

First of all, a brief introduction of artificial neural networks is given; the background and the necessity of the structure analysis problem are also stated. Then a geometrical interpretation of multilayer perceptron based on the geometrical meaning of the weights of a single hidden neuron is 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 is proposed based upon the geometrical interpretation. The effectiveness of this guideline is illustrated by a couple of simulation examples. Subsequently, the attention is shifted to the controversial issue of whether four-layered (with two hidden layers) MLP is superior to the three-layer MLP. With the aid of the geometrical interpretation and also through careful examination of the various contradictory results reported in the literature, it is 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, and for a certain class of problems the four-layered MLP outperforms three-layered MLP significantly.

After that, the overfitting problem of MLP is examined, a comparative study is carried out on various alleviating methods and the reasons behind these methods are reviewed based on the 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.

Afterward, another popular feedforward neural network, radial basis function network, is visited. A special additional input, which is the sum of the squares of the other inputs, is added to the standard multilayer perceptron, so that the multilayer perceptron works similarly to the radial basis function network with localized response. Specially, we will show a three-layered multilayer perceptron with exponential activation function and this kind of additional input is naturally a generalized radial basis function network and multilayer perceptron can be trained using the well-developed training strategies of multilayer perceptrons. Then a comparative study is conducted between multilayer perceptrons, multilayer perceptrons with additional inputs and radial basis function networks trained by various methods.

Finally, a conclusion of the whole thesis is presented and the direction of future research is also pointed.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Artificial Neural Networks

Artificial neural networks (usually shorten as "neural networks") are originally motivated from the biological neural networks such as the brain and human nervous system. The first artificial neural network is called *perceptron*, which is developed by Rosenblatt (1959) from the biological neuron model by McCulloch and Pitts (1943). Despite originating from the biological system, artificial neural networks are widely used as problem-solving algorithms rather than in developing them as accurate representations of the human nervous system (Ripley 1994). However, the artificial neural networks still emulate biological neural networks in following main aspects:

1. Each basic unit of the artificial neural networks is a simplified version of the biological neuron.
2. Each basic unit is connected to a massive network in parallel.
3. Each basic unit has an activation function.
4. Learning of the network is done by adjust the connections (weights) between the basic units.

There is still no formal definition of artificial neural networks, one recent definition was given by Haykin (1999): A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights are used to store the acquired knowledge.

Fig 1.1 gives a mathematic model of a simplest artificial neural network with only one basic unit. Three essential elements are noted: the connection weights, summation operator and the activation function. Another term bias adjusts the value of the summation. We may describe the model with equation (1.1), where $x_1, x_2, \cdots, x_m$ are the input signals; $w_1, w_2, \cdots, w_m$ are the connection weights; b is the bias; and $\varphi(\cdot)$ is the activation function. When outside signals are feed to the neural network; the inputs first go through the connection weights which lead to weighted inputs, then the summation operator effects, and finally the weighted summation of the inputs and bias are sent to the activation function to give the final output.

$$y_{out} = \varphi(\sum_{i=1}^{m} w_i x_i + b) \tag{1.1}$$

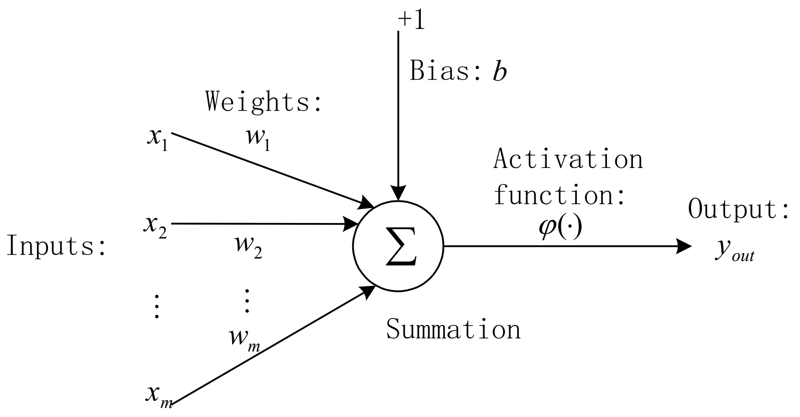


Fig. 1.1. A nonlinear model of a single neuron ANN

Normally, the artificial neural network contains many of this kind of basic units distributed in different layers. A more general structure of artificial neural network is provided in Fig. 1.2.

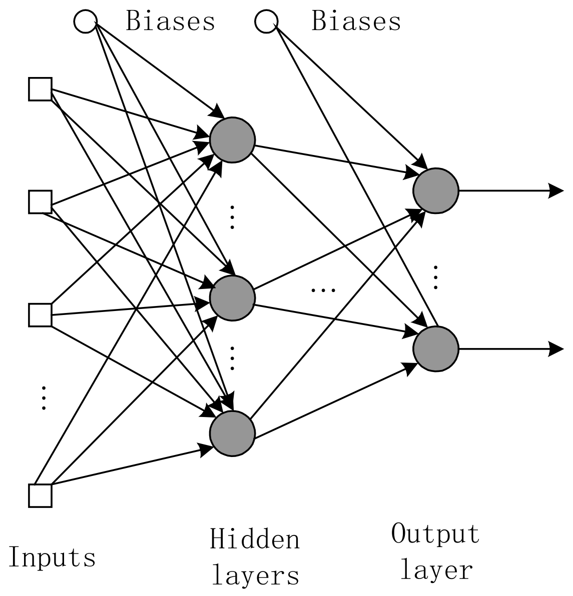


Fig. 1.2. General structure of a feedforward ANN

Please note that some neural networks do have reverse (feedback) signal flow like recurrent neural networks. In this thesis, feedforward neural networks such as multilayer perceptron and radial basis function neural networks are studied.

1.2 Statement of the Structure Analysis Problem of Neural Networks

Although neural networks are used widely and successfully in many application areas, how to select the structure of specified neural networks is still a very essential problem. For example, if the multilayer perceptron network is chosen, then the practitioner still faces many problems to decide the structure of multilayer perceptron to be used. Such as given the following function approximation example as shown in Figure 1.3, how many hidden layers to use and how many neurons to choose for each hidden layer?

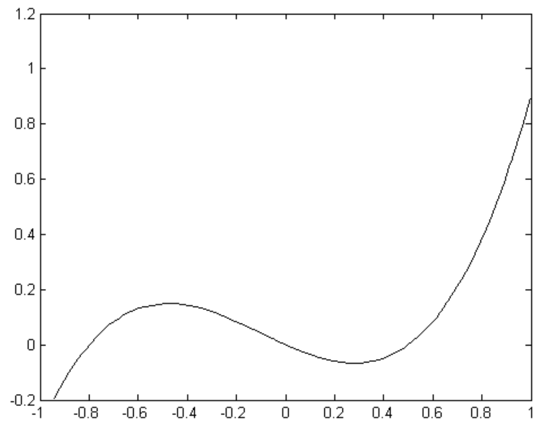


Fig. 1.3. A simple one-dimensional function approximation problem

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 just regarding the multilayer perceptron 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 work seem more of an art than a science. Without a deep understanding of the design parameters, some people still feel uneasy to use the multilayer perceptron 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.

Traditionally, the main focus regarding the architecture selection of MLP has been centered upon the growing and pruning techniques. (Mozer and Smolensky 1989; Karnin 1990; LeCun et al. 1990; Weigend et al. 1991; Hassibi et al. 1992; Reed 1993; Hush 1997). In network growing techniques, we often start with a small network to solve the problem at hand and add additional neurons or layers only if the current network is unable to meet the criterion. For network pruning, which is to choose a network larger than necessary at first, and then remove the redundant part. More efforts were put on the pruning techniques in the literature; the pruning techniques mainly include sensitivity calculation methods and regularization methods (Reed 1993). The sensitivity calculation methods usually estimate the “sensitivity” of each neuron or connection and delete those with less sensitivity or less importance (Mozer and Smolensky 1989; Karnin 1990; Reed 1993). The regularization methods incorporate an additional term in the standard error cost function. This additional penalty term is a complexity penalty, which is usually a function of the weights (Plaut et al. 1986; Chauvin 1989; Ji et al. 1990; Weigend et al. 1991; Nowlan et al. 1992; Moody and Rögnvaldsson 1997). One attractive advantage of the regularization methods is that the training and pruning are done simultaneously which will lead to a more optimal solution. However for the pruning algorithms, when to stop the pruning procedure or how to choose the regularization parameter is still a problem.

Recently, lots of attention has also been drawn on applying evolutionary algorithms to evolve both the parameters and architectures of the artificial neural networks (Alpaydim 1994; Jasic and Poh 1995; Sarkar and Yegnanarayana 1997; Castillo 2000). Such kind of hybrid algorithms is commonly referred to in the literature as evolutionary artificial neural networks (EANN); for a detailed survey see (Yao 1999). 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 multilayer perceptron. Thus how to simplify the task of architecture selection using this geometrical information is a very interesting and challenging problem.

The overfitting problem of neural networks is also essential, because in most cases what we focused is how the neural networks act with the unseen inputs, which is called generalization performance. Normally, we take for granted it is the size of the neural networks that dominate the generalization performance. However, Bartlett (1997) stated that the size of the weights is more important that the size of the network for generalization performance. So that a deep insight on how the structure of neural networks influences the generalization performance is desirable.

Radial basis function network is another very popular feedforward neural network. There is normally only one hidden layer in the structure, so choosing the number of hidden layers is not a problem for radial basis function network. But it still faces the problem of deciding the number of hidden neurons. Moreover, the radial basis function network has another problem of deciding the locations and the spreads of the basis functions. There are various methods to determine the locations and spreads of the basis functions, which are usually separated from the calculation of the output weights. One stage supervised training algorithms to decide all the parameters simultaneously are also available. However, the supervised training of radial basis function networks is immature comparing the well developed training algorithms for multilayer perceptrons. Thus, a comparative study of these available methods is also attractive.

The local responses of multilayer perceptrons with a certain class of additional inputs or normalized inputs are reported in the literatures (Casasent 1992; Maruyama et al. 1992; Sarajedini and Hecht-Nielsen 1992; Ridella et al. 1997). The connection between the different structured multilayer perceptrons and radial basis function networks is also a very interesting problem. A multilayer perceptron with additional second order inputs, which is the sum of the square of other inputs, can approximate a radial basis function arbitrarily. At the same time, another question rises: can such a multilayer perceptron represent a radial basis function network exactly? If the answer is positive, how does it perform comparing to the standard multilayer perceptron and radial basis function networks?

1.3 Thesis Outline

This thesis consists of five chapters.

Chapter 2 presents a geometrical interpretation of multilayer perceptron based on the geometrical meaning of the weights of a single hidden neuron, discusses the selection of the hidden neurons in three-layered multilayer perceptrons, and analyze the advantages offered by four-layered multilayer perceptrons.

Chapter 3 gives an overview of the overfitting problem, a comparative study is carried out on various alleviating methods for this problem and the reasons behind these methods are reviewed based upon the geometrical interpretation in Chapter 2.

In Chapter 4, another popular feedforward neural network, radial basis function network, is visited. A special additional input, which is the sum of the squares of the other inputs, is added to the standard multilayer perceptron, so that the multilayer perceptron works similarly to the radial basis function network with localized response. Specially, we will show a three-layered multilayer perceptron with exponential activation function and this kind of additional input is naturally a generalized radial basis function network and multilayer perceptron, which can by trained with the well developed training strategies of multilayer perceptrons. Then a comparative study is conducted between multilayer perceptrons, multilayer perceptrons with additional inputs and radial basis function networks trained by various methods.

Chapter 5 concludes the whole thesis and points out the direction of future research.

Chapter 2

Architecture Selection of Multi-layer Perceptron

2.1 Introduction

As mentioned in the previous chapter, 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? The common practice is still based on a trial and error method, which choosing the number of neurons manually until the network can solve the practical problem in hand. Traditionally, the main focus regarding the architecture selection of MLP has been centered upon the growing and pruning techniques (LeCun et al. 1990; Weigend et al. 1991; Hassibi et al. 1992; Hush 1997). Recently, evolutionary artificial neural networks (EANN) (Alpaydim 1994; Jasic and Poh 1995; Sarkar and Yegnanarayana 1997; Yao 1999; Castillo 2000) are alternative methods concerning the architecture selection problem. However, the adaptation speed of EANN is usually very slow due to its nature of population and random search.

In previous approaches, 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 chapter 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 this Chapter.

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 it is the minimal architecture itself. As we know, searching for a minimal or sub-minimal 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 the best 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 couple 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-layer MLP. With the aid of the geometrical interpretation and also through carefully examining the 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, and for a certain class of problems the four-layered MLP outperforms three-layered MLP significantly.

2.2 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

$$\varphi(v) = \begin{cases} 1, & v \geq 0.5 \\ v + 0.5, & -0.5 < v < 0.5 \\ 0, & v \leq -0.5 \end{cases} \tag{2.1}$$

and plotted in Figure 2.1.

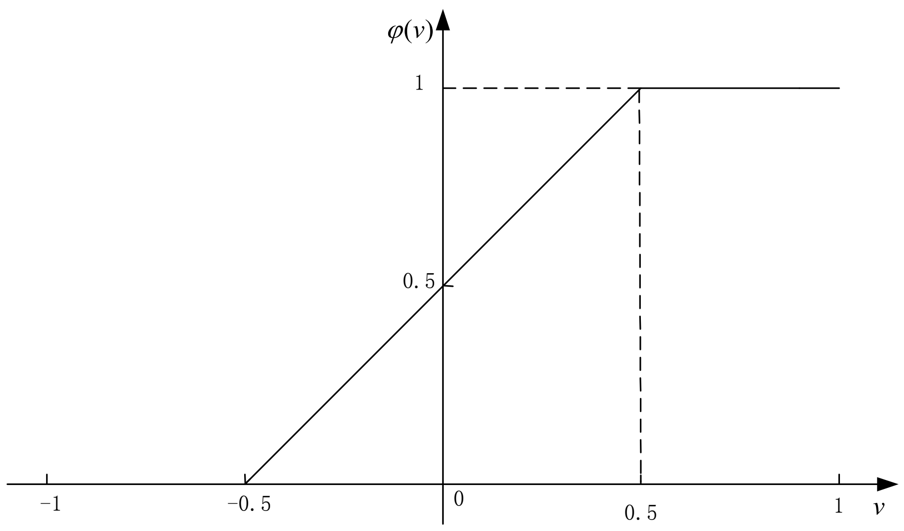


Fig. 2.1. Piecewise linear activation function.

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 x feeding into the network is

$$y(x) = \sum_{i=1}^{N} w_i^{(2)} \varphi(w_i^{(1)} x + b_i^{(1)}) + b^{(2)} \tag{2.2}$$

It is evident that $y(x)$ is just superposition of N piecewise-linear functions plus the bias. From (2.1) we know that each piecewise-linear function in (2.2) is described by

$$w_i^{(2)}\varphi(w_i^{(1)}x + b_i^{(1)}) = \begin{cases} w_i^{(2)}, & w_i^{(1)}x + b_i^{(1)} \geq 0.5 \\ w_i^{(2)}(w_i^{(1)}x + b_i^{(1)} + 0.5), & -0.5 < w_i^{(1)}x + b_i^{(1)} < 0.5 \\ 0, & w_i^{(1)}x + b_i^{(1)} \leq -0.5 \end{cases} \tag{2.3}$$

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

$$w_i^{(2)}\varphi(w_i^{(1)}x + b_i^{(1)}) = \begin{cases} w_i^{(2)}, & x \geq \dfrac{0.5}{w_i^{(1)}} - \dfrac{b_i^{(1)}}{w_i^{(1)}} \\ w_i^{(2)}(w_i^{(1)}x + b_i^{(1)} + 0.5), & \dfrac{-0.5}{w_i^{(1)}} - \dfrac{b_i^{(1)}}{w_i^{(1)}} < x < \dfrac{0.5}{w_i^{(1)}} - \dfrac{b_i^{(1)}}{w_i^{(1)}} \\ 0, & x \leq \dfrac{-0.5}{w_i^{(1)}} - \dfrac{b_i^{(1)}}{w_i^{(1)}} \end{cases} \tag{2.4}$$

The graph for this weighted piecewise linear function is plotted in Figure 2.2.

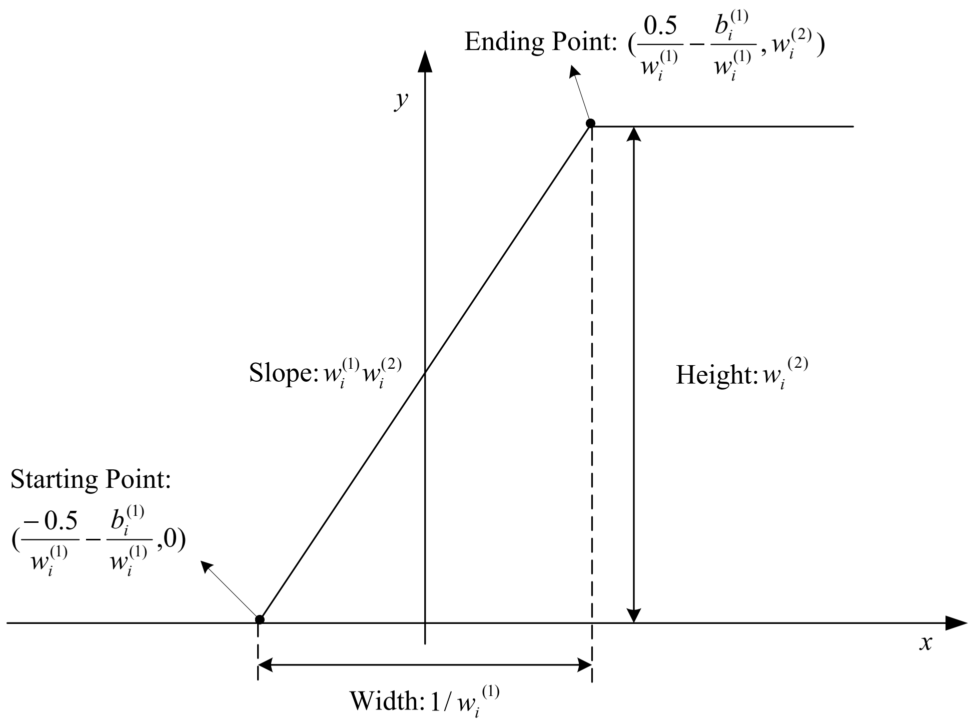

Ending Point: $(\frac{0.5}{w_i^{(1)}} - \frac{b_i^{(1)}}{w_i^{(1)}}, w_i^{(2)})$
y
Slope: $w_i^{(1)}w_i^{(2)}$
Height: $w_i^{(2)}$
Starting Point: $(\frac{-0.5}{w_i^{(1)}} - \frac{b_i^{(1)}}{w_i^{(1)}}, 0)$
x
Width: $1/w_i^{(1)}$

Fig. 2.2. Weighted piecewise linear function.

This piece-wise linear function has the same geometrical shape as that of (2.1), 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 equation (2.4) and Figure 2.2 that the width of the middle line segment is $\frac{1}{w_i^{(1)}}$, the height is $w_i^{(2)}$, the slope is therefore $w_i^{(1)}w_i^{(2)}$, 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 the 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.
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. Note that negative height implies negative slope of the middle line segment of basic building-blocks.
4) The biases in the hidden neuron govern the positions of the middle line segments of those basic building-blocks. By adjusting the values 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 and prove Theorem 2.1 as follows.

