

Spectral Approximation of Functions by Using Three-layered Feed-forward Neural Networks

Cesare Citterio, Andrea Pelagotti, Vincenzo Piuri, Luca Rocca
Department of Electronics and Information, Politecnico di Milano
piazza Leonardo da Vinci 32, I-20133 Milano, Italy
email: piuri@elet.polimi.it

Abstract

The universal approximation capability exhibited by one-hidden-layer neural network is analyzed in the frequency domain. Hidden neurons are studied in terms of spectral generators and the output neurons as units linearly combining the spectra. The learning phase is described in terms of spectral approximation: it is directed to reduce the distance between the reference function's spectrum and the output network's one. In this paper, we propose a new spectrum-based technique to train 1-N-1 networks which approximate $y=f(x)$ functions, with $x, y \in \mathbf{R}$; besides, this method takes in account the robustness of the resulting weights configuration.

1. Introduction

Functions approximation using connectionist supervised learning has been successfully applied to estimate functions starting from a representative set of input-output pair patterns. Imposing a certain degree of smoothness on the unknown function, one-hidden layer networks were proven to be universal approximators [1].

However, designing and tailoring neural network to solve a specific approximation problem may often become complex tasks, because they imply identification both of the topological structure (i.e., the size of the hidden layer) and of the operating parameter configuration (i.e., interconnection weights and thresholds). The traditional approach consists of the following steps: creation of a tentative network by exploiting the knowledge of experts, training the uncommitted structure by using the back-propagation algorithm or similar ones, and validation of the final configured network to verify the operation error and the generalization capabilities.

The number of input-output pairs to use during learning in order to reach a low validation error cannot be a priori computed. If few patterns are used, the network shows poor generalization capability because it has learned only one small part of the information contained in data and, as a consequence, it cannot predict the correct output corresponding to a never seen input. Vice versa, a great bulk of training patterns can make the training process too time consuming and, besides, more hidden neurons are often required; this implies that the error surface has more local minima and back-propagation algorithms convergence is more difficult.

To characterize a function, i.e., to represent the information contained in the function, we traditionally use the input-output pairs: however, this is not the only feasible approach. In Sections 2 and 3, we show how to represent a function $y=f(x)$ by its Fourier Transform spectrum and how to train a network in order to reproduce the same harmonics. By analyzing the spectrum of $f(x)$, an adequate topological neural structure can be identified and a system of spectral-training equations can be considered; in Section 4, the learning procedure is summarized and examples are given.

The learning procedure is not executed by applying patterns to the networks and then by trying to reduce the global error, but by adapting the connection weights to obtain the required harmonic components at network's output. This makes the learning faster and improves the generalization ability.

2. Spectrum generation by hidden neurons

Neural networks can be analyzed from a spectral point of view [2]. In a trained three-layered network, with a linear output layer, output neurons compute a linear combination of spectra generated by hidden neurons in order to obtain a good approximation of the required output spectrum, i.e. the spectrum of the function to be approximated. Now we briefly summarize the results presented in [2].

Given the periodic function $v=v(t)$ with period T , we can sample it and obtain $v=v(k)$, $k=0..N-1$, where N is the number of samples. According to Nyquist theorem [3], we choose the sample frequency f_c higher than double times the maximal harmonic frequency. We can analyze the spectrum of $v(t)$ by computing the Discrete Fourier Transform (DFT):

$$V(h) = \frac{1}{N} \sum_{k=0}^{N-1} v(k) e^{-j h k \phi} \quad \text{with } \phi = \frac{2\pi}{N} \quad (1)$$

where $V(0)$ is the offset (given by the mean of $v(k)$), and $|V(h)|$, $h=1..\lfloor N/2 \rfloor$, is the half of the h -th harmonic's amplitude. Note that $V(h)$ is periodic with period N , and that $V(N-h)=\text{conj}(V(h))$. The set of $V(h)$, $h=0..\lfloor N/2 \rfloor$, contains all spectral information about the function $v(t)$. The FFT (Fast Fourier Transform) algorithm can be used instead of the DFT to speed up the computation.

