APPROXIMATION OF A FRACTAL CURVE USING FEED-FORWARD NEURAL NETWORKS

HO WAI SHING

MPHIL

THE HONG KONG POLYTECHNIC UNIVERSITY

2000

Abstract

The approximation of fractal curves in the form of Brownian functions by two-layer feed-forward neural networks is studied. The network parameters are restricted within a finite range. For given realizations of the Brownian target function, all local minima in the output error measure with appreciable sizes of basins of attraction are located and found to be about a dozen in number in each case. The error follows a log-normal distribution which can be explained by a distribution of mean squared normal deviates. Its mean value exhibits simple scaling relationships with the number of hidden neurons and the number of training patterns. Our numerical findings are explained by comparison with a simple piecewise linear fit approach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Neural network is an inter-disciplinary field of research. Its study involves knowledge from neurobiology, cognitive science, computer science, statistical physics, etc. We will first give a brief overview of this subject. The motivation and the scope of the current work will then be introduced.

1.1 Biological neural networks

The human nervous system may be the most sophisticated structure in the world. It contains over 10^{11} neurons [1]. A neuron consists of a cell body, dendrites and an axon. The dendrites have branch structures which receive inputs from other cells. The axon carries the neuron's output to the dendrites of other neurons. The junction between an axon and a dendrite is called a synapse.

The connection among neurons is very extensive. Each neuron is connected to 1000 others. There are therefore more that 10^{14} synapses in the human brain. Neurons communicate among themselves through electrical signals. A neuron collects signals from its dendrites and sums up all influences which can either be excitatory or inhibitory. If the signal exceeds a certain threshold, the neuron then fires an output signal through the axon.

The brain is thus a massively parallel computational machine. Although its switch time is only a few milliseconds compared with nanoseconds for modern computers, its performance is definitively superior in certain areas such as pattern or speech recognition.

1.2 Artificial neural networks

An artificial neural network is an abstract simulation of a real nervous system. The first model was proposed by McCulloch and Pitts in 1943 in which a neuron was modeled as a binary device with a fixed threshold [2]. Subsequently, Hebb proposed self-organization in the network connections in a way that the signals themselves could strengthen the synaptic connections carrying them. This forms the foundation of the understanding of learning in nervous systems [1].

Further development of artificial neural networks may be classified into two approaches in general. One is a more biological approach. It aims at understanding of the functions of real nervous systems. There are numerous interesting and very active areas including anatomy and functional structure of the nervous system, signal transmission between neurons, cognition, attention, etc.

Another approach is more computation orientated. Conventional digital computers follow the Von Neumann paradigm. Computations follow very clear and explicit logics. This proves to be an extremely successful paradigm. Nevertheless, there are still certain drawbacks. Von Neumann computers may be difficult to program for complicated applications. Parallelization is often difficult. Fault tolerance in many case may not be satisfactory and a simple error in the program may be able to bring down the system completely.

As a result, neural network can be a useful alternative computational paradigm. Their construction and operation do not necessarily follow that of real neural networks but are engineered in order to perform specific applications. Most often, practical artificial neural networks are simulated on serial or parallel Von Neumann computers. Special hardware implementation is expensive and can only be found when computational speed is crucial.

There are many different types of neural network architecture. More important ones include the Hopfield network, perceptrons, multi-layer feed-forward networks, recurrent networks, Boltzmann machine, etc [3].

In neural network approximation, too many weights lead a curve fitted to follow all the small details or noise but is very poor for interpolation and extrapolation. The output function of the network has often fitted the data by developing some dramatic oscillations. Such function is said to be over-fitted to the data and gives a

poor representation of target function. The simple way to determine over-fitting is to divide the data into two parts. The training set consists mostly of data and the rest is testing set. If over-fitting occurs, the minimum root mean square error with respect to the training set decreases monotonically with weight increases, while the error of testing set is not. In our work, the degrees of freedom of networks are very smaller than the number of training pairs. The curves of target functions and output functions are also examined such that no over-fitting occurs.

1.3 Scope of thesis

Fundamental properties of neural networks of a certain architecture can often be studied by examining the learning of simple and well quantified tasks. Random uncorrelated patterns, for example, are convenient and widely used choices. In this work, we conduct a systematic study on approximation of a sophisticated yet well understood class of functions by the widely used two-layer feed-forward neural networks. The functions used are fractals obeying simple scaling properties. It turns out that the statistical properties of learning also inherit certain scaling behavior which can be understood easily.

In Chapter 2, we introduce the fractal function to be approximated, details of the neural network architecture and the training method. Chapter 3 presents all numerical measurements, focusing on the statistical properties of error measures of the approximations and in particular scaling behaviors. Chapter 4 examines analytically a simple curve fitting approach which explains most numerical results in Chapter 3. We conclude in Chapter 5 with some further discussions.

Chapter 2

Problem definition

2.1 Brownian functions as target functions

The fractal functions we choose to study in this work as the target functions are Brownian functions. They are self-affine fractals [4]. Denote the target function by $T(x)$. The input x is limited to $-l_x \leq x \leq l_x$. The Brownian functions we consider are displacement-time profiles of Brownian particles in one dimension. The input x is analogous to time while $T(x)$ is analogous to displacement. We assume that the particle starts at the origin and thus $T(-l_x) = 0$. Consider a set of N equally spaced discrete points at

$$x_\mu = -l_x + \mu\Delta x \tag{2.1}$$

where $\Delta x = 2l_x/N$ and $\mu = 1,2, \ldots N$. The function is defined recursively at these points by

$$T(x_\mu) = T(x_{\mu-1}) + \delta_\mu \tag{2.2}$$

The displacement δ_μ is assumed to be an independent normal deviate with mean 0 and standard deviation

$$\sigma_\delta = \sqrt{\frac{1}{N}} \tag{2.3}$$

The standard deviation $T(x_\mu)$ is thus

$$\sigma_T(x_\mu) = \sqrt{\frac{\mu}{N}} \tag{2.4}$$

In particular, $\sigma_T(l_x) = 1$. The value $T(l_x)$ at the other end point l_x, corresponding to the total displacement, thus also follows a normal distribution with unit standard deviation.

In principle, we obtain a Brownian function only when $N \to \infty$. At finite N, $T(x)$ is a pre-fractal function defined at discrete points at x_μ. Figure 2.1 shows an example of a target function generated using the above approach for $N = 125$.

2.2 Architecture and bounded weight space

This work focuses on two-layer feed-forward neural networks [3, 5]. They are widely used since they are relatively simple and can be trained for a large variety of problems. It has been proved that a two-layer feed-forward network is a universal function approximator [6, 7]. Given sufficient number of neurons, it can approximate any continuous function up to arbitrary accuracy [8, 9].

Since we are approximating a one to one function $T(x)$, the network we consider has one input and one output. The architecture is shown in Fig. 2.2. The circles denote the neurons and the lines represent the synaptic connections. Assume that there are n neurons in the lower layer, called the hidden layer. The ith hidden neuron receives the input value x and produces the activation

$$V_i(x) = \tanh(w_i^{(1)} x + \theta_i^{(1)}) \tag{2.5}$$

for $i = 1, 2..n$. They are then passed to the output neuron which generates the overall network output

$$O(x) = \sum_{i=1}^{n} w_i^{(2)} V_i(x) + \theta^{(2)} \tag{2.6}$$

The network parameters $w_i^{(1)}$ and $w_i^{(2)}$ are the weights of the synaptic connections and $\theta_i^{(1)}$ and $\theta^{(2)}$ are biases. Biases can also be regarded as weights of connections to inputs fixed at unity. The output neuron acts on the activation of the hidden neurons linearly. In contrast, the transfer function of the hidden neurons is the hyperbolic tangent function and leads to the overall nonlinearity of the network [10].