Theorem 2.1: 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 2.1 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 two end points. Denote the end points of the i^{th} line segment as $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$, where $x_0 = a$, and $x_N = b$. The width and height of the i^{th} line segment are then $x_i - x_{i-1}$ and $f(x_i) - f(x_{i-1})$ respectively.

Let's construct the 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 i^{th} neuron match the i^{th} line segment in $f(x)$. Therefore, the parameters of the MLP can be calculated as follows.

To match the width, we set

$$\frac{1}{w_i^{(1)}} = x_i - x_{i-1}, \ \mathrm{i} = 1, \ldots, \mathrm{N} \tag{2.5}$$

To match the height, we set

$$w_i^{(2)} = f(x_i) - f(x_{i-1}), \ \mathrm{i} = 1, \ldots, \mathrm{N} \tag{2.6}$$

To match the position, we set

$$\frac{-0.5}{w_i^{(1)}} - \frac{b_i^{(1)}}{w_i^{(1)}} = x_{i-1}, \ \mathrm{i} = 1, \ldots, \mathrm{N} \tag{2.7}$$

To match the final value of $f(x)$, we need to provide the offset term as

$$b^{(2)} = f(x_0) = f(a) \tag{2.8}$$

The parameters of the three-layered MLP are completely determined by equations (2.5) to (2.8). 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 $[x_{i-1}, x_i]$ comes only from the middle line segment provided by the i^{th} neuron. From (2.5) and (2.6), 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 $b^{(2)}$,

$$NN(x_0) = b^{(2)} \tag{2.9}$$

At point $x = x_1$, which is the end 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

$$NN(x_1) = w_1^{(2)} + b^{(2)} \tag{2.10}$$

Similar argument leads to

$$NN(x_i) = w_1^{(2)} + \cdots + w_i^{(2)} + b^{(2)}, \ \mathrm{i} = 1,2,\ldots,\mathrm{N} \tag{2.11}$$

From equation (2.6) and (2.8), it follows immediately that

$$NN(x_i) = f(x_i), \ \mathrm{i} = 0,1\ldots,\mathrm{N} \tag{2.12}$$

This completes the proof of Theorem 2.1.

<u>Comment 2.1:</u> The weights and biases constructed by equations (2.5) to (2.8) 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 let $w_i^{(1)} > 0$ in all our discussions so far. Without this

constraint, the sign of the slope of piecewise line is determined by $w_i^{(1)} w_i^{(2)}$, and consequently there are many other combinations of the building-blocks that can construct the same piecewise linear function exactly. This implies that the global minimum may not be unique in many cases.

Comment 2.2: In the proof given, 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 plotted in Figure 2.3, where solid line is the combination of two basic building-blocks, which are plotted with dash-dotted and dashed lines respectively.

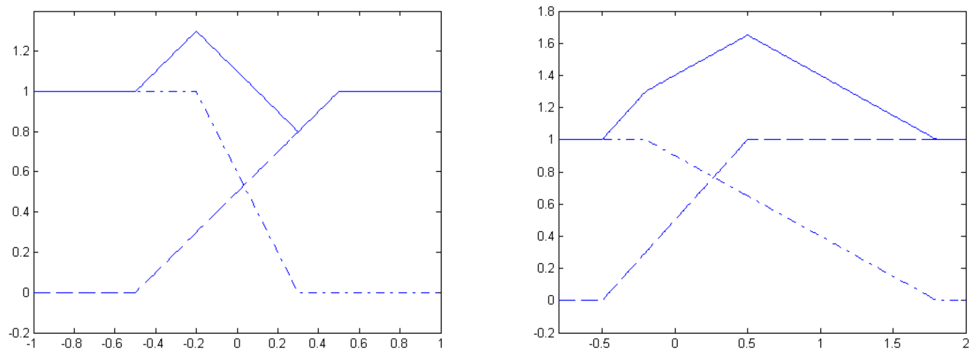


Fig. 2.3. Overlapping of basic building-blocks.

Comment 2.3: Since any bounded continuous function can be approximated arbitrarily closely by piecewise linear function, Theorem 2.1 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 (Hornik et al. 1989; Cybenko 1989; Funahashi; 1989). Although the proof is given only for the case of piecewise-linear activation functions, the geometrical nature of the proof presented in this Chapter makes this nice property of MLP much more transparent than other approaches.

Comment 2.4: 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 smoothed out as shown in Figure 2.4. 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 non-smooth end points, the MLP with sigmoid activation functions is more efficient to approximate smooth functions.

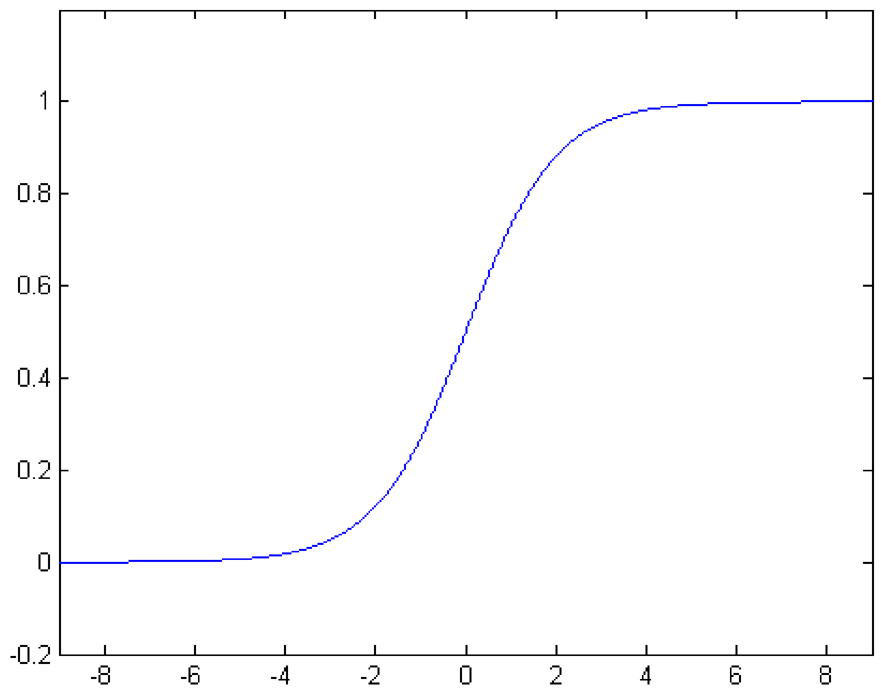


Fig. 2.4. Sigmoid activation function.

Comment 2.5: 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 example of such building-blocks is shown in Figure 2.5 where sigmoid activation functions are used.

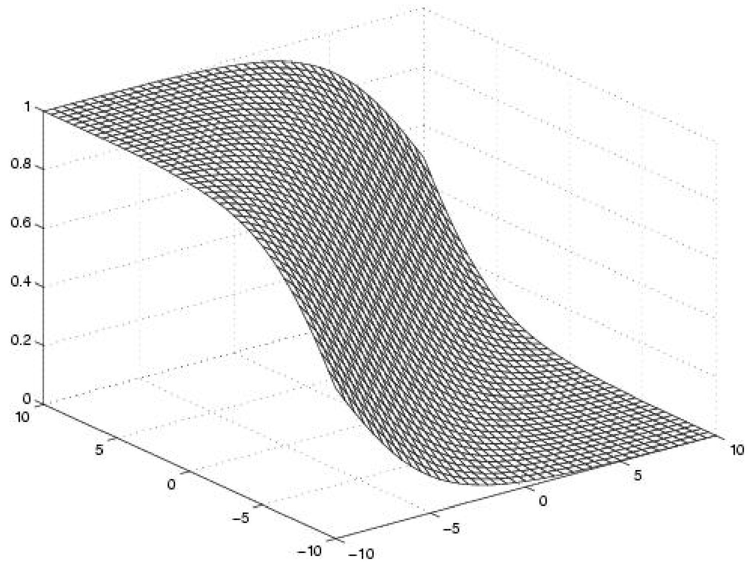


Fig. 2.5. Two-dimensional building-block.

2.3. 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, we suggest a simple guideline for choosing the number of hidden neurons for the three-layered MLP 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.

We have tested this guideline with extensive simulation studies. In all of the cases studied, 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 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 (Marquardt 1963; Mor 1977) (called “trainlm” in MATLAB) is used as the training algorithm. The Nguyen-Widrow method (Nguyen and Widrow 1990) is utilized to initialize the weights of the each layer of the MLPs.

Comment 2.6: The selection of the activation function and training algorithm is another interesting issue which was investigated by other papers (Hush and Salas 1988; Mennon et al. 1996; Amri 1998). We will not delve into this issue here. We choose “tansig” and “trainlm” just by simple trial and error studies.

Simulation 2.1: The target function is chosen as:

$$f(x) = x^3 + 0.3x^2 - 0.4x,\ x \in [-1,1] \tag{2.13}$$

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 is tried first. It turns out that 1-3-1 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 Figure 2.6, 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.

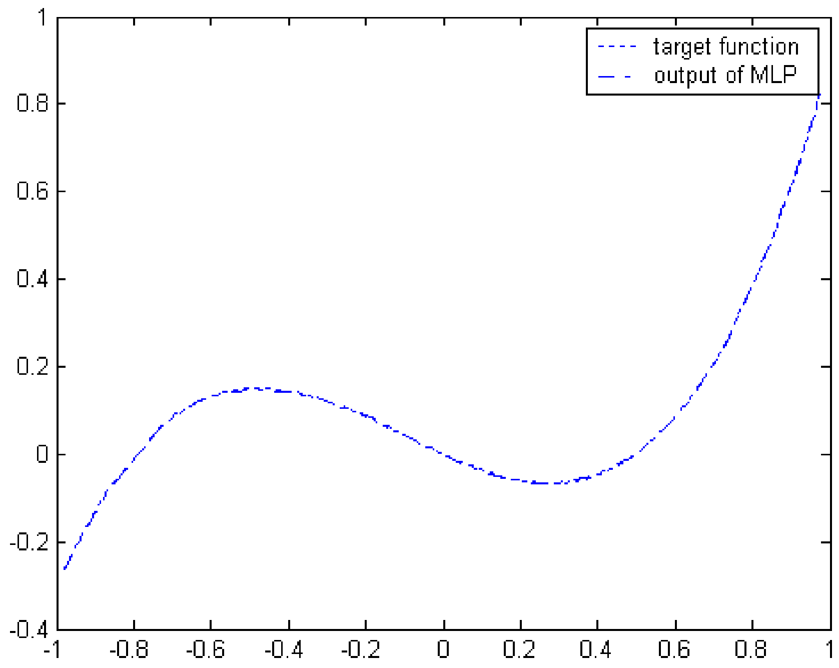
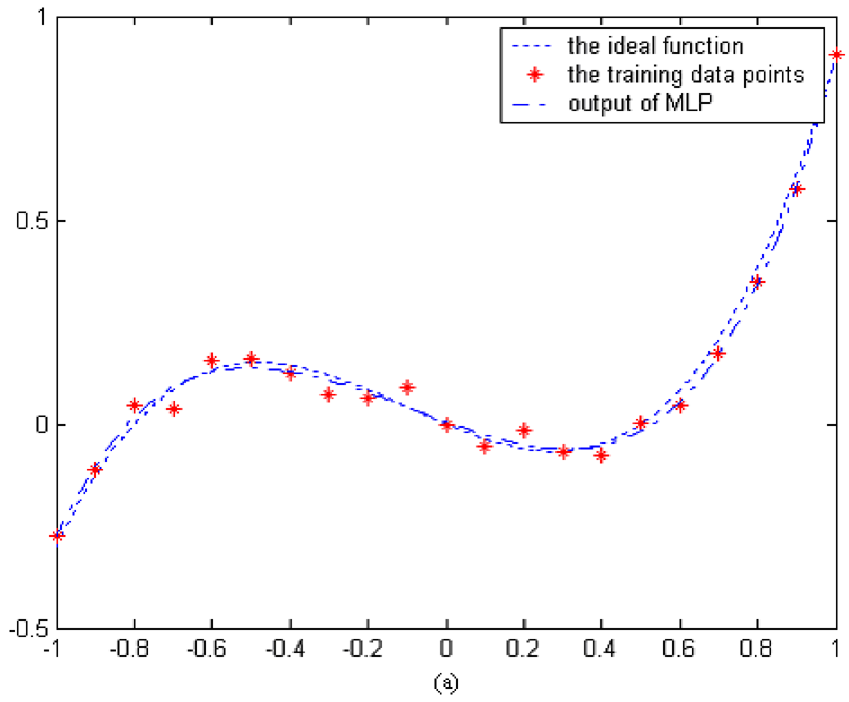


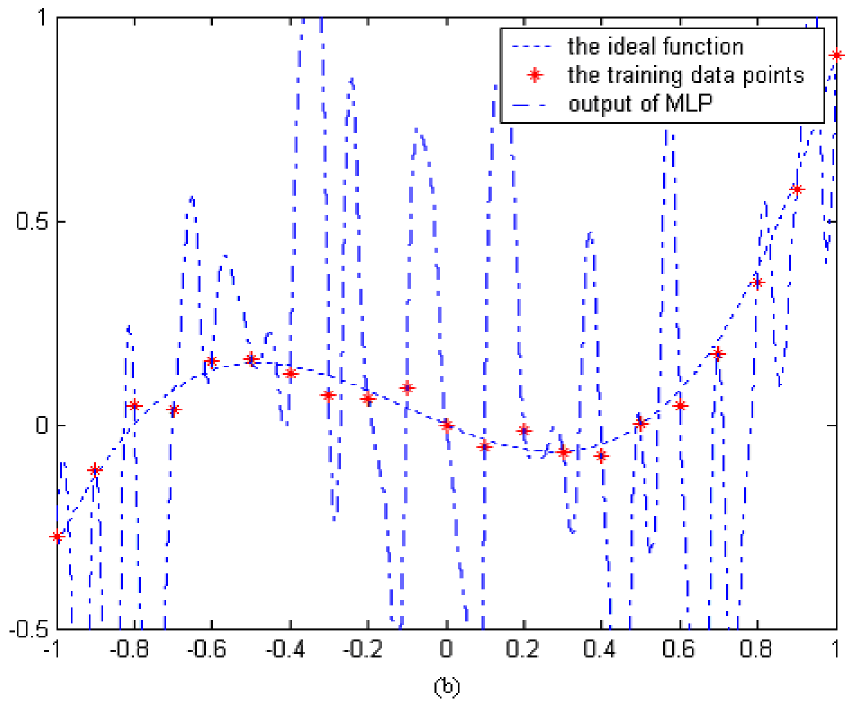
Fig. 2.6. A simple one-dimensional example.

Comment 2.7: 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 2.2: Assume the samples of the target function in Simulation One are corrupted by noises uniformly distributed in [-0.05, 0.05]. Both 1-3-1 and 1-50-1 are used to learn the same set of training data, and the test set contains 100 points uniformly randomly selected in [-1, 1]. The results are shown in Table 2.1 and are plotted in Figure 2.7.



(a) Approximation by 1-3-1



(b) Approximation by 1-50-1

Fig. 2.7. The noisy one-dimensional example.

TABLE 2.1

Significantly different performance 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

Comment 2.8: 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" with only 2 epochs.

Simulation 2.3: We intend to approximate a more complicated function as follows,

$$y = 0.5\sin(\pi x)^3 - \frac{2}{x^3+2} - 0.1\cos(4\pi x) + |x|, \qquad -1 \le x \le 1.6 \tag{2.14}$$

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.

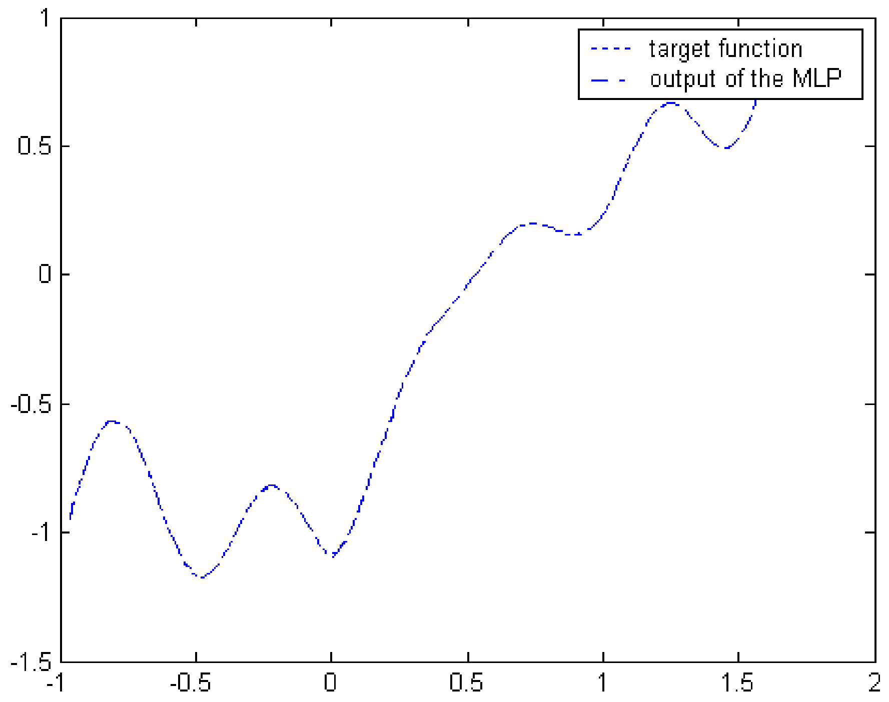


Fig. 2.8. A complicated one dimensional example.

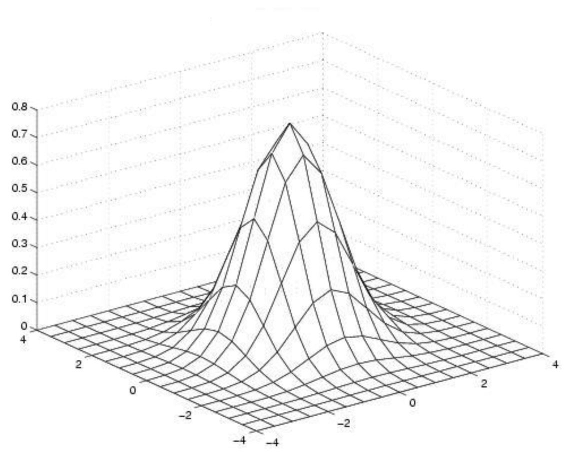
Comment 2.9: Smaller sized MLP such as 1-8-1 and 1-7-1 are also tested to solve this problem. Both of them are able to provide good approximations except in the small neighborhood around $x = 0$ where the error bound is bigger than 0.01 (but smaller than 0.04). The reader is referred back to Comment 2.2 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 2.4: We move on to consider a simple two-dimensional example, a Gaussian function described by

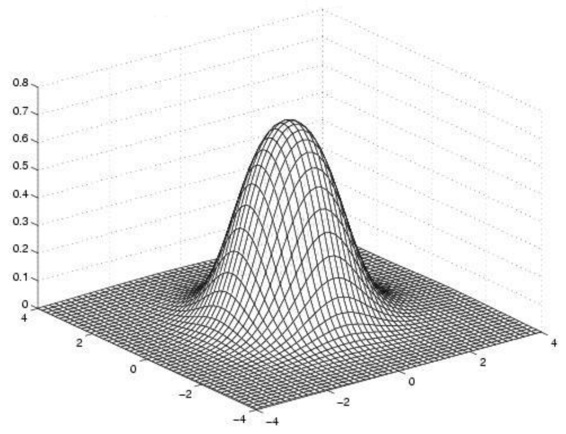
$$f(x,y) = \frac{5}{2\pi}\exp(-\frac{x^2+y^2}{2}),\ x,y \in [-4,4] \tag{2.15}$$

The training set comprises 289 points, which are chosen by uniformly partitioning the domain $x, y \in [-4,4]$ with grid size of 0.5. The test set contains 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}. The result is reasonably good as shown in Figure 2.9, if we consider the error bound of about 0.07 to be acceptable.

