Approximation with Artificial Neural Networks

MSc thesis

Balázs Csanád Csáji
Faculty of Sciences
Eötvös Loránd University
Hungary

Consultant:
Huub ten Eikelder
Faculty of Mathematics and Computing Science
Eindhoven University of Technology
The Netherlands

2001

Contents:

CONTENTS: 2
PREFACE: 3
1. INTRODUCTION: 4
1.1. About Neural Networks: 4
1.2. Artificial Neurons: 6
1.3. Types of Activation Functions: 7
1.4. Multilayer Feedforward Architecture: 8
2. COMPUTATION POWER: 9
2.1. Single-Layer Perceptrons: 9
2.2. Computation of Boolean Functions: 9
2.3. The Universal Approximation Theorem: 11
2.4. Arrangement of the Neurons: 12
3. SPECIAL CASES: 13
3.1. About B-Splines: 13
3.2. Case of the Threshold Function: 14
Theorem 3.2.1 (approximation with the threshold function): 14
Theorem 3.2.2 (estimating the error): 16
Corollary 3.2.3 (convergence): 17
Theorem 3.2.4 (optimality): 17
3.3. Case of the Piecewise Linear Function: 18
Theorem 3.3.1 (approximation with the piecewise linear function): 18
3.4. Case of the Logistic Function: 19
Theorem 3.4.1 (approximation with the logistic function): 19
4. GENERAL CASE: 22
4.1. Approximation with Arbitrary Activation Function: 22
Theorem 4.1.1 (universal approximation theorem): 22
4.2. Debao Chen's Theorem: 25
Theorem 4.2.1 (by Debao Chen in 1993): 25
5. IMPROVING THE APPROXIMATION: 26
5.1. General Multiresolution Algorithm: 26
5.2. About Wavelets: 29
5.3. Multiresolution Formulation of Wavelets: 31
5.4. Numerically Stable Reconstruction: 33
Theorem 5.4.1 (sufficient requirements for a frame): 34
5.5. Building Wavelets from Logistic Functions: 35
Theorem 5.5.1 (integral of the logistic wavelet): 36
Theorem 5.5.2 (logistic wavelets form a frame): 36
EPILOGUE: 40
APPENDIX$_1$: 41
APPENDIX$_2$: 43
BIBLIOGRAPHY: 45

Preface:

"Computer Science is no more about computers than astronomy is about telescopes."
E. W. Dijkstra

An artificial neural network is a (simplified) mathematical model of the human brain. Many different types of neural network models are suited, but we shall describe just one, called *feed-forward neural network* with one hidden layer (also called as multilayer perceptron with one hidden layer).

Since 1960 a number of results have been published showing that a multilayer neural network with only one hidden layer can approximate arbitrary well a continuous function of *n* real variables. E.g. poofs have been given by Cybenko (1989), Halbert White (1990) and Kurt Hornik (1991), the later proofs require less assumptions on the activation functions. The problem with these proofs is that they are not constructive (and not elementary). These theorems do not state how many neurons should be used in the hidden layer, and they do not claim anything about the error of the approximation with a given number of neurons. Our main aim was to prove the *universal approximation theorem* of the feed-forward artificial neural networks in a constructive (and may be elementary) way and if possible also estimate the error. The basic form of this theorem claims that every continuous function defined on a compact set of the n^{th} dimensional vector space over the real numbers can be arbitrary well approximated by a feed-forward artificial neural network with one hidden layer (with finite number of artificial neurons). In other words, the function space defined by the feed-forward artificial neural networks with one hidden layer is dense in $C(R^n)$.

In the first chapter we describe a few basic things about the artificial neural networks: e.g. the artificial neurons, the types of activation functions and the multilayer feed-forward architecture. After that, in the second chapter we describe the computation capacity of the feed-forward artificial neural networks, e.g. computation of boolean functions and the universal approximation theorem; and also in this chapter we introduce the method of *arranging the neurons* that allows us to build up different kind of *mother functions* (defined later) from the used activation function. We used this method to prove the different variations of the universal approximation theorem. In the third chapter we prove the universal approximation theorem for some special cases: for the three basic (most popular) activation function (the threshold/step function, the piecewise linear and the sigmoid/logistic function) only in one dimension. The fourth chapter consists a generalisation of our results for an arbitrary activation function and another theorem by Debao Chen (1993) that estimates the error of the approximation. During the fifth chapter we will improve the approximation by an iterative (successive/multiresolution) approximation method and by using the *theory of wavelets*. We use a previously defined method of arranging the neurons to build a mother wavelet (by the combination of several logistic activation functions) and with this technique we can give another proof. In the appendix there are some programs in Maple and Matlab that demonstrate our results.

Balázs Csanád Csáji
Eindhoven, June, 2001

1. Introduction:

"Numberless are the world's wonders,
but none more wonderful than man."
- Sophocles

1.1. About Neural Networks:

Neural networks, or *artificial neural networks* (ANN) to be more precise have been motivated right from its inception by the recognition that the human brain computes in an entirely different way from the conventional digital (von Neumann) computer. The brain is a highly *complex*, *nonlinear* and *parallel* computer (information processing system). It has the capacity to organise its structural constituents, known as *neurons*, so as to perform certain computations (e.g. pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. In its most general form, an artificial neural network is a machine that is designed to *model* the way in which the brain performs a particular task or function of interest. The ANNs are usually implemented by using electronic components or are simulated in software in a digital computer.

Neural networks present a technology that is rooted in many disciplines: neurosciences, mathematics, statistics, physics, computer-science and engineering. Neural networks find application in such diverse fields as modelling, time series analysis, pattern recognition, signal processing, and control by virtue of an important property: the ability to *learn* from input data with or without a teacher.

The use of neural networks offers the following useful properties and capabilities:

(1) *Nonlinearity*: an artificial neuron can be linear or nonlinear. Nonlinearity is a highly important property, particularly if the underlying physical mechanism responsible for generation of the input signal (e.g. speech signal) in inherently nonlinear.

(2) *Input-Output Mapping*: the network can learn an input-output mapping with a teacher with a method called supervised leaning. This involves modification of the synaptic weights by applying a set of labelled training samples.

(3) *Adaptivity*: neural networks have a built-in capacity to *adapt* their synaptic weights to changes in the surrounding environment.

(4) *Evidential Response*: In context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to be selected but also about the confidence in the decision made.

(5) *Contextual Information*: Knowledge is represented by the very structure and activation state of a neural network. Every neuron in the network is potentially affected by the global activity of all other neurons in the network.

Consequently, contextual information is dealt with naturally by a neural network.

(6) *Fault Tolerance*: a neural network implemented in a hardware form, has the potential to be inherently fault tolerant, or capable of robust computation, in the sense that its performance degrades gracefully under adverse operating conditions. Due the distributed nature of information stored in the network, a damage has to be extensive before the overall response of the network is degraded seriously. Thus, in principle, a neural network exhibits a graceful degradation in performance rather then a catastrophic failure.

(7) *VLSI Implementation*: The massively parallel nature of a neural network makes it potentially fast for computation of certain tasks. This same feature makes a neural network well suited for implementation using very-large-scale-integrated (VLSI) technology.

(8) *Uniformity of Analysis and Design*: Basically, neural networks enjoy universality as information processors, since the same notations is used in all domains involving the application of neural networks.

(9) *Neurobiological Analogy*: The design of neural network is motivated by analogy with the brain, which is a living proof that fault toleranct parallel processing is not only physically possible but also fast and powerfull. Neurobiologists look to (artificial) neural networks as a research tool for the interpretation of neurobiological phenomena.

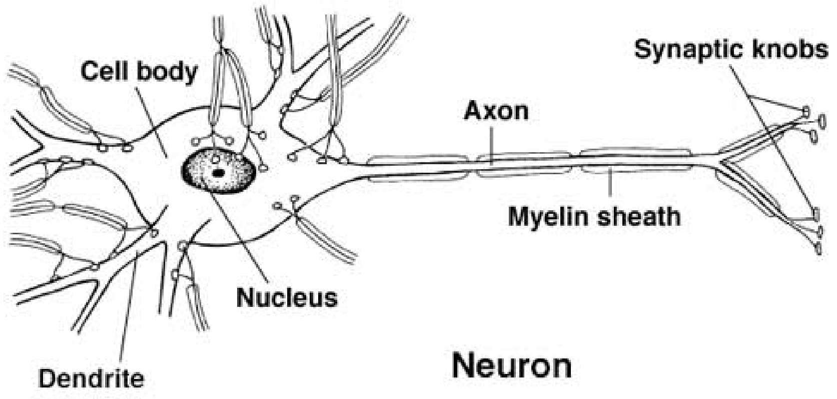


Figure 1.1
Draft picture of a "real" (non-artificial) neuron

1.2. Artificial Neurons:

A *neuron* is an information-processing unit that is fundamental to the operation of a neural networks. Figure 1.2 shows the *model* of a neuron, which forms the basis for designing (artificial) neural networks. The artificial neurons we use to build our neural networks are truly primitive in comparison to those found in the brain. An artificial neuron has several inputs but only one output. Here we identify four basic elements of the neuronal model:

(1) *Synapses* or *connecting links*, each is characterised by a *weight* or *strength* of its own. Specifically, a signal x_j at input of synapse j connected to a neuron is multiplied by the synaptic weight w_j.
(2) An *adder* for summing the input signals, weighted by the respective synapses of the neuron.
(3) An *activation function* for limiting the amplitude of the output of a neuron. Typically, the normalised amplitude of the output of a neuron is written as the closed unit interval [0,1] or alternatively [-1,1].
(4) The model of a neuron also includes an external *bias*, denoted by b, which has the effect of increasing or lowering the net input of the activation function.

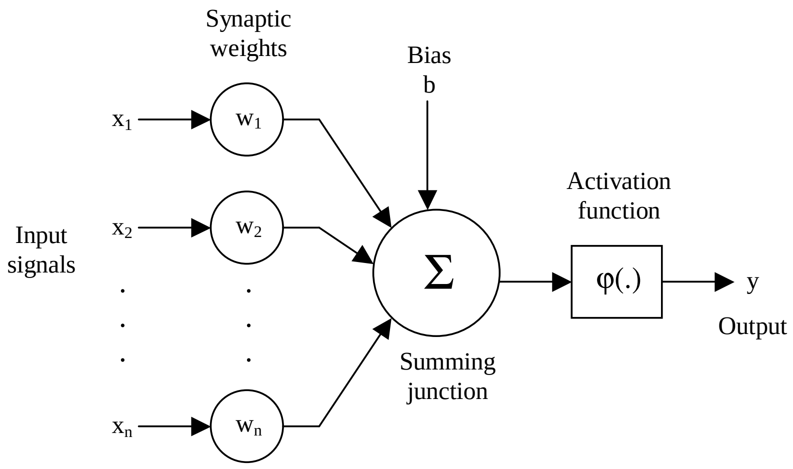


Figure 1.2
Nonlinear model of a neuron

In mathematical terms, we may describe a neuron by:

$$y = \varphi\left(\sum_{j=1}^{n} w_j x_j + b\right)$$

where $x_1,\ldots x_n$ are the input signals $w_1,\ldots,w_n$ are the synaptic weights of the neuron, b is the bias, φ is the activation function and y is the output signal of the neuron. Such a neuron (with a threshold type activation function) is referred in the literature as the *McCulloch-Pitts model,* in recognition of the pioneering work done by McCulloc and Pitts (1943).

1.3. Types of Activation Functions:

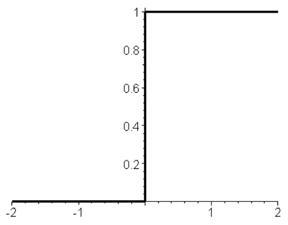
The activation function, denoted by $\varphi : \mathrm{R} \rightarrow \mathrm{R}$ defines the output of a neuron. During this paper when we speak of an activation function, we mean a function with the following assumptions: φ is *bounded* and:

$$\lim_{x \to +\infty} \varphi(x) = a \quad \text{and} \quad \lim_{x \to -\infty} \varphi(x) = b \qquad (a \neq b)$$

Here we identify three basic types of activation functions:

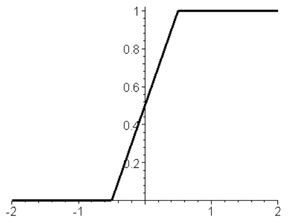
(1) *Threshold function*:

$$\varphi(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$



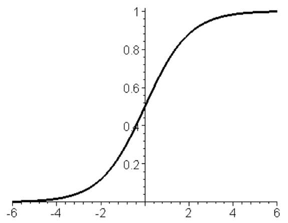
(2) *Piecewise linear function*:

$$\varphi(x) = \begin{cases} 1 & x \geq +\frac{1}{2} \\ x + \frac{1}{2} & \frac{1}{2} \geq x \geq -\frac{1}{2} \\ 0 & -\frac{1}{2} \geq x \end{cases}$$

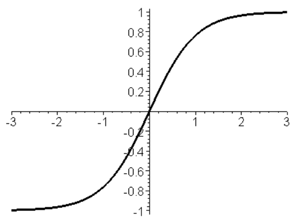


(3) *Sigmoid function (logistic function)*:

$$\varphi(x) = \frac{1}{1 + \mathrm{e}^{-x}}$$



The sigmoid function, whose graph is s-shaped, is by far the most common form of activation function used in construction of artificial neural networks. An example of the sigmoid function is the *logistic function*. An important feature of the sigmoid function that it is differentiable (whereas the threshold function is not).



