Neural Processing Letters (2024) 56:100
https://doi.org/10.1007/s11063-024-11518-y

Multi-back-propagation Algorithm for Signal Neural Network Decomposition

Paulo Salgado[1] · **T.-P. Azevedo Perdicoúlis**[2]

Accepted: 24 November 2023

Abstract
In this paper, a novel back-propagation error technique is presented. This neural network structure allows for two fundamental basic modes: (1) To decompose the neurones by transforming their variables, weights, and scalar functions into vectors. This conveys for the decomposition of the transfer function of every neurone (where the output variables are the components of the decomposition) and, consequently, to be written as the invariant sum of orthogonal functions, with the safeguard of preserving information This orthogonality is proven using Fourier theory. (2) In a second mode, a tuned neural network that occupies one of the channels of the neural network can see the weights of its supplementary channels adjusted to retain additional information. Only the decomposition algorithm of the network is presented here—Multi-back-propagation algorithm. The adopted methodology is validated step-by-step with some representative examples. Namely, to assess the performance of the splitting method, two different examples have been constructed from scratch: (1) a 2D classification problem and (2) a 3D surface. In both problems, the signal and transfer functions of the neural network are successfully decomposed without information losses. Therefore, since the main contribution of this work is to allow for the organisation of the information stored in neural network structure, through a split process, this promising method shows potential use in various areas—e.g. classification and/or pattern recognition problems, data analysis, modelling and so on. In the future, we expect to work further in the method computational aspects to render it more efficient, versatile and robust.

Keywords Back-propagation · Fourier series · Multivariable neurone · Neural network training · Network decomposition · Network splitting · Orthogonal decomposition

✉ T.-P. Azevedo Perdicoúlis
tazevedo@utad.pt

Paulo Salgado
psal@utad.pt

[1] Department of Engineering, ECT & CITAB, UTAD, Quinta de Prados, 5000-715 Vila Real, Portugal

[2] ISR, University of Coimbra & Department of Engineering, ECT, UTAD, Quinta de Prados, 5000-715 Vila Real, Portugal

Published online: 12 March 2024

1 Introduction

During the last decades, neural networks (NN) have been receiving much attention from the research community, especially in the area of artificial intelligence. The reward comes from the emulation of the human brain in the computing environment through mathematical models. A NN is a strongly connected structure of artificial neurones.

As is well known, an artificial neurone is a mathematical abstraction of the nerve cell,—an excitable cell that communicates with other cells via specialised connections. An artificial network It resembles a set of biological neurones only in a few aspects, with its main differences being the topology and the number of neurones used. Also, layers are calculated in a certain order while biological neurones can act asynchronously. The speed of information transfer and the learning process are other distinguishing features.

To mirror the brain, the perceived inputs are propagated through the network, changing on the way as a result of the connections' different weights, as well as the mixing and signal-triggering of different functions which characterise every neurone. As the information withheld by the NN is dependent on its structure and every neurone idiosyncrasy, the dimension of the structure, its topology and the type of its neurones define the NN capacity to retain specific information. Thence, an adequate learning algorithm should be used to adjust the weights within the NN either to retain required information or adjust the interconnection structure. In this way, this complex structure can resemble an artificial brain that is able to memorise, map information, and give answers according to the received inputs. Learning methods are classified into supervised, unsupervised, and reinforced [1–3].

Assuming the success of the chosen learning algorithm and that the NN is well-dimensioned, the network will produce the required information. Most of the learning methods give neither attention to the form of the information spread within the network nor to the physical meaning of the network weights. The information disseminates through the structure without any legibility nor inter-functionality of its local values.

The neurone is the NN basic unit that retains information about the structure in question through its weights, i.e., information about its integration and activation functions. Once its structure is chosen, a NN retains information in the weights of all its neurones. Whenever the information memorised in the network can be reorganised, particularly through the decomposition into parcels of information, a relevant question is whether this decomposition can be carried out locally in the neurones.

To answer this question, we need to prove whether it is possible to "split" a neurone, keeping the core of its structure but assuming all its variables to be multidimensional as well as the activation function decomposed into orthogonal components. Such a decomposition should be reversible in the sense that the sum of the sub-models is always equal to the transfer function of the initial neurone (before being split). Another issue is to present a method able to transfer information between neurones for this new model structure (i.e., vectorial signals and integration and activation functions) and that, at the same time, warrants immutability of the neurone and consequently of the whole NN.

Therefore, the present work aims to capacitate an already existing NN to re-arrange its information by transforming its scalar inputs, outputs and functions into multivariable entities to guarantee the decomposition of the signals that circulate within the NN. Crucial to this paradigm is to warrant that the transfer functions of the NN are not changed, to make sure that the information is preserved. This is possible since the decomposition of the multidimensional functions transmits the representation of the transfer function of every neurone as an invariant sum of parcel functions that equals its original value.

The back-propagation algorithm is traditionally used in supervised learning of NN [4]. It propagates the NN output error to every neurone and then, locally, makes the readjustment of the weights. Its objective is the NN to learn from the training data and then to be able to reproduce the data (this or other data) with the smallest error as possible. The algorithm that we propose in this work—Multi-back-propagation algorithm—has two phases. But its contribution is Phase-2, since Phase-1 is the NN that we always assume to exist. Phase-1, or learning phase, obtains the NN structure with some other method; e.g. the back-propagation algorithm. Phase-2, or the splitting phase, transforms the NN obtained at Phase-1 into a multidimensional version assuring that no information is lost and the NN output components are taken as answers of the decomposed NN. At any instant, the sum of the NN components should remain the same as the original NN (from Phase-1). Thus, the objective of this method is to back-propagate the separability factor of the output components, making sure that at every vectorial neurone of the NN arrives an adjustment factor of separability. Once holding this value, it is possible to locally transfer information between the sub-models of the vectorial neurone.

From the best of our knowledge, transfer function transformation and optimisation has received little research attention so far. The idea presented in this paper, i.e., to split the transfer function into orthogonal components is innovative and can be applied to all problems that benefit from NN readability. Moreover, it also makes possible to interrogate the NN about portions of the modelled knowledge. We envision its use, with clear benefits, for structural NN reorganisation and in the formulation of new machine learning strategies.