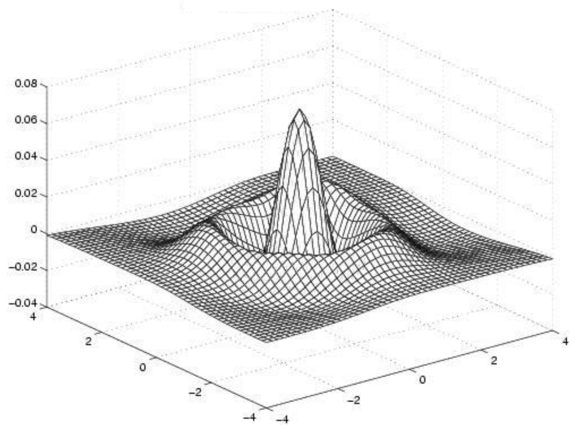
Comment 2.10: It is worth noting that the activation function used for the output neuron in Simulation Four is not the identity function, but the logistic function (called "logsig" in MATLAB). Since the sigmoid function has the property of flattening things outside of its focused domain, it is possible 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.



(a) Training data



(b) Output of the Neural Network



(c) Approximation error

Fig. 2.9. Approximation of Gaussian function.

Simulation 2.5: We consider a more complicated two-dimensional example as follows,

$$f(x,y) = 0.1x^2 - 0.05y^2 + \sin(0.16x^2 + 0.16y^2), \quad x, y \in [-4.5, 4.5] \tag{2.16}$$

The training set composes of 100 points, by uniformly partitioning the domain $x, y \in [-4.5, 4.5]$ 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 Figure 2.10. 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, the training error (MSE) decreases to 0.0009 and the test error (MSE) is 0.0018. The approximation result is quite well with error bound of 0.15, as shown in Figure 2.11.

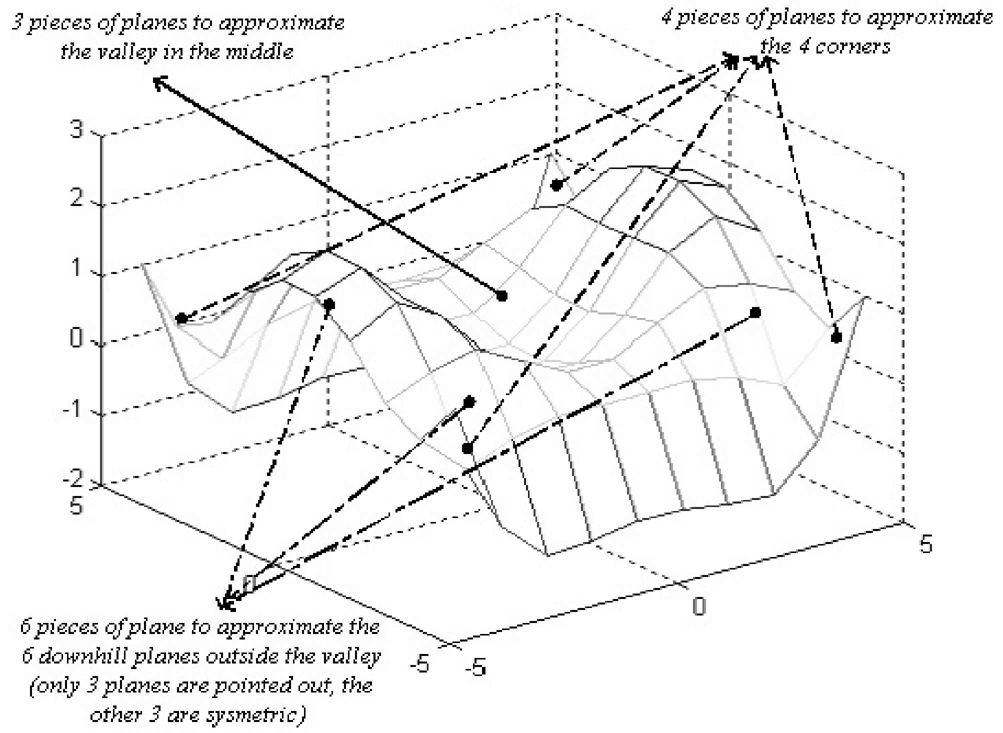
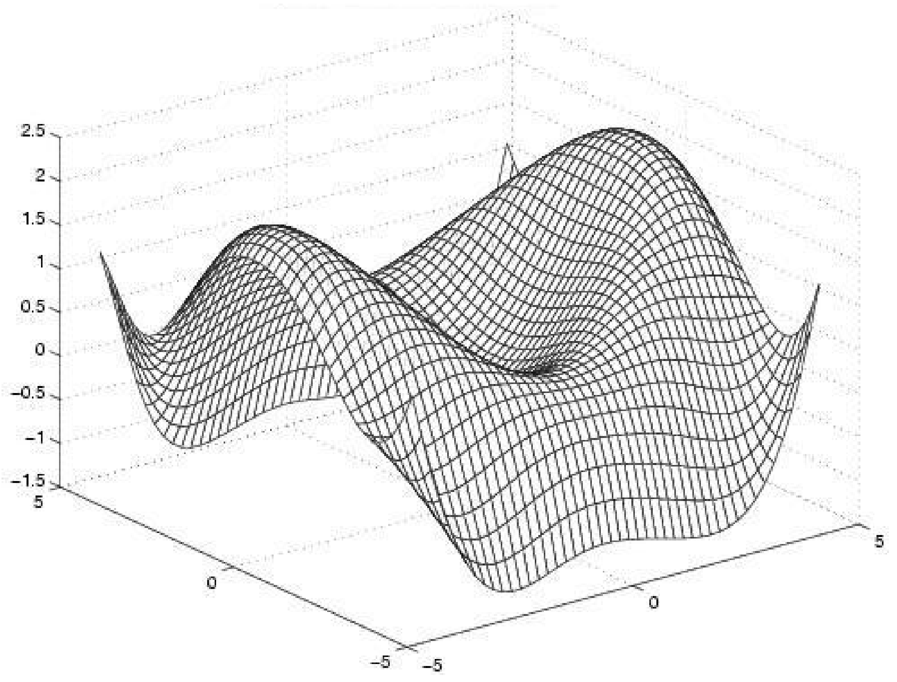
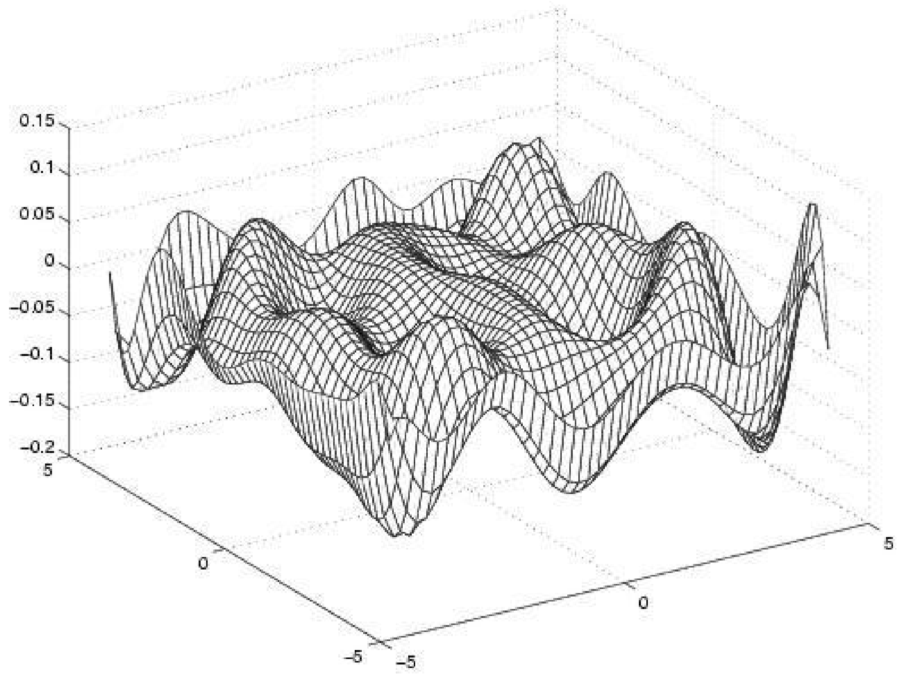


Fig. 2.10. Piecewise planes needed to construct the basic shape.



(a) Output of the Neural Network



(b) Approximation error

Fig. 2.11.A more complicated two-dimensional example.

It is observed that the local minimum 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 minimum problem, as well as to further decrease the error bound of the test set, evolutionary artificial neural networks (EANNs) are applied to this example. One of the popular EANN systems, EPNET (Yao and Liu 1997; Riessen et al. 1997), is adopted to solve the approximation problem for the function (2.16) with the same training set and test set 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 Figure 2.12, which is a simplified version of the flowchart in (Riessen et al. 1997).

The reader is referred to (Yao and Liu 1997; Riessen et al. 1997) for detailed description of the EPNET algorithm. The following comments are in order as follows to explain some of the blocks in the flowchart:

a) "MBP training" refers to training with the Levenberg-Marquardt algorithm (trainlm).
b) "MRS" refers to the modified random search algorithm, and the reader is referred to (Solis and Wets 1981) for further details.
c) "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 (Yao and Liu 1997; Riessen et al. 1997).

d) "Successful" means the validation error bound has been reduced substantially, for instance, by at least 10% in our simulations. The validation set contains 1000 random samples uniformly distributed in the domain of $[-4.5,4.5]\times[-4.5,4.5]$.

e) "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 2.2 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 (Yao and Liu 1997), the convergence speed is usually much slower as shown in Table 2.2.

TABLE 2.2

Performance comparison of EPNET with different initial populations

Structures of the initial population	Generations needed to meet the goal	The error bounds of the best network	The structures of the best network
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

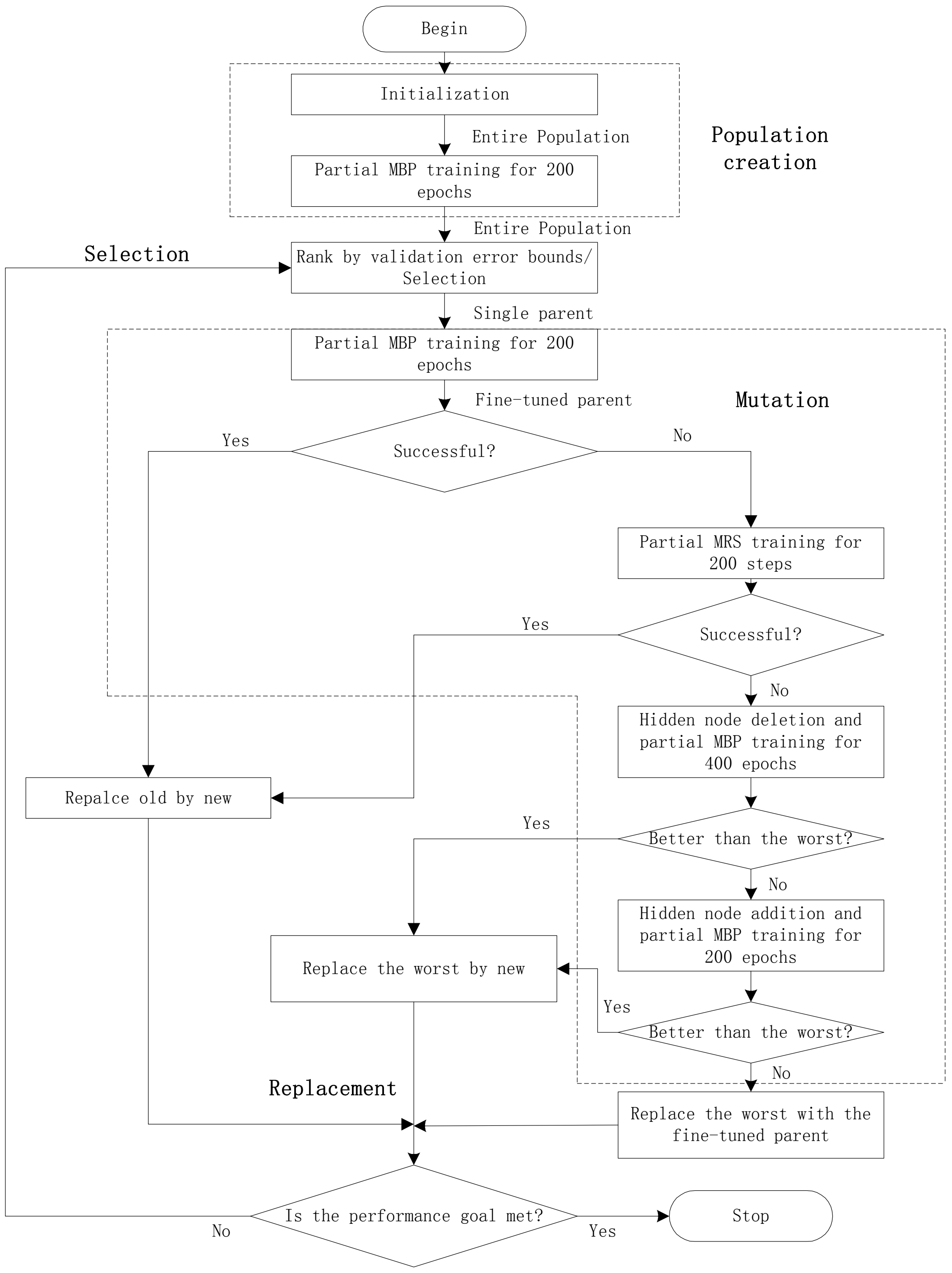

Begin
Initialization
Entire Population
Partial MBP training for 200 epochs
Population creation
Entire Population
Selection
Rank by validation error bounds/ Selection
Single parent
Partial MBP training for 200 epochs
Fine-tuned parent
Mutation
Yes
Successful?
No
Partial MRS training for 200 steps
Yes
Successful?
No
Hidden node deletion and partial MBP training for 400 epochs
Repalce old by new
Yes
Better than the worst?
No
Hidden node addition and partial MBP training for 200 epochs
Replace the worst by new
Yes
Better than the worst?
No
Replacement
Replace the worst with the fine-tuned parent
No
Is the performance goal met?
Yes
Stop

Fig. 2.12. Flowchart of the simplified EPNET.

Comment 2.11: 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 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 2.12: 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 minimum 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 2.13: 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 by other means. Either pruning and growing techniques (LeCun et al. 1990; Weigend et al. 1991; Hassibi et al. 1992; Hush 1997) or EANNs (Alpaydim 1994; Jasic and Poh 1995; Sarkar and Yegnanarayana 1997; Yao and Liu 1997; Yao 1999; Riessen et al. 1997; Castillo 2000) are then recommended to deal with such problems where geometrical information is hardly known.

2.4 Advantage Offered by Four-layered MLP

Whether adding another hidden layer to the three-layered MLP is more effective remains a controversial issue in the literature. While some published results (Chester 1990; Sontag 1992; Tamura and Tateishi 1997) suggest that four-layered MLP is superior to three-layered MLP from various points of views, other results (Villiers and Barnard 1992) claim 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 2.2.

One simple interpretation of four-layered MLP is just 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 sub-functions 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 and slope.

One obvious advantage gained by decomposing the target function into several sub-functions 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 sub-functions with simpler geometrical shapes and hence need less number of the building-blocks to construct.

Simulation 2.6: Consider the approximation problem in Simulation 2.3, the training set and the test set remain the same as those of Simulation 2.3. 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 error (MSE) reaches 2.03×10^{-5}, the test error (MSE) is 1.38×10^{-5} and the error bound of the test set is about 0.01. Due to the local minimum problem, it is hard to get a good result by only one trial, and the success rate is about one out of twenty, which is much less than the 90% success rate of 1-9-1 MLP.

Simulation 2.7: We also revisit the two-dimensional problem in Simulation 2.5 with the training data set and test data. 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

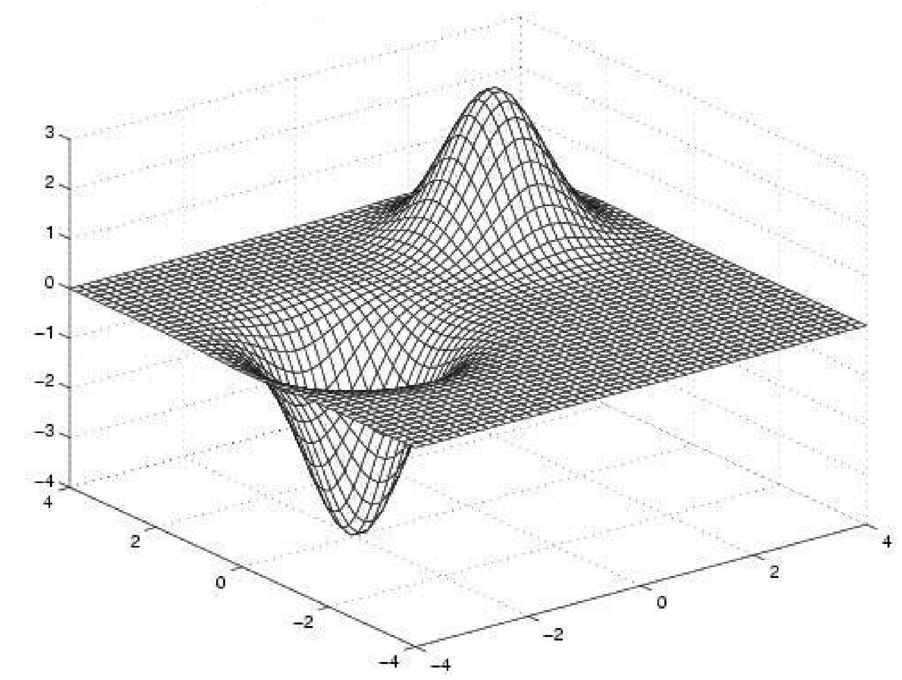
43, while the total number of the parameters for the former 2-13-1 network is 53. After 1241 epochs, the training error (MSE) decreases to 9.98×10^{-5}, the test error (MSE) is 1.09×10^{-4} and the test error bound is about 0.05.

From above two simulation examples, it is clear that four-layered MLP is more efficient than three-layered MLP in terms of the number of parameters needed to achieve similar performance. However, the difference between the 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 traps because of its more complicate structure as pointed out in (Villiers and Barnard 1992). But there are certain situations that four-layered MLP is distinctively better than three-layered MLP as shown below.