The state of the network is determined by the network parameters which can be expressed conveniently using a weight vector

$$\vec{W} = (w_i^{(1)}, \theta_i^{(1)}, w_i^{(2)}, \theta^{(2)}) \tag{2.7}$$

The number of parameters, which equals the degree of freedom of the network, is

$$n_f = 3n + 1 \tag{2.8}$$

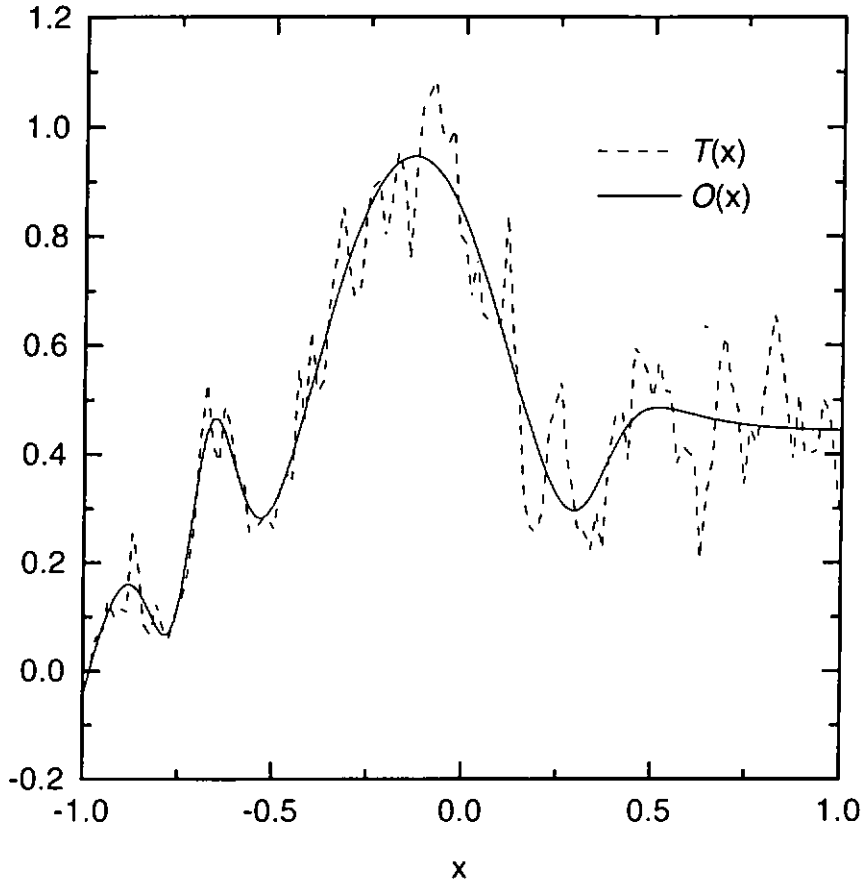

Figure 2.1: An example of a target function $T(x)$ in the form of a Brownian function with $N = 125$ sample points (dotted line). Also shown is the output function $O(x)$ of a trained neural network with $n = 5$ hidden nodes (solid line).

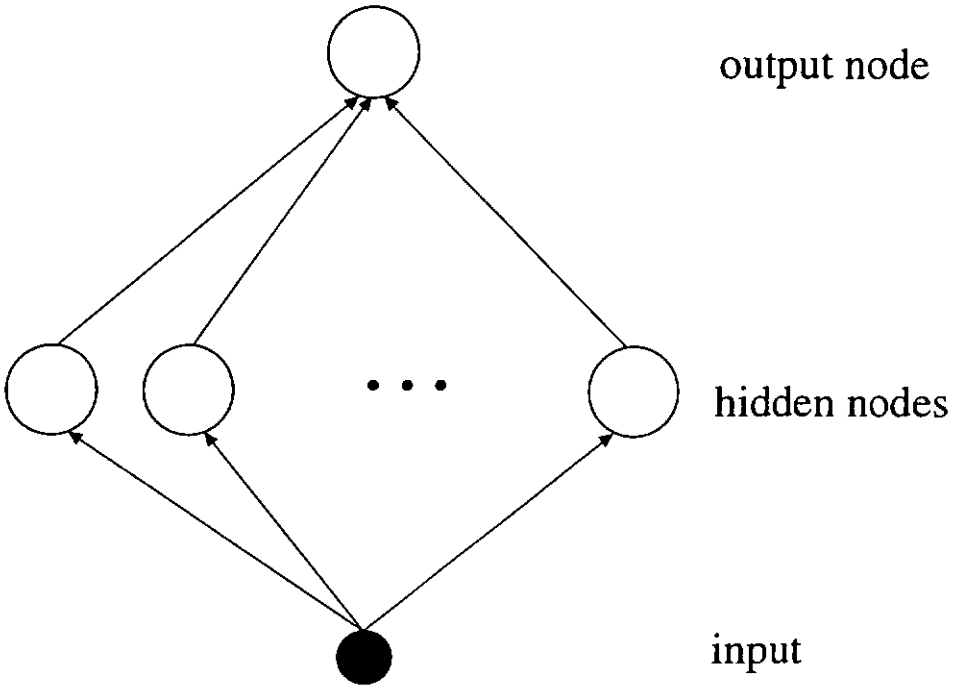

Figure 2.2: Architecture of a two-layer feed-forward neural network with one input, n hidden nodes and one output node.

The vector $\vec{W}$ is thus defined in an n_f-dimensional weight space.

In general, each network parameter can take any real value. However, as will be explained later, extreme values lead to problematic truncation errors in the numerical calculations and prohibit accurate computations. They are in fact also irrelevant for most practical applications. Therefore, we restrict all parameters to the range $[-L_W, L_W]$ and we put $L_W = 10$. The weight vectors we consider are all inside a bounded region in the weight space defined as

$$\mathcal{W}_B = \{\vec{W} : \mid W_i \mid \leq L_W\} \tag{2.9}$$

where W_i denotes the ith component of $\vec{W}$.

2.3 Training using conjugate gradient method

An error measure for the approximation of the target function by a neural network is defined as

$$E = \frac{1}{N} \sum_{\mu=1}^{N} [O(x_\mu) - T(x_\mu)]^2 \tag{2.10}$$

A network is trained by minimizing E with respect to $\vec{W}$. The most popular method for training a multi-layer feed-forward network is back-propagation, which is essentially a steepest decent method. It is well-known that landscape of E is complicated. We have found that training with back-propagation practically always ends up at nearly flat plateau instead of local minima. A more accurate conjugate gradient method for training is therefore used. We adopt basically the Polak and Ribiere approach of conjugate gradient method [11]. There are however slight modifications to be explained below since we consider network parameters constrained in a bounded region.

The network is first initialized with random weights and biases before training commences. It is conventional to choose parameters in the range [-1,1] with uniform probability. The initial parameters are thus described by a random vector in a region $\mathcal{W}_I$ defined by

$$\mathcal{W}_I = \{\vec{W} : \mid W_i \mid \leq 1\} \tag{2.11}$$

The conjugate gradient method involves successive line minimizations along conjugate directions. Assume that the jth line minimization session starts from $\vec{W}_j$ and is

carried out along direction $\vec{h_j}$ in the weight space. An initial interval $(\vec{a},\vec{b})$ is chosen as

$$\vec{a} = \vec{W}_j \tag{2.12}$$

and

$$\vec{b} = \vec{a} + \frac{\vec{h_j}}{|\vec{h_j}|} \tag{2.13}$$

The golden section search is applied along the downhill direction of $\vec{h_j}$ until a new triplet interval $(\vec{a},\vec{b},\vec{c})$ is found where the error measure at $\vec{b}$ is the least of three points.

The minimium point is obtained along $\vec{h_j}$ by using Brent's method which applied parabolic interpolation. The formula below is used to find the minimum of a parabola through three points $E(\vec{a})$, $E(\vec{b})$ and $E(\vec{c})$.

$$\vec{d} = \vec{b} - \frac{1}{2}\frac{(\vec{b}-\vec{a})^2[E(\vec{b})-E(\vec{c})] - (\vec{b}-\vec{c})^2[E(\vec{b})-E(\vec{a})]}{(\vec{b}-\vec{a})[E(\vec{b})-E(\vec{c})] - (\vec{b}-\vec{c})[E(\vec{b})-E(\vec{a})]} \tag{2.14}$$

If $\vec{d}$ is between $\vec{a}$ and $\vec{b}$, the point $\vec{b}$ replaces $\vec{c}$ and $\vec{d}$ replaces $\vec{b}$, otherwise; the point $\vec{b}$ replaces $\vec{a}$ and $\vec{d}$ replaces $\vec{b}$. Convergence in the line search is assumed when the fractional change in E from successive steps in Brent's method is less than 10^{-10}. However, there are further constraints in the line search. First, the parameter vector has to be kept inside the bounded weight space $\mathcal{W}_B$. Furthermore, we restrict the size of the displacement by enforcing

$$|\vec{W}_{j+1} - \vec{W}_j| \leq d \tag{2.15}$$

and we put $d = 0.01$. This is because we want to have a well-defined flow pattern in the minimization process to study quantities such as sizes of basins of attraction. We have found that further reducing d does not alter our results within our numerical errors.