The idea of splitting the NN is interesting and there are just a few papers on this topic. In [5] the NN automatically learns to split the network weights into either a set or a hierarchy of multiple groups that use disjoint sets of features, by learning both the class-to-group and feature-to-group assignment matrices along with the network weights. In [6], experiments show that in the case where the network performs two different tasks, the neurones naturally split into clusters, where each cluster is responsible for processing a different task. This behaviour not only corresponds to biological systems, but also allows for further insight into interpretability or continual learning. In [7], a new technique is proposed to split a convolutional network structure into small parts that consume lower memory than the original one. The split parts can be processed almost separately, which provides an essential role for better memory management. In [8] is proposed a novel, complete algorithm for the verification and analysis of feedforward NN ReLU-based. The algorithm, based on symbolic interval propagation, introduces a new method for determining split-nodes which evaluates the indirect effect that splitting has on the relaxations of successor nodes. A constructive algorithm is proposed in [9] for feedforward NN which uses node-splitting in the hidden layers to build large networks from smaller ones. Modification of the transfer function and the error back-propagation technique applied to NN can be seen in [4] and the transfer function pooling in [10].

In Sect. 2, the activation function is written as a Fourier sum of m terms to facilitate decomposition of the input and output signals. The optimality of this approximation is proven. In Sect. 3, the multidimensional neurone is formulated and the orthogonality of the partial activation functions is demonstrated. In Sect. 4, the new learning method is outlined—the Multi-back-propagation algorithm. This conveys the transfer of information between different parts of the decomposed neurones of a feedforward NN with the use of a reinforcement back-propagation technique. The demonstration of the decomposition procedure, as well as the effectiveness of the proposed splitting algorithm can be found in Sect. 5. Namely, (1) we illustrate the extension of the activation function to a periodic one; (2) we give an example where the activation function—the hyperbolic tangent—to demonstrate the virtue of the approximation of the output by an orthogonal sum; (3) The splitting of the NN transfer

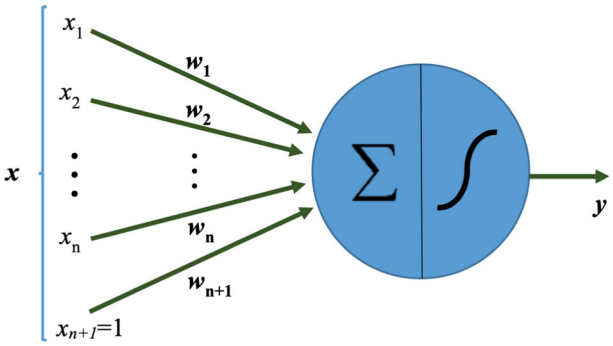


Fig. 1 Structure of an artificial neuron. Σ is the activation function and $\int$ the integration function

function, as well as the invariance of the sum of its parts, is substantiated in two different examples. The work concludes with Sect. 6 where the main conclusions are outlined. Also, some directions for future research in view to ripen the Multi-back-propagation method are stated.

2 Activation Function Orthogonal Decomposition

A classical NN is formed by elementary units—the neurones.
A single artificial neurone is represented mathematically as:

$$y = \varphi\left(\sum_{i=1}^{n+1} w_i x_i\right). \tag{1}$$

Considering that we have $a = \boldsymbol{w}^T \boldsymbol{x}$, where $\boldsymbol{w} = (w_1, w_2, \ldots, w_n, w_{n+1})^T$, $w_i \in \mathbb{R}, i = 1, \ldots, n$, is the vector of the weights and $\boldsymbol{x} = (x_1, x_2, \ldots, x_n, x_{n+1})^T$ the input vector which means that it takes n inputs $x_i \in \mathbb{R}, i = 1, \ldots, n$, with each one having its own weight, $w_i, i = 1, \ldots, n$. A bias input exists at every node and is represented by a constant $x_{n+1} = 1$ and the adjusted weight w_{n+1}. Thence $y = \varphi(a)$.
Two mathematical functions are the basis of its structure. The neurone calculates the sum of the weighted inputs—the integrative function (Σ)—and passes the result through φ, that represents the neurone's activation potential. Typically, φ is non-linear and has the shape of a sigmoid — the activation function ($\int$). Usual choices are the hyperbolic tangent, the Heaviside function or the LeRu [11, 12]. φ is often monotonic increasing, continuous, differentiable and bounded. This function is used to pass the information further to the network, with the activation lower and upper bounds represented by $-\delta$ and $+\delta$, respectively. A threshold level is used to shift the action potential value of the neurone to the network, represented here by the bias of every neurone, x_{n+1}.

2.1 Orthogonal Decomposition of φ

The activation function defines how the output signal of the integration function is transformed into an output of the neurone. Without loss of generality, φ is a monotonously increasing saturated function. Define the saturation level as δ and the interval where φ is defined as

$\left[\underline{a}, \overline{a}\right]$, such that $\varepsilon = \varphi(\underline{a}) + \delta = \delta - \varphi(\overline{a})$. For what is meant the extremes of the interval to be sufficiently close to the saturation point. Furthermore, φ is odd, and then we have that $\underline{a} = -\overline{a}$ and $\underline{a} = -\varphi^{-1}(\delta - \varepsilon)$, $\overline{a} = \varphi^{-1}(\delta - \varepsilon)$, where ε is the required precision. Next, a periodic extension of function φ in the interval $\left[2\underline{a}, 2\overline{a}\right]$ is defined:

$$\Phi(a) = \begin{cases} \varphi(-a + 2\underline{a}) : 2\underline{a} \le a \le \underline{a} \\ \varphi(a) \qquad\quad : \underline{a} \le x \le \overline{a} \\ \varphi(-a + 2\overline{a}) : \overline{a} \le a \le 2\overline{a}. \end{cases} \tag{2}$$

Function $\Phi(a)$ is periodic of fundamental period $T = 2\left(\overline{a} - \underline{a}\right)$, i.e., $\Phi(a + kT) = \Phi(a), k \in \mathbb{Z}$. The extension of φ by a periodic function is illustrated in Fig. 3 of Sect. 5.1.

Remark 1 The decomposition of the activation function into Fourier series requires the activation function to be bounded (saturates), and therefore its transformation into a periodic function. For no saturating activation functions the same method may be used using other type of orthogonal decomposition as for instance Legendre, Chebyshev or Hermite polynomials [13, 14].