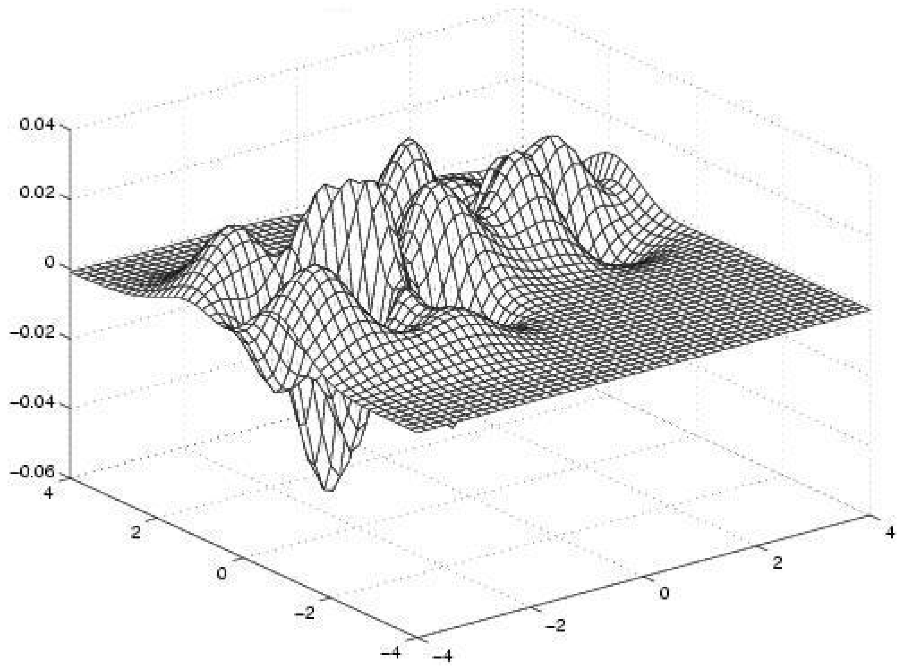
Simulation 2.8: Consider an example (Sarle 2002) made of a Gaussian hill and a Gaussian valley as follows,

$$f(x,y)=3\exp(-(x-2)^2-(y-2)^2)-4\exp(-(x+2)^2-y^2),\quad x,y\in[-4,4] \qquad (2.17)$$

The training set consists of 1681 points, which are sampled by uniformly partitioning the domain $x,y\in[-4,4]$ with grid size of 0.2. The test set comprises 1000 points randomly chosen from the same domain. A 2-4-2-1 network is used to approximate it quite well as shown in Figure 2.13. The training error (MSE) is reduced 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's consider another related hill and valley example.



(a) Output of the Neural Network



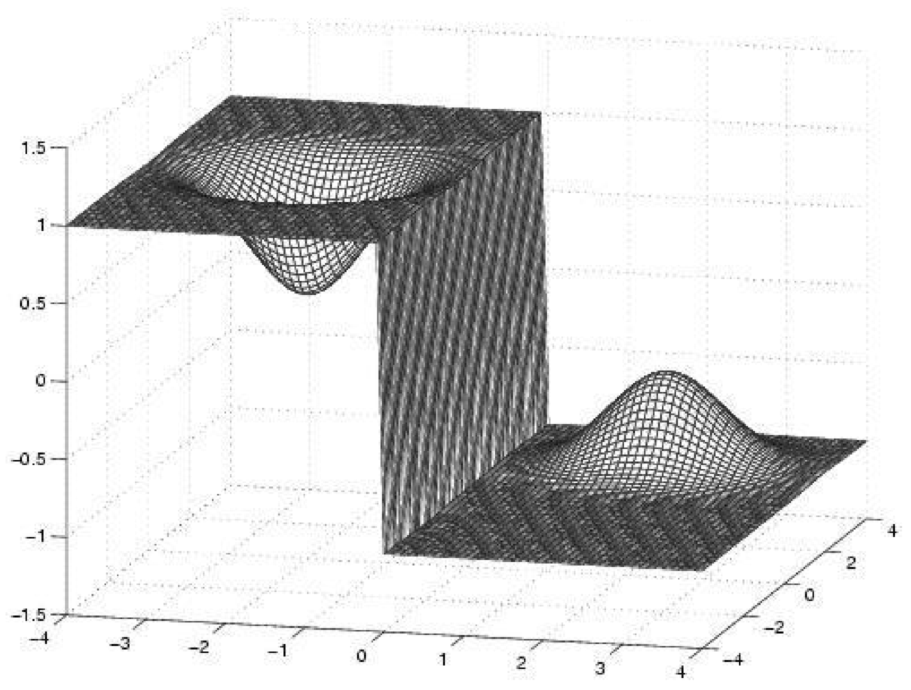
(b) Approximation error

Fig. 2.13. Approximating hill and valley with a 2-4-2-1 MLP.

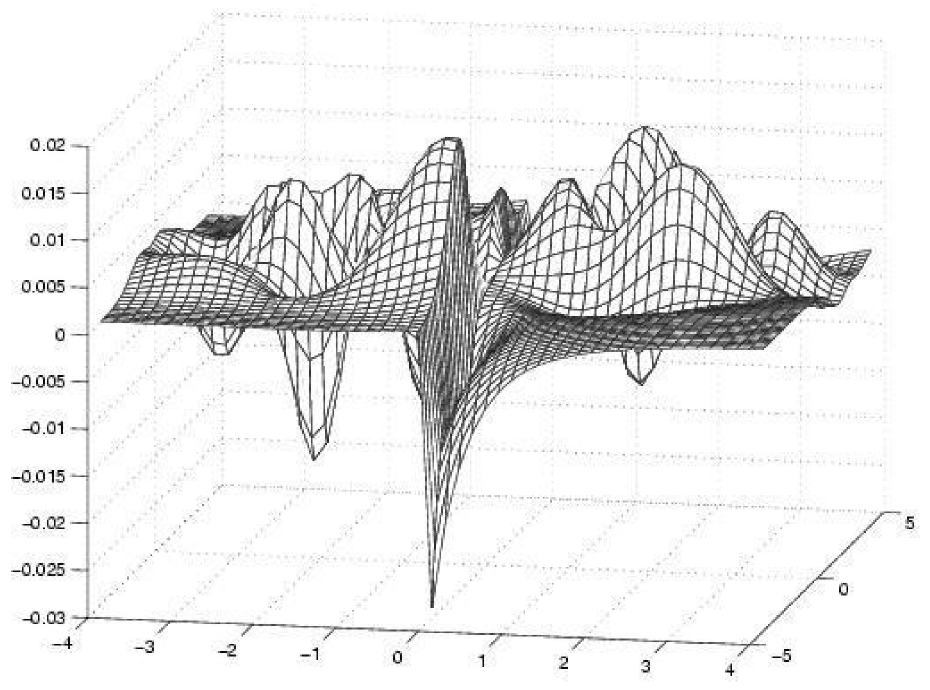
Simulation 2.9: It is still a hill and valley problem as described below and shown in Figure 2.14,

$$f(x,y)=\begin{cases}0.6\exp(-(x-2)^2-(y-2)^2)-0.8\exp(-(x+2)^2-y^2)-1, x\in[0,4], y\in[-4,4]\\ 0.6\exp(-(x-2)^2-(y-2)^2)-0.8\exp(-(x+2)^2-y^2)+1, x\in[-4,0), y\in[-4,4]\end{cases} \tag{2.18}$$

The training set consists of 6561 points, which are chosen by uniformly partitioning the domain $x,y\in[-4,4]$ with grid size of 0.1. The test set composes of 2500 points randomly chosen from the same domain. At first glance of the geometrical shape of this function, it appears more complicated than the previous example because of the jump in the planes, 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 Figure 2.14.



(a) Output of the Neural Network



(b) Approximation error

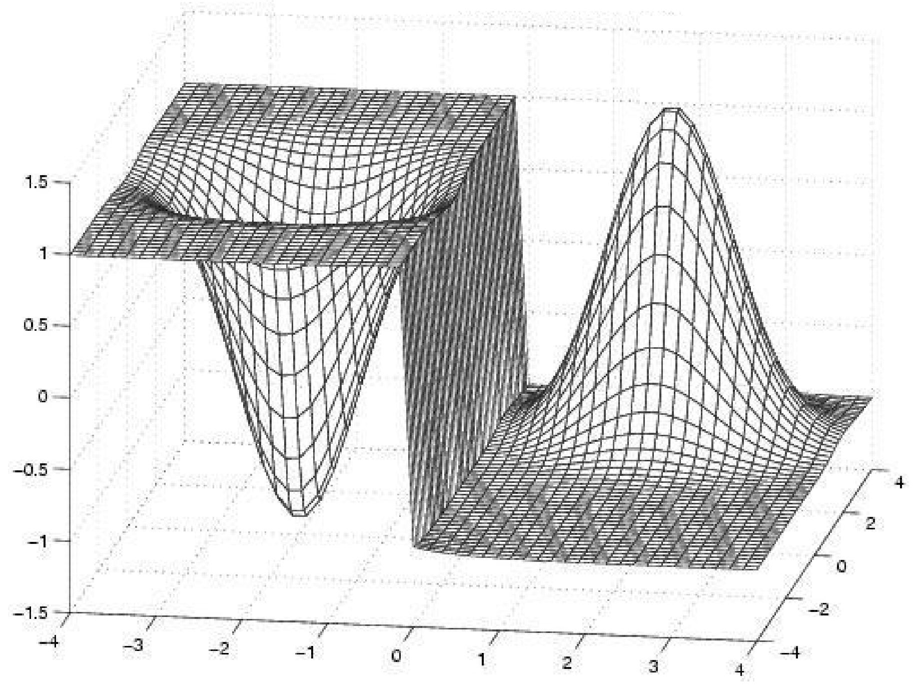
Fig. 2.14. Approximating hill and valley by a 2-5-1 network.

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 2.8 lie in the middle, while those in Simulation 2.9 are located on the top as well as at the bottom. It is noticed previously in the Gaussian function example (Simulation 2.4) that the sigmoid 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 2.9 with three-layered MLP than the function in Simulation 2.8. To verify this explanation, we increase the height of the hill as well as the depth of the valley in Simulation 2.9 such that they are higher or lower than the two flat planes, then it becomes very difficult to approximate with three-layered MLP, as shown in the following simulation.

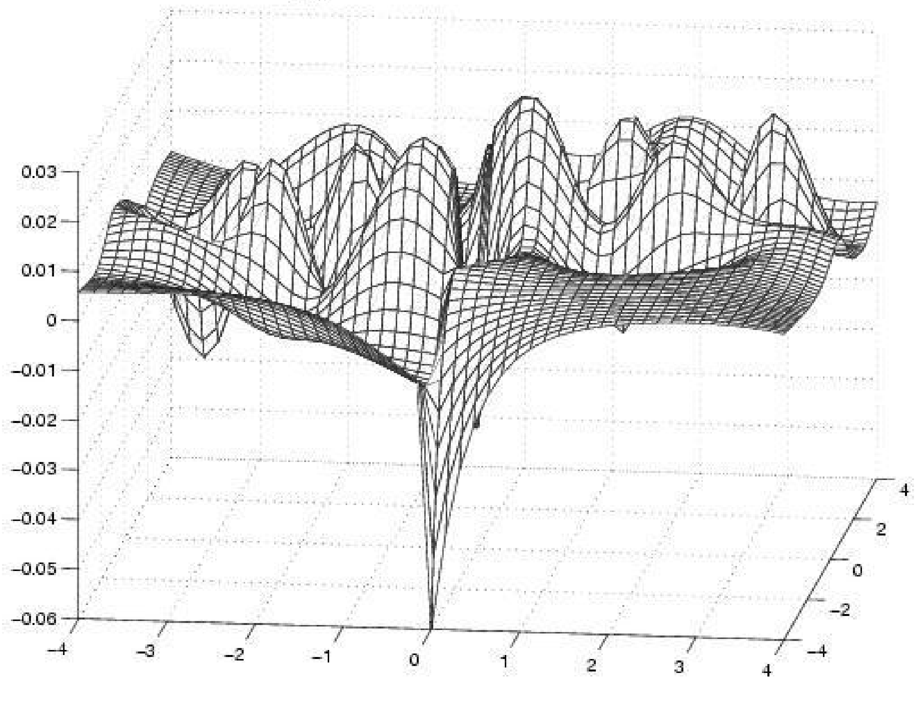
Simulation 2.10: We slightly change the approximation problem in Simulation Nine as follows.

$$f(x,y)=\begin{cases}2.3\exp(-(x-2)^2-(y-2)^2)-2.4\exp(-(x+2)^2-y^2)-1, x\in[0,4], y\in[-4,4]\\ 2.3\exp(-(x-2)^2-(y-2)^2)-2.4\exp(-(x+2)^2-y^2)+1, x\in[-4,0), y\in[-4,4]\end{cases} \quad (2.19)$$

The difference between this example and 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 set and test set are 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 plotted in Figure 2.15.



(a) Output of the Neural Network

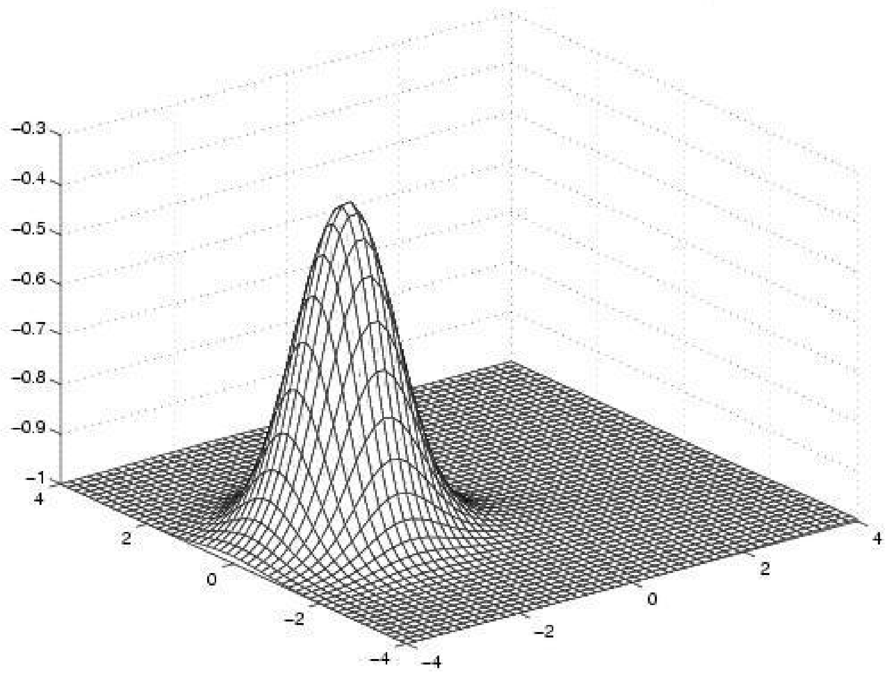


(b) Approximation Error

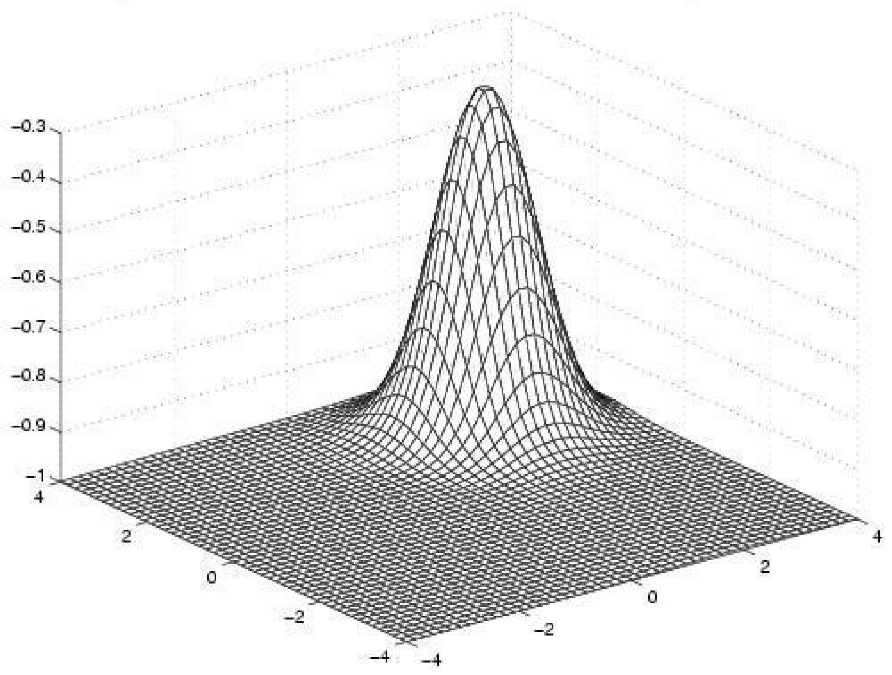
Fig. 2.15. The modified hill and valley example.

From above discussion the reason why a simple 2-4-2-1 four layered MLP can approximate hill and valley very well should be also clear now. As we discussed before, the four-layered MLP has the capability of decomposing the task of approximating one target function into tasks of approximating sub-functions. If the target function with flat regions in the middle as in the case of Simulation 2.8 and 2.10 can be decomposed into linear combination of sub-functions 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 sub-function 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 2.8 are plotted out in Figure 2.16, which are interestingly in the shape of a hill with flat areas around. It is apparent that these two sub-functions which are constructed by three-layered MLPs can easily combine into a shape consisting of a hill and a valley by subtraction.

Comment 2.14: 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 Figure 2.16, as shown in Figure 2.17, when different initial weights 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.