Let $y=f(x)$, $x \in [x_1, x_2]$, be the function to be approximated by the neural network. First, we transform $f(x)$ in a periodic function, by applying to its input a periodic signal such as:

$$x = \frac{x_1 + x_2}{2} + \frac{x_2 - x_1}{2} \sin\left(\frac{2\pi}{T} t\right) \quad (2)$$

To sample $y(t)=f(x(t))$, we substitute the ratio t/T by k/N ; we obtain $y=y(k)$. Finally, we compute the DFT. To avoid aliasing, N must be greater than $f_c T$, where f_c is the Nyquist frequency. To compute f_c , we make some experiments by considering large values of N and by checking this adopted value with respect to the up-limit frequency, i.e., the lowest frequency above which harmonics are null (or below a given small threshold).

Let's consider a three-layered network with N_h hidden neuron, a single input and a single output. The hyperbolic tangent is used as output function for the hidden neurons, while the output neuron is linear. We obtain:

$$\hat{y}(x) = d + \sum_{i=1}^{N_h} c_i \tanh(b_i x + a_i) \quad (3)$$

where $\hat{y}$ is the network's output, d and a_i are the biases, c_i are the output neuron's weights, and b_i are the input neurons' ones. We apply $x(k)$ to the network's input and we compute the DFT($\hat{y}(k)$). Due to the linear properties of the DFT, the network's output spectrum is a linear combination of the hidden neurons' output spectra:

$$DFT[\hat{y}(k)] = DFT[d] + \sum_{i=1}^{N_h} c_i DFT[\tanh(b_i x(k) + a_i)] \quad (4)$$

Consider the single neuron's spectrum when it is stimulated by a sinusoid function as $x(k)$. In Fig. 1, three different pairs (a, b) are used to obtain three different spectra, associated to different output saturations: low ($a=0.5$, $b=0.1$), middle ($a=1.5$, $b=5$), and high ($a=3$, $b=20$). We note that a neuron's spectrum is always band-limited and the band increases when the saturation increases.

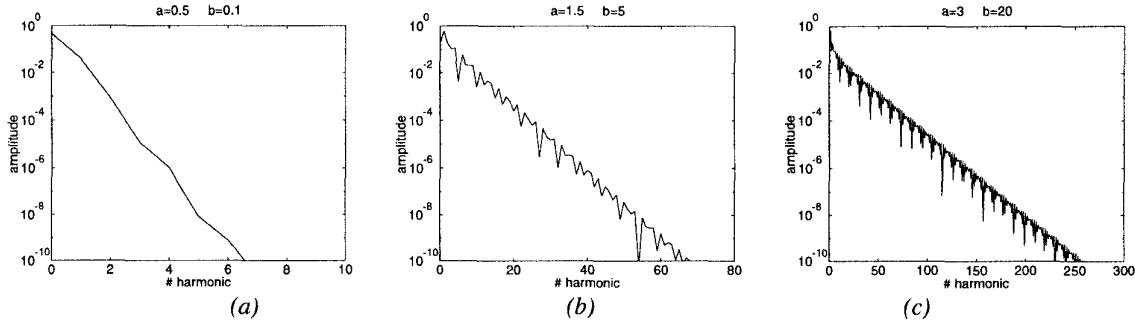


Fig. 1 - Examples of neurons' spectra.

Our goal is to make $\hat{y}$ as much as possible similar to y : as a consequence, $\hat{y}$ and y must have the same spectrum. We assume that $y(k)$ is band-limited, i.e., the amplitudes of the harmonics above a certain frequency are below a given small threshold (for instance, 10^{-4} times the first harmonic amplitude). Let N_a be the number of harmonics inside the band, except the offset $V(0)$. To approximate the function $y=f(x)$, we must approximate its first N_a harmonics by means of the neural network characterized by the same band. From Eq. 4, we can derive Eq. 5:

$$\begin{bmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,N_h} \\ v_{2,1} & v_{2,2} & \dots & v_{2,N_h} \\ \dots & \dots & \dots & \dots \\ v_{N_a,1} & v_{N_a,2} & \dots & v_{N_a,N_h} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \dots \\ c_{N_h} \end{bmatrix} = \begin{bmatrix} V(1) \\ V(2) \\ \dots \\ V(N_a) \end{bmatrix} \quad d = V(0) - \sum_{j=1}^{N_h} c_j v_{0,j} \quad (5)$$

where each column v_j is the vector composed by the first N_a harmonics of the j -th hidden neuron's spectrum (except the offset); the vector $\mathbf{V}$ at the second member is composed by the first N_a harmonics of y (except the offset). The output offset is corrected by Eq. 7. Note that $\mathbf{V}$ is composed by real or imaginary numbers only, not by complex ones. This is due to the fact that $y=f(x)$ receives a sinusoid function with a null angle of phase; in particular, if h is odd then $V(h)$ is imaginary, otherwise it is real. We can simplify Eq. 6 by replacing $\mathbf{v}$ by $\text{real}(\mathbf{v})+\text{conj}(\mathbf{v})$ and $\mathbf{V}$ by $\text{real}(\mathbf{V})+\text{conj}(\mathbf{V})$: the equation becomes therefore real.