Theorem 2.1 states that if function Φ complies with certain assumptions then it can be written as an infinite sum—using the Fourier Series.

Theorem 2.1 (Approximation of the activation function by a Fourier Series) *Assuming that the periodic extension Φ has been defined in a way that is continuous and absolutely integrable over its period, so that the Dirichlet conditions are fulfilled, hence:*

$$\Phi(a) = S_m(a) + R_m(a) \tag{3}$$

where

$$S_m(a) := \sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n a), \tag{4}$$

$$R_m(a) := \sum_{n=m+1}^{\infty} \hat{\Phi}_n \sin(\omega_n a), \tag{5}$$

$$\hat{\Phi}_n = \frac{2}{T} \int_{-T/2}^{T/2} \sin(\omega_n a)\, \Phi(a) da, \tag{6}$$

with $w_n := (2n - 1)\omega_0$ *and* $\omega_0 = \dfrac{2\pi}{T}$.

Proof See Appendix A. □

The following corollary asserts that if some assumptions on Φ are fulfilled, its approximation by a Fourier sum can be as accurate as required by controlling the number of terms.

Corollary 2.1.1 *In the conditions of Theorem 2.1, and for an input signal that is real,* $\lim_{m\to\infty} \left|\Phi(a) - S_m(a)\right| \to 0$ *and the truncated Fourier series of degree m, $S_m(a)$, is the best approximation of this order in L^2 ($[-T/2, T/2]$) to function $\Phi(a)$.*

Proof From the Riesz-Fischer Theorem [15] we know that if $S_m(a)$ is the Fourier Series of m terms for the square-integrable function $\Phi(a)$ then

$$\lim_{m\to\infty} \left|\Phi(a) - S_m(a)\right| \to 0. \tag{7}$$

Moreover, the truncated Fourier Series of degree m, $S_m(a)$, is the best approximation of this order in L^2 ($[-T/2, T/2]$) to function $\Phi(a)$ [16]. That is, if $\Phi(a) \in L^2$ ($[-T/2, T/2]$) and $\zeta_1, \zeta_2, \ldots, \zeta_m$ are real numbers, then:

$$\left|\Phi(a) - S_m(a)\right| \leq \left|\Phi(a) - \sum_{n=1}^{m} \zeta_n \sin(\omega_n a)\right| \tag{8}$$

and equality holds only when $\zeta_n = \hat{\Phi}_n, n = 1, \ldots, m$. □

Lemma 2.1 states that the amount of energy contained in function Φ in a certain finite interval equals the energy of the Fourier series in the same interval.

Lemma 2.1 *For Φ defined in $L^2\left([2\underline{a}, 2\overline{a}]\right)$, we have*

$$\int_{-T/2}^{T/2} |\Phi(a)|^2 da = \int_{-T/2}^{T/2} |S_m(a)|^2 dx + \int_{-T/2}^{T/2} |R_m(a)|^2 da.$$

Proof See Appendix A. □

Corollary 2.1.1 *In the conditions of Lemma 2.1, we have:*

$$\frac{1}{T} \int_{-T/2}^{T/2} |S_m(a)|^2 da = \sum_{n=1}^{m} \hat{\Phi}_n^2. \tag{9}$$

Also, the mean quadratic error is the sum of the energies of the neglected harmonics:

$$\frac{1}{T} \int_{-T/2}^{T/2} |R_m(a)|^2 da = \sum_{n=m+1}^{\infty} \hat{\Phi}_n^2. \tag{10}$$

Lemma 2.2 states that for an odd function Φ, the coefficients of the Fourier series are calculated back in the interval $\left[\underline{a}, \overline{a}\right]$ where φ is defined before (2), i.e. $\hat{\Phi}_n$ is twice the respective $\hat{\varphi}_n$.

Lemma 2.2 *For an odd function Φ, the coefficients of the Fourier series in (6) can be redefined as:*

$$\hat{\Phi}_n = \frac{4}{T} \int_{-T/4}^{T/4} \sin(\omega_n a)\, \Phi(a) da.$$

Proof See Appendix A. □

And then is easily concluded that the energy of Φ in $\left[-\frac{T}{2}, \frac{T}{2}\right]$ is twice the energy of φ in $\left[-\frac{T}{4}, \frac{T}{4}\right]$.

Remark 2 For an odd function $\varphi(a)$

$$\frac{1}{2} \int_{-T/2}^{T/2} |\Phi(a)|^2 da = \int_{-T/4}^{T/4} |\varphi(a)|^2 da.$$

This remark follows directly from the Parseval-Rayleigh's identity, since $\varphi(a)$ coincides with $\Phi(a)$ in the interval $\left[-\frac{T}{4}, \frac{T}{4}\right]$.

Remark 3 For an odd function

$$R_m(a) = \sum_{n=m+1}^{\infty} \hat{\Phi}_n \sin(\omega_n a) = 2 \sum_{n=m+1}^{\infty} \hat{\varphi}_n \sin(\omega_n a)$$

with

$$\hat{\varphi}_n = \frac{1}{2}\hat{\Phi}_n = \frac{2}{T} \int_{-T/4}^{T/4} \sin(\omega_n a)\, \varphi(a) da$$

From what is clear that the error of the approximation of function $\Phi(a)$ by the Fourier Series is double of the same error for function $\varphi(a)$.

This remark follows from Lemma 2.2.

Remark 4 Knowing (5), then

$$\frac{2}{T} \int_{-T/4}^{T/4} |R_m(a)|^2 da = \sum_{n=m+1}^{\infty} \hat{\Phi}_n^2.$$

This remark follows from Lemma 2.2.

3 Multidimensional Neurones

In this section, we consider the multivariable counterpart of the neurone formalised in Sect. 2. The neurone input and output are multidimensional variables with every component being a fraction of the original signal. Hence

$$\begin{aligned} \varphi : \mathbb{R} &\longrightarrow \mathbb{R}^{\ell} \\ a &\rightsquigarrow y = \varphi(a), \end{aligned}$$

where $\boldsymbol{a} = \sum_{i=1}^{n+1} w_i \vec{x}_i$, $w_i, \vec{x}_i \in \mathbb{R}^p$. The multidimensional case is represented for $p = 2$ and $\ell = 2$ in Fig. 2, i.e., every input has got two different channels leading to the duplication of the integration/ activation functions, that are orthogonal between themselves, as it is demonstrated in Theorem 3.1.