These activation functions defined above range from 0 to +1. It is sometimes desirable to have the activation function range from –1 to +1 in which case the activation function assumes an antisymmetric form with respect to the origin. Specifically, the threshold function becomes the *signum function*. For the corresponding form of a sigmoid function we may define the *hyperbolic tangent function*:

$$\varphi(x) = \tanh(x)$$

Allowing an activation function of the sigmoid type to assume negative values.

1.4. Multilayer Feedforward Architecture:

In a *layered* neural network the neurons are organised in the form of layers. We have at least two layers: an *input* and an *output layer*. The layers between the input and the output layer (if any) are called *hidden layers*, whose computation nodes are correspondingly called *hidden neurons* or *hidden units*. The source nodes in the input layer of the network supply respective elements of the activation pattern (input vector), which constitute the input signals applied to the neurons (computation nodes) in the second layer (i.e., the first hidden layer). The output signals of the second layer are used as inputs to the third layer, and so on for the rest of the network. A layer of nodes projects onto the next layer of neurons (computation nodes), but not vice versa. In other words, this network is strickly a *feedforward* or *acyclic* type. The neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the output (final) layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer. Neural networks with this kind of architecture is also called as *multilayer perceptron*.

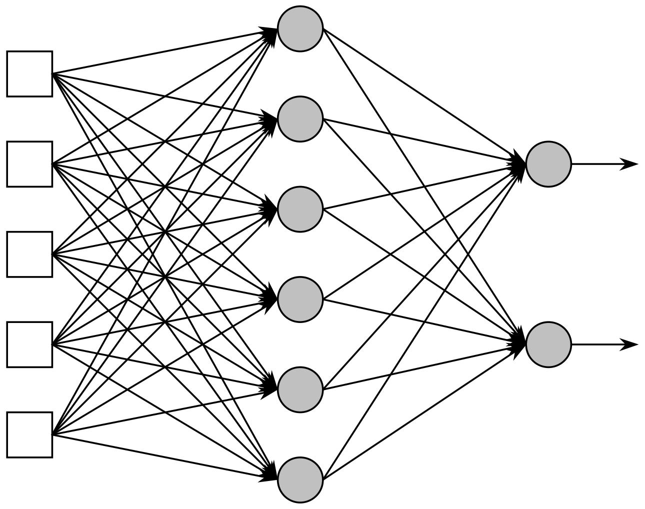


Figure 1.3
A fully connected feed-forward neural network with one hidden layer

The function of hidden neurons is to intervene between the external input and the network output in some useful manner. By adding one or more hidden layers, the network is enabled to extract high-order statistics. In a rather loose sense the network acquires a *global* perspective despite its local connectivity due the extra synaptic connections and the extra dimension of neural interactions. The neural network is said to be *fully connected* in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. (otherwise the network is called *partially connected*)

2. Computation Power:

"O, it is excellent to have a giant's strength;
but it is tyrannous to use it like a giant."
William Shakespeare

In this chapter we study the computation power of the feedforward artificial neural networks: we are interested in what kind of problems can be solved with neural networks? First we examine the feedforward neural networks without hidden neurons. Next we show that every boolean function can be computed by a feedforward neural network with one hidden layer. Finally we consider the *universal approximation theorem*, which claims that every continuous functions defined on a compact set can be arbitrary well approximated with a neural network with one hidden layer. At the end of the chapter we introduce a method called *arranging the neurons* that will help us building constructive proofs during the following chapters.

2.1. Single-Layer Perceptrons:

A single layer perceptron is the simplest form of a neural network used for the classification of patterns. Basically, it consists of a single neuron with adjustable synaptic weights and bias. It can be easily shown that a finite set of training samples can be classified correctly by a single-layer perceptron if and only if it is *linearly separable* (i.e. patterns with different type lie on opposite sides of a *hyperplane*). Thus e.g. if we look at the boolean functions (using the identification *true* = 1 and *false* = 0) it is clear that the "*and*" or the "*or*" functions can be computed by a single neuron (e.g. with the *threshold* activation function) but the "*xor*" (exclusive or) is not. A neuron can be trained with the *perceptron learning rule* given by Rosenblatt (1958).

2.2. Computation of Boolean Functions:

It is a well known result that if we use the *threshold* function as an activation function then we can compute every boolean functions with a multilayer perceptron with one hidden layer (using the identification *true* = 1 and *false* = 0). We look over the proof briefly: if we consider a function $f:\{0,1\}^n \to \{0,1\}$ then we can write f in *disjunctive normal form*[1], i.e. f can be written as a disjunction of clauses c_j:

$$f(x_1,...,x_n) = c_1 \vee c_2 \vee ... \vee c_p \qquad \text{where} \qquad c_j = l_{j1} \wedge l_{j2} \wedge ... \wedge l_{jq_j}$$

Each clause c_j is a conjunction of literals l_{jk}, and each literal l_{jk} is a parameter x_i or its negation $\neg x_i$. The values of a clause can be computed with one layer in the following way: assume that clause c is given by (we can rename the inputs if it is necessary):

$$c = x_1 \wedge ... \wedge x_r \wedge \neg x_{r+1} \wedge ... \wedge \neg x_{r+s}$$

[1] It is a standard result that every boolean function can be written in disjunctive normal form

To compute this clause we use a single artificial neuron: we can use $w_i = 1$ if $1 \le i \le r$, $w_i = -1$ if $r+1 \le i \le r+s$ and $w_i = 0$ if $r+s+1 \le i \le n$ with $b = 0.5 - r$:

$$0.5 - r + \sum_{i=1}^{n} w_i x_i = 0.5 - r + \sum_{i=1}^{r} x_i - \sum_{i=r+1}^{r+s} x_i \ge 0$$

$$\text{if and only if} \quad \forall i \in \{1,...,r\} : x_i = 1 \wedge \forall i \in \{r+1,...,r+s\} : x_i = 0$$

For the computation of *f* we now construct a multilayer perceptron: the first layer consists of *p* neuron that computes the clauses $c_1,...,c_p$ in the way described above. The output layer consists of one neuron with *p* inputs, that simply computes the disjunction of values of the *p* clauses. It is easily seen that the disjunction can be computed by a neuron with all weights equal to 1 and the threshold equals to –0.5. ■

Before we continue our examination of the computing power of neural networks, we formalise the output of a feedforward neural networks with one hidden layer and with one linear output unit as a function of the inputs:

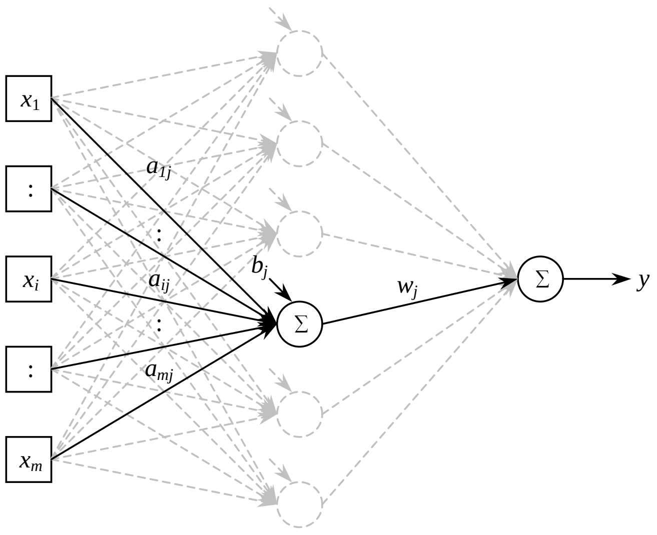


Figure 2.1
A feedforward neural network with one hidden layer

If we have a *multilayer perceptron* neural network with one hidden layer that consists of *n* hidden unit, and the network has *m* inputs (see Figure 2.1) then we can formalise the output of the network as a function of the inputs by:

$$(A_n f)(x_1,..., x_m) = \sum_{j=1}^{n} w_j \varphi(\sum_{i=1}^{m} a_{ij} x_i + b_j)$$

where a_{ij} is the weight of the synapse that goes form the input x_i to the j^{th} hidden neuron, b_j is the bias of the j^{th} hidden neuron, φ is the activation function, and w_i is the weights of the synapse that goes to the (linear) output neuron form the j^{th} neuron.

2.3. The Universal Approximation Theorem:

The universal approximation theorem claims that the standard multilayer feed-forward networks with a single hidden layer that contains finite number of hidden neurons, and with arbitrary activation function[2] are universal approximators in C(R^m). Kurt Hornik (1991) showed that it is not the specific choice of the activation function, but rather the *multilayer feedforward architecture* itself which gives neural networks the potential of being universal approximators. The output units are always assumed to be linear. For notational convenience we shall explicitly formulate our results only for the case where there is only one output unit. (The general case can easily be deduced from the simple case.) The theorem in mathematical terms:

Theorem 2.3.1 (universal approximation theorem):
Let $\varphi(.)$ be an arbitrary activation function. Let $X \subseteq R^m$ and X is compact. The space of continuous functions on X is denoted by C(X). Then $\forall f \in C(X)$, $\forall \varepsilon > 0$: $\exists n \in N$, a_{ij}, b_i, $w_i \in R$, $i \in \{1...n\}$, $j \in \{1...m\}$:

$$(A_n f)(x_1,..., x_m) = \sum_{i=1}^{n} w_i \varphi(\sum_{j=1}^{m} a_{ij} x_j + b_i)$$

as an approximation of the function f(.); that is

$$\| f - A_n f \| < \varepsilon$$

(In the notation $A_n f$, n shows the number of hidden neurons) How to measure the accuracy of approximation depends on how to measure closeness between functions, which in turn varies significantly with the specific problem to be dealt with. In many application, it is necessary to have the network *simultaneously* well on all input samples. In this case, closeness measured by the uniform distance between functions, that is:

$$\| f - A_n f \|_\infty = \sup_{\underline{x} \in X} | f(\underline{x}) - (A_n f)(\underline{x}) |$$

In other applications, we think of inputs as random variables and are interested in the *average performance*:

$$\| f - A_n f \|_p = \sqrt[p]{\int_X | f(\underline{x}) - (A_n f)(\underline{x}) |^p \, d\mu(\underline{x})}$$

$1 \leq p < \infty$, the most popular choice being p = 2, corresponding to the mean square error. Of course, there are many more ways of measuring closeness of functions. In particular, in many applications, it is also necessary that the *derivates* of the approximating function implemented by the network closely resemble those of the function to be approximated, up to some order.

[2] φ is an activation function if and only if φ is bounded and $\lim_{x \to +\infty} \varphi(x) = a$, $\lim_{x \to -\infty} \varphi(x) = b$, $a \neq b$

2.4. Arrangement of the Neurons:

Sometimes it is a good idea to build a different kind of *mother function* for the approximation from our activation functions. (We can build a function space with a translation and dilation of a mother function, and we want to prove that this function space is dense in another function space (e.g. C(R^m))) The simplest thing that we could do is to make couples from the neurons and build a function that has some useful property. (E.g. zero or “small” outside of an interval or forms a wavelet)

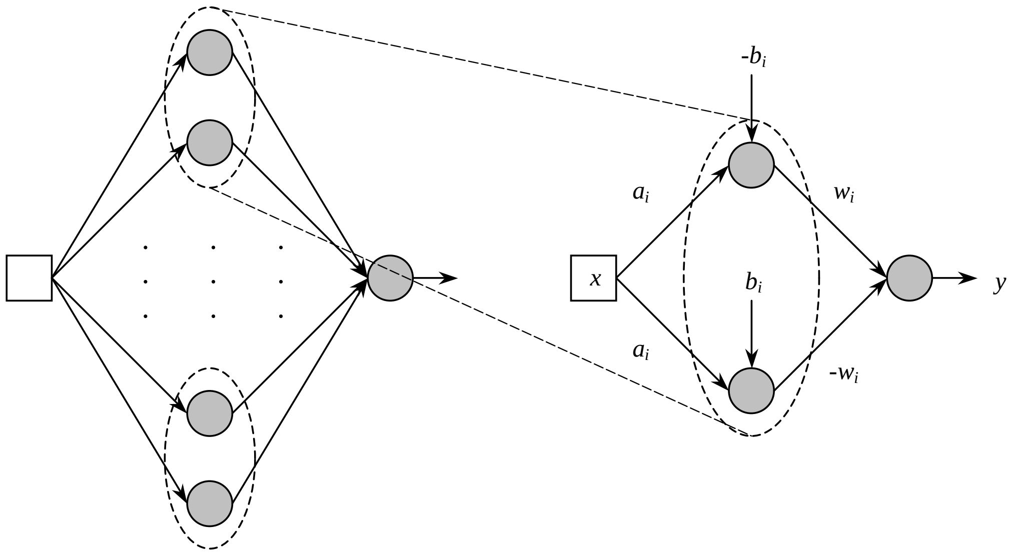


Figure 2.2
Arrangement of the neurons in couples

We can take the difference of two shifted activation function to build (define) our new basis/mother function:

$$\psi(x,a,b) = \varphi(ax+b) - \varphi(ax-b)$$

Thus, for example the approximation in R is looks like this (see Figure 2.2):

$$\psi_i(x) = \psi(x - x_i, a_i, b_i)$$
$$(A_{2n} f)(x) = \sum_{i=1}^{n} w_i \, \psi_i(x)$$

The method of arranging the neurons is only a notation, it doesn’t effect the neural network itself, but sometimes it is very useful for building a constructive proof. Sometimes (e.g. building a mother wavelet) we could use more then two neurons to build a new function.