Following Ref. [11], the conjugate direction $\vec{h}_j$ along which the line search is carried out follows

$$\vec{h}_{j+1} = \vec{g}_{j+1} + \frac{(\vec{g}_{j+1} - \vec{g}_j)\cdot\vec{g}_{j+1}}{\vec{g}_j \cdot \vec{g}_j}\vec{h}_j \tag{2.16}$$

where

$$\vec{g}_j = -\nabla E(\vec{W}_j) \tag{2.17}$$

If $\vec{W}_{j+1}$ is at the boundary of $\mathcal{W}_B$ and the downhill direction of $\vec{h}_{j+1}$ points outside $\mathcal{W}_B$, $\vec{h}_{j+1}$ is replaced by an appropriate projection on the boundary so that minimization can be constrained to $\mathcal{W}_B$. A candidate for a minimum is found when the fractional change in E between successive line search sessions is less than 10^{-18}.

The resulting candidate is not always a minimum but can often be a saddle point. This is at first sight surprising. The slightest perturbation inherited from the random initial conditions should have allowed the network to flow pass an unstable point. The problem lies with the existence of valleys in the error landscape aligned along a high symmetry axis, such as one in which two weights are equal. During flow along the valley, $\vec{W}_j$ attains the symmetry exactly within our machine precision. It thus loses the tiny perturbation needed to escape from a saddle point at the bottom of the valley.

To overcome the problem, we perturb the candidate for the minimum. Specifically, we reposition $\vec{W}_j$ randomly in a small hypercube with side 0.001 centered at $\vec{W}_j$. That means every weight and bias is perturbed by an independent random number with absolute value smaller than 0.0005. Care is taken to bring $\vec{W}_j$ back to the bounded region $\mathcal{W}_B$ if it crosses the boundary. The same approach of conjugate gradient search is then conducted again. Upon the next convergence at the same tolerance of 10^{-18}, we declare location of a minimum if the network configuration comes back to the same candidate. Otherwise, search continues. The minimum can either be a local or the global one.

We have found that there are abundances of valleys with near flat bottoms which render location of minima very challenging. As a result, the exceptionally small tolerance 10^{-18} is essential. Machine precision thus becomes a concern. We have checked that our numerical results do not suffer from false minima or machine error by detecting no significant change when the tolerance is shifted for example by two orders of magnitude. We have set the boundary of the region $\mathcal{W}_B$ at $L_W = 10$. For larger values, valleys with even more levelled bottoms arise and machine with higher precision is needed to allow for a sufficiently small tolerance.

Natural gradient descent also leads to outperforms gradient descent in the transient, significnatly shortening or even removing plateaus in the transient generalization that typically hamper gradient descent training [12, 13].It was recently proposed by Amari as an alternative method for adapting the parameters of a statistical model

on-line using an underlying Riemannian parameter space to redefine the direction of steepest descent. The training error is defined as the negative log likelihood of the data under probabilistic model, then the direction of steepest descent in Riemannian space is found by premulitplying the error gradient with the inverse of the Fisher information matrix.

In our work, the geometry of error surfaces near mimima is a bottom of very shallow steep-sided valley. Standard gradient descent will perform many small steps in going down a long, narrow valley. Conjugate gradient method is more efficient to pass through this region to reach a minimum. The performance of natural gradient descent bases on gradient descent. We believe that its efficiency is between conjuage gradient method and standard gradient descent for the problem like this.

Chapter 3

Numerical results

3.1 Geometry of error surfaces

The error measure E defined in Eq. (2.10) is a n_f-dimensional function of all the network parameters. In general, it has a rugged landscape. For a network with $n = 5$ hidden neurons, the dimension is $n_f = 16$ according to Eq. (2.8). We have trained such a network using a Brownian target function with $N = 125$ sample points and obtained an optimized set of parameters denoted by $\vec{W}_m$. The vector $\vec{W}_m$ thus corresponds to a minimum of E subject to the constrain of the bounded weight space $\mathcal{W}_B$.

The geometry of the error function at the neighborhood of a minimum is most relevant to the efficiency of training. To visualize the geometry of such a high dimensional function, we can only resort to examination of cross-sections. We consider one-dimensional cross-sections created by fixing all but one component in $\vec{W}_m$. Figure 3.1(a) shows five cross-sections produced by varying only the weights $w_i^{(1)}$ connecting the input and the hidden nodes for $i = 1$ to 5. Specifically, we plot the function $E(\vec{W}_m + s\hat{w}_i^{(1)})$ parametrized by s, where "$\hat{}$" denotes a unit vector. Similarly, Fig. 3.1(b) plots $E(\vec{W}_m + s\hat{\theta}_i^{(1)})$ which are cross-sections along the biases of the hidden neurons. Finally, Fig. 3.1(c) plots $E(\vec{W}_m + s\hat{w}_i^{(2)}))$ and $E(\vec{W}_m + s\hat{\theta}^{(2)})$ associated with the weights and bias of the output node. Similar two-dimensional cross-sections have been presented in Ref. [14] for different problems.

In Fig. 3.1, we observe that tuning some parameters create big changes in E while some others lead to practically no variation. This actually corresponds to a

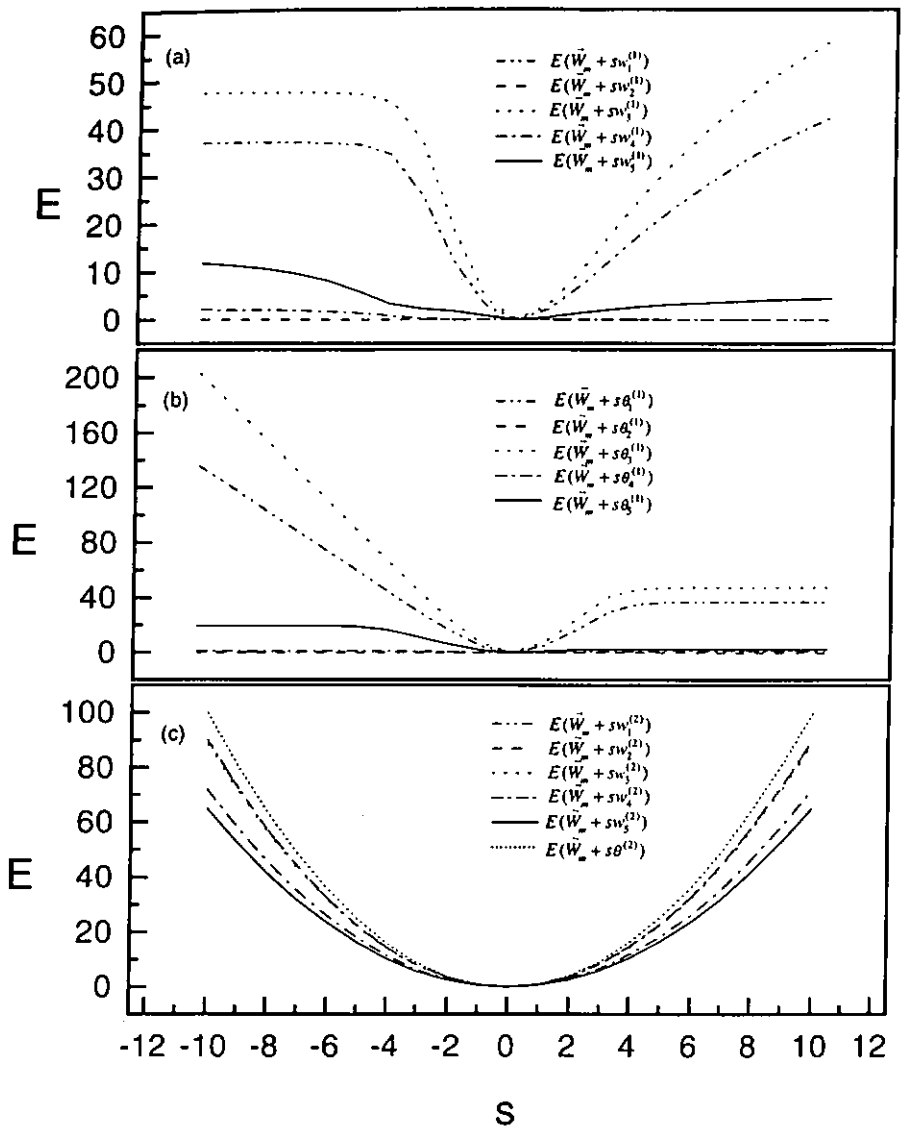

Figure 3.1: One-dimensional cross-sections of the error measure E cutting through a local minimum $\vec{W}_m$ along axes of the network parameters $w_i^{(1)}$ (a), $\theta_i^{(1)}$ (b), $w_i^{(2)}$ and $\theta^{(2)}$ (c) respectively.

valley with very levelled bottom discussed in Sec. 2.3. The popular back-propagation method will crawl very slowly along such a valley in zig-zag manner [11]. Conjugate gradient method is much more efficient for problems like this.