In the multidimensional formulation, the counterpart of $\boldsymbol{x}$ in (1) is $\boldsymbol{X} = \begin{pmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_{n+1} \end{pmatrix}$, where $\vec{x}_i = (x_i(1), x_i(2), \ldots, x_i(p))$, for $i = 1, \ldots, n$. To decompose x_i into p components: $x_i(k) = x_i \beta_i(k)$, with $\vec{\beta}_i^T = (\beta_i(1), \ldots, \beta_i(p))$, $\sum_{k=1}^{p} \beta_i(k) = 1$ and x_i in (1) becomes $x_i = \sum_{k=1}^{p} x_i(k)$. Therefore, $\boldsymbol{B} = \left(\vec{\beta}_1\ \vec{\beta}_2 \cdots \vec{\beta}_{n+1}\right)^T$ is the matrix of splitting factors of

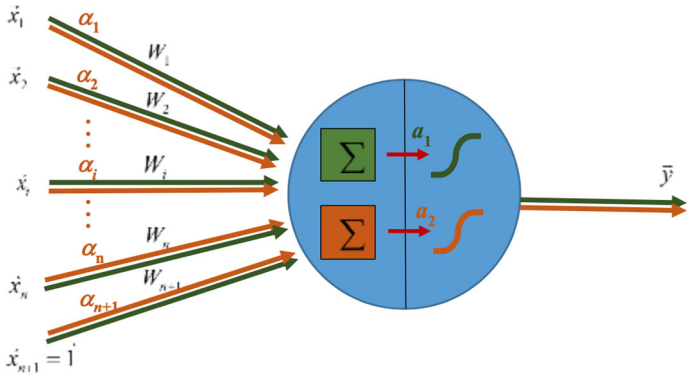


Fig. 2 Structure of the multidimensional neurone with $\ell = 2$

input $\boldsymbol{x}$. Thence $\boldsymbol{X} = \begin{pmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_{n+1} \end{pmatrix} = \begin{pmatrix} X_1 & X_2 & \cdots & X_p \end{pmatrix}$, $X_k \in \mathbb{R}^{n+1}$ is a column vector that contains the component-k of every input $\vec{x}_i$. Hence $X_k = B_k \odot \boldsymbol{x} = diag(B_k) \cdot \boldsymbol{x}, \quad k = 1, \ldots, p$, where $\odot$ is the Hadamard product, $\cdot$ is the inner product and $diag(B_k)$ is the diagonal matrix whose main diagonal is vector B_k—corresponding to the fraction-k of every input x_i. An alternative way to write matrix X is using the Khatri-Rao product:

$$\boldsymbol{X} = \boldsymbol{B} * \boldsymbol{x} = \begin{pmatrix} \vec{\beta}_1^T \\ \vec{\beta}_2^T \\ \vdots \\ \vec{\beta}_{n+1}^T \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n+1} \end{pmatrix} = \begin{pmatrix} \vec{\beta}_1^T \otimes x_1 \\ \vec{\beta}_2^T \otimes x_2 \\ \vdots \\ \vec{\beta}_{n+1}^T \otimes x_{n+1} \end{pmatrix}. \tag{11}$$

In a similar manner, the output $y = \begin{pmatrix} y(1) & y(2) & \cdots & y(\ell) \end{pmatrix}$, with $y = \sum_{k=1}^{\ell} y(k)$, $y(k) = \zeta(k)y$ and $\sum_{k=1}^{\ell} \zeta(k) = 1, \zeta = \begin{pmatrix} \zeta(1) & \cdots & \zeta(\ell) \end{pmatrix}$. That is, $Y = \zeta \otimes y$ with ζ being the splitting factor the output. Note that here we have assumed that $\ell = p$. The information contained in a neurone is strongly dependent on the weight values, w_i, and its transfer within the decomposed neurone can be done by assigning different weights to the different channels of the same input. Thence, consider $\vec{\tau}_i^T = \begin{pmatrix} \tau_i(1) & \cdots & \tau_i(p) \end{pmatrix}$ the splitting factor of the weights w_i, i.e., $w_i(k) = w_i \tau_i(k)$, $w_i = \sum_{k=1}^{p} w_i(k)$. Define $\boldsymbol{T} = \begin{pmatrix} \tau_1 & \tau_2 & \cdots & \tau_{n+1} \end{pmatrix}^T$ and then $\boldsymbol{W} = \boldsymbol{T} * \boldsymbol{w}$, likewise as in (11). For simplicity sake, in what follows, we consider $p = \ell = 2$. Hence $w_i = \begin{pmatrix} \tau_i & 1 - \tau_i \end{pmatrix}$ and
$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{\tau} & \mathbf{1} - \boldsymbol{\tau} \end{pmatrix}, \boldsymbol{\tau} \in \mathbb{R}^{n+1}$ and $\boldsymbol{\tau}(i) = \tau_i$. Also $\boldsymbol{B} = \begin{pmatrix} \boldsymbol{\beta} & \mathbf{1} - \boldsymbol{\beta} \end{pmatrix}, \boldsymbol{\beta} \in \mathbb{R}^{n+1}$ and $\boldsymbol{\beta}(i) = \beta_i, i = 1, \ldots, n$, $\mathbf{1}$ is a vector of ones.

A standard neurone has several input signals, which are weighted and conducted to the cell nucleus and, from there—through the activation function—to the output. All variables and the image of the function are scalar. With the herein proposed method, the structure of the neurone is replicated to accept vector inputs. The activation function of the neurone is decomposed into orthogonal functions, taking advantage of its representation as a Fourier series, and assuming it to be a periodic function. Each component of the activation function uses as input the weighted sum value of the associated coordinate of the input vector. In

this decomposition process, the weights of the original NN are distributed across various channels, always ensuring that the sum of the output components of the neurone is equal to the value of the standard neurone output. The weight distribution that occurs in each neurone is adjusted by a supervised sharing learning technique. Let $\zeta(k)$ be the separability vector of sample-k from the training set. With this new algorithm, the splitting factors $\zeta(k)$ are back-propagated by the structure of the NN to the input of each neurone, where the weights W are then decomposed into various channels.
The values $\zeta(k)$, $k = 1, ..., \ell$, are the result of applying a certain criterion of separability to the output signal, based on dissimilarity measure or decomposition techniques of the transfer function. This process is not considered in this study. The result of this splitting is that the flow of information in the NN happens through multi-dimensional channels. The coordinates of the output vector of the NN are the components of the decomposition of the NN response. In this section, we explain how to decompose every input, as well as the respective weights, into different channels. The next theorem gives a representation for each of its component.