3. Special Cases:

"If it doesn't kill me,
it makes me stronger."
- Friderich Nietzsche

Our main aim during this paper is to prove the *universal approximation theorem* in a constructive (and may be elementary) way and, if possible, get an estimation of the error. Instead of doing this directly in a general form, first we will try to prove the theorem for the three most popular type of activation functions: for the *threshold* (*step*) function, for the *piecewise linear* function and for the *logistic* (*sigmoid*) function. We do this not just because these cases have a great practical importance, but also because when we are dealing with the general case we will find these cases (especially the case of the threshold function) a great help. The main reason of this is that with our assumptions on the activation function (it has a limit in plus and minus infinity, it is bounded) we can mimic the approximation with the threshold function (with an other activation function). During this chapter we will assume that our compact set is the unit hypercube [0,1] and we will formulate our results only for $\mathrm{R} \to \mathrm{R}$ functions. We will give a very short introduction to the spline theory, because we will use that during the proof of some theorems.

3.1. About B-Splines:

The *B-splines* are piecewise functions which can be built by translation and dilation of the following mother functions:

$$\mathrm{B}_1(x) = \begin{cases} 1 & -\frac{1}{2} \le x < \frac{1}{2} \\ 0 & \textit{otherwise} \end{cases}$$

$$\mathrm{B}_2 = \mathrm{B}_1 * \mathrm{B}_1$$
$$\mathrm{B}_3 = \mathrm{B}_1 * \mathrm{B}_1 * \mathrm{B}_1, \text{ and so on}$$

It is clear that *B_1* is a unit step function, and is piecewise constant. It's called B-spline of degree 0 (order 1). It turns out that *B_2* is piecewise linear, *B_3* is piecewise quadratic, and so forth. We can calculate *B_2* as follows:

$$\mathrm{B}_2(x) = \int_{-\infty}^{\infty} \mathrm{B}_1(y)\,\mathrm{B}_1(x-y)\,dy = \int_{-1/2}^{1/2} \mathrm{B}_1(x-y)\,dy = \begin{cases} 1+x & -1 \le x \le 0 \\ 1-x & 0 < x \le 1 \\ 0 & \textit{elsewhere} \end{cases}$$

The theory of splines can be generalised to R^m (e.g. with *Box splines*) and it can be shown that every continuous function can be arbitrary well approximated with the linear combination of this kind of functions. We will show that with the method of *arranging the neurons* (we described this method in the second chapter) we can build up the spline basis and that makes the *spline theory* directly acceptable for our case.

3.2. Case of the Threshold Function:

In this part of the chapter, first we will give a constructive proof by the aid of spline-theory. After that we will build another approximation that has some optimality property. We will give an estimation of the error for that approximation, and as a consequence of this estimation we will prove the universal approximation theorem in an alternative way. Finally we will show that this approximation is optimal in the sense that if the biases are independent from the function that we want to approximate then no better approximation is possible.

Theorem 3.2.1 (approximation with the threshold function):
An arbitrary continuous function, defined on $[0,1]$ *can be arbitrary well uniformly approximated by a multilayer feed-forward neural network with one hidden layer (that contains only finite number of neurons) using McCulloch-Pitts model neurons*[3] *in the hidden layer and a linear neuron in the output layer. In mathematical terms:*

Let $\varphi(.)$ *be the threshold function. Then* $\forall f \in \mathrm{C}([0,1])$*,* $\forall \varepsilon > 0$: $\exists\, n \in \mathrm{N}$, w_i, $b_i \in \mathrm{R}$, $i \in \{0 \ldots n\}$:

$$(A_n f)(x) = \sum_{i=1}^{n} w_i \varphi(x + b_i)$$

as an approximation of the function f(.); that is

$$\sup_{x \in [0,1]} \left| (A_n f)(x) - \mathrm{f}(x) \right| < \varepsilon$$

Proof (3.2.1):
With the method of arranging the neurons, we can build the 1st order B-spline mother function (also known as the Haar function[4]):

$$\psi(x,1,\frac{1}{2}) = \varphi(x + \frac{1}{2}) - \varphi(x - \frac{1}{2}) = \mathrm{B}_1(x)$$

It follows from the *spline theory* that every continuous function could be arbitrary uniformly approximated by linear combination of the translation and dilation of this function. ■

Comment:
Not only the continuous functions can be arbitrary uniformly approximated with the Haar basis, but every function that $\forall \varepsilon > 0$ has finite number of split points which are larger than ε.

[3] artificial neurons with the threshold activation function
[4] Haar already proved the approximation for this kind of functions in 1910. The theory of splines and wavelets are generalisation of his work.

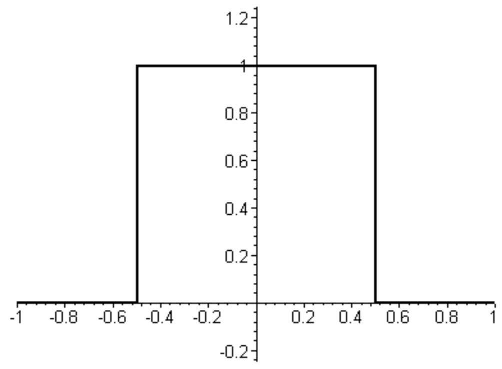
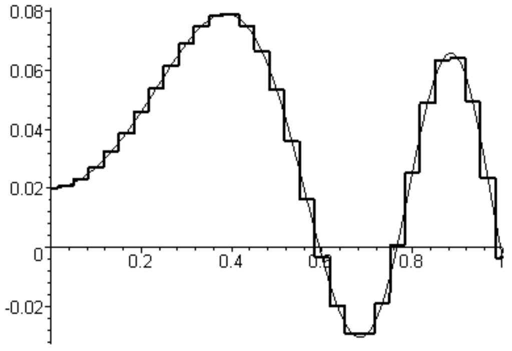



Figure 3.1
The mother function of the Haar basis (a = 1, b = ½); Approximation with the Haar basis (n = 30)

Comment:
If we want an approximation with n + 1 Haar functions, the simplest construction is:

$$(A_{n+1} f)(x) = \sum_{i=0}^{n} w_i \, \psi_i(x,b) \,, \; \forall i \in \{0...n\}: \; w_i = f(x_i) \,, \; b = \frac{1}{2n}$$

We excluded a from the parameters, because changing a has no real effect on the threshold function. Proving the approximation of the feed-forward neural networks by building the Haar basis (1st order spline basis) is theoretically correct, but that construction doesn't give us an effective way of approximating functions. We can notice that the number of used neurons can be reduced:

$$\sum_{i=0}^{n} w_i \, \psi_i(x,b) = \sum_{i=0}^{n} w_i \big(\varphi_i(x-b) - \varphi_i(x+b)\big) = w_0\varphi_0(x-b) + \sum_{i=1}^{n} (w_i - w_{i-1})\varphi_i(x-b) \,,$$

where $x_i = \dfrac{i}{n}$, $b = \dfrac{1}{2n}$, $\psi_i(x,b) = \psi(x - x_i, b)$, $\varphi_i(x) = \varphi(x - x_i)$, $\forall i \in \{0...n\}$.
At the left side 2(n+1) neurons required, but with a single trick, we can reduce it to: n+1. With the reduced number of neurons the weights of the approximation are:

$$(A_{n+1} f)(x) = \sum_{i=0}^{n} w_i \varphi_i(x-b)$$

$$w_0 = f(x_0)$$
$$w_i = f(x_i) - \sum_{j=0}^{i-1} w_j$$

By choosing the weights and the biases in a special way we can build an approximation with some “optimality” property. All approximation with the threshold function on [0,1] look like this:

$$0 \le x_1 < x_2 < \ldots < x_n \le 1$$
$$(A_n f)(x) = \sum_{i=1}^{n} w_i \varphi(x - x_i)$$

For the easier notations we will use $x_0 = 0$ and $x_{n+1} = 1$. It is clear that if our activation function is the *threshold* function then the approximation will be constant in $[b_i, b_{i+1})$ for every $0 \le i \le n$. If we use biases which aren’t depend on f (if we know nothing about the structure of the function that we want to approximate) then we can build an optimal approximation by choosing the parameters of the approximation $(0 \le i \le n)$:

$$x_i = \frac{i-1}{n}$$
$$w_i^{opt} = \frac{\chi(f,i)}{2} - \sum_{j=1}^{i-1} w_j$$

$$\chi(f,i) = \sup_{x \in [x_i, x_{i+1})} f(x) + \inf_{x \in [x_i, x_{i+1})} f(x)$$

$$(A_n^{opt} f)(x) = \sum_{i=1}^{n} w_i^{opt} \varphi(x - x_i)$$

First we have to prove that with this construction every continuous function can be arbitrary uniformly approximated. We will do this by estimating the error of the approximation. To do this we need the definition of the *modulus of continuity*:

$$\omega(f;\delta) = \sup\{ |f(x) - f(y)| : x, y \in D_f \wedge |x - y| \le \delta \}.$$

It is clear from the definition that if f is continuous and a_n has the property $\lim_{n\to\infty} a_n = 0$ then $\lim_{n\to\infty} \omega(f; a_n) = 0$. And $\forall \varepsilon > 0 : \omega(id; \varepsilon) = \varepsilon$, where *id* is the identity function. ($id(x) = x$)

Theorem 3.2.2 (estimating the error):

$$\forall n \in \mathrm{N}:\ \forall f \in [0,1] \to \mathrm{R}:\ \left\| f - A_n^{opt} f \right\|_\infty \le \frac{\omega(f; \frac{1}{n})}{2}$$

Proof (3.2.2):
First we prove that:

$$\forall i \in \{1 \ldots n\} : \forall x \in [x_i, x_{i+1}) : (A_n^{opt} f)(x) = \frac{\chi(f,i)}{2}$$

Fix an $i \in \{1...n\}$ and let $x \in [x_i, x_{i+1})$. It's clear from the construction that:

$$\varphi(x - x_i) = \begin{cases} 1 & j \le i \\ 0 & j > i \end{cases}$$

Then:

$$(A_n^{opt} f)(x) = \sum_{j=1}^{n} w_j^{opt} \varphi(x - x_j) = \sum_{j=1}^{i} w_j^{opt} = w_i^{opt} + \sum_{j=1}^{i-1} w_j^{opt} =$$
$$= \frac{\chi(f,i)}{2} - \sum_{j=1}^{i-1} w_j^{opt} + \sum_{j=1}^{i-1} w_j^{opt} = \frac{\chi(f,i)}{2}$$

We can formulate the error of the approximation on the interval $[x_i, x_{i+1})$ by:

$$d_i(f, A_n^{opt} f) = \sup_{x \in [x_i, x_{i+1})} \left| f(x) - (A_n^{opt} f)(x) \right| = \sup_{x \in [x_i, x_{i+1})} \left| f(x) - \frac{\chi(f,i)}{2} \right| \le \frac{\omega(f; \frac{1}{n})}{2}$$

The last step follows from:

$$\omega(f; \frac{1}{n}) = \sup\left\{ |f(x) - f(y)| : x, y \in [0,1] \wedge |x - y| \le \frac{1}{n} \right\} \ge$$
$$\ge \sup\left\{ |f(x) - f(y)| : x, y \in [x_i, x_{i+1}) \wedge |x - y| \le \frac{1}{n} \right\} =$$
$$= \sup\left\{ |f(x) - f(y)| : x, y \in [x_i, x_{i+1}) \right\} = \sup_{x \in [x_i, x_{i+1})} f(x) - \inf_{x \in [x_i, x_{i+1})} f(x) =$$
$$= 2 \cdot \sup\left\{ \left| f(x) - \frac{\chi(f,i)}{2} \right| : x \in [x_i, x_{i+1}) \right\}$$

The total error of the approximation is the maximal error on the intervals:

$$\left\| f - A_n^{opt} f \right\|_\infty = \max_{i \in \{1...n\}} d_i(f, A_n^{opt} f) \le \frac{\omega(f; \frac{1}{n})}{2}. \quad \blacksquare$$

Corollary 3.2.3 (convergence):
If f is continuous then $\lim_{n \to \infty} \left\| f - A_n^{opt} f \right\|_\infty = 0$.