3.2 Almost exhaustive search of minima

We consider again a fixed target function with $N = 125$ sample points. Totally 3000 independent conjugate training sessions are then carried out. Most minima are reached far more than once and we have found only 18 distinguished local minima. Mirror reflections in symmetrical parts of the weight space have been regarded as the same minimum. We have repeated the calculation using two other target functions. 11 and 20 minima are found in these two cases respectively. We also count the frequency f that each minimum is reached and is tabulated in Table 3.2 together with the associated error measure E. The relative frequency, or probability of occurence, p is obtained from $p = f/3000$ and the results are plotted in the form of bar charts in Figs. 3.2(a), (b) and (c) for the three target functions respectively. Recall that in each training session, we initialize the network parameter vector randomly in the region $\mathcal{W}_I$. We are therefore only investigating minima with basins of attraction intersecting $\mathcal{W}_I$. The volume of the basin of attraction of a minimum inside $\mathcal{W}_I$ is proportional to p. We observe from Table 3.2 or Fig. 3.2 that one single minimum dominates respectively in all three cases. These minima in general have small values of E, but are not necessarily the global minimum such as in cases (a) and (c).

We observe that there are only about a dozen of minima with appreciable basins of attraction. There are however 7 and 2 minima visited only once in case (a) and (c) respectively according to Table 3.2. Minima with very small basins of attraction are thus common. This also implies that we have not located all minima exhaustively in the current scale of search.

All minima we have found lie on the boundary of the region $\mathcal{W}_B$. That means they would not be minima if we have not constrained the search in $\mathcal{W}_B$. We check that this is not an artifact of our algorithm as follow. We choose a random vector inside $\mathcal{W}_B$ away from its boundary. The output function of this network is taken as the target function. In subsequent training sessions, we can indeed reach this global minima with error $E = 0$ inside $\mathcal{W}_B$ without problem.

Table 3.1: The error measure E and the frequency of occurence f of all minima found. Each of the columns (a), (b) and (c) lists data for a different target function. In each case, 3000 training sessions have been conducted.

(a)		(b)		(c)	
$E/10^{-5}$	f	$E/10^{-5}$	f	$E/10^{-5}$	f
771	201	876	1874	460	83
779	1	959	2	472	134
786	2473	959	194	513	1677
796	1	1013	38	516	7
801	2	1023	659	520	3
817	1	1026	24	538	46
821	27	1028	25	547	319
839	115	1039	13	555	34
840	20	1042	7	565	1
857	1	1167	155	573	9
869	1	1168	9	634	4
899	1			787	60
903	68			793	12
936	2			797	37
949	2			888	2
950	76			894	29
987	7			1004	489
1000	1			1084	17
				1100	1
				1114	36

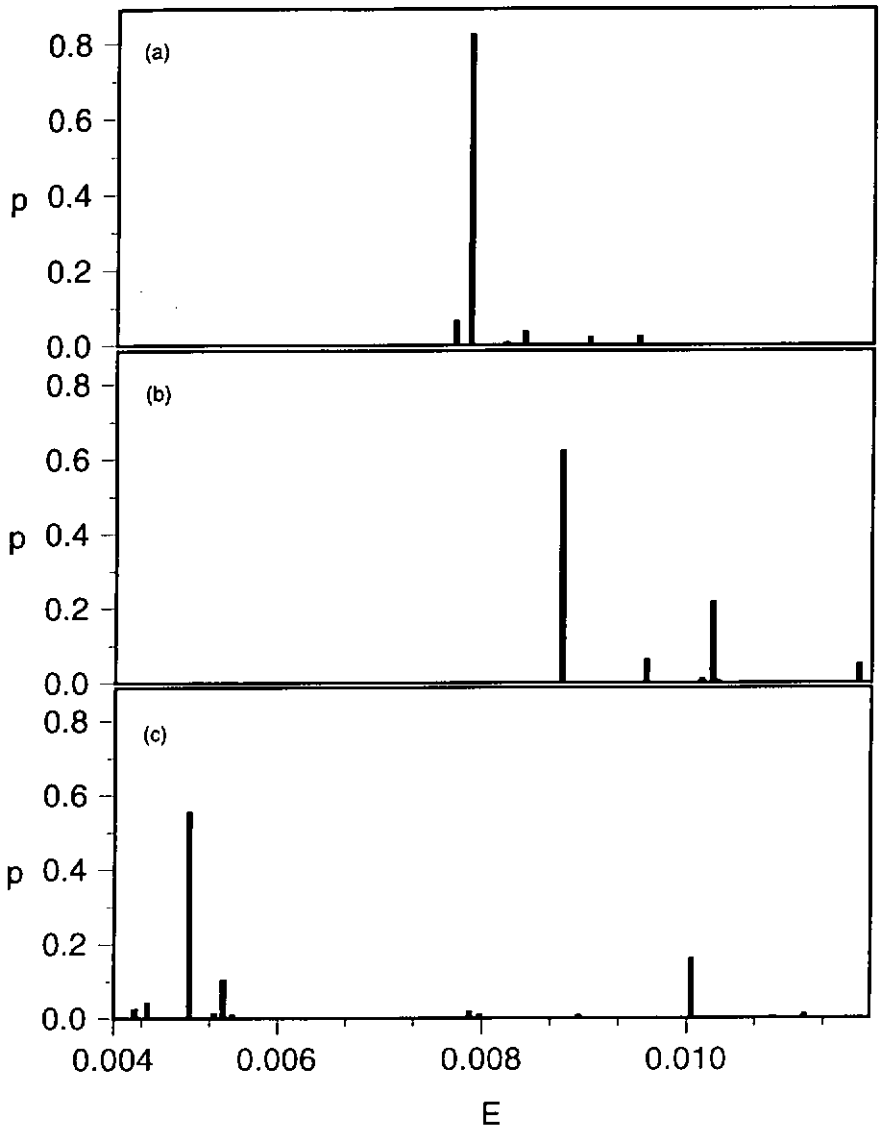

Figure 3.2: Bar charts of the occurence probability p against error measure E of all local minima found. Three different target functions are considered for (a), (b) and (c) respectively.

3.3 Scalings of error measure E

We have just observed in the last section that every target function has a characteristic distribution of minima. In the rest of this work, we will focus on statistical properties averaged over an ensemble of target functions. Specifically, a different realization of target function is used every time a minimum is obtained in a training session. We also broaden the scope to study the effects of changing the number of hidden neurons n in the network and the number of sample points N in the target function.

For a range of n and N, we have obtained in each case 3000 minima using different target functions as described above. The ensemble averaged error $\bar{E}$ can hence be calculated. Figure 3.3 plot $\bar{E}$ as a function of N for $n = 2, 3$ and 5. We observe that $\bar{E}$ rises from some small values at small N. In fact, if $N \leq n_f$, where n_f is the number of parameters in the network given in Eq. (2.8), the task is learnable and in general we can have $\bar{E} = 0$. The results presented focus on the $N > n_f$ case corresponding to unlearnable tasks. As N increases, the larger number of sample points lead to a more complicated target function. The approximation thus becomes more difficult and $\bar{E}$ increases. However, for even larger N, $\bar{E}$ saturates. This occurs when the additional sample points in the target function begin to add only fine details irrelevant for the fitting with finite precision.

We now compare the three curves in Fig. 3.3 for different numbers of hidden nodes n. For increasing values of n, $\bar{E}$ becomes smaller since a larger network can approximate finer details of the target function. Furthermore, $\bar{E}$ saturates at a larger value of N. This is because a larger network can fit through more sample points and thus N has to be larger until certain points will be irrelevant for the fitting. The shape of the three curves are quite similar. They can be approximately collapsed into a single curve by rescaling both $\bar{E}$ and N appropriately. Figure 3.4 plots $\bar{E}/n^{-\beta}$ against N/n with $\beta = 0.9$, which gives the best collapse. This implies a scaling form

$$\bar{E} = n^{-\beta} g(N/n) \tag{3.1}$$

The scaling function $g(z)$ is an increasing function which has practically converged to a constant at $z \simeq 25$. We have chosen to perform most of our computations for the case $n = 5$ and $N = 125$, which is in the saturated regime.