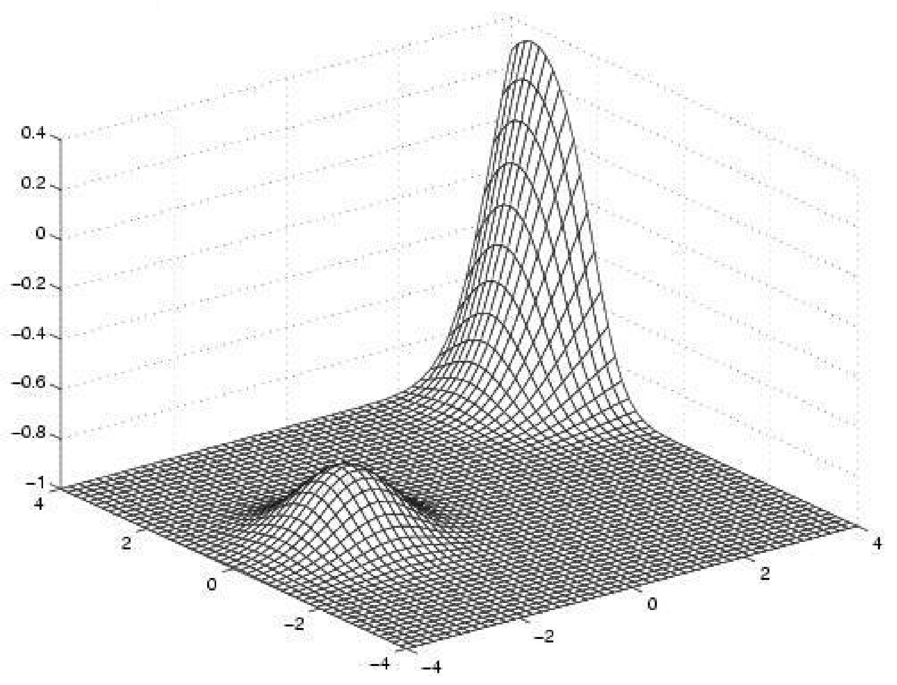


(a) Output of the first hidden neuron

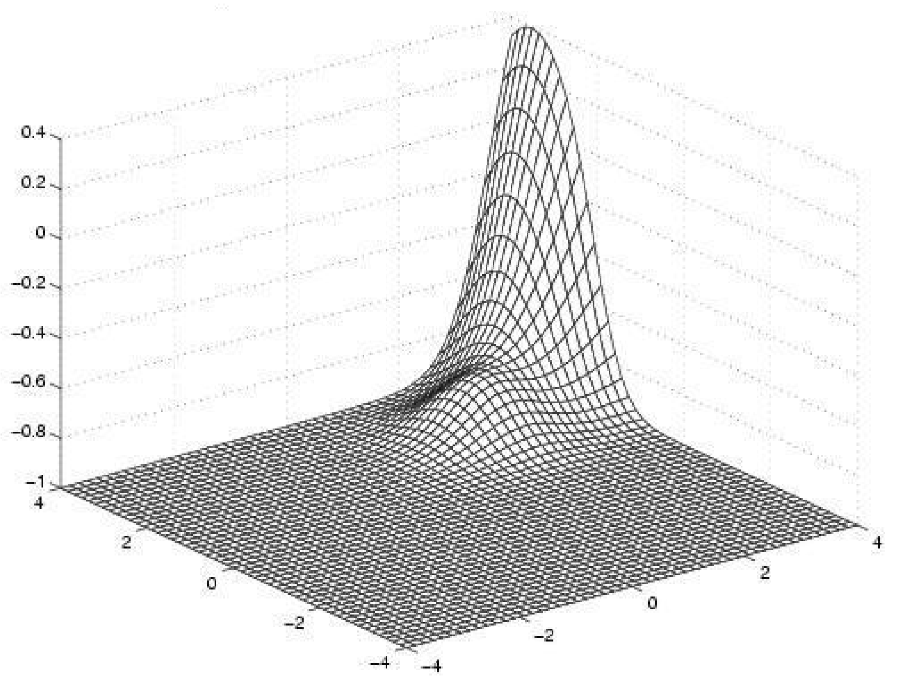


(b) Output of the second hidden neuron

Fig. 2.16. The 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

Fig. 2.17. The outputs of the neurons in the second hidden layer with different initialization.

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 2.15: The Gaussian hill and valley example is the most well known example (Sarle 2002) 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 (Chester 1990), which was not well founded.

Comment 2.16: Sontag (1992) proved 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 (Sontag 1992) 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.

2.5 Conclusions

A geometrical interpretation of MLPs is suggested in this Chapter, 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 have 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 sub-functions 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 various simulation studies that these guidelines are very effective for searching of the minimal structure of the MLP, which is very critical in many application problems.

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 (1992; 1993), 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 to high dimensional problems because the basic geometrical shapes of high dimensional target functions are very difficult to determine. Consequently, how to extract the basic geometrical shape of a high dimensional target function from the available training data would be a very interesting and challenging problem.

Chapter 3

Overfitting Problem of MLP

3.1 Overfitting Problem Overview

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.

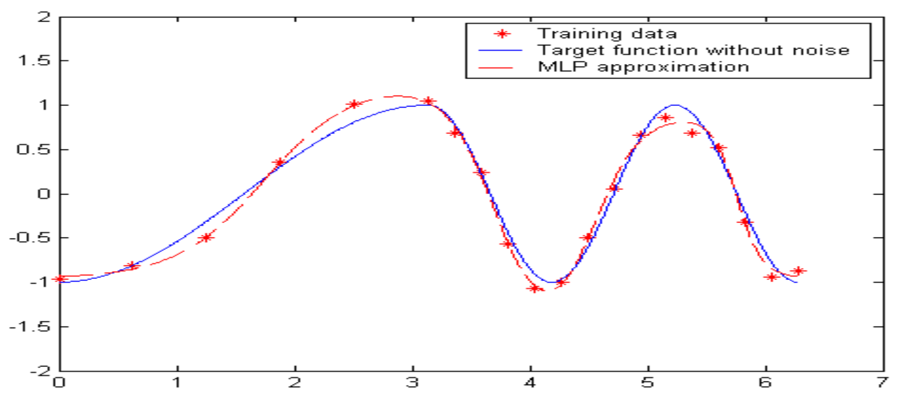
Model selection mainly focuses on the size of the neural network, i.e. the number of weights, while most other approaches are related to the size of the weights, directly or indirectly. They are actually the two aspects of the complexity of the networks. Therefore it is of great interest to gain deeper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

Based on the geometrical interpretation presented in Chapter 2, how the number and the size of the weights influence the over-fitting problem will then be clearly explained. Various 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.

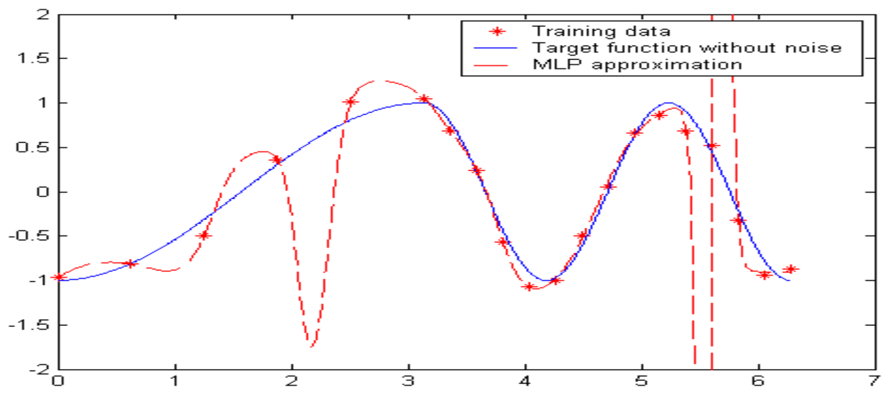
Applying the geometrical interpretation, a brief overview of over-fitting and some popular approaches to improve generalization will be discussed in this Chapter. An example of over-fitting problem (Caruana et al. 2000) is illustrated in Figure 3.1, which is a function approximation with a three-layered (one hidden layered) MLP. The training dataset is created by

$$y = \begin{cases} -\cos(x) + v & 0 \le x < \pi \\ \cos(3(x-\pi)) + v & \pi \le x \le 2\pi \end{cases} \tag{3.5}$$

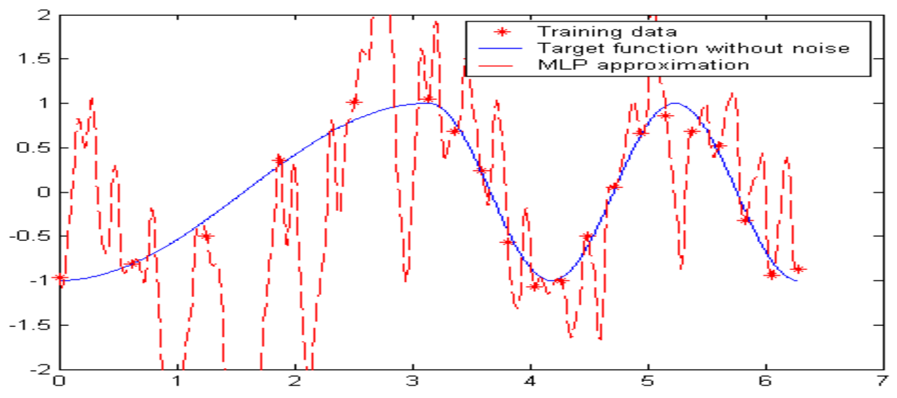
And the noise v is uniformly distributed within [–0.25,0.25]. The MLP is trained with Levenberg-Marquardt algorithm using Neural Network toolbox of MATLAB. With 4 hidden neurons, the approximation is fairly good. When the number of hidden neurons increases, significant over-fitting and poor generalization 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.



(a) 1-4-1



(b) 1-10-1



(c) 1-100-1

Figure 3.1: 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 (1997) made a surprising observation that for valid generalization, the size of the weights is more important that the size of the network, 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.

3.2 Comparative Study of Available Methods

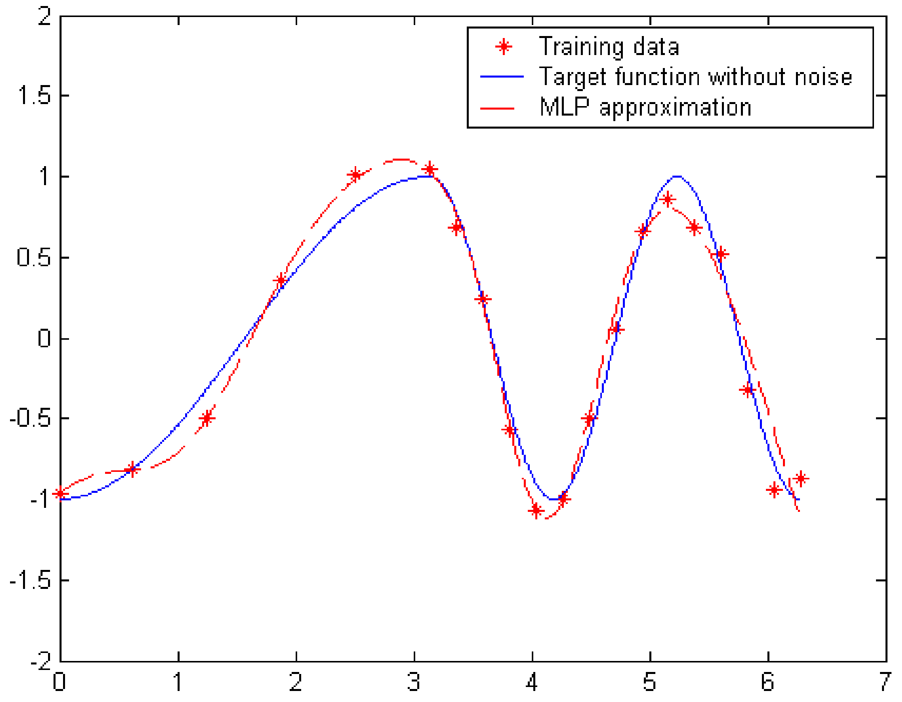
3.2.1 Model Selection

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, some much simpler guidelines have already been proposed in Chapter 2.

Following the guidelines, 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 3.1. 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 as shown in Chapter 2.

3.2.2 Early Stopping

Early stopping is another popular method to overcome the over-fitting problem in the training progress (Sarle 1995). The main idea is to stop training when the validation error goes up. Figure 3.2 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. Confining the size of the weights to be small is also a good remedy to alleviate the over-fitting problem as discussed before.



1-100 –1

Figure 3.2: Early stopping for overcoming over-fitting problem

3.2.3 Regularization Methods

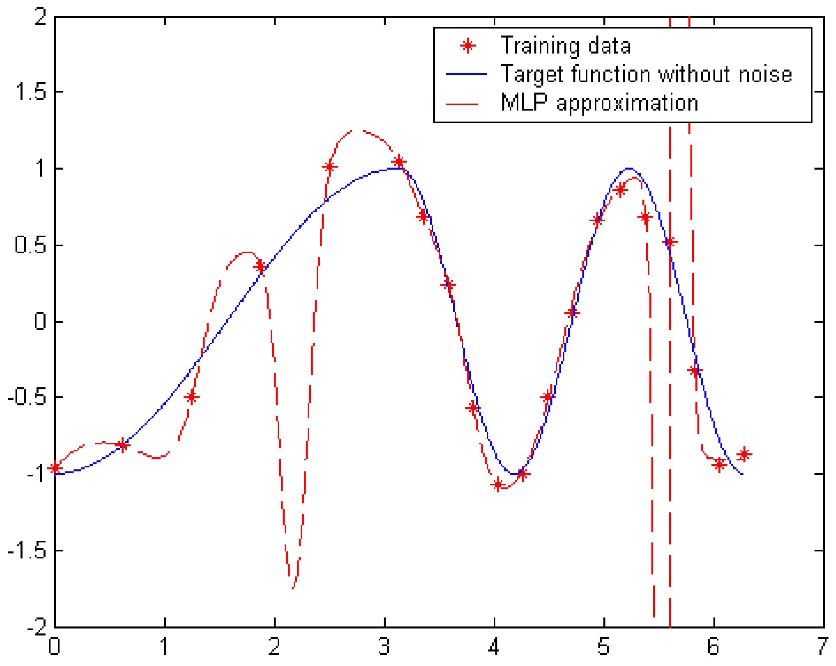
Conventionally, the training progress is to minimize the cost function $F = E_D$, where E_D is the summation of the squared errors. Regularization methods add a penalty term to the cost function. Usually, the penalty term is a function of the weights, which is called complexity penalty. Then the cost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

Weight decay (Plaut et al. 1986) 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 (Weigend et al. 1991) 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.

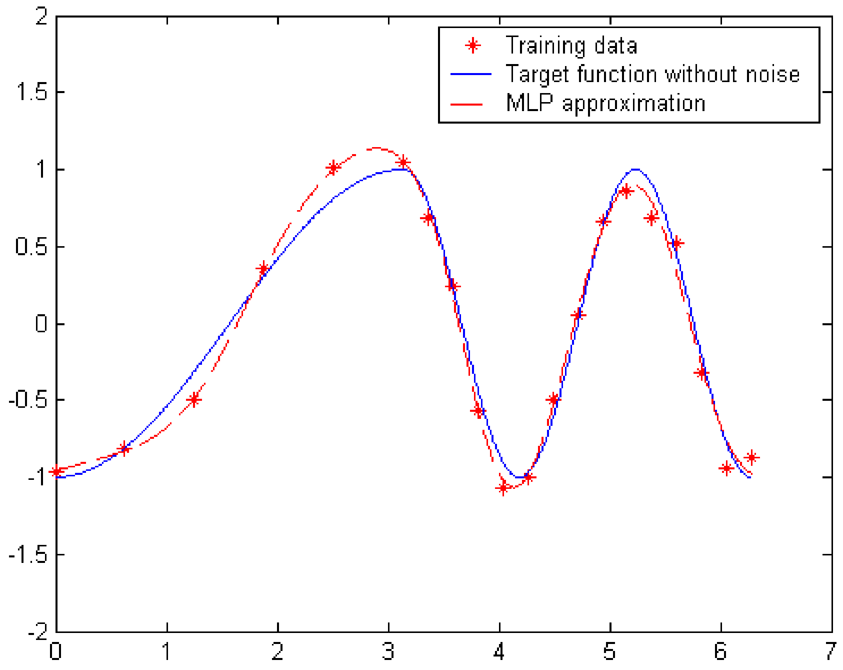
A more recent regularization method proposed by Moody and Rögnvaldsson (1997) work much better than the standard weight decay and weight elimination. In their approach, for the case of one-dimensional map, the complexity penalty E_w for the first order local smoothing regularizer can be reduced to $E_w = \sum_{i=1}^{N} (w_i^{(2)} w_i^{(1)})^2$, 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 (MacKay 1992a; MacKay 1992b) 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 3.3. 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 (MacKay 1992b). Furthermore, this breaking down may also depend upon the initialization of the parameters as observed from our simulation studies.

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 $y = \sin(\pi x) + 0.2\sin(10\pi x)$. 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 3.4, 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.

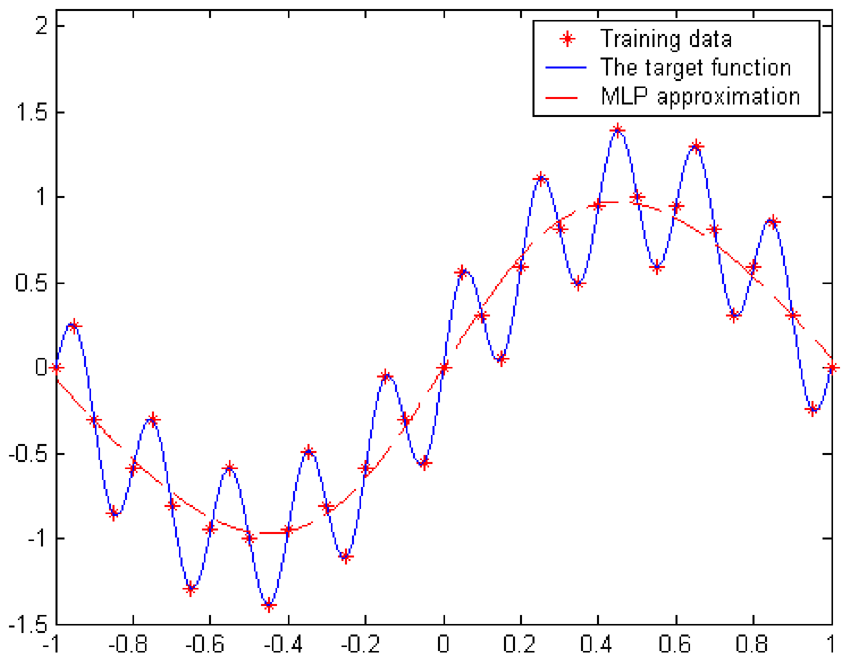


(a) 1-10-1 without Bayesian regularization

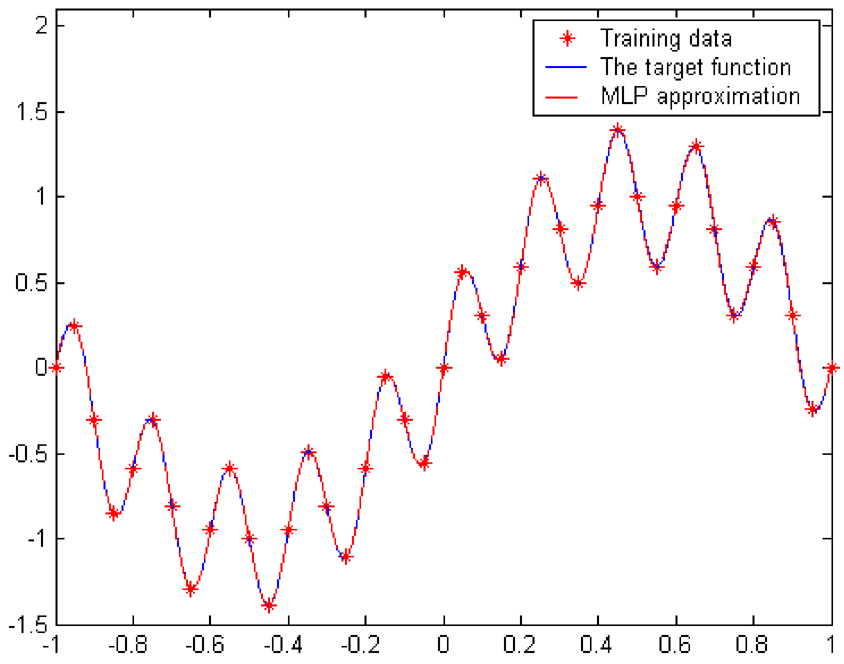


(b)1-10-1 with Bayesian regularization

Figure 3.3: Bayesian regularization for overcoming over-fitting problem



(a) 1-21-1 with Bayesian regularization



(b) 1-21-1 without Bayesian regularization

Figure 3.4: A simple example where Bayesian regularization fails

3.3 Conclusions

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 in Chapter 2 is revisited. 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.