Theorem 3.2.4 (optimality):
$\forall n \in \mathbb{N}$: $A_n^{opt} f$ *is optimal in the sense that if the biases are independent from f then if*

$$\exists i \in \{1...n\} : x_i \ne \frac{i-1}{n} \quad \vee \quad \exists i \in \{1...n\} : w_i \ne \frac{\chi(f,i)}{2} - \sum_{j=1}^{i-1} w_j$$

then:

$$\exists\, f \in [0,1] \to \mathrm{R} : \left\| f - A_n f \right\|_\infty > \frac{\omega(f;\frac{1}{n})}{2}$$

Proof (3.2.4):

Case 1: Assume that $\exists i \in \{1...n\}: x_i \neq \frac{i-1}{n}$. In that case $\exists\, j \in \{0...n\}: x_{j+1} - x_j > \frac{1}{n}$. ($x_0$ = 0, x_{n+1} = 1). Let $\delta = x_{j+1} - x_j$. It is clear that $A_n f$ is constant on $[x_j, x_{j+1})$. Let $f = id$. Then $d_j(f, A_n f) \geq \frac{\delta}{2}$. Thus $\left\| f - A_n f \right\|_\infty \geq \frac{\delta}{2} > \frac{1}{2n} = \frac{\omega(id;\frac{1}{n})}{2}$.

Case 2: We can assume that $\forall i \in \{1...n\}: x_i = \frac{i-1}{n}$ otherwise we already proved in Case 1 that exists such an *f*. Assume that $\exists i \in \{1...n\}: w_i \neq \frac{\chi(f,i)}{2} - \sum_{j=1}^{i-1} w_j$. Let *f* = *id*. In that case $\exists\, j \in \{0...n\}: \forall x \in [x_j, x_{j+1}) : (A_n f)(x) = c \neq \frac{\chi(id,i)}{2} = x_j + \frac{1}{2n}$. If $c > \frac{\chi(f,i)}{2}$ then $\left| f(x_j) - (A_n f)(x_j) \right| = c - x_j > x_j + \frac{1}{2n} - x_j = \frac{1}{2n}$. If $c < \frac{\chi(f,i)}{2}$ then $\lim_{x \uparrow x_{j+1}} \left| f(x) - (A_n f)(x) \right| = x_{j+1} - c > x_{j+1} + \frac{1}{2n} - x_{j+1} = \frac{1}{2n}$. ■

3.3. Case of the Piecewise Linear Function:

Theorem 3.3.1 (approximation with the piecewise linear function):
Let $\varphi(.)$ be the piecewise linear function. Then $\forall f \in \mathrm{C}([0,1])$, $\forall \varepsilon > 0$: $\exists n \in \mathrm{N}$, w_i, a_i, $b_i \in \mathrm{R}$, $i \in \{0...n\}$:

$$(A_n f)(x) = \sum_{i=1}^{n} w_i \varphi(a_i x + b_i)$$

as an approximation of the function f(.); that is

$$\sup_{x \in [0,1]} \left| f(x) - (A_n f)(x) \right| < \varepsilon$$

Proof (3.3.1):
With the method of arranging the neurons, we can build the mother function (the *tent function*) of the 2nd order (1st degree) spline basis:

$$\psi(x, 1, \frac{1}{2}) = \varphi(x + \frac{1}{2}) - \varphi(x - \frac{1}{2}) = \mathrm{B}_2(x)$$

It's a well-known result that every continuous function can be arbitrary uniformly approximated with the 2nd order spline basis. ■

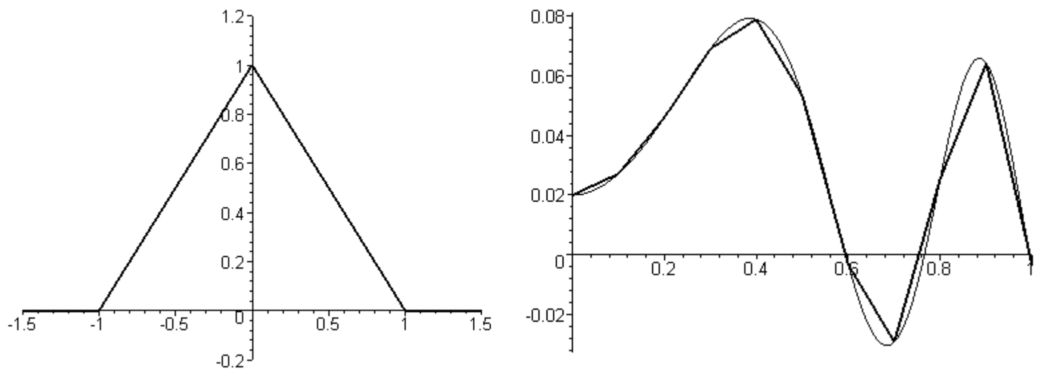


Figure 3.2
The tent mother function; Approximation with tent functions (n = 10)

Comment:
We can use the trick of reducing the number of neurons again, and the similar recursive formulas are valid for the weights as in case of the threshold function.

3.4. Case of the Logistic Function:

Theorem 3.4.1 (approximation with the logistic function):
An arbitrary continuous function, defined on $[0,1]$ *can be arbitrary well uniformly approximated by a multilayer feed-forward neural network with one hidden layer using the logistic function as activation function* (φ).

Idea of a proof (3.4.1):
With the notations of theorem 7.2. We can build a bell shaped basis/mother function again with the trick of arranging the neurons (we can take the difference of two shifted logistic function):

$$\psi(x,a,b) = \varphi(ax+b) - \varphi(ax-b)$$
$$\psi_i(x,a) = \psi(x-x_i,a,b) \qquad x_i = \frac{i}{n}$$

Without the loss of generality we can fix *b* to an arbitrary nonzero value. We can notice that the width of the bells depends only on the parameter *a* and the height of the bells depends on the parameter *b*. For the proof we want the weighted sum of our basis functions to interpolate at the points x_i:

$$(A_{2(n+1)} f)(\underline{w},\underline{a},x) = \sum_{j=0}^{n} w_j \, \psi_j(x,a_j)$$

$$\forall i \in \{0 \ldots n\}: \ (A_{2(n+1)} f)(\underline{w},\underline{a},x_i) = \mathrm{f}(x_i)$$

Now we can define the following vectors and matrix to compute the weights:

$$\underline{b} = \left[\mathrm{f}(x_0) \quad \mathrm{f}(x_1) \quad \ldots \quad \mathrm{f}(x_n)\right]$$

$$\underline{w} = \left[w_0 \quad w_1 \quad \ldots \quad w_n\right]$$

$$\underline{\underline{G}}(\underline{a}) = \begin{bmatrix} \psi_0(x_0, a_0) & \psi_0(x_1, a_0) & \cdots & \psi_0(x_n, a_0) \\ \psi_1(x_0, a_1) & \psi_1(x_1, a_1) & \cdots & \psi_1(x_n, a_1) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_n(x_0, a_n) & \psi_n(x_1, a_n) & \cdots & \psi_n(x_n, a_n) \end{bmatrix}$$

$$\underline{w} \cdot \underline{\underline{G}}(\underline{a}) = \underline{b}$$

$$\underline{w} = \underline{b} \cdot \underline{\underline{G}}(\underline{a})^{-1}$$

We have to prove that $\underline{\underline{G}}(\underline{a})^{-1}$ is exists. We can use Gershgorin's theorem to estimate the eigenvalues of the matrix, because our G matrix has it's large elements in it's diagonal (with a suitable $\underline{a}$):

$$\underline{\underline{G}}(\underline{a}) \approx \begin{bmatrix} big & small & \cdots & small \\ small & big & \cdots & small \\ \vdots & \vdots & \ddots & \vdots \\ small & small & \cdots & big \end{bmatrix}$$

Note that we can find Gershgorin's theorem at the end of the chapter. Thus we can claim that $\underline{\underline{G}}(\underline{a})^{-1}$ exists if:

$$\forall i \in \{0 \ldots n\}: \sum_{j \neq i} \psi_i(x_j, a_i) < \psi_i(x_i, a_i)$$

We can notice that $\psi_i(x_i, a_i)$ does not depends on a_i and it is equal to $\psi_0(0) = \varphi(b) - \varphi(-b)$. We should find a suitable $\underline{a}$ (for a specific n) which satisfies this inequality and we have to prove that the error of the approximation is getting smaller if n getting larger to prove the theorem. □

Comment:
A theoretically correct proof will be shown in the next chapter for the approximation with the *logistic* function. The reason that we have included this kind of approach is that this will give a practically effective way of approximation (if we could guess a good $\underline{a}$). We don't even need to use the bells: if we use directly the *logistic* functions then the $\underline{\underline{G}}(\underline{a})$ will be upper triangonal dominant, and will be invertable for a suitable $\underline{a}$, but the proof will be more difficult. (But for practical reasons that is the advised way)

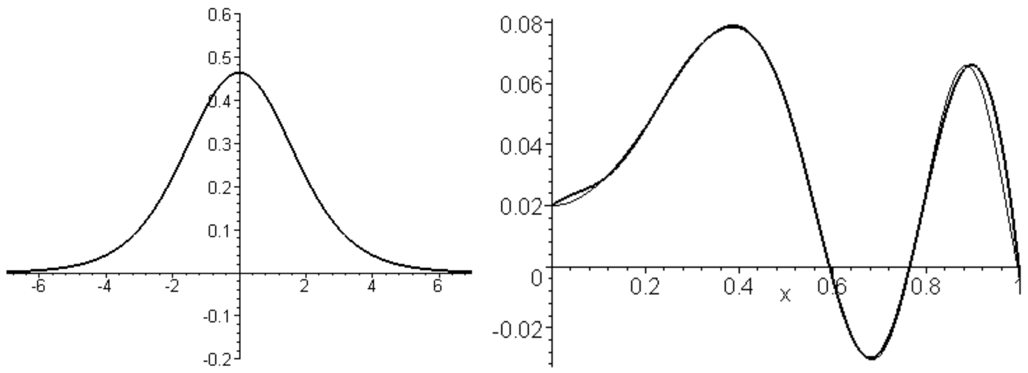


Figure 3.3
The bell shaped basis function (a = 1, b = 1); Approximation with the bells (n = 8)

Hypothesis:
This kind of approach can be generalised to any monotone-increasing activation function that has $\varphi(0) \neq 0$. For example for the *threshold function* and the *piecewise linear function*. (See *Appendix$_1$* for an example in *MapleV*)

We used Gershgorin's theorem during the idea of proof of *theorem 3.4.1*:

Gershgorin's theorem:
Each eigenvalue of $A = (a_{ij})_{i,j=1}^{n}$ lies in at least one of the Gershgorin disks:

$$\left\{ z : \left| z - a_{ii} \right| \leq \sum_{j \neq i} \left| a_{ij} \right| \right\} \qquad i \in \{1..n\}$$

Comment:
If each pair of the *n* Gershgorin disks has an empty intersection, then each disk contains exactly one eigenvalue of *A*, which is therefore simple.

4. General Case:

"We think in generalities,
but we live in detail."
Alfred North Whitehead

Now we consider a more general case: in the first part of this chapter we prove the approximation for an arbitrary activation function, however we still restrict ourselves to one dimension. Generalisation to higher dimensions is nontrivial. See the *Epilogue* for more details about this problem. We use our results from the previous chapter (*Special Cases*), specially the results for the threshold function. In the second part of this chapter we present a theorem given by Debao Chen in 1993 that directly acceptable to estimate the error of a multilayer feedforward artificial neural network with a given number of hidden units.

4.1. Approximation with Arbitrary Activation Function:

Theorem 4.1.1 (universal approximation theorem):
An arbitrary continuous function, defined on [0,1] *can be arbitrary well uniformly approximated by a multilayer feed-forward neural network with one hidden layer (that contains only finite number of neurons) using neurons with arbitrary activation functions in the hidden layer and a linear neuron in the output layer. Formally:*

Let $\varphi(.)$ *be the arbitrary activation*[5] *function. Then* $\forall f \in \mathrm{C}([0,1])$, $\forall \varepsilon > 0$: $\exists\, n \in \mathrm{N}$, w_i, a_i, $b_i \in \mathrm{R}$, $i \in \{0...n\}$:

$$(A_n f)(x) = \sum_{i=1}^{n} w_i \varphi(a_i x + b_i)$$

as an approximation of the function f(.); that is

$$\sup_{x\in[0,1]} \left|(A_n f)(x) - \mathrm{f}(x)\right| < \varepsilon$$

Proof (4.1.1):
Because φ is an activation function: it is bounded and we can assume without the loss of generality that $\lim_{x\to+\infty} \varphi(x) = 1$ and $\lim_{x\to-\infty} \varphi(x) = 0$. First we will assume that $\forall x : \varphi(x) \in [0,1]$. We already proved the universal approximation theorem in the *theorem 3.2.1*, *3.2.3* in two different ways in the case when our activation function is the *threshold* (*step*) function. We also have an estimation of the error for this case in *theorem 3.2.2*. We will show that every function that can be approximated by a linear combination of threshold functions $\tau(x - x_i)$, can also be approximated by a linear combination of functions of the form $\varphi(m(x - x_i))$. The idea is simply that a threshold function $\tau(x - x_i)$ can be approximated by $\varphi(m(x - x_i))$ if m is large

[5] φ is an activation function if and only if φ is bounded and $\lim_{x\to+\infty} \varphi(x) = a$, $\lim_{x\to-\infty} \varphi(x) = b$, $a \neq b$

enough. We will show that if we have an approximation with a maximal error of ε by a linear combination of threshold functions then there exists an $m > 0$ such that if we change all the $\tau(x - x_i)$ into $\varphi(m(x - x_i))$ then the error of the approximation will be less or equal then 4ε.