We now examine $\bar{E}$ at saturation for $n = 1$ to 5 by setting $N = 25n$. Each value

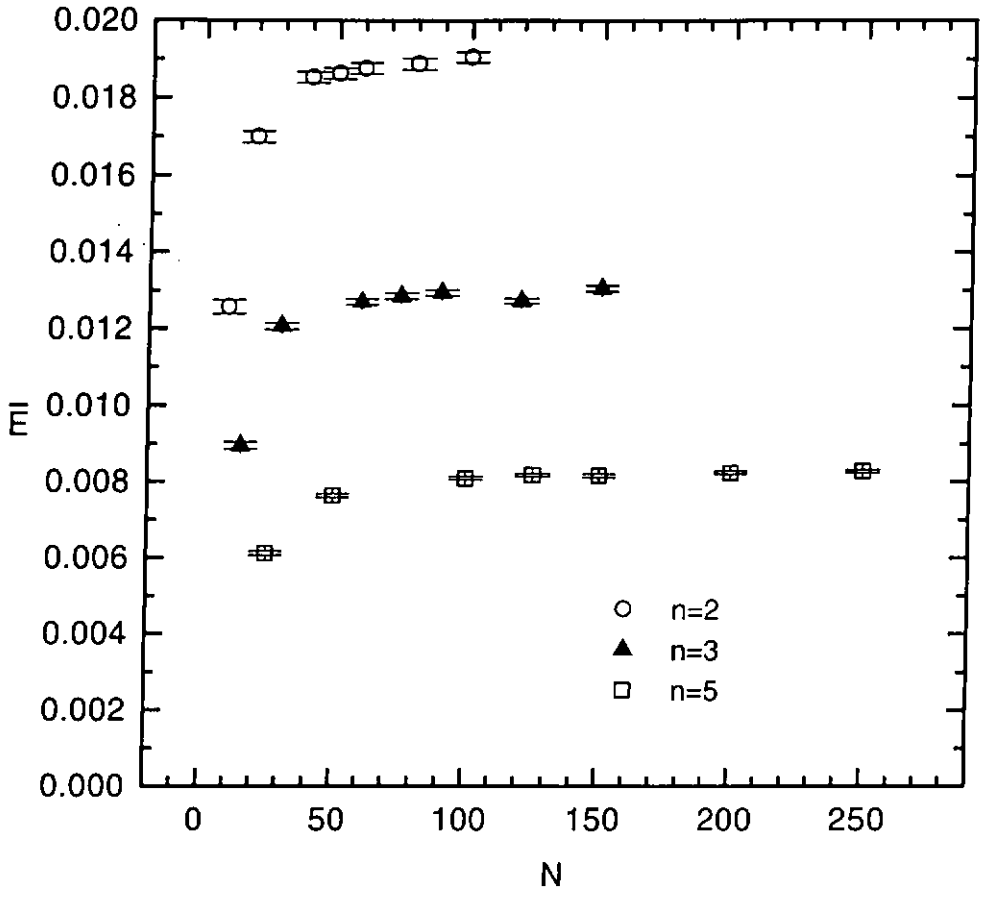

Figure 3.3: Plot of the average error measure $\bar{E}$ against the number of sample points N in the target function. The three curves result from networks with number of hidden nodes $n = 2, 3$ and 5 respectively.

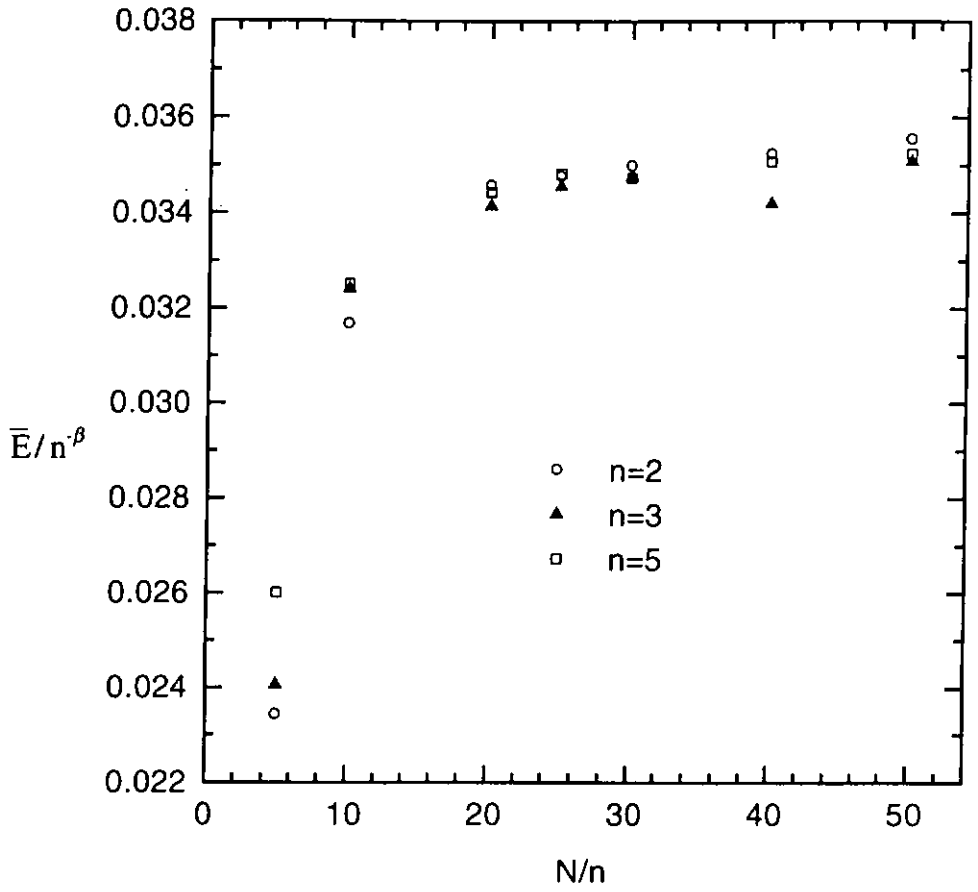

Figure 3.4: Data collapse of the values of $\bar{E}$ from Fig. 3.3 with $\beta = 0.9$.

of $\bar{E}$ has been averaged over 3000 realizations of target functions. The result is shown in Fig. 3.5 in a log-log plot. The linear relation implies

$$\bar{E} \sim n^{-\beta} \tag{3.2}$$

which is a particular case of Eq. (3.1) for $N/n \geq 25$. We obtain a more objective estimate of $\beta = 0.942$ from the slope of the fitted line in Fig. 3.5. At $n = 5$ and hence $N = 125$, $\bar{E} = 8.18 \times 10^{-3}$. This value will be compared with other results later.

3.4 Statistical properties of output error

We investigate again the $n = 5$ and $N = 125$ case. The error measure E defined in Eq. (2.10) is a squared error averaged over N sample points x_μ. For any given input value x, define the output error ϵ by

$$\epsilon(x) = O(x) - T(x) \tag{3.3}$$

We have obtained 30000 values of ϵ taken randomly at various input values from 3000 target and output function pairs. They are histogrammed and the probability distribution $P(\epsilon)$ is hence calculated. Figure 3.6 shows the result. The distribution $P(\epsilon)$ is well fitted by the normal distribution

$$P(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_\epsilon} \exp\left\{-\frac{\epsilon^2}{2\sigma_\epsilon^2}\right\} \tag{3.4}$$

with a standard deviation $\sigma_\epsilon = 0.0855$. From Eqs. (2.10) and (3.3), we should have

$$\bar{E} = \sigma_\epsilon^2 \tag{3.5}$$

and hence $\sigma_\epsilon = 0.0904$ using the value of $\bar{E}$ found in Sec 3.5. The discrepancy is due to slight deviations from the normal distribution especially for large magnitudes of ϵ.

We also studied the auto-correlation function of $\epsilon(x)$ defined as

$$C(r) = < \epsilon(x)\epsilon(x+r) > \tag{3.6}$$

The brackets denote averaging over both realizations of target functions and input values x satisfying $x \geq -l_x$ and $x + r \leq l_x$. We have computed $C(r)$ for $0 \leq r \leq 2l_x$ from 3000 target and output function pairs. Figure 3.7 shows the correlation function for $r < 0.65$. At $r = 0$, $C(r)$ reduces to σ_ϵ^2. It decreases quickly as r increases and then oscillates between positive and anti-correlation. It can be regarded as a short range correlation since it practically vanishes for $0.4 \leq r \leq 2$.