Theorem 3.1 (Decomposition of the integration function) *Considering all the previous definitions, the splitting of the integration function, for* $p = 2$, *i.e.,* $a = \boldsymbol{w}^T \boldsymbol{x} = a_1 + a_2$ *becomes*

$$a_1 = \boldsymbol{w}^T \operatorname{diag}(\boldsymbol{\tau} \odot \boldsymbol{\beta})\boldsymbol{x}, \tag{12}$$

$$a_2 = \boldsymbol{w}^T \operatorname{diag}(I - \boldsymbol{\tau} \odot \boldsymbol{\beta})\boldsymbol{x}. \tag{13}$$

Proof See Appendix B. □

Define

$$\boldsymbol{\alpha} := \boldsymbol{\tau} \odot \boldsymbol{\beta} \in \mathbb{R}^{n+1} \tag{14}$$

which is a new parameter vector associated with the network input that specifies how the information is broken down. E.g. in the bidimensional case, if $\alpha_1 = 0.5$ it means that the first and second components of x_1 have the same amount of information. *Au contraire* if $\alpha_1 = 1$, there will be no information stored in the second component of the same input (this is what happens without splitting). Next, Theorem 3.2 gives a Fourier decomposition for the output components.

Theorem 3.2 (Splitting of the Output) *Assuming the input decomposed as in Theorem 3.1,* $y \approx \varphi(a)$ *can be split in the following manner:* $\varphi = \varphi_1(a_1, a_2) + \varphi_2(a_1, a_2)$, *where*

$$\varphi_1(a_1, a_2) = \sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n a_1) \cos(\omega_n a_2), \tag{15}$$

$$\varphi_2(a_1, a_2) = \sum_{n=1}^{m} \hat{\Phi}_n \cos(\omega_n a_1) \sin(\omega_n a_2) \tag{16}$$

and $y_i = \varphi_i(a_1, a_2,), i = 1, 2.$

Proof See Appendix B. □

Remark 5 From Theorem 3.2, one may infer:

$$a_1 = 0 \implies \varphi\,(a_1 + a_2) = \varphi_2\,(a_1, a_2) = \sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n a_2)$$

$$\text{and } \hat{\Phi}_n = \frac{2}{T}\int_{-T/2}^{T/2} \sin(\omega_n a_2)\,\varphi(a_2)dx$$

$$a_2 = 0 \implies \varphi(a_1 + a_2) = \varphi_1(a_1, a_2) = \sum_{n=1}^{m} \hat{\Phi}_n \sin \omega_n a_1$$

$$\text{and } \hat{\Phi}_n = \frac{2}{T}\int_{-T/2}^{T/2} \sin(\omega_n a_1)\,\varphi(a_1)dx.$$

Remark 6 Theorem 3.2 may be generalised to the decomposition of φ into three or more orthogonal components. E.g. $\varphi(a_1 + a_2 + a_3) = \varphi_1(a_1, a_2, a_3) + \varphi_2(a_1, a_2, a_3) + \varphi_3(a_1, a_2, a_3) + \varphi_4(a_1, a_2, a_3)$, where $\{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$ are orthogonal functions between themselves.

Axiom 3.1 The activation function $\varphi(a)$ obeys the following properties:

Subspace projection $\varphi(a) = \varphi_1(a + 0) = \varphi_2(0 + a)$
Decomposition $\varphi(a_1 + a_2) = \varphi_1(a_1 + a_2) + \varphi_2(a_1 + a_2)$
Additivity $\varphi(a_1 + a_2) = \varphi_1(a_1 + a_2) + \varphi_2(a_1 + a_2)$

Theorem 3.3 proves the interesting property of the orthogonality of the components of the output between themselves.

Theorem 3.3 (complementarity of $\varphi_i, i = 1, 2$) *The activation function components,* $\varphi_1(a_1 + a_2)$ *and* $\varphi_2(a_1 + a_2)$, *are orthogonal between themselves.*

Proof See Appendix B. □

For computational purposes, there is also a representation for the splitting factor of the output, given in Theorem 3.4. That is, $\varphi_1(a_1, a_2) = \zeta\varphi(a)$.

Theorem 3.4 (Output split factor) *The split factor for the output is*

$$\zeta = \frac{1}{2}\left(1 + \frac{\sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n(a_2 - a_1))}{\varphi(a)}\right), \tag{17}$$

with $a = a_1 + a_2$.

Proof

$$\begin{aligned}
\zeta &= \frac{\varphi_1(a_1, a_2)}{\varphi(a)} \\
&= \frac{\sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n a_1)\cos(\omega_n a_2)}{\sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n(a_1 + a_2))} \\
&= \frac{1}{2}\frac{\sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n(a_1 + a_2)) - \sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n(a_1 - a_2))}{\sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n(a_1 + a_2))} \\
&= \frac{1}{2}\left(1 + \frac{\sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n(a_2 - a_1))}{\varphi(a)}\right)
\end{aligned}$$

□

Remark 7 If $a_2 = 0$, then $a = a_1$, $\zeta = 0$. If $a_1 = 0$, then $a = a_2$, $\zeta = 1$. If $a_1 = a_2$, then $\zeta = \frac{1}{2}$.

4 Multi-back-propagation Algorithm

The back-propagation method [17] is used to train a feedforward NN to fit a data set, $\{x_j, d_j\}_{j=1,\ldots,N}$, where x_j is the input and d_j the desired output. That is, the weights $W \in \mathbb{R}^N$ are fine-tuned based on the current error rate, where every $d_j = y_j + e_j$ and $y_j = NN(x_j, W)$. Prior to the running of the algorithm, the user needs to choose the number of layers, the number of neurones in each layer, the integration and activation functions, the number of iterations and the learning rate. The NN weights, together with the bias of every neurone, are initially randomised and then iteratively readjusted, by sending messages forward and backward alternatively in the NN, until the error becomes acceptable [18]. In the end of this process, the adjustment function has been found. This identification process corresponds to a supervised learning method, widely used in parametric learning of NN, conventionally called NN back-propagation algorithm. Whenever this process proves to be suitable, the NN weights encode the information of the training data set. This is here referred to as the learning phase or Phase-1.