Let an approximation of f by a linear combination of threshold functions with a maximal error of ε be given by:

$$(A_n^{\text{th}} f)(x) = \sum_{i=1}^{n} a_i \tau(x - x_i)\,.$$

It is not a restriction to assume that the points x_i are increasing and in the interval [0,1]: $0 \le x_1 < x_2 < \ldots < x_n \le 1$.

Since f is continuous at the points x_i and $| f(x) - \sum_{i=1}^{n} a_i \tau(x - x_i) | < \varepsilon$, we obtain that for $i > 0$, the coefficients $|a_i| < 2\varepsilon$. Only if $x_1 = 0$, this may not hold for $i = 0$. Therefore we assume for the moment that either $x_1 > 0$ or that $|a_1| < 2\varepsilon$. We now compute the maximal error of the approximation:

$$(A_n f)(x) = \sum_{i=1}^{n} a_i \varphi(m(x - x_i)).$$

We define δ by $\delta = \frac{1}{2} \min_{i \ne j} | x_i - x_j |$. We now consider two cases:

1. Suppose that $| x - x_i | \ge \delta$ for all $i \in \{0, \ldots, n\}$. Then the limiting behaviour of φ implies that $| \tau(x - x_i) - \varphi(m(x - x_i)) |$ can be made smaller than a given ε_1 by taking m large enough. This implies that

$$\left| \sum_{i=1}^{n} a_i \tau(x - x_i) - \sum_{i=1}^{n} a_i \varphi(m(x - x_i)) \right| < \varepsilon_1 \sum_{i=1}^{n} | a_i |$$

   if m is large enough.

2. Suppose that $| x - x_j | < \delta$ for some $j \in \{0, \ldots, n\}$. Then

$$| a_j \tau(x - x_j) - a_j \varphi(m(x - x_j)) | < | a_j | < 2\varepsilon.$$

   For $i \ne j$ we infer from the definition of δ that $| x - x_i | \ge \delta$, which means that these terms can be bounded in the same way as in case 1.

Summarising we have shown that for all $x \in [0,1]$ that

$$|\sum_{i=1}^{n} a_i \tau(x-x_i) - \sum_{i=1}^{n} a_i \varphi(m(x-x_i))| < \varepsilon_1 \sum_{i=1}^{n} |a_i| + 2\varepsilon$$

Since ε_1 can be made as small as needed by increasing m, the right hand side of this term can be made smaller than 3ε. Our assumption was that the error of the approximation with the threshold function is at most ε, hence we have shown that f can be approximated uniformly on [0,1] by $\sum_{i=1}^{n} a_i \varphi(m(x-x_i))$ with an error at most $3\varepsilon + \varepsilon = 4\varepsilon$. It remains to get rid of the assumption that either $x_1 > 0$ or that $|a_1| < 2\varepsilon$. For if $x_1 = 0$ and $a_1 \geq 2\varepsilon$, then in case 2 the term $|a_j \tau(x-x_j) - a_j \varphi(m(x-x_j))|$ cannot be bounded by 2ε. However in this case $a_1 \tau(x-x_1) = a_1$, and this can be approximated uniformly by $a_1 \varphi(m(x+b))$ by taking b large enough. Now we go back to the assumption: $\forall x : \varphi(x) \in [0,1]$. If it not holds, then we must use the bound ($B = \|\varphi\|_\infty = \sup_{x \in (-\infty,\infty)} |\varphi(x)|$) in our estimations, that makes them slightly more complicated: In the first case ($|x-x_i| \geq \delta$ for all $i \in \{0,...,n\}$) we can get the same estimation, but in the second case ($|x-x_j| < \delta$ for some $j \in \{0,...,n\}$) we can only get:

$$|a_j \tau(x-x_j) - a_j \varphi(m(x-x_j))| < (B+1)|a_j| < 2\varepsilon(B+1)$$

which implies:

$$|\sum_{i=1}^{n} a_i \tau(x-x_i) - \sum_{i=1}^{n} a_i \varphi(m(x-x_i))| < \varepsilon_1 \sum_{i=1}^{n} |a_i| + 2\varepsilon(B+1) < (2B+2)\varepsilon$$

Thus we proved, that error of the new approximation will be at most $(2B + 3)\varepsilon$, thus the convergence is also proved in that way. ■

In a concrete case the bound given in this theorem can be improved: assume for instance that $\varphi(x) = \frac{1}{1+e^{-x}}$, which means we consider the standard logistic function. A simple computation shows that now: $|\tau(x-x_i) - \varphi(m(x-x_i))| \leq \max(e^{-m|x-x_i|}, \frac{1}{2})$

Then it is easily verified that the bound above can be replaced by

$$|\sum_{i=1}^{n} a_i \tau(x-x_i) - \sum_{i=1}^{n} a_i \varphi(m(x-x_i))| < 2Be^{-m\delta} + 2Be^{-3m\delta} + 2Be^{-5m\delta} + ... + \varepsilon =$$

$$= 2B \frac{e^{-m\delta}}{1-e^{-2m\delta}} + \varepsilon$$

where $B = \max_i |a_i|$. This allows to compute the actual value of m in a concrete case.

4.2. Debao Chen’s Theorem:

Definition:
Fixing an activation function φ and an integer *n*, we consider the class of functions of the form ($x\in R$):

$$g(x)=\sum_{i=1}^{n} a_i\varphi(mx+k_i)$$

By varying the parameters $a_i\in R$, $m\in N$, and $k_i\in Z$, we obtain the class being considered, and we denote it by $\Phi(\varphi,n)$. A question about “degree” of approximation can now be posed: How well can a function $f\in C([0,1])$ be approximated by elements of $\Phi(\varphi,n)$? To make the question precise, define:

$$\mathrm{dist}(f,\Phi(\varphi,n))=\inf\left\{\|f-g\|_\infty : g\in\Phi(\varphi,n)\right\}$$

Theorem 4.2.1 (by Debao Chen in 1993):

$$\forall f\in C([0,1]):\ \ \mathrm{dist}(f,\Phi(\varphi,n))\le\|\varphi\|\,\omega(f,\frac{1}{n}),$$

where $\|\varphi\|=\sup\limits_{x\in(-\infty,\infty)}|\varphi(x)|$.

Idea of the proof (4.2.1):
Because *f* is an activation function it is bounded and we can assume without the loss of generality that $\lim\limits_{x\to+\infty}\varphi(x)=1$ and $\lim\limits_{x\to-\infty}\varphi(x)=0$. Having a fixed *n* we define:

$$(L_m f)(x)= f_0\varphi(mx+m)+\sum_{i=1}^{n-1}(f_{i-1}-f_i)\varphi(mx-mx_i),$$
$$\text{where } x_i=\frac{i}{n},\ \ f_i=f(x_i),\ i\in\{0..n\}$$

Observe that if *m* is multiple of *n*, then $L_m f\in\Phi(\varphi,n)$. Debao Chen showed that if *m* goes to infinity through the multiples of *n* then the error goes to the expected one. □

Comment:
The complete proof can be found in the book of Ward Cheney and Will Light (A Course in Approximation Theory).

Comment:
With this theorem of Debao Chen the Universal Approximation Theorem is proved, because: $\|\varphi\|$ is constant and $\lim\limits_{n\to\infty}\omega(f;\frac{1}{n})=0$ if *f* is continuous.

5. Improving the Approximation:

“To find a fault is easy;
to do better may be difficult.”
- Plutarch

Now we try to improve the approximation with a multiresolution approach. First we will present an algorithm that gives such an approximation with a monotone increasing activation function. This algorithm has been motivated by wavelets, so it will help us understand the wavelet approach. The idea of the multiresolution is that first we build a raw approximation with a few (e.g. one) neurons, and after we add some details by approximating the error on smaller (e.g. half size) intervals. After this we can add some more details again by approximating the error on smaller intervals, and so on. The advantages of the multiresolution approach are:

- Suppose we have a artificial neural network that approximates a function with ε_1 error and we want a better approximation, e.g. with $\varepsilon_2 < \varepsilon_1$ error. With the algorithm presented in the previous chapters we need to build a completly new neural network, but with the multiresolution algorithm we can simply add a few more neurons to the original neural network to improve the accuracy of the approximation.

- If we have a physically built artificial neural network, then one of its useful properties is its robustness. But with the methods defined in the previous chapters, the neural network is unstable in the sense that if one of the neurons is damaged, then the whole networks goes wrong. But with this multiresolution approach there is a great probability that if a neuron is damaged, then the error is increasing only slightly in a small interval.

Disadvantage of this approach that in the most cases the neural network is redundant. Later we will define *wavelets* and prove that we can build a *mother wavelet* with the linear combination of several *logistic* function. We will prove that with this kind of wavelets a numerically stable computation of the wavelet coefficients (synaptic weights) is possible because these wavelets (the wavelets that are generated from our mother wavelet) form a *frame*.

5.1. General Multiresolution Algorithm:

We assume, that our activation function is *monotone-increasing* and:

$$\lim_{x\to+\infty}\varphi(x)=1, \qquad \lim_{x\to-\infty}\varphi(x)=0 \qquad \text{and} \qquad \varphi(0)\neq 0$$

And the function *f* that we want to approximate is continuous on [0,1]. Our main aim is to construct an approximation with the following property:

$$\forall x\in[0,1]: \sum_{n=1}^{\infty}\sum_{i=1}^{2^{n-1}} w_{n,i}\varphi_{n,i}(x) = f(x)$$

We define the sequence of the knots ($\xi_{n,i}$) and the shifts ($\eta_{n,i}$) by:

$$\xi_{n,i} = \frac{2i-1}{2^n} \quad \text{and} \quad \eta_{n,i} = \frac{i-1}{2^{n-1}}$$

We can easily check, that the knots and shifts has following properties $i \in \{1...2^{n-1}\}$:

$$\eta_{n,i} = \eta_{n+1,2i-1}, \qquad \eta_{n,i} < \xi_{n,i} < \eta_{n,i+1}, \qquad \eta_{n,i+1} - \eta_{n,i} = \frac{1}{2^{n-1}}$$

The approximation in the step *n* will approximate *f* in the knots ($\xi_{n,i}$). We define our basis functions by (a_n is the scaling parameter, it is something like 2^n)

$$\varphi_{n,i} = \varphi(a_n(x - \eta_{n,i}))$$

Then the nth level of the approximation $(L_n f)(x)$ and the approximation in the step *n* $(A_n f)(x)$ are:

$$(L_n f)(x) = \sum_{i=1}^{2^{n-1}} w_{n,i}\varphi_{n,i}(x)$$

$$(A_n f)(x) = \sum_{i=1}^{n} (L_i f)(x),$$

where $w_{i,j}$ is such that:

$$\forall i \in \{1...2^{n-1}\} : (A_n f)(\xi_{n,i}) = f(\xi_{n,i})$$

It is clear that for the first level it must be: $w_{1,1} = \dfrac{f(\xi_{1,1})}{\varphi_{1,1}(\xi_{1,1})}$

If our activation function is such that: $\varphi(x) = 0$ if $x < 0$ (like the *threshold* function) then we can define the weights like:

$$\forall i \in \{1...2^{n-1}\} : w_{n,i} = \frac{f(\xi_{n,i}) - (A_{n-1} f)(\xi_{n,i})}{\varphi_{n,i}(\xi_{n,i})} - \sum_{j=1}^{i-1} w_{n,j}\varphi_{n,j}(\xi_{n,i})$$

In the general case it can be done by matrix inversion (suppose we computed the weights for all the levels lesser then *n*):

$$\underline{\underline{G_n}} = \left(\varphi_{n,i}(\xi_{n,j})\right)_{i,j=1}^{2^{n-1}}, \qquad \underline{F_n} = \left(f(\xi_{n,i}) - (A_{n-1} f)(\xi_{n,i})\right)_{i=1}^{2^{n-1}}, \qquad \underline{W_n} = \left(w_{n,i}\right)_{i=1}^{2^{n-1}}$$

$$\underline{W_n} \cdot \underline{\underline{G_n}} = \underline{F_n} \Leftrightarrow \underline{W_n} = \underline{F_n} \cdot \underline{\underline{G_n}}^{-1}$$

(Of course, first we have to prove that $\det(\underline{\underline{G_n}}) \neq 0$) If we use the threshold function as activation function then we can prove the convergence of our algorithm in a similar way that we used in the *theorem 3.2.3*. In that case we can get an upper estimation of the error:

$$\forall n \in \mathrm{N}:\ \forall f \in [0,1] \to \mathrm{R}:\ \left\| f - A_n f \right\|_\infty \leq \omega(f; \frac{1}{2^{n-1}})$$

It is an important thing that we recognise that $A_n f$ requires 2^n-1 hidden neurons with this algorithm! So as we see in Debao Chen's theorem the error could be smaller thus this algorithm doesn't give an optimal approximation. It is redundant. We lost the optimality, but we gain some other good properties (see the beginning of this chapter). It depends on the task what kind of approximation must we choose. For practical reasons it is advised to swap the activation function at every level of the approximation to equalise that the functions have their small values (because they are monotone-increasing) at the beginning of the interval and their large values at the end of the interval (if we do so, we need a little change in the shift sequence). In the *Appendix*$_2$ we can find a *Matlab* program that computes the approximation with that algorithm. Figure 5.1 shows how the neurons in the hidden layer are arranged to form the levels of the multiresolution approximation.