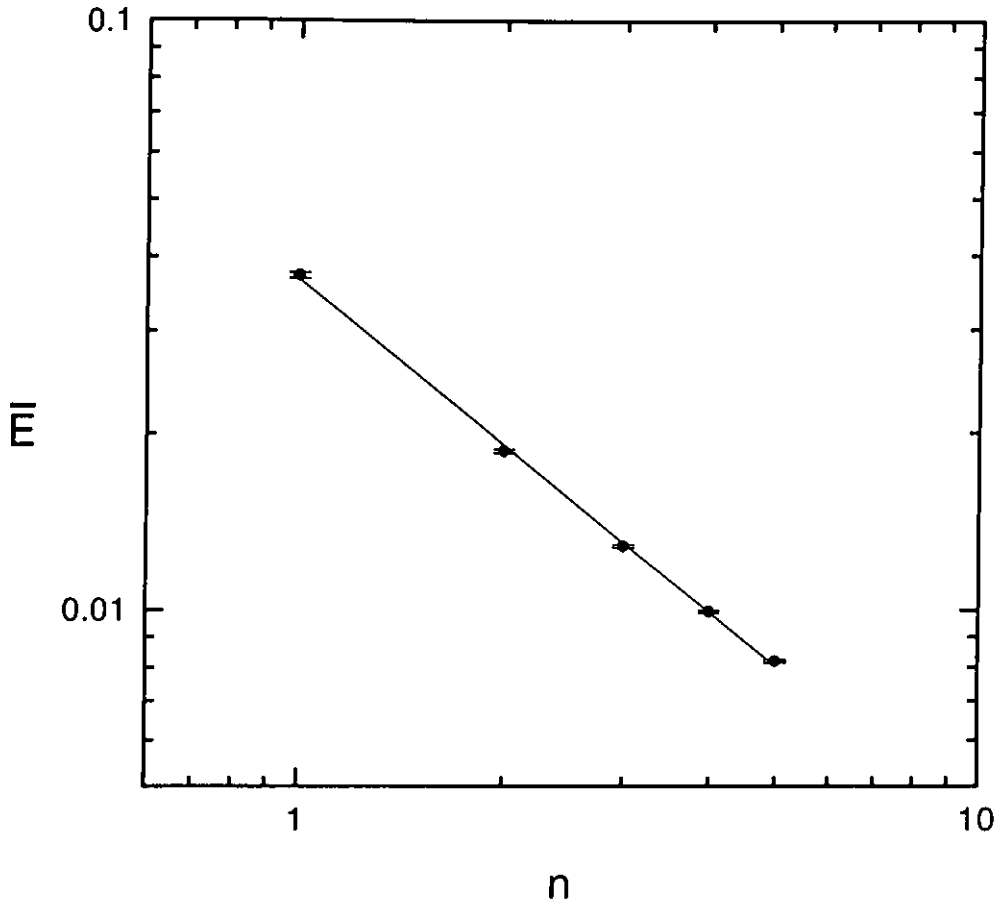

Figure 3.5: Log-log plot of the average error measure $\bar{E}$ against the number of hidden nodes n at saturation. The slope of the fitted line is -0.942.

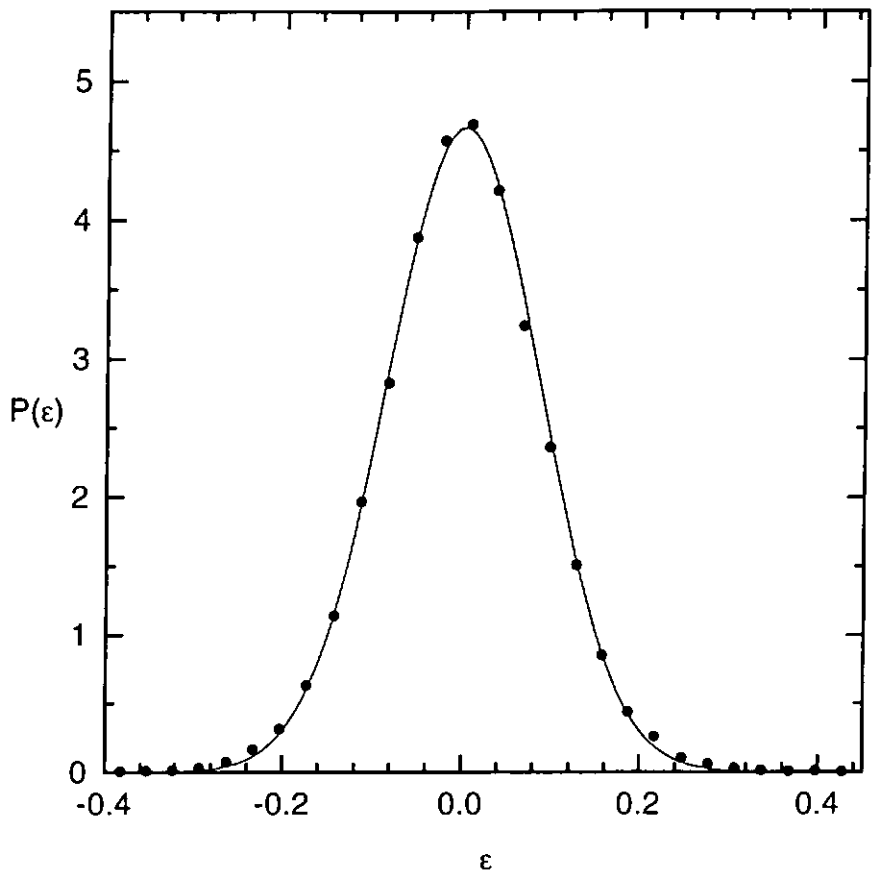

Figure 3.6: Probability distribution $P(\epsilon)$ of the output error ϵ. The fitted line follows the normal distribution with standard deviation $\sigma_\epsilon = 0.0855$.

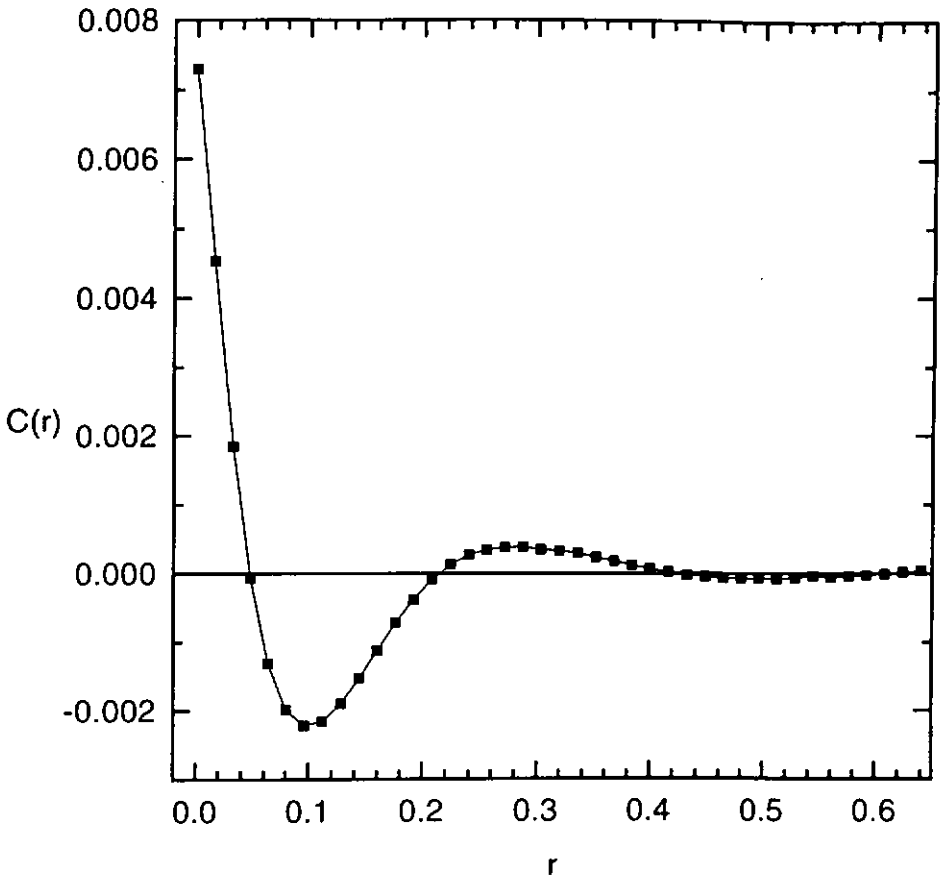

Figure 3.7: Auto-correlation function $C(r)$ of the output error ϵ.

3.5 Statistical properties of error measure E

We now examine the distribution of the 3000 values of the error measure E obtained in 3000 independent training sessions using different target functions. From the histogram of the values, the probability distribution $P(E)$ is calculated. Figure 3.8 shows the result. We have found that the data can be nicely fitted by a log-normal distribution given by

$$P(E) = \frac{1}{\sqrt{2\pi}\sigma_l} \exp\left\{-\frac{(\ln E - \mu_l)^2}{2\sigma_l^2} - \ln E\right\} \tag{3.7}$$

It means that $\ln E$ follows a normal distribution with mean μ_l and standard deviation $\ln \sigma_l$. Note that the distributions of E and $\ln E$ are related by $P(E) = P(\ln E)d(\ln E)/dE$ and the last $\ln E$ term in the exponent in Eq. (3.7) comes from the $d(\ln E)/dE$ factor. The fitted curve is shown in Fig. 3.8 as a solid line. We obtain $\mu_l = -4.86$ and $\sigma_l = 0.251$. The value $\exp(< \ln E >) = \exp(\mu_l) = 7.75 \times 10^{-3}$ is close to $\bar{E} = 8.18 \times 10^{-3}$ obtained in Sec. 3.5. For this rather narrow distribution of E, similarity between the two values are expected.