3. Tuning the spectrum generators

Eq. 5 implies the correct training of the network if and only if y and $\hat{y}$ have the same band, i.e., the i -th harmonic with $i > N_a$ can be neglected. Since the spectrum of $\hat{y}$ is a linear combination of the hidden neurons' spectra, we can reach this goal by tuning the hidden neurons' weights (a_i and b_i) in order to limit their spectral bands to the desired value.

The validation error is upper limited by the sum of the inside band harmonic error and the outside band one:

$$\max |y - \hat{y}| \leq \sum_{i=0}^{N_a} |A_i - \hat{A}_i| + \sum_{i=N_a+1}^{\infty} |A_i - \hat{A}_i| \quad (6)$$

where A_i is the amplitude of the i -th harmonic of y , $\hat{A}_i$ is the i -th harmonic of $\hat{y}$. If Eq. 5 is satisfied, the first summation is zero and the validation error is equal to the outside band harmonic error. Therefore, we initially choose N_a according to the maximal validation error we tolerate. The final approximation error is upper limited by the sum of the amplitudes of all the harmonics we ignore; when N_a increases, the maximal validation error decreases.

A simple way to solve the system (5) consists of choosing $N_h = N_a$ and applying a classical algorithm to solve a linear equations' system (e.g., the Cramer's method). Since the matrix $\mathbf{v}$ must be not singular in order to obtain one and only one solution $\mathbf{c}$, we must tune the hidden neurons to make their spectra linearly independent.

The hidden layer is built step by step, progressively. First, it is initialized to an empty layer. Then, one neuron at a time is added: its weights a_i and b_i are randomized and linearly-independence and band-limited conditions are checked. If these conditions are satisfied, the most-recently inserted neuron is retained; otherwise, it is discarded and weights are randomized again. At the end, we obtain a layer composed by N_h hidden neurons. Output neuron's weights are computed by solving the system (5).

This training method is very fast but it has one drawback: weights c_i may be very large with respect to the output range [2]. As a consequence, many significant digits are required to compute correctly the weighted sum in the output neuron. Besides, very high weights in the output neuron may increase outside-band harmonics' amplitudes, because they are amplified by the high coefficients: in this case, the band of the network's output is no more limited by the first N_a components and, as a consequence, the validation error increases.

This drawback is due to the fact that the system is ill-conditioned [4]. In fact, if the coefficient matrix $\mathbf{v}$ is non-singular, one and only one solution is assured but nothing can be stated about possible computing problems related to the limited precision of the computer. If the system is bad conditioned, its solution is greatly affected by errors due to the finite precision in the number representation and its value may be very large. To avoid this events, we should configure the hidden neurons so that their harmonics generate a coefficient matrix with a low condition number. The solution of Eq. 5 can be expressed by:

$$c_i = \frac{\det u_i}{\det \mathbf{v}} \quad \forall i = 1 \dots N_h \quad (7)$$

where u_i is the matrix obtained by replacing the i -th column of $\mathbf{v}$ with $\mathbf{V}$. Note that the denominator is shared by all c_i . Whenever a group of columns is closed to be linearly dependent, the associated c_i are greater than the others. The rational behind this can be simply explained by noting that the matrices u_i associated to the linearly independent columns are nearly to be singular, because they contain the group of columns which is closed to be dependent. To improve the conditioning of system (5), we must tune the hidden neurons associated to the highest weights c_i in order to generate spectra (i.e., columns in matrix $\mathbf{v}$) more strongly linearly independent. In next section we propose a learning procedure which take into account all these remarks.