Chapter 4

From Multilayer Perceptron to Radial Basis Function Network

4.1. Introduction to Radial Basis Function Network

Radial Basis Function Network (RBFN) is another popular feedforward neural network that is widely used in classification, regression and function approximation problems. The main difference between the MLP is that the activations of the hidden neurons of RBFN depend on the distance of an input vector from a prototype vector whereas MLP calculate the inner product of the input vector and the weight vector.

Normally, radial basis function networks have three layers with different roles. The input layer (sensors) connects the network to the environment. The hidden layer performs the key nonlinear transformation from the input space to the high dimensional hidden space in the network. The output layer gives a weighted linear combination of the hidden neuron activations. The structure of radial basis function network is shown in Figure 4.1. The kth output of the network should be:

$$f_k(X) = \sum_{i=1}^{M} w_{ki} h_i(X) \tag{4.1}$$

where X is the input vector, $h_i(X)$ is the i^th basis function and w_{ki} is the weight from the i^th basis function. The basis functions are normally multivariate Gaussian functions:

$$h_i(X) = \exp(-\frac{\|X-\mu_i\|^2}{2\sigma_i^2}) \tag{4.2}$$

where μ_i is the center of the prototype vector, σ_i is the spread of the Gaussian function and $\|X-\mu_i\|^2$ is the squared Euclidean distance between the input vector and the prototype vector.

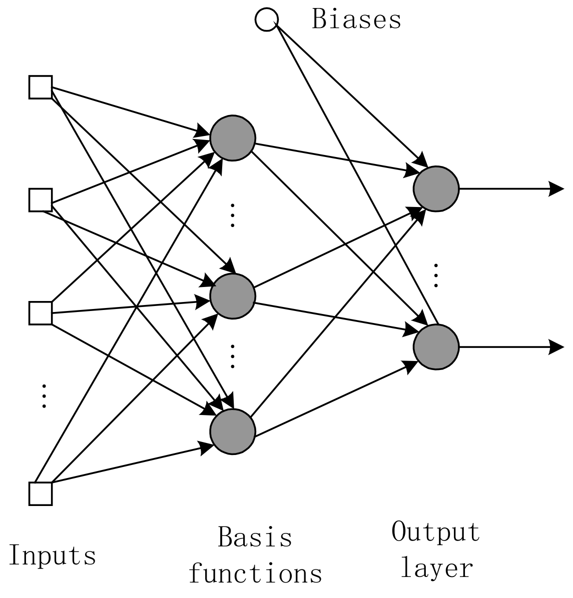


Figure 4.1 Three-layered structure of radial basis function network

A very important and interesting property is that RBFN is naturally related to the regularization network and some statistical concepts especially in classification areas. Comparing to multilayer perceptron networks, these links make radial basis function networks able to be trained by different and fast training methods (such as clustering and EM methods). The training of RBFN is usually separated in to two stages.

4.1.1 Two-stage Training of Radial Basis Function Networks

Although simultaneous adjustment of all the parameters of RBFN is also available, in practice, the estimation of the parameters is often separated into two stages:

1. Determine the centers μ_i and the relative spreads σ_i.
2. Estimate the output weights based on the previously determined centers and spreads.

Both these two stages can be solved quickly using batch mode methods. Although this kind of separation may lead to a sub-optimal solution as compared to the simultaneous training of the whole network, the difference of final performance is not that large. Actually, in many situations, it even can provide better solutions considering the finite training data and computational resource.

In the first stage, only the part of the training information is used. The centers and spreads can be determined without the target (label) information. So the learning is unsupervised at first. Once the centers and spreads are set, supervised learning will be conducted to calculate the output weights.

Random Selection of Centers

The most convenient and fast way is to choose fixed parameters for the basis functions. The locations of the centers may be simply randomly chosen from the training data set, sometimes, even the whole training data set. This is considered to be a "sensible" approach, since the training data are distributed in a representative manner for the problem at hand (Lowe 1989). Specifically, a radial basis function centered at μ_i is defined as:

$$H(\|X-\mu_i\|^2)=\exp(-\frac{n}{d_{\max}^2}\|X-\mu_i\|^2),\ \ i=1,2,\ldots,n \tag{4.3}$$

Where n is the number of centers and $d_{\max}$ is the largest distance between the chosen center vectors. Then the spread (standard deviation) of all the Gaussian basis functions is:

$$\sigma=\frac{d_{\max}}{\sqrt{2n}} \tag{4.4}$$

So that each individual radial basis function will not be too steep or too flat. Small spread can lead to less smooth functions. Another empirical method of choosing the spreads is to set the spread to be 1.5 to 2 times of the average distance to L nearest neighbors (Ghosh and Nag, 2000). Once the location of centers and the spreads are determined, the network can be treated as a single-layer network with linear output neurons. So that least-squares solution can be applied to get the weights:

$$W^T=H^+D \tag{4.5}$$

where D is the target vector in the training set, H^+ is the pseudo-inverse of the basis function matrix H. This kind of random selection of centers seems somehow rough, but it is often used because such ad hoc procedure is very fast (Bishop 1995). And it actually works satisfactorily in many practical issues.

Clustering Algorithms

A more suitable approach is to choose the centers using clustering algorithms, which can separate the given training points into subsets. Then the location of the centers can be obtained by calculating the geometric mean of the points in the subsets. There are many of such clustering algorithms. Among all, self-organized learning or K-means clustering algorithm (Mcqueen 1967; Duda and Hart 1973, Moody and Darken 1989; Kohonen 1990) is widely used. The K-means algorithm partitions the training data points into K subsets S_j by minimizing criterion with the clusters:

$$J = \sum_{i=1}^{K} \sum_{n \in S_i} \left\| X^n - \mu_i \right\|^2 \tag{4.6}$$

where μ_i is the center of the ith subset (N_i is the number of points in the ith subset):

$$\mu_i = \frac{1}{N_i} \sum_{n \in S_i} X^n \tag{4.7}$$

The partition of the data set is normally at random at first. Then the centers for each of the subsets are calculated using equation 4.6. After that, each data point is reassigned to the nearest center calculated. This procedure is iterated until there is no further change in the partition. Although the above clustering procedure is a batch one, sequential clustering is also available (Haykin 1999). Similar spreads determination and output weights linear least-squared solution can be applied after the location of centers is settled down.

4.1.2 One-stage Supervised Training of Radial Basis Function Networks

Radial basis function network is a specific feedforward neural network. It can also be trained in a similar way to the multilayer perceptron. The first step is also to define the cost function, which is usually the sum-squared error.

$$E = \frac{1}{2} \sum_{j=1}^{N} \sum_{k=1}^{O} e_k(j)^2 \ , \quad \text{where } e_k(j) = y_k(j) - d_k(j) \tag{4.9}$$

Hence, the error gradient for linear output weights and bias:

$$\frac{\partial E(n)}{\partial w_{ki}(n)} = \sum_{j=1}^{N}\sum_{k=1}^{O} e_{kj}(n)\exp(-\frac{\left\|X_j - \mu_i(n)\right\|^2}{2\sigma_i^{\,2}(n)}) \tag{4.10}$$

$$\frac{\partial E(n)}{\partial b_j(n)} = \sum_{j=1}^{N}\sum_{k=1}^{O} e_{kj}(n) \tag{4.11}$$

The error gradient for the location and spread of the centers:

$$\frac{\partial E(n)}{\partial \mu_i(n)} = \sum_{j=1}^{N}\sum_{k=1}^{O} e_{kj}(n)w_{ki}\exp(-\frac{\left\|X - \mu_i(n)\right\|^2}{2\sigma_i^2(n)})\frac{(X - \mu_i(n))}{\sigma_i^2(n)} \tag{4.12}$$

$$\frac{\partial E(n)}{\partial \sigma_i(n)} = \sum_{j=1}^{N}\sum_{k=1}^{O} e_{kj}(n)w_{ki}\exp(-\frac{\left\|X - \mu_i(n)\right\|^2}{2\sigma_i^2(n)})\frac{\left\|X - \mu_i(n)\right\|^2}{\sigma_i^3(n)} \tag{4.13}$$

Where $e_{kj}(n)$ is the error signal of the kth output neuron respect to the jth training point at time n. Actually the gradient learning has an effect similar to a clustering effect (Poggio and Girosi, 1990). After the error gradients are ready, we can easily update these parameters with a set of learning rates η_l for different parameters. Like the supervised training of multilayer perceptron, the choice of the learning rate is also a problem. When the learning rate is too small, the convergence speed will be very slow; when the learning rate is too large, the learning procedure maybe unstable. To alleviate the influence of the chose of learning rate, adaptive learning rate with momentum is adopted in the later simulation studies. The NETtalk experiment (Wettschereck and Dietterich 1992) indicated the generalization performance of the supervised trained RBFN is better than those by two-stage trained ones. However, supervised training is computationally expensive comparing to two-stage training.

4.1.3 Difference Comparing to Multilayer Perceptron

Radial basis function network and multilayer perceptron are widely used, since they are both universal approximators. However, there are important differences between these two kinds of neural networks:

1. The hidden units of MLP compute the weighted linear summations of the inputs, where the hidden units of RBFN calculate the distance between the input vector and the prototype vectors (i.e. the centers).
2. The respond of RBFN is localized and the network can be adjusted locally with the new inputs.
3. The MLP can have a complex structure with many layers whereas the RBFN normally has only one hidden layer.
4. The parameters of MLP are usually adjusted simultaneously at one time; whereas the training of RBFN is mostly separated to two stages.

4.2. MLP with Additional Second Order Inputs

Although there are major differences between the multilayer perceptron and the radial basis function networks, they do have connection between each other. Maruyama, Girosi and Poggio have reported that for normalized inputs, multilayer perceptron network can always simulate a Gaussian radial basis function network (Maruyama et al. 1992). Wilensky and Manukian (1992) also proposed the Projection Neural Network where two different transformations from an N-dimensional input to the N+1 dimensional transformed input space were introduced, and resulted in localized reponses. All the dimensions should be recalculated in the both transformations.

Wilamowski and Jaeger (1996; 1997) also raised a simple transformation of input patterns onto a hypershpere in augmented space, and the efficiency of this method is also experimentally verified. Omohundro (1989) also mentioned a MLP with additional input which is the sum of the squares of other inputs may have localized responses like RBFNs. This kind of additional input increases the input dimension by one, which actually transforms the input to a hyperbolic surface. Casasent Networks (Casasent 1992; Sarajedini 1992) are the practical approaches of this concept, which allows either MLP or RBFNs, or combinations of these two. A more recent report is given by Ridella et al. (1997), the proposed circular backpropagation (CBP) network is also a MLP with additional input which is the sum of the squares of the original inputs. The structure of CBP network is given in Figure 4.2.

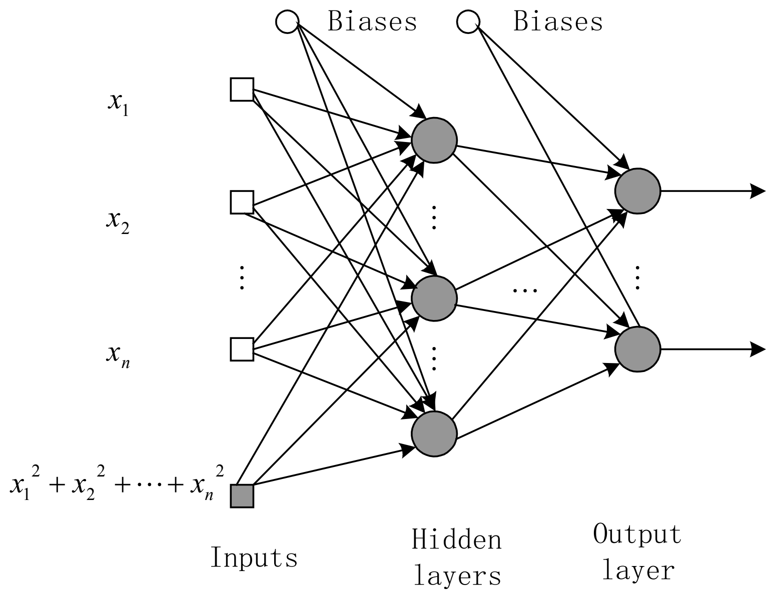


Fig. 4.2. Structure of CBP network

It is clear from Figure 4.2 that the CBP network will become a standard MLP if the weights connected to the additional input are set to zero. The three-layered CBP network (with only one hidden layer) can also approximate a RBFN with same hidden neurons. At the unit level, the CBP model can be described in the following form:

$$h(x,w) = b + \sum_{i=1}^{d} w_i x_i + w_{d+1} \sum_{i=1}^{d} x_i^{\,2} \tag{4.14}$$

It is possible to obtain another form by simple algebraic transformations:

$$h(x) = g(\|x-c\|^2 + \theta) \tag{4.15}$$

where $g = w_{d+1}$ which decides the spread of the Gaussian like function, $c_i = -w_i / 2w_{d+1}$ are the centers of these Gaussian like function, and $\theta = \frac{1}{w_{d+1}}(b - \sum_{i=1}^{d} \frac{w_i^{\,2}}{4w_{d+1}})$ works like a bias for the Gaussian like function. The activation function for CBP is sigmoidal:

$$\varphi(h) = \frac{1}{1+e^{-h}} \tag{4.16}$$

Let $h' = g\|x-c\|^2$, multiply it by an arbitrary constant which can be taken from the output weights:

$$k\varphi = \frac{k}{1+e^{-(h'+g\theta)}} = e^{h'} \frac{ke^{g\theta}}{e^{h'}e^{g\theta}+1} \tag{4.17}$$

We can choose k and $g\theta$ that let $\frac{ke^{g\theta}}{e^{h'}e^{g\theta}+1}$ arbitrarily close to 1 (Ridella et al. 1997), and then the CBP network can approximate the relevant RBFN if the remaining parts of the output weights are identical to those of the RBFN. Similarly, a three-layered CBP network with hyperbolic tangent function can also approximate a relevant RBFN with same number of hidden neurons. If exponential activation function $\varphi(h) = e^{-h}$ is selected in the CBP network, then this exponential CBP (ECBP) network is naturally a RBFN. Since the output of the exponential neuron becomes:

$$\varphi(h) = e^{-(g(\|x-c\|^2+\theta))} = e^{-g\|x-c\|^2} \cdot e^{-g\theta} \tag{4.18}$$

The term $e^{-g\theta}$ can be easily absorbed in the output weights. In the RBFNs the spreads of the Gaussian functions are always positive, but in the ECBP network the spread $g = w_{d+1}$ can be negative. So that, the ECBP network is actually a generalized version of the RBFN.

4.3. Comparative Study

To illuminate the effectiveness of the CBP network and the proposed ECBP network, a lot of simulation studies are carried out. Various training methods of RBFNs are also examined. The performances are compared together with multiplayer perception networks.

4.3.1. Simulational Setup

All of the simulation studies are conducted in MATLAB and based on the NETLAB toolbox and the Neural Network toolbox of MATLAB. All the networks have only one hidden layer and the activation functions for the output neurons are all identical functions (called "purelin" in MATLAB). The following are the detail settings of the networks to be compared:

1. **MLP**: The Levenberg-Marquardt algorithm (Marquardt 1963, Mor 1977) is used; the activation functions in the hidden layers are all hyperbolic tangent function (called "tansig") and the Nguyen-Widrow method (Nguyen and Widrow 1990) is utilized to initialize the weights of the each layer of the MLPs.

2. **CBP**: The same settings as those of MLP are adopted.
3. **ECBP**: Same settings as those of MLP are used except the activation function used in the hidden layer is exponential function.
4. **RBFN 1**: The centers are randomly chosen from the training samples following the method mentioned in section 4.1.1. Least Square method is selected to calculate the output weights of the network.
5. **RBFN 2**: k-means clustering method is adopted here, and the spread is the same as that of RBFN-1.
6. **S-RBFN**: A one-stage supervised learning algorithm with momentum and adaptive learning rate is selected to training the RBFN.

The maximum iterations for supervised networks (MLP, CBP, ECBP and S-RBFN) are fixed at 10000, and the maximum iterations for the clustering algorithm are 1000, unless specified. Each result in the tables is the optimal one from ten trials. Those did not achieve the performance goal within specified epochs were marked with “Failed”.

4.3.2. Simulational Results

Simulation 4.1: Consider the approximation problem in Simulation 3.1 with the same training and test data sets. Table 4.1 gives the results of the minimum number of hidden neurons needed to reach the test goal, which is set at 0.0001.

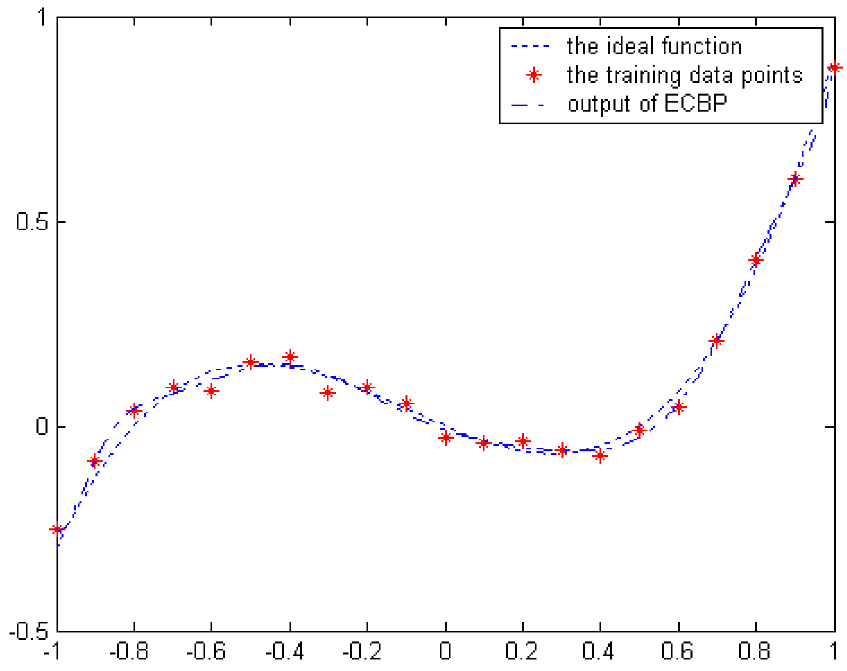
TABLE 4.1

The minimum number of hidden neurons needed for simulation 4.1

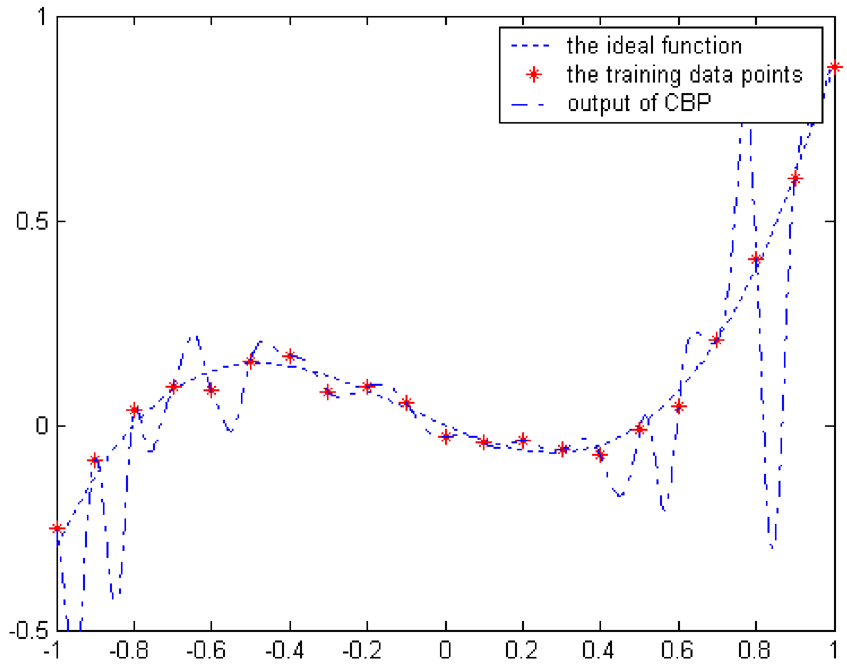
MLP	CBP	ECBP	RBFN 1	RBFN 2	S-RBFN
3	2	2	5	5	3

Comment 4.1: From Table 4.1, we can see that fewer number of hidden neurons is needed for supervised training. We also noted that for CBP and ECBP networks, a network with only 2 hidden neurons is sufficient for satisfactory approximation. That is maybe because of the additional input weights incorporated in the CBP and ECBP networks. The total number of free parameters for MLP with 3 hidden neurons is 10, and those for CBP and ECBP with 2 hidden neurons are already 9. The clustering method does not work very efficiently compared to S-RBFN which adjusts the centers, spreads and weights at the same time.