Assuming the decomposition of the information into several parcels to be possible, the herein proposed method will perform the adjustment task at the multidimensional neuron level, transferring the information between the respective sub-models. This is a multidimensional version of the error back-propagation technique, whose main objective is to maximise the separability criterion of the output components of the NN—Phase-2 or splitting phase.

Thus, once the adjustment function to the data has been found—Phase-1—the splitting phase—Phase-2—takes place: the neurones are forced to divide the data representing the adjustment function into two connected parts, which can be considered as the first and the second components. To do this, conceive the data set to be such that $\boldsymbol{d}_j = (d_j(1), d_j(2))$ with $d_j = d_j(1) + d_j(2)$ and $d_j(\ell) \approx y_j(\ell)$, $\ell = 1, 2$, where $y_j = y_j(1) + y_j(2)$ is calculated as in Theorem 3.2. The weights, W, and the splitting coefficients, ζ in (17), are randomised to initialise the Multi-back-propagation calculations. To calculate the splited output to a desired precision, the fraction square mean error (FSME) is minimised:

$$\min_W \sum_{j=1}^{N} \|\boldsymbol{e}_j\|^2 = \min_W \frac{1}{N} \sum_{j=1}^{N} 2\left(e_j(1) - e_j\right) e_j(1) + e_j^2. \tag{18}$$

Since

$$d_j = y_j + e_j \Leftrightarrow e_j = y_j - d_j$$

$$\begin{aligned} \text{with } d_j &= d_j(1) + d_j(2) \implies d_j(2) = d_j - d_j(1) \\ y_j &= y_j(1) + y_j(2) \implies y_j(2) = y_j - y_j(1) \end{aligned}$$

$$\begin{aligned} \implies \|\boldsymbol{e}_j\|_2^2 &= e_j(1)^2 + e_j(2)^2 \\ &= \left(d_j(1) - y_j(1)\right)^2 + \left(d_j(2) - y_j(2)\right)^2 \\ &= \left(d_j(1) - y_j(1)\right)^2 + \left(\left(d_j - y_j\right) - \left(d_j(1) - y_j(1)\right)\right)^2 \\ &= e_j(1)^2 + \left(e_j - e_j(1)\right)^2. \end{aligned}$$

So $\min_{W,\alpha} \|\boldsymbol{e}_j\|_2^2 \implies \min_{W,\alpha} e_j(1)^2 + \left(e_j - e_j(1)\right)^2$. □

4.1 Algorithm Outline

Based on the results found in Sect. 3 an algorithm is outlined next for the multi-dimensional version of the NN—a multidimensional back-propagation algorithm. To make the proposed algorithm easier to understand, it will be presented for dimension two.

The aim of the algorithm is to find a multilayer NN able to approximate the data set to a prescribed precision, that is, given $\{x_j, d_j\}_{j=1,\ldots,N}$, obtain y_j such that $\frac{1}{N}\sum_{j=1}^{N}(d_j - y_j)^2 < \varepsilon$, where ε is the required precision. Moreover, y_j is split into two components such that $d_j(\ell) \approx y_j(\ell)$, $\ell = 1, 2$, and $d_j = d_j(1) + d_j(2)$.

This algorithm takes place at Phase-2 and has got two main stages. The first stage is optional since the multidimensional NN may already exist. Otherwise, Stage-1 takes place (as described below) by transforming the NN into a multidimensional NN. To do so, all variables are transformed from scalars to vectors and the activation function is decomposed in an orthogonal way. The new parameters are initialised.

After Phase-1, the NN have a multidimensional structure, where one of the neurone's sub-models will be a copy of the original neurone, and the others being devoid of information (zero-valued weights). Phase-2 of the algorithm will then transfer information between the sub-models (dimensions) of the neurone using a process identical to the back-propagation algorithm. However, in this case, back-propagation is not performed on the approximation error of the NN output, but rather on the separability value of the output components of the multidimensional neural network, taking (18).

Stage–1 Transforms a NN into its multidimensional version.

- Write $\varphi(a)$ as a Fourier Series of m terms: $S_m(a)$
- Decompose the activation functions using Theorem 3.2.
- Initialise randomly the NN bias and splitting parameters $\boldsymbol{\alpha}$.

Stage–2 Propagating the separability error criterium backwards. Minimise FMSE using (18) by tuning $\boldsymbol{\alpha}$.

1. Calculate $y_j(1)$ and $y_j(2)$ using the splitting factor calculated in (17). I.e.
   (a) Decompose the integration function using Theorem 3.1.
   (b) Calculate the splitting factor ζ using (17).
   (c) Calculate $y_j(1)$ and $y_j(2)$ using $y_j(1) = \zeta y_j$ and $y_j(2) = (1-\zeta)y_j$, $j = 1, \ldots, N$.
2. Calculate $e_j(1) = y_j(1) - d_j(1)$ and $e_j = y_j - d_j$, $j = 1, \ldots, N$.
3. Evaluate $\frac{dE}{de_j(1)} = \frac{2}{N}\sum_{j=1}^{N}\left(2e_j(1) - e_j\right)$ to guide the search for maximum splitting into components $y_j(1)$ and $y_j(2)$.
4. Propagating the error backwards. Readjust $\boldsymbol{\alpha}$ to minimise (18).

In summary, the final objective of the algorithm is to decompose the network into two complementary parts. To do this, it minimises (18) taking into account maximum separability, which is achieved by taking advantage of component orthogonality. Obviously, every obtained component can be further split.