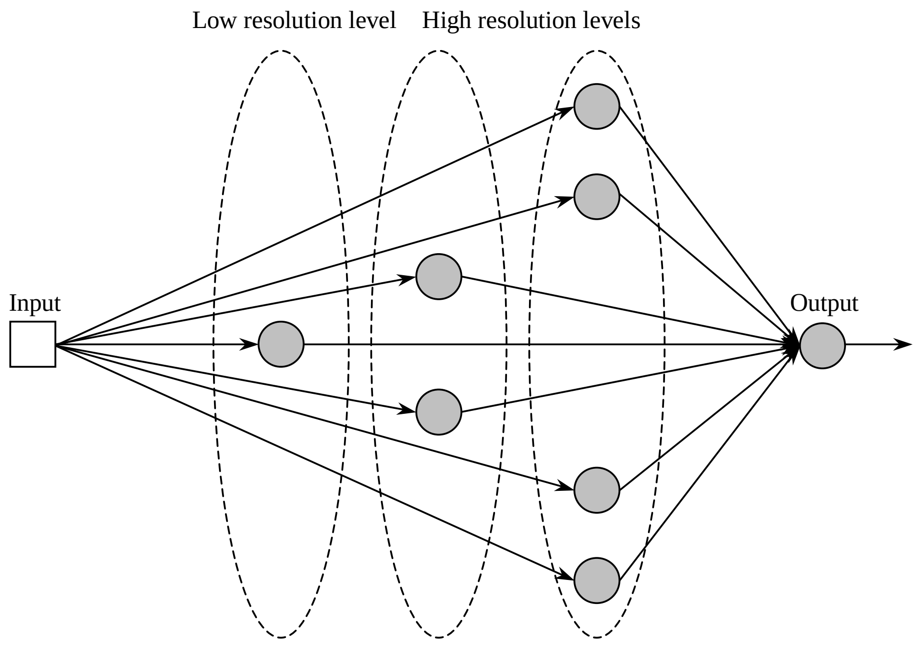


Figure 5.1
Arranging the neurons in the only hidden layer into the levels of the multiresolution approximation

5.2. About Wavelets:

The goal of most modern wavelet research is to create a set of basis functions (or general expansion functions) and transforms that will give an informative, efficient, and useful description of a function or a signal. If the signal is represented as a function of time, wavelets provide efficient localisation in both time and frequency or scale. A space that is particularly important in signal processing is the $L^2(R)$. We use that space in this chapter instead of C(R). Consequently the used norm is the 2nd norm instead of the supremum norm. We formalise our results in the whole real line, but if we want to build a wavelet approximation with a feedforward neural network with *finite* number of hidden units then we must restrict ourselves to a compact set. Another central idea of the wavelets is the *multiresolution* where the decomposition of a signal is in terms of resolution of detail. Before delving into the details of wavelets and their properties, we need to get some idea of their general characteristics and what we are going to do with them: a signal or function *f*(*x*) can often be better analysed, described, or processed if expressed as a linear decomposition by:

$$f(x) = \sum_l a_l \, \psi_l(x)$$

where l is an integer index for the finite or infinite sum, a_l are the real-values expansion coefficients, and $\psi_l(t)$ are a set of real-valued functions of t called the expansion set. If the expansion is unique, the set is called *basis* for the class of functions that can be so expressed. If the basis is *orthogonal*, meaning:

$$\langle \psi_k, \psi_l \rangle = \int_{-\infty}^{\infty} \psi_k(x)\, \psi_l(x) dx = 0 \qquad k \neq l$$

with the usual *inner product*, then the coefficients can be calculated by:

$$a_k = \langle f, \psi_k \rangle = \int_{-\infty}^{\infty} f(x)\, \psi_k(x) dx$$

If the basis set is not orthogonal, then a dual basis set $\tilde{\psi}_k(t)$ exists such that the inner products with the dual basis gives the desired coefficients. For the *wavelet expansion*, a two-parameter system is constructed such that:

$$f(x) = \sum_k \sum_j a_{j,k} \, \psi_{j,k}(x)$$

where both j and k are integer indices and the $\psi_{j,k}(t)$ are the wavelet expansion functions that usually form an orthogonal basis. The wavelet expansion set is not unique. There are many different wavelet systems that can be used effectively. The set of expansion coefficients $a_{j,k}$ are called *discrete wavelet transform* (DWT). If the signal is a function of continuous variable and a transform that is a function of two continuous variables is desired, the *continuous wavelet transform* (CWT) can be defined by (if the wavelets are not orthogonal, then we must use the dual basis again):

$$F(a,b) = \int_{-\infty}^{\infty} f(x)\,\psi_{a,b}(x)dx\,,$$

with an inverse transform of:

$$f(x) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(a,b)\,\psi_{a,b}(x)da\,db\,,$$

where $\psi_{a,b}(x)$ is usually defined by ($a \neq 0$):

$$\psi_{a,b}(x) = |a|^{-\frac{1}{2}}\,\psi\left(\frac{x-b}{a}\right)$$

In both cases (the discrete and the continuous wavelet transform) we will assume that:

$$\int_{-\infty}^{\infty} \psi(x)dx = 0$$

We can notice, that if our activation function is a *mother wavelet* then we can build up a function by a feedforward artificial neural network with one hidden layer using the discrete wavelet coefficient as weights of the linear output neuron (modification possible). The class of neural networks which use (mostly orthogonal) wavelets as activation functions is called: *wavelet neural networks*. We will show that we don't need to build in directly the wavelets into the activation function, we can build wavelets from the standard *logistic* functions (or from the *threshold* or the *piecewise-linear* functions).

Wavelet expansions and wavelet transforms have proven very efficient and effective in analysing a very wide class of signals and phenomena. The main reasons of this:

1. The size of wavelet coefficients $a_{j,k}$ drop off rapidly with j and k for a large class of signals. This property is being called an *unconditional basis* and it is why wavelets are so effective in signal and image compression, denoising, and detection.

2. The wavelet expansion allows a more accurate local description and separation of signal characteristics (e.g. then a Fourier transform). A wavelet expansion coefficient represents a component that is itself local and is easy to interpret. The wavelet expansion may allow a separation of components of a signal that overlap in both time and frequency.

3. Wavelets are adjustable and adaptable. Because there is not just one wavelet, they can be designed to fit individual applications. They are ideal for adaptive systems (e.g. neural networks) that adjust themselves to suit a signal.

4. The generation of wavelets and the calculation of discrete wavelet transform is well matched to the digital computer. It can be shown that the defining equation for a wavelet uses no calculus. There are no derivates or integrals, just multiplications and additions – operations that are basic to digital a computer.

5.3. Multiresolution Formulation of Wavelets:

Both the mathematics and practical interpretations of wavelets seem to be best served by using the concept of resolution to define the effect of changing scale. To do this we will start with a *scaling function* $\varphi(t)$[6] rather than directly with the *wavelet* $\psi(t)$. Almost all of the so called first generation wavelets is constructed from a scaling function. After the scaling function is defined from the concept of resolution, the wavelet functions will be derived from it. A set of scaling functions can be defined form the basic scaling function by integer translation:

$$\varphi_k(x) = \varphi(x-k) \qquad k \in \mathrm{Z} \qquad \varphi \in \mathrm{L}^2(\mathrm{R})$$

The subset of $\mathrm{L}^2(\mathrm{R})$ spanned by these functions is defined by:

$$V_0 = \overline{\underset{k\in Z}{Span}\{\varphi_k(x)\}}$$

The over bar denotes closure. This means that

$$\forall f(x) \in V_0 \;:\; f(x) = \sum_k a_k \varphi_k(x)$$

One can generally increase the size of the subspace spanned by changing the time scale of the scaling function. A two-dimensional family is generated from the basic scaling function by scaling and translation:

$$\varphi_{j,k}(x) = 2^{j/2}\varphi(2^j x - k)$$

whose span over k is:

$$V_j = \overline{\underset{k\in Z}{Span}\{\varphi_k(2^j x)\}} = \overline{\underset{k\in Z}{Span}\{\varphi_{j,k}(x)\}}$$

For $j > 0$ the span can be larger since $\varphi_{j,k}(t)$ is narrower and is translated in smaller steps. It, therefore can represent finer detail. For $j < 0$, $\varphi_{j,k}(t)$ is wider and is translated in larger steps. So these wider scaling functions can represent only coarse information, and the space they span is smaller. A requirement for a scaling function that it must be satisfy the following:

[6] In the previous chapters we used φ to identify the activation function, but now we use it to identify the scaling function, because it is the usual notation for that. (We hope it won't be confusing)

$$\forall j \in \mathrm{Z} : V_j \subset V_{j+1}$$

with

$$\bigcap_{j\in Z} V_j = \{0\} \qquad \text{and} \qquad \overline{\bigcup_{j\in Z} V_j} = \mathrm{L}^2(\mathrm{R})$$

The space that contains high resolution signals will contain those of lower resolution also. Because of the definition of V_j, the spaces have to satisfy a natural scaling condition:

$$f(x) \in V_j \Leftrightarrow f(2x) \in V_{j+1}$$

The important features of a signal can better be described or parameterised, not by using $\varphi_{j,k}(t)$ and increasing j to increase the size of the subspace spanned by the scaling function, but by defining a slightly different set of functions $\psi_{j,k}(t)$ that span the differences between the spaces spanned by the various scales of the scaling function. These functions are called *wavelets*. We now define the wavelet spanned subspace W_0 by orthogonal complement of V_0 in V_1:

$$V_1 = V_0 \oplus W_0$$

which extends to:

$$V_2 = V_0 \oplus W_0 \oplus W_1$$

In general:

$$\mathrm{L}^2 = V_0 \bigoplus_{i=0}^{\infty} W_i$$

when V_0 is the initial space spanned by the scaling function. The initial space is arbitrary and could be chosen at a higher or lower resolution:

$$\mathrm{L}^2 = V_j \bigoplus_{i=j}^{\infty} W_i$$

or even $j = -\infty$ is possible:

$$\mathrm{L}^2 = \bigoplus_{i=-\infty}^{\infty} W_i$$

We have now constructed a set of functions that could span all of $f \in \mathrm{L}^2(\mathrm{R})$:

$$f(x) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} a_{j,k} \psi_{j,k}(x)$$

Note that things slightly more complicated when we use non-orthogonal wavelets.
We will use the following way to generate wavelets from our *mother wavelet*:

$$\psi_{m,n}(x) = a_0^{-m/2} \psi(a_0^{-m} x - nb_0),$$

where $a_0 > 1$, $b_0 > 0$.