The origin of the log-normal form of $P(E)$ is purely empirical. It is known that log-normal distributions often result from multiplicative processes. We have suspected that E can be expressed as a product of random numbers each of which may quantify the progress in a certain stage of the minimization process. However, we have not observed any indication of multiplicative nature when examining the change in the intermediate values of E during training.

We therefore suggest an alternative fit for the distribution $P(E)$ based on the distribution $P(\epsilon)$ of the network output error ϵ. First note that the error measure E defined in Eq. (2.10) can be expressed in terms of ϵ as

$$\bar{E} = \frac{1}{N}\sum_{\mu=1}^{N} \epsilon(x_\mu)^2 \tag{3.8}$$

If $N \to \infty$ and the correlation of ϵ is of sufficiently short range, central limit theorem implies that $P(\bar{E})$ follows a normal distribution. However, for the moderate value of N in our consideration, $P(\bar{E})$ deviates significantly from a normal one.

In Sec. 3.4, it has been shown numerically that ϵ follows a normal distribution given in Eq. (3.4). Let

$$y = \epsilon^2 \tag{3.9}$$

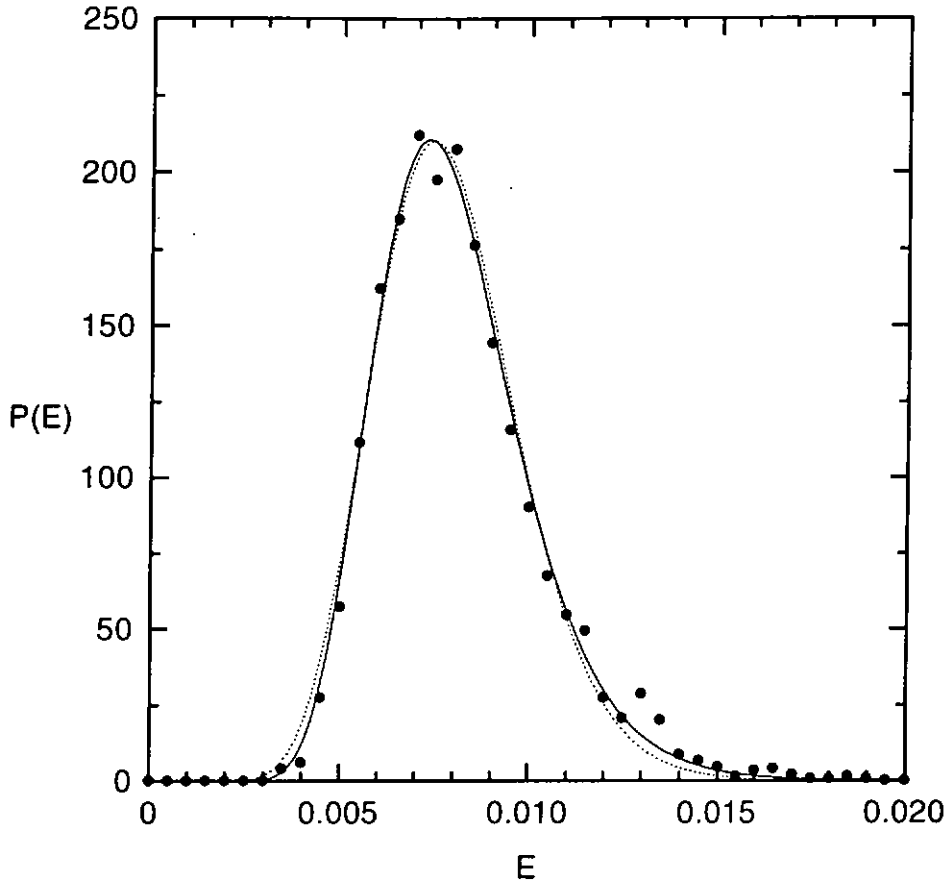

Figure 3.8: Probability distribution $P(E)$ of the error measure E. The solid and the dotted lines are fits using Eqs. (3.7) and (3.17) respectively.

The distribution of y is given by

$$P(y) = P(\epsilon)\left(2\frac{d\epsilon}{dy}\right) \tag{3.10}$$

The factor 2 comes from consideration of both $\pm\epsilon$. Using Eq. (3.4), we get

$$P(y) = \frac{1}{\sqrt{2\pi y}\sigma_\epsilon}\exp\left\{-\frac{y}{2\sigma_\epsilon^2}\right\} \tag{3.11}$$

It has also be reported in Sec. 3.4 that the values of ϵ at neighboring sample input points are correlated. The values of y are therefore also correlated. Since the correlation is of short range, we assume that it effectively extends across λ sample input values. The effective number of independent terms m is given by

$$m = N/\lambda \tag{3.12}$$

According to Eqs. (3.8) and (3.9), E is the average of y over all sample points. It should be approximately equal to the average over only the independent values. Therefore, we have

$$E = \frac{1}{m}\sum_{\alpha=1}^{m} y_\alpha \tag{3.13}$$

where the m values of y denoted by y_α are now independent.

Let

$$S = \sum_{\alpha=1}^{m} y_\alpha \tag{3.14}$$

so that $E = S/m$. The distribution of S is given by the m-fold convolution of $P(y)$ defined in Eq. (3.11). Using induction, it is straight-forward to prove that

$$P(S) = \frac{1}{\Gamma(\frac{m}{2})S}\left(\frac{S}{2\sigma_\epsilon^2}\right)^{\frac{m}{2}}\exp\left(-\frac{S}{2\sigma_\epsilon^2}\right) \tag{3.15}$$

The distribution of E is given by

$$P(E) = P(S)(\frac{dS}{dE}) \tag{3.16}$$

and we get

$$P(E) = \frac{1}{\Gamma(\frac{m}{2})E}\left(\frac{mE}{2\sigma_\epsilon^2}\right)^{\frac{m}{2}}\exp\left(-\frac{mE}{2\sigma_\epsilon^2}\right) \tag{3.17}$$

We fit the numerical data with this expression of $P(E)$ and the resulting curve is also shown in Fig. 3.8 as a dotted line. The fitted parameters are $\sigma_\epsilon = 0.0889$ and $m = 32.9$.

This value of σ_ϵ is in good agreement with 0.0855 obtained directly from $P(\epsilon)$ in Section 3.4. From the result, there are effectively 33 independent values of ϵ out of $N = 125$ samples. The correlation extends across $\lambda = N/m \simeq 3.80$ consecutive input values. This corresponds to input values differing by up to $r = 2L_x/m \simeq 0.0608$. Comparing with the auto-correlation function of ϵ in Fig. 3.7, this value seems to underestimate the range of the correlation. This may be because of the presence of anti-correlation which cancels partly the effects of correlation and gives a short effective range.

From Fig. 3.8, the numerical data are fitted quite well by both the log-normal distribution in Eq. (3.7) and the distribution of the mean squared normal deviates in Eq. (3.17). Since we have not identified a reason for the log-normal form, it is tempting to conclude that it arises simply from its apparently accidental similarity to Eq. (3.17). However, closer examination of Fig. 3.8 reveals that the log-normal distribution in fact gives a slightly better fit. This is also true for other networks with more hidden neurons investigated in smaller scale simulations. Discrepancy in the description using Eq. (3.17) is not surprising in view of the over-simplified correlation assumed. However, we do not know if there is any physical reason behind the log-normal distribution which better describes the numerical data.

Chapter 4

Piecewise linear fit

The approximation of functions using neural networks can be regarded as a curve fitting approach. There are numerous other fitting techniques available. Many properties we have observed here for the neural network approximation are also shared by other fitting methods. In this chapter, we consider a particularly simple although not very accurate approach of piecewise linear fit. It gives simple qualitative and sometimes quantitative explanations of most numerical results in this work.

For the two-layer network we consider in this work with n hidden neurons, the total number of weights and biases is n_f given in Eq. (2.8). It is only fair to make comparisons with an alternative fitting method involving also n_f degrees of freedom. We consider a continuous function $s(x)$ consisting of $n_f - 1$ linear segments of equal width. Assume that every segment contains M sample input values so that

$$N = (n_f - 1)M = 3nM \tag{4.1}$$

where Eq. (2.8) is used. A sample input point x_μ is on the boundary of a segment if μ is a multiple of M, i.e. $\mu = \alpha M$ for some integer α.