5 Discussion, Tests and Results

This section contains the assessment of different stages of the Multi-back-propagation algorithm and, in particular, of the neurone decomposition. Namely:

A. The validation of the approximation of an activation function—the hyperbolic tangent—by a Fourier Series of m terms. See Sect. 5.1.
B. The illustration of the decomposition of the activation function. See Sect. 5.2.
C. To illustrate the performance of the Multi-back-propagation algorithm, two different examples are implemented, a 2D classifier and the decomposition of a 3D network. Both examples demand the estimation of a NN with high precision for a non-linear target function. Hence:

    1. The decomposition of a 2D classification example into two sub-classifiers is presented in Sect. 5.3.
    2. The decomposition of a 3D NN whose transfer function, $y = NN(x_1, x_2)$, has the shape of a volcano is presented in Sect. 5.4. The network is split into two sub-models, $y = NN_1(x_1, x_2) + NN_2(x_1, x_2)$, where one identifies a mountain, $NN_1(x_1, x_2)$, and the other, $NN_2(x_1, x_2)$, the volcano crater, that once added up returns the volcano.

For both problems, the transfer function, that is the sum of the estimated sub-models, is not changed and is equal to the initial one, which has been identified using the standard back-propagation algorithm.
To assess the transfer of information in every NN layer, namely at the n inputs, the following metrics is used:

$$TI = \frac{\sum_{i=1}^{n} |\alpha_i||w_i|}{\|\tilde{w}\|}, \tag{19}$$

where n are the inputs of the neuron. $\boldsymbol{\alpha}$ and $\boldsymbol{w}$ are as defined in Sect. 3. This metrics is used to assess the information transfer in each neurone of every layer. It is calculated as the product of the α parameters by the weights of the normalised neurone. E.g for $n = 2$ and $p = 2$ it translates into $TI = \dfrac{|\alpha_1 w_1| + |\alpha_2 w_2|}{\sqrt{w_1^2 + w_2^2}}$.
The code [19] has been written in Matlab/ Octave without using a specific toolbox but implementing everything from scratch. Instead, Matlab served as the coding platform, chosen for its graphical capabilities.
In the implementation of both back-propagation algorithms, the classic (scalar version) and the multidimensional version, the gradient-descent method has been used; this is a method simple to implement but of slow convergence. Although the computational results were quite good, its convergence might be improved by choosing some non-linear methods as for instance the Levenberg-Marquardt. However, since the results obtained in this work were quite satisfactory, we do not consider this to be an essential issue in the work.

5.1 Approximation of φ by a Fourier Series

To assess the decomposition of the activation function into orthogonal components, we consider $\varphi(a) = \tanh(a)$, an activation function commonly chosen [12]. Φ is defined as an extension of the hyperbolic tangent $\varphi(a) = \dfrac{\left(e^a - e^{-a}\right)}{\left(e^a + e^{-a}\right)}$, with $\underline{a} = -4$ and $\overline{a} = 4$. As $\delta = 1$, then $\varepsilon = -1 - \varphi(-4) = 1 - \varphi(4) \approx 6.7 \times 10^{-4}$.
To obtain the following result, we apply Lemma 2.1 to $\varphi(x) = \tanh(a)$.

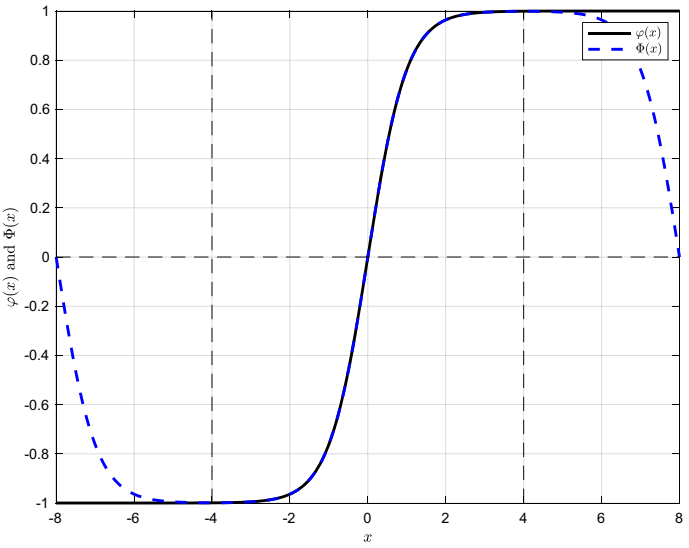


Fig. 3 Representation of $\Phi(a)$, $a \in \left[-\frac{T}{2}, \frac{T}{2}\right]$ and period $T = 16$

Corollary 1 *Considering* $\varphi(x) = \tanh(a)$ *approximated by the Fourier Series of m terms,* $S_m(a)$, *in interval* $\left[-\frac{T}{4}, \frac{T}{4}\right]$, *thence the value of the mean quadratic error is*

$$\int_{-T/4}^{T/4} R_m^2(a)da = \frac{T}{2}\left(1 - \frac{4}{T}\tanh\left(\frac{T}{4}\right) - \frac{1}{2}\sum_{i=m+1}^{\infty}\hat{\Phi}_n^2\right). \tag{20}$$

Proof

$$\int_{-T/4}^{T/4} |\varphi(a)|^2 da = \int_{-T/4}^{T/4} |S_m(a)|^2 da + \int_{-T/4}^{T/4} |R_m(a)|^2 da \Leftrightarrow$$

$$\Leftrightarrow \frac{2}{T}\int_{-T/4}^{T/4} |R_m(a)|^2 da = \frac{2}{T}\int_{-T/4}^{T/4} |\varphi(a)|^2 da - \frac{2}{T}\int_{-T/4}^{T/4} |S_m(a)|^2 da \Leftrightarrow$$

$$= \frac{2}{T}\int_{-T/4}^{T/4} |\tanh(a)|^2 da - \frac{2}{T}\int_{-T/4}^{T/4} |S_m(a)|^2 da \Leftrightarrow$$

$$= \frac{2}{T}|a - \tanh(a)|_{-T/4}^{T/4} - \frac{T}{2}\sum_{i=m+1}^{\infty}\hat{\Phi}_n^2 \Leftrightarrow$$

$$= 1 - \frac{4}{T}\tanh\left(\frac{T}{4}\right) - \frac{1}{2}\sum_{i=m+1}^{\infty}\hat{\Phi}_n^2.$$

□

Table 1 $\varphi(x) = \tanh(x)$, $\omega_0 = \pi/8$, $m = 8$, $T = 16$

F. coeff/value	F. coeff/value	F. coeff/value
$\hat{\Phi}_1 = 1.19611056;$	$\hat{\Phi}_4 = 0.02088747;$	$\hat{\Phi}_7 = 0.00048394;$
$\hat{\Phi}_2 = 0.25297128;$	$\hat{\Phi}_5 = 0.00611703;$	$\hat{\Phi}_8 = 0.00020280.$
$\hat{\Phi}_3 = 0.07212369;$	$\hat{\Phi}_6 = 0.00178229;$	