5.4. Numerically Stable Reconstruction:

With the wavelets, functions can be characterised by means of their wavelet coefficients $\langle f, \psi_{m,n} \rangle$ (or $\langle f, \tilde{\psi}_{m,n} \rangle$ if the wavelets are not orthogonal, where $(\tilde{\psi}_{m,n})_{m,n\in Z}$ is the dual basis). But we want more than characterisability: we want to reconstruct f in a numerically stable way from the coefficients. In order for such an algorithm to exist, we must be sure that if the sequence $\left(\langle f_1, \psi_{m,n} \rangle\right)_{m,n\in Z}$ is "close" to the sequence $\left(\langle f_2, \psi_{m,n} \rangle\right)_{m,n\in Z}$ then necessarily f_1 and f_2 were "close" as well. For that we need a topology on the function space and on the sequence space. On the function space L^2(R) we already have its Hilbert space topology; on the sequence space we will choose a similar l^2 –topology, in which the distance between sequences $c^1 = (c^1_{m,n})_{m.n\in Z}$ and $c^2 = (c^2_{m,n})_{m.n\in Z}$ is measured by:

$$\left\| c^1 - c^2 \right\|^2 = \sum_{m,n\in Z} \left| c^1_{m,n} - c^2_{m,n} \right|^2$$

This implicitly assumes that the sequences $\left(\langle f, \psi_{m,n} \rangle\right)_{m,n\in Z}$ are in l^2(Z^2). That means that:

$$\sum_{m,n\in Z} \left| \langle f, \psi_{m,n} \rangle \right|^2 < \infty$$

In practice this is no problem. As we will see below, any reasonable wavelet (which means that ψ has some decay in both time and frequency, and that its integral is zero), and any choice for $a_0 > 1$, $b_0 > 0$ leads to:

$$\text{(1)} \qquad \sum_{m,n\in Z} \left| \langle f, \psi_{m,n} \rangle \right|^2 \leq B \| f \|^2$$

We will assume that it holds. With the $\left(\langle f, \psi_{m,n} \rangle\right)_{m,n\in Z}$ interpretation of "closeness" the stability requirement means that if $\sum_{m,n\in Z} \left| \langle f, \psi_{m,n} \rangle \right|^2$ is small then $\| f \|^2$ should be small as well. In particular it holds if there exists an $\alpha < \infty$ so that

$$\sum_{m,n\in Z} \left| \langle f, \psi_{m,n} \rangle \right|^2 < 1 \Rightarrow \| f \|^2 \leq \alpha$$

Now we take an arbitrary $f \in$ L^2(R) and define:

$$\tilde{f} = \frac{1}{\sqrt{\sum_{m,n\in Z} \left| \langle f, \psi_{m,n} \rangle \right|^2}} f$$

Clearly:

$$\sum_{m,n\in Z}\left|\left\langle \tilde{f},\psi_{m,n}\right\rangle\right|^2 < 1 \qquad \text{hence} \qquad \left\|\tilde{f}\right\|^2 \le \alpha$$

But this means:

$$\left(\sum_{m,n\in Z}\left|\left\langle f,\psi_{m,n}\right\rangle\right|^2\right)^{-1}\left\|f\right\|^2 \le \alpha$$

or

(2) $$A\left\|f\right\|^2 \le \sum_{m,n\in Z}\left|\left\langle f,\psi_{m,n}\right\rangle\right|^2$$

For some $A=\alpha^{-1}>0$. On the other hand, if this holds for all *f*, then the distance $\left\|f_1-f_2\right\|$ cannot be arbitrary large if $\sum_{m,n\in Z}\left|\left\langle f_1,\psi_{m,n}\right\rangle-\left\langle f_2,\psi_{m,n}\right\rangle\right|^2$ is small. Combining (1) and (2) we obtain that $\{\psi_{m,n} : m,n\in Z\}$ should constitute a *frame*:

Definition (frame):
A family of functions $(\varphi_j)_{j\in J}$ in a Hilbert space *H* is called a *frame* if:

$$\exists A>0 : \exists B<\infty : \forall f\in H : A\left\|f\right\|^2 \le \sum_{j\in J}\left|\left\langle f,\varphi_j\right\rangle\right|^2 \le B\left\|f\right\|^2 .$$

We call *A* and *B* the *frame bounds*. If the two bounds are equal, then we call the frame a *tight frame*. (A tight frame with frame bounds 1 constitues an orthonormal basis)

Thus we can conclude that: to have a numerically stable reconstruction algorithm for *f* from $\left\langle f,\psi_{m,n}\right\rangle$, we should require that $\psi_{m,n}$ constitute a frame.

Ingrid Daubenchies proved in (Daubenchies, 1992) that:

Theorem 5.4.1 (sufficient requirements for a frame):
Let $\hat{\psi}$ be the Fourier transform of $\hat{\psi}$. If ψ, a_0 are such that

(1) $\inf\limits_{1\le|\xi|\le a_0}\sum_{m=-\infty}^{\infty}\left|\hat{\psi}(a_0^m\xi)\right|^2 > 0$,

(2) $\sup\limits_{1\le|\xi|\le a_0}\sum_{m=-\infty}^{\infty}\left|\hat{\psi}(a_0^m\xi)\right|^2 < \infty$,

(3) $\beta(s)=\sup\limits_{\xi}\sum_{m}\left|\hat{\psi}(a_0^m\xi)\right|\left|\hat{\psi}(a_0^m\xi+s)\right|$ decays at least as fast as $(1+|s|)^{-(1+\varepsilon)}, \varepsilon>0$

then there exists $(b_0)_{\text{thr}}>0$ such that the $\psi_{m,n}$ constitute a frame for all choice of $b_0<(b_0)_{\text{thr}}$. (where $\psi_{m,n}=a_0^{-m/2}\psi(a_0^{-m/2}x-nb_0)$). Moreover (2) and (3) are satisfied if, e.g. $|\hat{\psi}(\xi)| < c|\xi|^{\alpha}(1+|\xi|)^{-\gamma}$ for some $\alpha > 0, \gamma > \alpha + 1$.

5.5. Building Wavelets from Logistic Functions:

Now we go back to our original problem: approximation with feedforward neural networks. We will show that we can build a suitable *mother wavelet* by linear combination of several *logistic* activation function (with the method of *arranging the neurons*). If we can prove that, then we have also proved the *universal approximation theorem* in an alternative – multiresolution – way, because, as we saw, the function space defined by wavelets is dense in $L^2(R)$. We will use φ in its original meaning: it will be the activation function of an artificial neuron (specially the standard logistic function). We do it because we do not need a scaling function any more. (We directly define our wavelet, not with a scaling function) To show that our constructed function is a wavelet we must prove, that its integral on the real line is zero, and the functions that are generated from it forms a frame.

Definition (logistic wavelet):
If φ is the standard logistic function ($\varphi(x) = (1+e^{-x})^{-1}$) then we define the *logistic mother wavelet* by:

$$\psi(x) = -c_1\varphi(a_1x+b_1) + c_1\varphi(a_1x-b_1) + c_2\varphi(a_2x+b_2) - c_2\varphi(a_2x-b_2)$$

where $c_1, c_2, a_1, a_2, b_1, b_2 \in$ R. We will restrict the parameters to let the integral of the logistic wavelet be zero. (That is a requirement for a mother wavelet) The shape of the wavelet looks like the *Mexican hat*. (See the left part of Figure 5.2)

Comment:
Note that these wavelets are *not* orthogonal! Thus if we want to compute the wavelet coefficients then we need the dual basis.

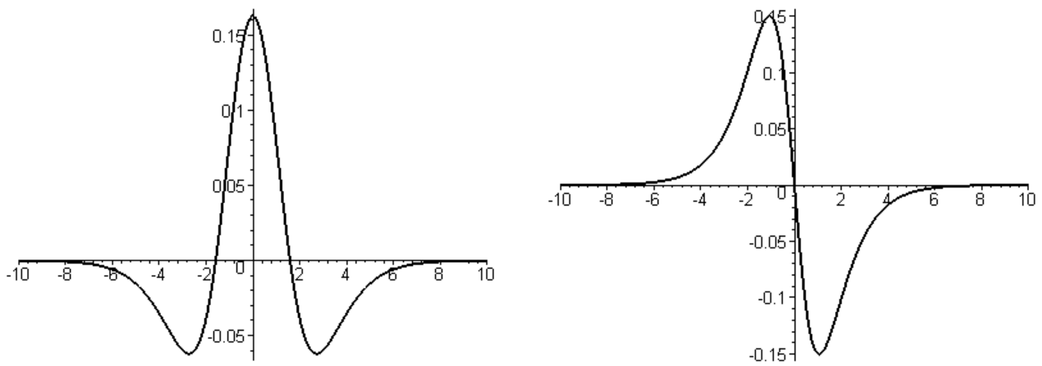


Figure 5.2
The logistic mother wavelet; An alternative way to define a logistic mother wavelet

Comment:
An alternative way could be to construct a mother wavelet from the standard logistic function is to simply take the difference of two logistic function: $\psi_{alt}(x) = c_1\varphi(a_1x) - c_2\varphi(a_2x)$. $(a_1 \neq a_2)$ (See the right part of Figure 5.2) In that case the integral of the function is trivially zero.

We now examine what kind of parameters are accepted to construct the logistic mother wavelet:

Theorem 5.5.1 (integral of the logistic wavelet):

$$\int_{-\infty}^{\infty}\psi(x)dx = 0 \Leftrightarrow \frac{c_1b_1}{a_1} = \frac{c_2b_2}{a_2}$$

Proof (5.5.1):

$$\int_{-\infty}^{\infty}\psi(x)dx = c_1\int_{-\infty}^{\infty}(-\varphi(a_1x+b_1)+\varphi(a_1x-b_1))dx + c_2\int_{-\infty}^{\infty}(\varphi(a_2x+b_2)-\varphi(a_2x-b_2))dx =$$

$$= \frac{c_1}{a_1}\int_{-\infty}^{\infty}(-\varphi(x+b_1)+\varphi(x-b_1))dx + \frac{c_2}{a_2}\int_{-\infty}^{\infty}(\varphi(x+b_2)-\varphi(x-b_2))dx = -\frac{2c_1b_1}{a_1} + \frac{2c_2b_2}{a_2}$$

because $\int_{-\infty}^{\infty}(\varphi(x+b_i)-\varphi(x-b_i))dx = 2b_i$. ■

After that we will assume this holds for the parameters. Now we show that with the logistic type of wavelets a numerically stable reconstruction is possible, because these type of wavelets form a *frame*. We will prove that with the *theorem 5.4.1*, so we will need the *Fourier transform* of our wavelet. We will use the following definition of the Fourier transform:

$$\mathrm{F}[f](w) = \int_{-\infty}^{\infty} e^{-iwx} f(x)dx$$

Note that the usual definition of the Fourier transform contains a constant (like $1/\sqrt{2\pi}$), but we only need the Fourier transform to estimate a function, thus we skipped this constant to made our formulas simpler. We will use the following basic properties of the Fourier transform:

$$\mathrm{F}[g \circ f](w) = e^{-ibw}\,\mathrm{F}[f(x)](w) \qquad \text{where} \qquad g(x) = x+b$$

$$\mathrm{F}[g \circ f](w) = \frac{1}{a}\mathrm{F}[f(x)]\left(\frac{w}{a}\right) \qquad \text{where} \qquad g(x) = ax$$

Theorem 5.5.2 (logistic wavelets form a frame):
The wavelets that are generated from the logistic mother wavelet by $\psi_{m,n} = a_0^{-m/2}\psi(a_0^{-m/2}x - nb_0)$ (where $a_0 > 1$, $b_0 > 0$) form a frame.

Proof (5.5.2):
First we will compute the Fourier transform of the logistic (mother) wavelet. After that we will show that the preconditions of the *theorem 5.4.1* are hold for our wavelet, thus the functions generated from that form a frame. We will express the Fourier transform of our wavelet by the Fourier transform of its derivate:

$$F[\psi'](w) = \int_{-\infty}^{\infty} e^{-iwx}\psi'(x)dx = \underbrace{\left[e^{-iwx}\psi(x)\right]_{-\infty}^{\infty}}_{=0} - \int_{-\infty}^{\infty} e^{-iwx} iw\,\psi(x)dx = -iw\int_{-\infty}^{\infty} e^{-iwx}\psi(x)dx =$$
$$= -iwF[\psi](w)$$

Note that we used the rule of *partial integration*. Now we have to compute the Fourier transform of ψ':

$$\psi'(x) = -c_1a_1\varphi'(a_1x+b_1) + c_1a_1\varphi'(a_1x-b_1) + c_2a_2\varphi'(a_2x+b_2) - c_2a_2\varphi'(a_2x-b_2)$$

Hence

$$F[\psi'](w) = -c_1a_1e^{-iw\frac{b_1}{a_1}}\frac{1}{a_1}F[\varphi']\left(\frac{w}{a_1}\right) + c_1a_1e^{iw\frac{b_1}{a_1}}\frac{1}{a_1}F[\varphi']\left(\frac{w}{a_1}\right) +$$
$$+c_2a_2e^{-iw\frac{b_2}{a_2}}\frac{1}{a_2}F[\varphi']\left(\frac{w}{a_2}\right) - c_2a_2e^{iw\frac{b_2}{a_2}}\frac{1}{a_2}F[\varphi']\left(\frac{w}{a_2}\right) =$$
$$2ic_1\sin\left(\frac{wb_1}{a_1}\right)F[\varphi']\left(\frac{w}{a_1}\right) - 2ic_2\sin\left(\frac{wb_2}{a_2}\right)F[\varphi']\left(\frac{w}{a_2}\right)$$

To compute this we only need the Fourier transform of the first derivate of our logistic activation function:

$$\varphi'(x) = \frac{e^{-x}}{\left(1+e^{-x}\right)^2} = \frac{1}{2+2\cosh(x)} = \frac{1}{4\cosh^2\left(\frac{x}{2}\right)}$$
$$F[\varphi'](w) = \int_{-\infty}^{\infty} e^{iwx}\varphi'(x)dx = \int_{-\infty}^{\infty}\frac{e^{iwx}}{4\cosh^2\left(\frac{x}{2}\right)}dx = 2\int_0^{\infty}\frac{\cos(wx)}{4\cosh^2\left(\frac{x}{2}\right)}dx = \frac{w\pi}{\sinh(w\pi)}$$