4. Spectral learning

The learning procedure is summarized as follows:

1. Compute the spectrum of $y(k)$.
2. Choose N_a according to the given maximum validation error, which is estimated by the sum of all the outside band harmonics (i.e., harmonics after N_a -th one).
3. Set $N_h = N_a$ to guarantee the existence of one solution for the system (5).
4. Build the hidden layer by adding one neuron at a time, randomizing its weights a_i and b_i and checking the linear-independence and the band-limited conditions (see section 3).
5. Compute c_i by solving Eq. 5.
6. While $\max(\text{abs}(c_i)) < G$ do
 - 6.1. Randomize the weights a_i and b_i associated to the biggest $|c_i|$ and compute v again until $\max(\text{abs}(c_i))$ decreases.

where G is the desired upper bound of the set of c_i (for example, we set $G=50$). The more G is small, the more the validation error is closed to the estimated value based on the sum of the outside band harmonics of $y(k)$. Too small values of G make difficult the termination of the learning procedure.

In Fig. 2, an interesting experiment is shown: we compare the network (which approximates the function in Fig. 2a) after the first five steps (Fig. 2b) and at the end (Fig. 2c) of the learning procedure. The differences between the two networks, in terms of validation error and output spectrum, give importance to the c_i reduction routine.

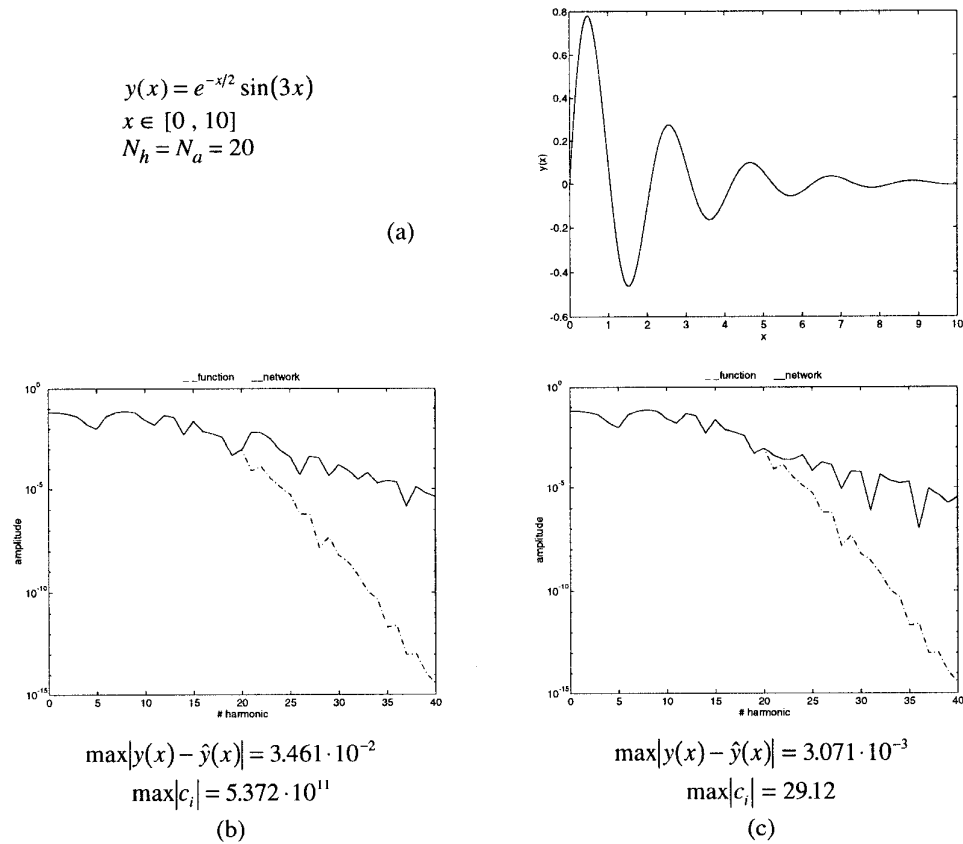
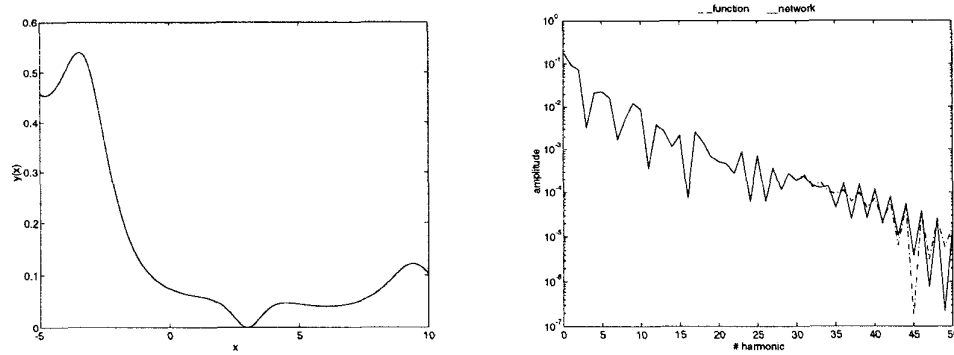