Simulation 4.2: The noisy approximation problem in simulation 3.2 is revisited here, and the result is shown in Figure 4.3. With the minimum structure, all the networks generalize well. But when the number of hidden neurons is increased to 50, the RBFNs (except for the clustering method, since the number of centers exceeds the number of training data) and the ECBP network still result in good generalization performance. But the CBP and MLP trained with Levenberg-Marquardt algorithm result in "overfitting".



(a) Approximation by ECBP



(b) Approximation by CBP

Fig. 4.3. Approximation with 50 hidden neurons

Comment 4.2: The difference of generalization performance between CBP and ECBP comes from the difference of the activation functions. The hyperbolic tangent function is more efficient than the exponential function. Hence, it is more prone to overfitting problem (Caruana et al. 2000).

Simulation 4.3: The more complicated one-dimensional example in simulation 3.3 is reconsidered here. The training and test sets remain the same. Table 4.2 gives the results of the minimum number of hidden neurons to reach the error bound 0.025.

TABLE 4.2

The minimum number of hidden neurons needed for simulation 4.3

MLP	CBP	ECBP	RBFN 1	RBFN 2	S-RBFN
9	6	6	41	41	Failed

Comment 4.3: Again, CBP and ECBP give the best approximation results considering the number of hidden neurons, even the total number of free parameters. The minimum number of the parameters of MLP is 28, those for CBP and ECBP is 25. All the RBFN networks do not give good approximation performance, which is related to the near linear part in the middle of the function to be approximated, since RBFN networks are very inefficient in approximating linear or constant function. The failure of S-RBFN maybe attributed to either slow convergence rate of the learning algorithm or local minima problem.

Simulation 4.4: Consider the two-dimensional example in Simulation 3.5 with the same training set and test set. The minimum numbers of hidden neurons to reach the error bound 0.15 are given in Table 4.3.

TABLE 4.3

The minimum number of hidden neurons needed for simulation 4.4

MLP	CBP	ECBP	RBFN 1	RBFN 2	S-RBFN
13	6	6	Failed	Failed	Failed

Comment 4.4: The total number of parameters of the minimum structure of CBP and ECBP is 25, and that for MLP is 40. So that CBP and ECBP is very efficient in terms of the number of parameters. Moreover, the CBP can even reach an error bound less than 0.04, which is hard even for EANNs (see Simulation 2.5 in Chapter 2). All of the RBFNs failed because of the severe “overfitting” problem for this particular case.

Simulation 4.5: We revisit the Gaussian hill and valley problem in Simulation 3.3. The training and test sets are still the same. The clustering method does not work well since the training data partition the input space uniformly. The target error bound set for approximation is 0.05, which is quiet strict. If the centers of the RBFN are chosen randomly from the training data, the number of the centers should be about 100 to achieve approximation goal. If the training is continued beyond the targeted error bound, after 5000 epochs, the approximation error bound can be reduced to about 5×10^{-5} for CBP networks and 1.5×10^{-10} for supervised training. For ECBP networks, as shown in Figure 4.4, the error bound can be even reduced to 4×10^{-15} after amazingly 63 epochs. Since the training data is uniformly distributed, the performance of RBFN-1 and RBFN-2 is much worse than S-RBFN.

TABLE 4.4

The minimum number of hidden neurons needed for simulation 4.5

MLP	CBP	ECBP	RBFN 1	RBFN 2	S-RBFN
30	2	2	93	92	2

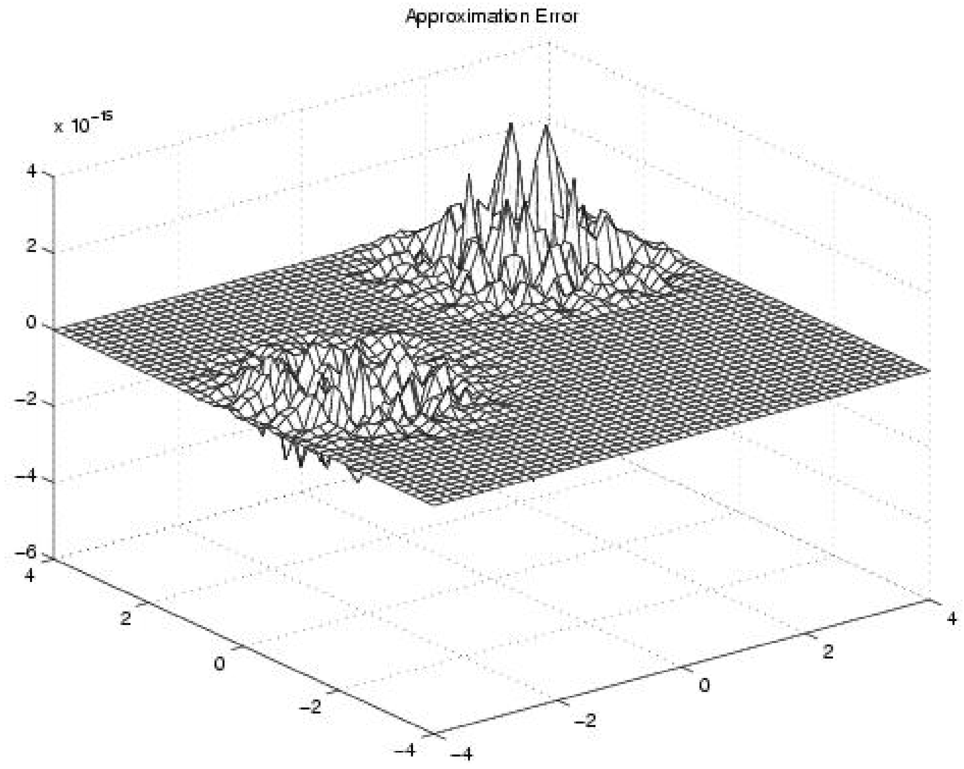


Fig. 4.4: The approximation error of ECBP network

<u>Comment 4.5</u>: If an additional hidden layer is added to the MLP, a simpler 2-4-2-1 network can be searched out to meet the requirement of error bound, which is still much more complex than the structures of CBP, ECBP and Supervise RBFN. The input weights of ECBP are [-4.0000 -4.0000 1.0000; 4.0000 0.0000 1.0000]. Since the centers of the equivalent RBFN network should be $c_i = -w_i / 2w_{d+1}$ and the

spread $g = w_{d+1}$, the centers are (2, 2) and (-2, 0) and the spreads are both 1, which is exactly the centers and spreads for the original Gaussian hill and valley. This fact inspires the idea partially trained ECBP network may be used to initialize the centers of the RBFNs. Since this kind of supervised training has an effect similar to the clustering procedure. Since this supervised training seems to be more efficient when the training data do not naturally appear in groups or clusters, we can expect this method of initializing RBFNs to be efficient in these cases. We can also expect that the total time of training maybe shorter than that of S-RBFN.

Simulation 4.6: we consider the classical two-spiral problem (Lang 1989; Fahlmann 1989), the training set is shown in Figure 4.5. Table 4.5 gives the minimum number of hidden neurons of those networks to achieve zero misclassification.

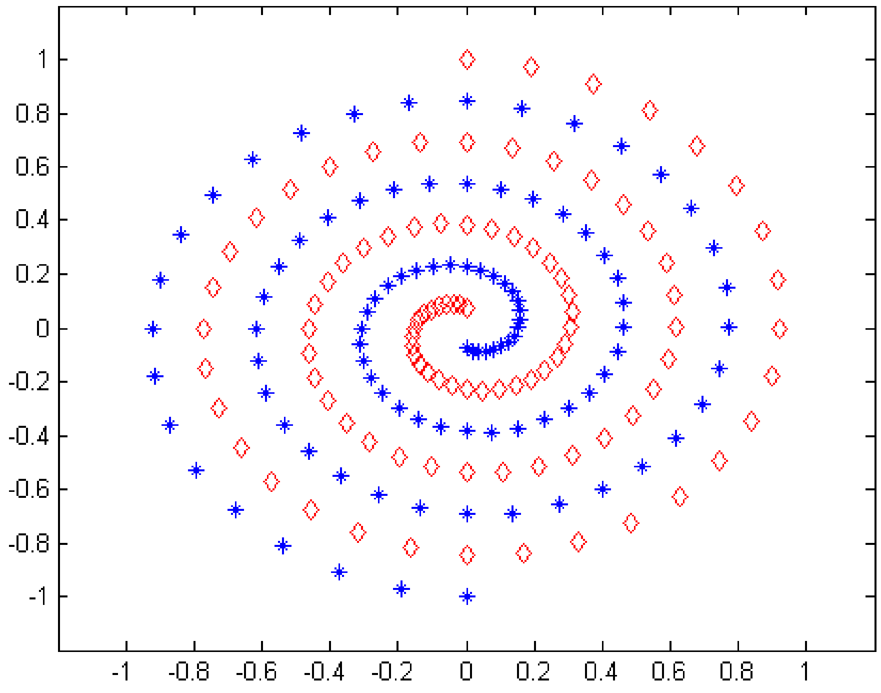


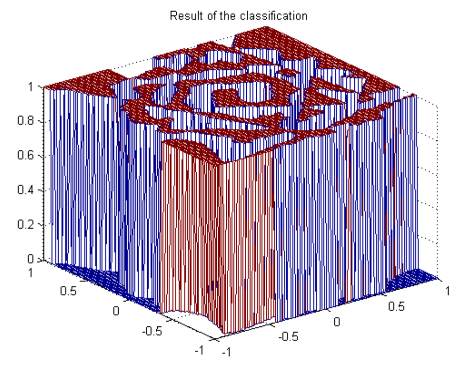
Fig. 4.5. The two-spiral problem

TABLE 4.5

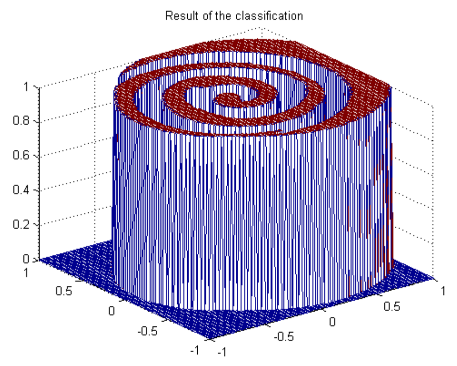
The minimum number of hidden neurons needed for simulation 4.6

MLP	CBP	ECBP	RBFN 1	RBFN 2	S-RBFN
29	8	42	130	98	Failed

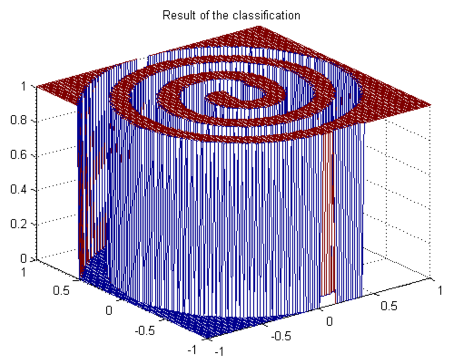
Comment 4.6: Ridella et al. (1997) mentioned that a RBFN with 42 hidden neurons can solve the two-spiral problem with considerable optimization efforts, but no details are given. In the above simulation studies, no specific optimization effort is added, so that most of RBFNs failed to give the correct classification. Theoretically, an RBFN with randomly selected centers can solve this problem with 97 (half of the total training points) hidden neurons, but it is almost impossible to select all these 97 centers in on class. Hence, about 130 centers are needed to correctly classify these two classes. In Figure 4.6, very interestingly, all the decision boundaries are smooth except that for MLP. That is because the decision boundary for a single neuron in MLP is linear and global, and the others are circular and localized.



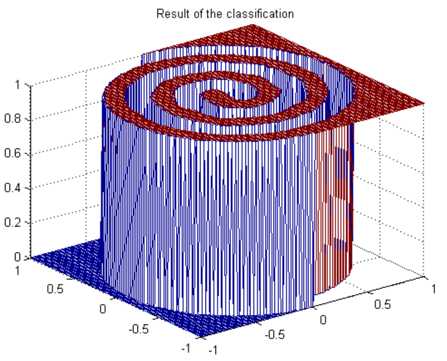
(a) MLP with 29 HN



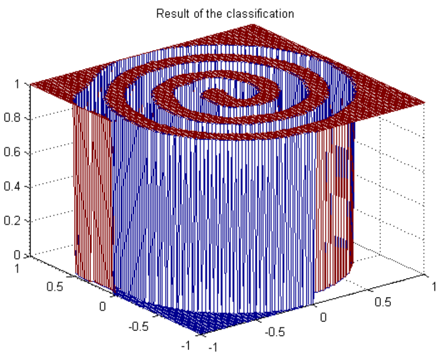
(b) CBP with 8 HN



(c) ECBP with 42 HN



(d) RBFN1 with 130 HN

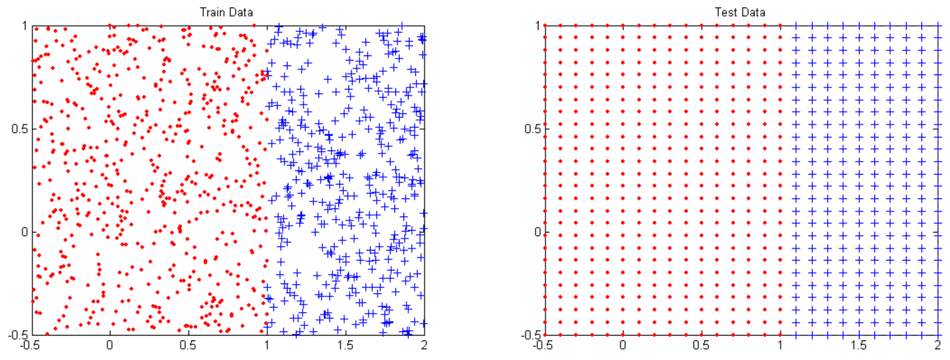


(e) RBFN-2 with 98 HN

Fig 4.6. Approximation results for the two-spiral problem

<u>Simulation 4.7</u>: Here, we go to a simple linear separation problem, where the line $x=1$ separates the input space to two classes. . The training data comprises 1000 points randomly selected from the input space. And the test data set consists 676 points, which are sampled by uniformly partitioning the input space by grid size 0.1×0.06.

The training and test data sets are shown in Figure 4.7. Table 4.6 shows the minimum number of hidden neurons needed for the different networks to achieve zero misclassification.



(a) Training data (b) Test data

Fig. 4.7. The training and test data of the simple linear separation problem

TABLE 4.6

The minimum number of hidden neurons needed for simulation 4.7

MLP	CBP	ECBP	RBFN 1	RBFN 2	S-RBFN
1	1	2	5	7	10

Comment 4.7: We can see that the MLP works best since the decision boundary of MLP is naturally linear which is very suitable for this problem. The CBP network also works very efficiently as it can represent a MLP exactly. However since the structure is more complex than the standard MLP, the performance of CBP network is not as good as that of MLP. Theoretically, the ECBP network can also be reduced to a MLP if the weights connecting to the additional input are set to zero. But the exponential transfer function is not as efficient as the hyperbolic tangent function and ECBP is more close to RBFNs, so that more hidden neurons are needed for ECBP. The RBFNs

are also able to solve this linear separation problem with zero error. However more hidden neurons are needed for their localize response. No advantage of the clustering method is observed, since the training data is randomly chosen from the input space.

Simulation 4.8: This is a two dimensional classification problem of two classes of Gaussian distributions C_1 and C_2. The conditional probability density functions for the two classes are:

$$f(X \mid C_1) = \frac{1}{2\pi\sigma_1^2}\exp(-\frac{1}{2\sigma_1^2}\|X-\mu_1\|^2) \tag{4.17}$$

$$f(X \mid C_2) = \frac{1}{2\pi\sigma_2^2}\exp(-\frac{1}{2\sigma_2^2}\|X-\mu_2\|^2) \tag{4.18}$$

Where the mean vectors μ_1, μ_2 are [0;0] and [2;2], the variance for the two classes σ_1^2 and σ_2^2 are 1 and 4 respectively. The training data set consists 500 points from each distribution as shown in Figure 4.8. The classification results are shown in Figure 4.9.