The piecewise linear function $s(x)$ can be completely specified by its values at all boundary points and thus has n_f free parameters $s(x_{\alpha M})$ where $\alpha = 0$ to $n_f - 1$. It is intuitively clear that $s(x)$ is also a universal approximator. The precision can be made arbitrarily high provided that n_f is sufficiently large. To construct a piecewise linear function $s(x)$ which approximates a target function $T(x)$, we could apply a least square fit. However, for simplicity in further analysis, we adopt a trivial fitting procedure by setting the free parameters as

$$s(x_{\alpha M}) = T(x_{\alpha M}) \tag{4.2}$$

We now examine the accuracy of this simple fitting approach. Following our previous notations, we denote the error of the fit at input value x again by $\epsilon(x)$, i.e.

$$\epsilon(x) = s(x) - T(x) \tag{4.3}$$

Since both s and T follows normal distributions, ϵ also follows a normal distribution as observed numerically in Sec. 3.4 for neural network approximation.

According to Eqs. (4.2) and (4.3), $\epsilon(x_{\alpha M}) = 0$, which means that the error vanishes at the end points of all segments. Every segment of ϵ is thus isolated and has identical statistical properties. From the scaling property of Brownian functions in Eq. (2.4), the standard deviation σ_ϵ of ϵ scales with the number of sample points M in a segment as

$$\sigma_\epsilon \sim \sqrt{\frac{M}{N}} \tag{4.4}$$

The pre-factor will also be calculated later. Using Eqs. (4.1), we get

$$\sigma_\epsilon \sim \frac{1}{\sqrt{n}} \tag{4.5}$$

From Eq. (3.5),

$$\bar{E} \sim \frac{1}{n} \tag{4.6}$$

Comparison with Eq. (3.2) implies that the scaling exponent β is 1. This value is consistent with the numerical estimate 0.942 for the neural network case. This is of no accident. In this simple consideration, linear segments are used to fit features in the target function at various positions. For neural networks, the hyperbolic tangent outputs of the hidden neurons behave similarly and are responsible for approximations at different regions, although there are complications due to variation in scales. Furthermore, we expect that the value $\beta = 1$ is exact for neural networks in the limit $n \to \infty$ and $L_W \to \infty$. Taking $n \to \infty$ dilutes the boundary effects at input values $x = \pm l_x$. Large L_W is necessary for the network to be a universal approximator generating features at arbitrarily small wavelength.

We now explain the full scaling form in Eq. (3.1). The prefactors in the scaling relations for σ_ϵ and $\bar{E}$ in Eqs. (4.5) and (4.6) respectively depend on the number of sample points $M = N/3n$ in a segment. For example, if $M = 1$, $s(x)$ trivially goes through every sample points and $\sigma_\epsilon = \bar{E} = 0$. Therefore, Eq. (4.6) becomes

$$\bar{E} \sim \frac{1}{n} g_0\left(\frac{N}{3n}\right) \tag{4.7}$$

The function $g_0(M)$ increases with M and approaches a constant when the abundance of sample input points can be approximated by a continuum. Defining g by reparametrizing g_0, we arrive at the scaling form in Eq. (3.1) with $\beta = 1$. Similar to the argument discussed above, we believe that this scaling form is exact also for the neural network case in the appropriate limit.

The above calculations highlight the fact that the scaling properties of the error measures are simple results inherited from related scalings of Brownian functions. We now calculate the actual value of σ_ϵ and $\bar{E}$ which involves slightly more algebra. We have pointed out that all segments of ϵ have identical statistical properties. Let us focus on the first segment corresponding to $\alpha = 1$. For $1 \leq \mu \leq M$, we have

$$\epsilon(x_\mu) = \frac{\mu}{M} T(x_M) - T(x_\mu) \tag{4.8}$$

From Eq. (2.2), the Brownian function T can be expressed as a sum of displacements, i.e.

$$T(x_\mu) = \sum_{\gamma=1}^{\mu} \delta_\gamma \tag{4.9}$$

Substituting the above expression into Eq. (4.8) and some rearrangements, we have

$$\epsilon(x_\mu) = \sum_{\gamma=1}^{\mu} (\frac{\mu}{M} - 1)\delta_\gamma + \sum_{\gamma=\mu+1}^{M} \frac{\mu}{M} \delta_\gamma \tag{4.10}$$

Since all δ's are independent variables, the standard deviation $\sigma_\epsilon(x_\mu)$ of $\epsilon(x_\mu)$ can be computed easily and we obtain

$$\sigma_\epsilon(x_\mu) = \sigma_\delta \sqrt{\frac{\mu(M-\mu)}{M}} \tag{4.11}$$

where σ_δ has been given in Eq. (2.3). The standard deviation of ϵ for arbitrary input values is obtained from the mean of the variance, i.e.

$$\sigma_\epsilon^2 = \frac{1}{M} \sum_{\mu=1}^{M} \sigma_\epsilon(x_\mu)^2 \tag{4.12}$$

If M is large corresponding to the saturated case, it can be approximated by an integral

$$\sigma_\epsilon^2 = \frac{1}{M} \int_0^M \sigma_\epsilon(x_\mu)^2 d\mu \tag{4.13}$$

Applying Eqs. (4.11), (2.3) and (4.1), we get

$$\sigma_\epsilon = \sqrt{\frac{1}{18n}} \tag{4.14}$$

and hence from Eq. (3.5),

$$\bar{E} = \frac{1}{18n} \tag{4.15}$$

For $n = 5$, we obtain $\sigma_\epsilon = 0.105$ and $\bar{E} = 0.0111$. These values are only 24% and 35% larger than 0.0855 and 8.18×10^{-3} obtained numerically in Sections 3.4 and 3.5 respectively for function approximations with neural networks. Our simple approach thus gives a fairly successful estimate of the errors in neural network approximations.

Chapter 5

Conclusion

We have investigated approximations of Brownian functions by two-layer feed-forward neural networks. By using a conjugate gradient training method and restricting the network parameters within a finite range, all local minima with appreciable sizes of basins of attraction are located and found to be in the order of a dozen in number. A valley in the error surface with a very levelled bottom which can hinder training using back-propagation is revealed in one-dimensional cross-sectional plots. The network output error ϵ follows a normal distribution while its mean squared value E obeys a log-normal distribution. We have no physical explanation of the log-normal distribution which may simply result from its accidental similarity to the distribution of mean squared normal deviates. The dependence of the ensemble averaged value of E on the number of neurons in the hidden layer and the number of input values used in training follows simple scaling rules. Our numerical findings are explained by comparison with a simple piecewise linear fit approach.

Multi-layer neural networks possess complicated error landscape and analytical treatment is difficult. The current work extracts simple relations for their behaviors in learning a specific model problem. We hope that the results can serve as useful hints for other practical problems. Some relations may even be readily applicable since there are signals in e.g. electrical circuits and financial markets admitting fractal properties similar to those of Brownian functions.

Bibliography

[1] P. Peretto, An Introduction to the modeling of neural networks, Cambridge University Press (1992).

[2] L. Fausett, Fundamentals of neural networks, Prentice Hall (1994).

[3] J. Hertz, A. Krogh, and R.G. Palmer, Introduction to the theory of neural computation, Addison Wesley (1991).

[4] B.B. Mandelbrot, The fractal geometry of nature, Freeman (1982).

[5] T.L.H. Watkin, and A. Rau, Rev. Mod. Phys. **65**, 499 (1993).

[6] G. Cybenko, Mathematics of control, Signals and Systems **2**, 303 (1989).

[7] K. Hornik, M. Stinchcombe, and H. White, Neural Networks **2**, 359 (1989).

[8] A. Priel, M. Blatt, T. Grossman, E. Domany, and I. Kanter, Phys. Rev E **50**, 577 (1994).

[9] W. Nadler and W. Fink, Phys. Rev. Lett. **78**, 555(1997).

[10] G.B. Huang and H.A. Babri, IEEE Trans. on Neural Networks **9** 1045 (1998).

[11] W.H. Press et al., Numerical recipes in C, 2nd Ed., Cambridge (1992).

[12] M. Rattray, D. Saad, and S. Amari, Phys. Rev. Lett. **81**, 5461 (1998).

[13] M. Rattray and D. Saad, Phys. Rev. E **59**, 4523 (1999).

[14] D.R. Hush, B. Horne, and J.M. Salas, IEEE Trans. on System, Man, and Cybernetics **22**, 1152 (1992).