Fig. 2 - Function approximation after 5 steps (b) and at the end of the learning (c).

Figs. 3 and 4 show other networks obtained by applying the whole spectral learning procedure. The validation error is closed to the value of the N_a -th harmonic of $y(k)$, which is therefore a good estimator of the final error.

The learning time is considerably smaller than the one required by traditional error backpropagation algorithms. For example, the networks in Fig. 3 and 4 require a spectral learning time of about 8 minutes on a Pentium-based IBM-compatible PC running a dedicated C program, while a Quasi-Newton backpropagation algorithm [4] needs

several hours to reach the same validation error on the same computer with a suited dedicated C program. Besides, this method avoid any overfitting error because the training problem is solved from a spectral point of view.



$$y(x) = \frac{1}{x+7} \arctan(x-3)^2, \quad x \in [-5, 10], \quad N_h = N_a = 30$$

Fig. 3

$$\max |y(x) - \hat{y}(x)| = 6.917 \cdot 10^{-4}, \quad \max |c_i| = 52.50$$

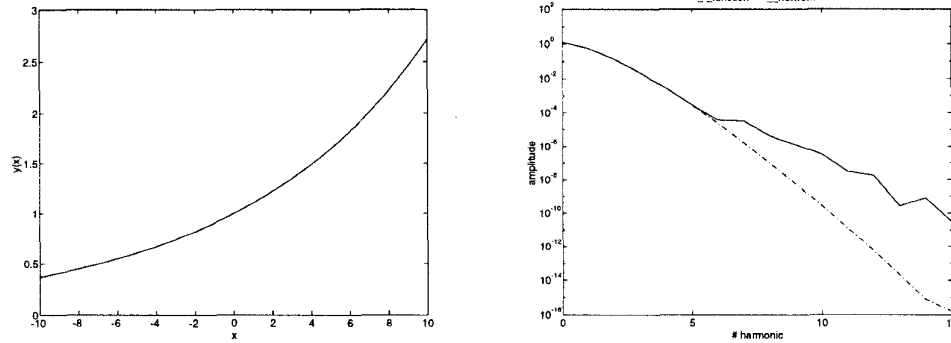


Fig. 4

$$y(x) = e^{x/10}, \quad x \in [-10, 10], \quad N_h = N_a = 5$$

$$\max |y(x) - \hat{y}(x)| = 1.901 \cdot 10^{-4}, \quad \max |c_i| = 19.32$$

5. Conclusions

In this paper, we proposed an original learning method to enhance the training time for the function approximation problem. This technique is based on the spectral analysis and allows to synthesize a neural structure capable to approximate a $y=f(x)$ function by imposing that the network's spectrum is equal to the function's one. Learning consists first of solving a linear system composed by N_h equation (being N_h the number of hidden neurons) and then of improving the problem conditioning by tuning the hidden neurons' weights in order to minimize the magnitude of the output weights. Extensive experiments have shown both effectiveness and efficiency of our method. In particular, the spectral learning procedure saves a lot of computing time with respect to traditional error backpropagation algorithms, without increasing the learning error. Besides our method avoids any overfitting error.

6. References

- [1] K. Hornik, M. Stinchcombe, H. White, "Multilayer feedforward networks are universal approximators", Neural Networks, vol. 2, 1989.
- [2] Andrea Pelagotti, Vincenzo Piuri, "Spectral Analysis and Synthesis of Three-layered Feed-forward Neural Networks for Function Approximation", Proc. MicroNeuro, Lausanne, Switzerland, February 1996.
- [3] A.V.O. Oppenheim, R.W. Schaffer, "Digital signal processing", Prentice Hall, 1975.
- [4] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, "Numerical Recipes in C", Cambridge University Press, 1988.