We calculated this integral by the formulas that can be find in (Gradshteyn, 1980). Finally we have got the Fourier transform of our logistic wavelet:

$$F[\psi](w) = \frac{-1}{iw}\left(2ic_1\sin\left(w\frac{b_1}{a_1}\right)F[\varphi']\left(\frac{w}{a_1}\right) - 2ic_2\sin\left(w\frac{b_2}{a_2}\right)F[\varphi']\left(\frac{w}{a_2}\right)\right) =$$

$$= -\frac{2c_1}{w}\sin\left(\frac{wb_1}{a_1}\right)\frac{\frac{w\pi}{a_1}}{\sinh\left(\frac{w\pi}{a_1}\right)} + \frac{2c_2}{w}\sin\left(\frac{wb_2}{a_2}\right)\frac{\frac{w\pi}{a_2}}{\sinh\left(\frac{w\pi}{a_2}\right)} =$$

$$= -\frac{2\pi c_1}{a_1}\frac{\sin\left(\frac{wb_1}{a_1}\right)}{\sinh\left(\frac{w\pi}{a_1}\right)} + \frac{2\pi c_2}{a_2}\frac{\sin\left(\frac{wb_2}{a_2}\right)}{\sinh\left(\frac{w\pi}{a_2}\right)}$$

With this Fourier transform we will show by *theorem 5.4.1* that the logistic wavelets form a frame. First we will show that for all w∈ R:

$$\forall\alpha > 0 : \forall\gamma > \alpha + 1 : \left|F[\psi](w)\right| < c\left|w\right|^{\alpha}\frac{1}{(1+\left|w\right|)^{\gamma}}$$

Note that this requires $F[\psi](0) = 0$, but we required that from the parameters. From the series expansions we easily obtain that:

$$\frac{\sin(ax)}{\sinh(bx)} = \frac{a}{b} + O(x^2) \qquad \text{for} \qquad x \to 0$$

Using this formula for the computed Fourier transform we get:

$$(1) \qquad F[\psi](w) = \underbrace{-\frac{2\pi c_1}{a_1^2}\frac{b_1}{\pi / a_1} + \frac{2\pi c_2}{a_2^2}\frac{b_2}{\pi / a_2}}_{=0} + O(w^2) = O(w^2) \qquad \text{for} \qquad w \to 0$$

On the other hand:

$$\left|\frac{\sin(ax)}{\sinh(bx)}\right| \le \frac{1}{\sinh(bx)} \le \frac{1}{2}e^{-\left|bx\right|}$$

which implies:

$$(2) \qquad \forall w \in R : \left|F[\varphi](w)\right| \le \left|\frac{2\pi c_1}{a_1}\right|\frac{1}{2}e^{-\left|\frac{\pi w}{a_1}\right|} + \left|\frac{2\pi c_2}{a_2}\right|\frac{1}{2}e^{-\left|\frac{\pi w}{a_2}\right|}$$

From (1) and (2) we easily obtain that for instance:

$$\left|F[\psi](w)\right| < c\left|w\right|\frac{1}{(1+\left|w\right|)^3}$$

Which implies the second and the third condition of the *theorem 5.4.1*. The only remaining thing that we must prove is that:

(3) $$\inf_{1 \le |\xi| \le a_0} \sum_{m=-\infty}^{\infty} \left| \hat{\psi}(a_0^m \xi) \right|^2 > 0$$

From (2) we conclude that there exist positive constants c_1, c_2 such that:

$$\left| F[\psi](w) \right| < c_1 e^{-c_2 |w|}$$

consequently:

$$\left| F[\psi](a_0^m w) \right|^2 < c_1^2 e^{-2c_2 a_0^m |w|}$$

Since $a_0 > 1$ this means the series (3) converges very fast if $m \to \infty$. Also since for $m \to -\infty$:

$$\left| F[\psi](a_0^m w) \right|^2 < c_3 \left(a_0^m w \right)^2 = c_3 \frac{w^2}{a_0^{-2m}} \qquad (m > 0,\ a_0 < 1)$$

Hence also this converges if $m \to -\infty$. We can compute the exact values of the frame bound if necessary by the formulas that can be find in (Daubenchies, 1992). ■

Epilogue:

"You react, Critias, as if I professed to know the answers to the questions I ask, and could explain it all to you if I wished. However, the opposite is true, for I am also inquiring into the truth of these matters."
- Socrates, quoted in Plato's Charmides

We have some useful results about the approximation with feedforward artificial neural networks, however there are several open questions that came up and we had not got enough time the analyze and answer them all. Here a brief list of the open questions:

- Generalization of our result to $\mathrm{R} \to \mathrm{R}^m$ functions is easy (almost trivial), because we used biases that are independent from the function that we want to approximate. But how could we generalise to $\mathrm{R}^m \to \mathrm{R}$ functions?

- We saw that our approximation can be *optimal* (the error will be minimal) in the case when the biases are independent from the function that we want to approximate (f). But how could we build an optimal approximation if the biases can depend on f?

- Is the well know *backpropagation* learning algorithm give this optimal solution? If yes, how can we prove this? If not how can we build a learning algorithm that give an optimal solution?

- If we use smooth functions as activation function (e.g. we use the *logistic function*) it is possible that the derivates of the approximation approximates the derivates of the function that we want to approximate up to some order. How could we choose the parameters if we want an approximation with this property? Or how steep must we set our function?

- How could we find a numerically effective (not only stable) way to compute the wavelet coefficients of a function with our mother wavelets?

- We can change the property of our wavelet by varying the parameters. For the fast computation it will be nice to have such a mother wavelet that has: $a_0 = 2$ and $b_0 = 1$. How must we choose the parameters to reach this?

- How can we find the dual basis to our wavelets?

- How can we find a suitable $\underline{\alpha}_n$ to finish the first proof that we tried to give for the *theorem 3.4.1*?

- Proving the convergence of our multiresolution algorithm (the algorithm that don't use wavelets) is easy when our activation function is such that: $\varphi(x) = 0$ if $x < c$. But how can we prove the convergence for the general case?

Appendix$_1$:

"One must learn by doing the thing;
for though you think you know it,
you have no certainty until you try."
- Sophocles

For constructing an approximation of an $f \in C([0,1])$ with an artificial neural network which has a monotone-increasing activation functions (*act*) that has $act(0) \neq 0$ we can use the following *MapleV* program: to build the approximation for a given number of interpolating points (n) and for a given width (a) of the bells (the program returns a lower estimation of the error):

We need the *linear algebra* package to build the approximation:

```
restart;
with(linalg);
```

The three "classical" activation function (needless to say, we can use any other activation function with the given properties):

The *threshold function*:

```
act := proc(x)
   if (x>=0) then
      RETURN(1);
   else
      RETURN(0);
   fi;
end;
```

The *piecewise linear function*:

```
act := proc(x)
   if (x>=1/2) then
      RETURN(1);
   elif (x>=-1/2) then
      RETURN(x+1/2);
   else
      RETURN(0);
   fi;
end;
```

The *logistic function*:

```
act := proc(x)
   RETURN(1/(1+exp(-x)))
end;
```

Building the *bells* (we can fix *b* to an arbitrary non-zero value):

```
bell := proc(x,a)
   RETURN(act(a*x+b)-act(a*x-b))
end;
```

The approximation of *f* with an artificial neural network with *2(n+1)* hidden neurons (*n+1 bells* with *a* width):

```
approx := proc(f,n,a)
   local F, B, v, i, j, k, temp, y, maxe;
   global g, W;
   B := linalg[matrix](n+1,n+1);
   F := linalg[vector](n+1);
   W := linalg[vector](n+1);
   for i from 1 to n+1 do
      F[i] := evalf(f((i-1)/n));
   od;
   for i from 1 to n+1 do
      for j from 1 to n+1 do
         B[i,j] := evalf(bell((i-1)/n-(j-1)/n,a));
      od;
   od;
   W := multiply(F, inverse(B));
   g := 'g';
   g := proc(x)
      local temp;
      temp := add(W[i]*bell(x-(i-1)/n,a),i=1..n+1);
      RETURN(temp);
   end;
   maxe := 0;
   for k from 1 to 20*n+1 do
      y := evalf((k-1)/(20*n));
      temp := f(y) - g(y);
      temp := evalf(temp);
      temp := abs(temp);
      if temp > maxe then maxe := temp fi;
   od;
   RETURN(maxe);
end;
```

To build the approximation of *f* with 11 *bells* (22 *hidden neurons*), we can use:

```
approx(f,10,20);
```

We can plot the original function and its approximation by typing:

```
plot([f,g],0..1);
```

Appendix$_2$:

"I hear, I forget;
I see, I remember;
I do, I understand!"
- Chinese proverb

This appendix contains of a *Matlab* program which can be use to build a multiresolution approximation of a given function (defined on [0,1]). The file *act.m* contains our activation function. We can use the built in logistic function as an activation function (if we have the *neural networks toolbox* installed):

```
function ret = act(x);
ret = logsig(10*x);
return;
```

Or we can define the threshold (or any other) function by:

```
function ret = act(x);
for i = 1:length(x)
   if x(i) < 0
      ret(i) = 0;
   else
      ret(i) = 1;
   end;
end;
return;
```

The successive (multiresolution) approximation program (The input parameter *fx* contains an arbitrary vector of the function values in [0,1] and *n* is the number of layers that will be used during the approximation. We don't have to use *m*):

```
function ap = succ(fx, n, m)

if (nargin < 3)
   m = n;
end;

step = 1/(length(fx)-1);
x  = 0:step:1;

if (m==1)
   ap = act(x) .* fx(round(length(fx)/2));
   if (n == 1)
      plot(x,fx,x,ap);
      max_error = max(abs(fx - ap))
   end;
   return;
end;
```

```
% pre-processing the indexes:

for i = 1:2:2^(m)
   shift((i+1)/2)  = (i-1)./(2^m);
   middle((i+1)/2) = i./(2^m);
   ind((i+1)/2) = round(((length(fx)-1)*i )./(2^m));
end;

temp = succ(fx,n,m-1);
ap(1:length(fx)) = 0;

for i = 1:2^(m-1)
   coeff(i) = (fx(ind(i))-temp(ind(i)));
   for j = 1:(i-1)
      coeff(i) = coeff(i)-(act(m*(middle(i)-
                 shift(j))) .* coeff(j));
   end;
   coeff(i) = coeff(i)/act(m*(middle(i)-shift(i)));
   ap = ap +  act(m*(x-shift(i))) .* coeff(i) ;
end;

ap = ap + temp;

if (m == n)
     plot(x,fx,x,ap);
     max_error = max(abs(fx - ap))
end;
```

First we need to put the values of the function on [0,1] (in an arbitrary equidistant set of points) into a vector, than we can use the program by:

```
x  = 0:0.001:1;
fx = sin(4*x+1)+(x+1).^2+1;

succ(fx,6)
```

The program returns a lower estimation of the error and plots the original function and its approximation.

Bibliography:

(1) G. Cybenko: Approximation by superpositions of sigmoidal functions. Mathematics of Control, Signals, and Systems. 1989.

(2) Kurt Hornik: Approximation Capabilities of Multilayer Feedforward Networks. Neural Networks, vol. 4, 1991.

(3) Kurt Hornik, Maxwell Stinchcombe and Halbert White: Multilayer Feedforward Networks are Universal Approximators. Neural Networks, vol. 2, 1989.

(4) Halbert White: Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings. Neural Networks, vol. 3, 1990.

(5) Pierre Baldi: Computing with Arrays of Bell-Shaped and Sigmoid Functions. IEEE Transactions on Neural Networks, vol. 5, 1994.

(6) Simon Haykin: Neural Networks, A comprehensive foundation. 2nd edition. Prentice Hall, 1999.

(7) Eduard Aved'yan: Learning Systems. Springer, 1995.

(8) Mohamad H. Hassoun: Fundamentals of Artificial Neural Networks. MIT Press, 1995.

(9) C. J. Thornton: Techniques in Computational Learning: an introduction. Chapmann and Hall Computing, 1992.

(10) Francoise Chatelin: Eigenvalues of Matrices. John Wiley and Sons Ltd, 1993.

(11) Ward Cheney, Will Light: A Course in Approximation Theory. Brooks/Cole, 2000.

(12) C. Sidney Burrus, Ramesh A. Gopinath, Haitao Guo: Introduction to Wavelets and Wavelet Transforms. Prentice Hall, 1998.

(13) Ingrid Daubenchies: Ten Lectures on Wavelets. Capital City Press, 1992.

(14) Ingrid Daubenchies: Different Perspectives on Wavelets. American Mathematical Society, 1993.

(15) I. S. Gradshteyn and I. M. Ryzhik: Table of Integrals, Series and Products. Academic Press, 1980