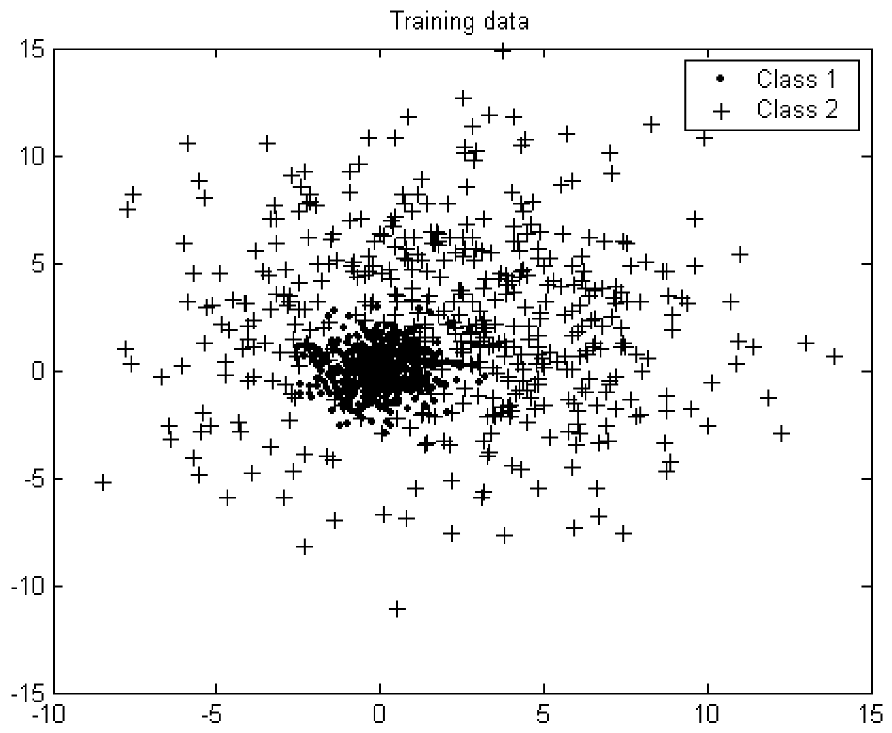


Fig. 4.8. The classification problem of two Gaussian distribution

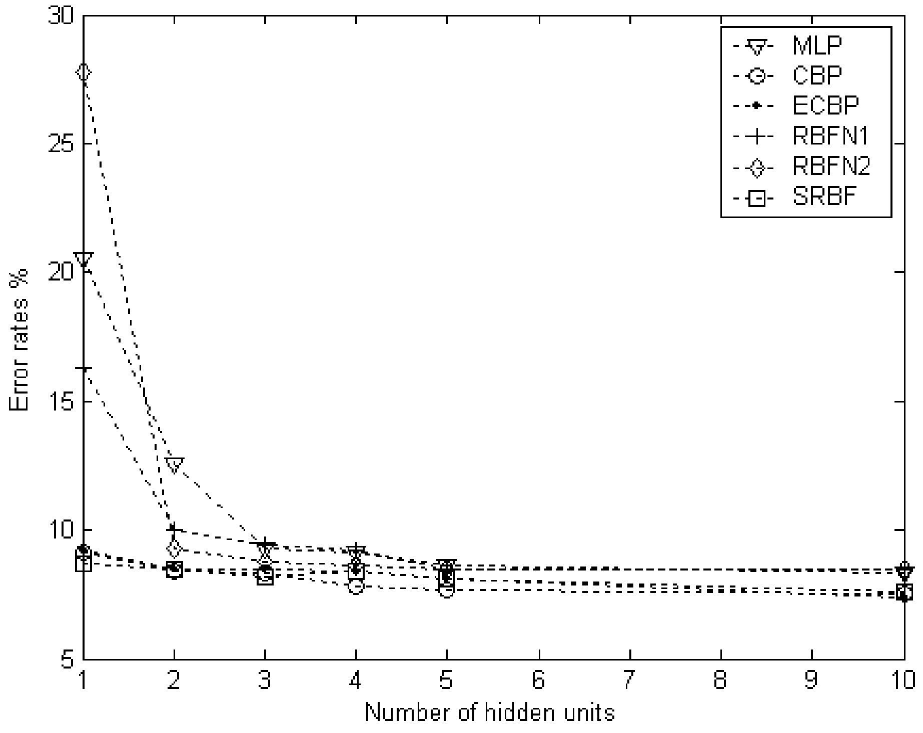


Fig. 4.9. The classification results for the two Gaussian distributions

Comment 4.8: From Figure 4.9 we can see both the CBP and ECBP networks outperform MLP and RBFN networks significantly. Supervised one-stage trained RBFN network also has better performance than the other algorithm in terms of classification accuracy. Again, the clustering algorithm does not perform better than other algorithms even comparing to the simple random selection centers method.

4.4. Conclusions

A brief introduction of radial basis function networks is given in this chapter. The various two-stage training methods of radial basis function networks, as well as the one-stage supervised training, are investigated. The difference between the MLP and RBFNs is also discussed.

MLPs with additional second order inputs, which can produce localize response fields, are introduced, especially the circular backpropagation (CBP) network by by Ridella et al. (1997). The CBP network can be reduced to a standard MLP if the weights connecting to the additional input are set to zero. The CBP also can approximate a RBFN arbitrarily.

After a careful study of the CBP network, a modified CBP network (ECBP) is proposed. The ECBP network replaced the sigmoidal activation function in the hidden layer with exponential function $\varphi(h) = e^{-h}$. Then the ECBP network becomes naturally a generalized version of RBFNs, which can represent a RBFN easily. Since the CBP and ECBP are actually special cases of multilayer perceptron networks, the fast training strategies such as Levenberg-Marquardt algorithm (Marquardt 1963; Mor 1977) of MLP can be easily implemented in these networks.

Because the natural connection between the ECBP and RBFN networks, we also can see the possibility of initializing RBFN networks with partially trained ECBP networks, as an alternative solution to determine the centers. The ECBP network is more prone to local minima problem compared to the CBP networks, since the exponential activation function is unbounded and not as efficient as the sigmoidal functions.

Comparative studies have been conducted between the MLP, CBP, ECBP and RBFN networks. In the comparative studies, the CBP and ECBP outperform the MLP and RBFNs for most of the cases of the approximation and classification problems. It is interesting to note that the clustering method of RBFN networks do not work efficiently as expected comparing to the simple method of random selection of the centers, especially when there are no obvious groups or clusters in the training data set. It appears that there is really no need to do any clustering for RBFN and simply using random selection, or replacing RBFN with CBP or ECBP would achieve better performance. Although there do exist some examples in the literature (Chen et al. 1993, Su and Chou 2001), where the clustering methods have very satisfactory performances, it remains to be seen whether it is indeed superior in terms of accuracy to other approaches for these problems since no comparative studies were reported.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

In this thesis, the structure selection and analysis problem of neural networks is carefully examined, especially for those feedforward neural networks such as multilayer perceptron and radial basis function networks.

First of all, a geometrical interpretation of MLPs is suggested based on the special geometrical shape of the activation function. The basic building blocks provided by the hidden layer of the three-layered MLP are very close to the piecewise lines (or piecewise hyperplanes in high dimensional cases). We can set the widths, heights and positions of these building blocks arbitrarily by adjusting the weights and biases. The four-layered MLP is interpreted simply as linear combination of multiple three-layered MLPs that have 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 that construct corresponding sub-functions that would combine into an approximation of the target function.

Three guidelines for selecting the architecture of the multilayer perceptron are proposed based on this geometrical interpretation. It is demonstrated by numerous simulation studies that these guidelines are very effective for searching of the minimal structure of the MLP, which is very critical 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 also useful to explain 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 2.1. At the same time, this geometrical interpretation may provide a light to illuminate the advantage of MLP over other conventional linear regression methods, shown by Barron (1992; 1993), 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.

Over-fitting problem 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, based upon the geometrical interpretation of MLP, 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. 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.

The structure analysis problem is unsurprisingly extended to the radial basis function networks, because of the similarity to the multilayer perceptron networks. The various two-stage training methods of radial basis function networks, as well as the one-stage supervised training, are investigated. The difference between the MLP and radial basis function networks is also discussed. Then the connection between MLP and RBF network is visited. The MLP with additional special second order inputs can produce localize response fields (Casesant 1992; Ridella 1997). The recent result is given by

Ridella etc. for their circular backpropagation (CBP) network. The CBP network can be reduced to a standard MLP if the weights connecting to the additional input are set to zero, and also can approximate a radial basis function network arbitrarily. After a careful study of the CBP network, a modified CBP network (ECBP) is proposed. The ECBP network replaced the sigmoidal activation function in the hidden layer with exponential function $\varphi(h) = e^{-h}$. Then the ECBP network becomes naturally a generalized version of radial basis function networks, which can represent a radial basis function network easily, as shown in equation (4.16). Since the CBP and ECBP are actually special cases of multilayer perceptron networks, the mature training strategies such as Levenberg-Marquardt algorithm (Marquardt 1963, Mor 1977) of MLP can be easily implemented in these networks.

After that, comparative studies are conducted between the MLP, CBP, ECBP and RBF networks. In the comparative studies, the CBP and ECBP outperform the MLP and RBF networks in the most cases of the approximation and classification problems in sense the size of the structure. And the clustering methods of RBF networks do not work efficiently as expected considering that much more effort needed comparing to the simple method of random selection of the centers, especially when there are no obvious groups or clusters in the training data set.

5.2 Future works

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

The ECBP network is naturally a radial basis function network when the weights connecting to the additional input w_{d+1} are positive. In this case, the parameters of the ECBP network are directly connected to a radial basis function network. So that we also can see the possibility of initializing RBF networks with partially trained ECBP networks. But the efficiency of this novel two-stage training algorithm of RBF networks is still to be proved. Another relative problem to solve is that: how the ECBP network works during the training procedure when the weights w_{d+1} are constrained to positive?

Since the CBP and ECBP networks are actually still multilayer perceptron networks, they may have additional hidden layers other than the number of hidden layers is normally constrained to one in those RBF networks. An additional hidden layer maybe helpful in the standard MLPs, as shown in Chapter 2, but the influence of this additional hidden layer to CBP and ECBP networks are still to be examined.

The advantage of CBP and ECBP network in classification problems is mainly because their sophisticated decision boundaries (either global or local) comparing to the standard MLPs and RBF networks. The activation function in the hidden layer has a great influence on the decision boundary. Details can be found in survey (Duch et al. 1999) on neural transfer functions. The CBP and ECBP networks are actually mixtures of the inner product and distance based activation functions. Dorffner (1994) proposed conic section activation functions as a unified framework for MLP and RBF networks, where straight lines, circles and ellipses are special cases of conic sections. And these conic section activation functions are also combinations of inner product and distance based activation functions. This approach gives more freedom for the decision boundaries in the classification problems. However, it also leads to more complex nonlinear optimization problems that are more prone to local minima. Hence, the evolutionary algorithms may be considered to solve these nonlinear optimization problems in the training procedures.

References

Alpaydim, E., 1994, "GAL: Networks that grow when they learn and shrink when they forget," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 8, no. 1, pp. 391-414.

Amri, S., 1998, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251-276.

Barron, A.R., 1992, "Neural net approximation," in *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 69-72, New Haven, CT.: Yale University.

Barron, A.R., 1993, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, pp. 930-945.

Bartlett, P. L., 1997, "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, The MIT Press.

Bishop, C., 1995, "Neural Networks for Pattern Recognition,", *Oxford University Press.*

Caruana, R., Lawrence, S., and Giles, C. L., 2000, “Overfitting in Neural Networks: Backpropagation, Conjugate Gradient, and Early Stopping,” *Neural Information Processing Systems*, Denver, Colorado, Nov. 28-30, 2000.

Casasent, D. and E. Barnard, 1992, Adaptive clustering neural net for picewise nonlinear discriminant surfaces, *International Joint Conference on Neural Networks*, I, pp. 423-427.

Castillo, P.A., J. Carpio, J. J. Merelo, V. Rivas, G. Romero and A. Prieto, 2000, “Evolving Multilayer Perceptrons,” *Neural Processing Letters,* vol. 12, no. 2, pp. 115-127.

Chen, S., B. Mulgrew, and P. M. Grant, 1993, “A clustering technique for digital communications channel equalization using radial basis function networks”, *IEEE Transactions on Neural Networks*, Vol. 4, no. 4, pp. 570-579.

Chester, D.L., 1990, “Why two hidden layers are better than one,” *International Joint Conference on Neural Networks (IJCNN-90)*, Lawrence Erlbaum, vol. 1, pp. 265-268.

Cybenko, G., 1989, “Approximation by superpositions of sigmoidal function,” *Math., Control, Signals, and Systems*, vol. 2, pp. 303-314, 1989.

Duda, R. O., and P. E. Hart, 1973, *Pattern Classification and Scene Analysis*, New York: Wiley.

Fahlmmann, S. E., and C. Lebiere,1989, “The cascade-correlation learning architecture”, in *Advances in Neural Information Processing Systems II*, San Mateo, CA: Morgan Kaufmann, pp. 524-532.

Funahashi, K., 1989, “On the approximate realization of continuous mappings by neural networks,” *Neural Networks*, vol. 2, pp. 183-192.

Ghosh, J. and Nag, A, 2000, “An Overview of Radial Basis Function Networks,” *Radial Basis Function Neural Network Theory and Applications*, R. J. Howlerr and L. C. Jain (Eds), Physica-Verlag.

Hassibi, B., D. G. Stork and G. J. Wolff, 1992, “Optimal brain surgeon and general network pruning,” in *Proceedings of IEEE International Conference on Neural Networks*, vol. 1, pp. 293-299, San Francisco.

Haykin, S., 1999, “Neural networks: a comprehensive foundation,” *Prentice Hall*, New Jersey, second edition.

Hornik, K., M. Stinchcombe and H. White, 1989, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, pp. 359-366.

Hush, D. R. and J. M. Salas, 1988, “Improving the learning rate of back-propagation with the gradient reuse algorithm,” in *Proceedings of IEEE International Conference on Neural Networks*, vol. 1, pp. 441-447, San Diego, CA.

Hush, D. R. 1997, “Learning from examples: From theory to practice,” Tutorial #4, *1997 International conference on Neural Networks*, Houston, June, 1997.

Jasic, T. and H. Poh, 1995, “Artificial and real world mapping problems,” *Lecture Notes in Computer Science,* vol. 930, pp. 239-245, Springer-Verlag.

Lang, K., and M. Witbrock, 1989, “Learning to tell two spirals apart”, in *Proceedings of Connectionist Models Summer School*, pages 52--59. Morgan Kaufmann.

LeCun, Y., J. S. Denker and S. A. Solla, 1990, “Optimal brain damage,” *Advances in Neural Information Processing Systems*, vol. 2, pp. 598-605, San Mateo, CA: Morgan Kaufmann.

Lowe, D., 1989, “Adaptive radial basis function nonlinearities, and the problem for generalization,” *First IEE International Conference on Artificial Neural Networks*, pp.171-175, London.

MacKay, D.J.C., 1992a, “Bayesian Interpolation,” *Neural Computation*. Vol.4, pp. 415-447.

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

Marquardt, D. W., 1963, “Nonlinear modeling,” *Journal of the Society for Industrial and Applied Mathematics,* vol. 11, pp. 431-441.

Maruyama, M., F. Girosi, and T. Poggio, 1992, “A connection between GRBF and MLP”, *A.I. Memo*, No. 1291, MIT Artificial Intelligence Lab.

McCulloch, W.S. and W. Pitts, 1943, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol.5, pp.115-133.

Mennon, A., K. Mehrotra, C. K. Mohan and S. RanKa, 1996, “Characterization of a class of sigmoid functions with applications to neural networks,” *Neural Networks*, vol. 9, pp. 819-835.

Moody, J. and C. Darken, 1988, “Learning with localized receptive fields,” In Touretzky et al. editors, *Proceedings of the 1988 Connectionist Models Summer School*, San Mateo, CA, Morgan Kaufmann.

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

Mor, J.J., 1977, “The Levenberg-Marquardt Algorithm: implementation and theory, in numerical analysis,” G. A. Watson, ed., *Lecture Notes in Mathematics*, vol. 630, Springer-Verlag, Berlin, pp. 105-116.

Nguyen, D. and B. Widrow, 1990, “Improving the learning speed of 2-layer neural network by choosing initial values of the adaptive weights,” in *Proceedings of International Joint Conference on Neural Networks (IJCNN-90),* vol. 3, pp. 21-26.

Omohundro, S. M., 1989, “Geometric learning algorithms,” *Technical Report 89-041*, *International Computer Science Institute*, Berkeley, CA.

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

Poggio, T., and F. Girosi, 1990, “Networks for approximation and learning”, *Proceedings of the IEEE*, Vol. 78, pp 1481-1497.

Ridella, S., Rovetta, S. and Zunino, R., 1997, “Circular backpropagation networks for classification”, *IEEE Transactions on Neural Networks,* vol.8, no. 1, pp. 84-97.

Riessen, G.A., Williams, G.J. and Yao, X., 1997, “PEPNet: parallel evolutionary programming for constructing Artificial Neural Networks,” *Proceedings of the 6th International Conference, Evolutionary Programming*, pp. 35-45.

Rosenblatt, F., 1959, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review* 65, pp. 386-408.

Sarle, W.S., 1995, “Stopped Training and Other Remedies for Overfitting,” *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, pp. 352-360.

Sarle, W., 2002 “How many hidden layers should I use?,” *Newsgroup: comp.ai.neural-nets FAQ*, Available at: ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hl.

Sarajedini, A., and Hecht-Nielsen, R., 1992, “The best of both worlds: Casasent networks integrate multilayer perceptrons and radial basis functions”, *International Joint Conference on Neural Networks*, III, pp. 905-910.

Sarkar, M. and B. Yegnanarayana, 1997, “Evolutionary programming-based probabilistic neural networks construction technique,” in *Proceedings of International Joint Conference on Neural Networks (IJCNN-97)*, *Part 1*, pp. 456-461.

Solis, F. and Wets, R.-B., 1981, “Minimization by random search techniques,” *Mathematics of Operation Research*, vol. 6, no. 1, pp. 19-30.

Sontag, E.D., 1992, “Feedback stabilization using two-hidden-layer nets,” *IEEE Transactions on Neural Networks*, vol. 3, pp. 981-990.

Su, M.C., and C.H.Chou, 2001, “A modified version of the k-means algorithm with a distance based on cluster symmetry,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 23, No. 6, pp. 674-680.

Tamura, S. and Tateishi, M., 1997, “Capabilities of a four-layered feed-forward Neural Network: four layers versus three,” *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 251-255.

Tarassenko, L. and S. Roberts, 1994, “Supervised and unsupervised learning in radial basis function classifiers”, IEE Proceedings: Visual, Image and Signal Processing, vol. 141, pp. 210-216.

Villiers, J.D. and Barnard, E., 1992, “Backpropagation neural nets with one and two hidden layers,” *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 136-141.

Weigend, A.S., D. E. Rumelhart and B. A. Huberman, 1991, “Generalization by weight-elimination with application to forecasting,” *Advances in Neural Information Processing Systems*, vol. 3, pp. 875-882, San Mateo, CA: Morgan Kaufmann.

Wettschereck, D., and T. Dietterich, 1992, “Improving the performance of radial basis function networks by learning center locations,” *Advances in Neural Information Processing Systems,* vol. 4, pp.1133-1140, San Mateo, CA: Morgan Kaufmanm.

Wilamowski, B. M. and Jaeger, R. C., 1996, “Implementation of RBF type networks by MLP networks,” *Proceedings of the IEEE International Conference on Neural Networks (ICNN96),* pp. 1670-1675.

Wilamowski, B. M. and Jaeger, R. C., 1997, “Implementation of RBF type networks by sigmoidal feedforward neural networks,” *Intelligent Engineering Systems Through Artificial Neural Networks* vol. 7, ed. C. H. Dagli and others, New York, pp. 183-188.

Wilensky, G. and N. Manukian, 1992, The projection neural networks, International Joint Conference on Neural Networks, II, 358-367.

Yao, X. and Liu, Y., 1997, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks,* vol. 8, pp. 694-713.

Yao, X., 1999, "Evolutionary artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423-1447.

List of Publications

The author has contributed to the following publications:

C. Xiang, S.Q. Ding and T.H. Lee, "Geometrical Interpretation and Architecture Selection of MLP", *IEEE Transactions on Neural Networks* (To appear as regular paper).

S.Q. Ding and C. Xiang, "Overfitting problem: a new perspective from the geometrical interpretation of MLP", in *Proceedings of the Third International Conference on Hybrid Intelligent Systems*, Melbourne, Australia, 14-17 December, 2003. A. Abraham ed. al., *Design and Application of Hybrid Intelligent Systems*, IOS Press, pp. 50-57, 2003.

C. Xiang, S.Q. Ding and T.H. Lee, "Geometrical Interpretation of MLP with Application to Architecture Selection", *the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Pan Pacific, Singapore 15-18 December, 2003.

S.Q. Ding and C. Xiang, "From Multilayer Perceptrons to Radial Basis Function Networks: A Comparative Study", *2004 IEEE Conference on Cybernetics and Intelligent Systems* (Accepted).