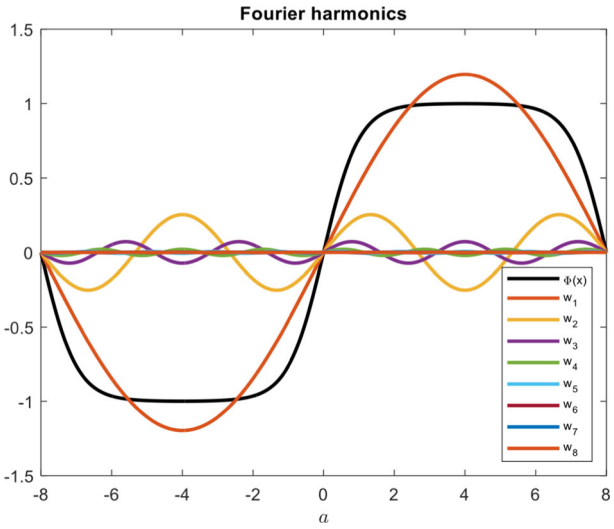


Fig. 4 Fourier harmonics representation: $\varphi(x) = \tanh(x)$, $\omega_0 = \pi/8$, $m = 8$ and $T = 16$

Corollary 2 *Assuming the saturation level* $\delta = 1$ *for the activation function* $\varphi(a) = \tanh(a)$, *the value of the mean quadratic error in intervals* $\left[-\infty, -\frac{T}{4}\right] \cup \left[\frac{T}{4}, \infty\right]$ *is*

$$\int_{-\infty}^{-T/4} R_m^2(x)dx + \int_{T/4}^{\infty} R_m^2(x)dx = 2\int_{-\infty}^{-T/4} (1+\tanh(x))^2 dx$$
$$= 4\left(\log\left(2\cosh\frac{T}{4}\right) - \frac{T}{4}\right) + 2\tanh\frac{T}{2} - 2. \quad (21)$$

Proof The result is obtained after a few standard calculations. □

Take $m = 8$ and $T = 16$ and then the hyperbolic tangent activation function: $\varphi(x) = \tanh(x) \approx \sum_{n=1}^{m=8} \hat{\Phi}_n \sin\omega_m a$. Table 1 have the Fourier Series coefficients values, $\hat{\Phi}_n$, $n = 1, \cdots, m$. Fourier's harmonics terms are show in Fig. 4 as well as the Fourier series approximation.
From Formulas (20) and (21) the errors are calculated:

$[-T/4, T/4]$: $R_m^2(x) = 1.0 \times 10^{-7}$
$[-\infty, -T/4] \cup [T/4, \infty]$: $R_m^2(x) = 6.7 \times 10^{-4}$

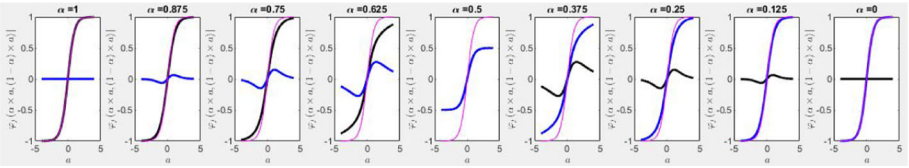
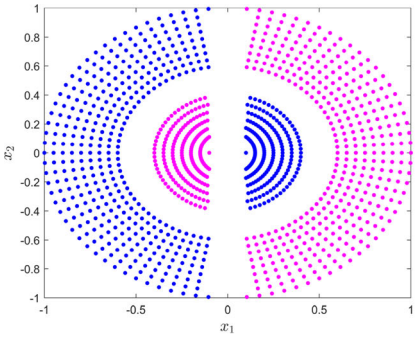
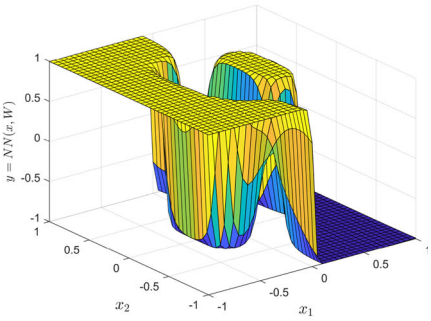


Fig. 5 Decomposition of $\tanh(a)$ into orthogonal components $\varphi_1(a_1, a_2)$ and $\varphi_2(a_1, a_2)$ using different splitting values



(a) Result obtained with the standard back-propagation algorithm



(b) Transfer function of the classifier: $y_j = NN(x_j, W)$

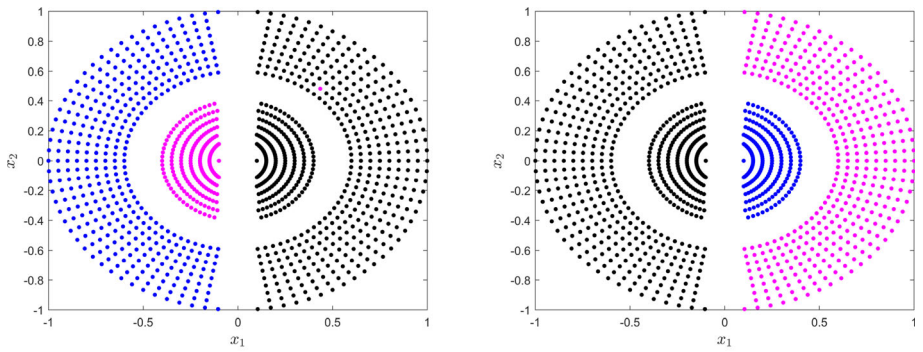
Fig. 6 Training data for the classification problem: blue—class+1, $d = +1$; magenta—class-1, $d = -1$

5.2 Orthogonal Decomposition of φ

Considering $\varphi(a) = \tanh(a)$, Fig. 5 shows its decomposition into orthogonal components $\varphi_1(a_1, a_2)$ and $\varphi_2(a_1, a_2)$ using different splitting values λ, i.e., $a_1 = \lambda a$ and $a_2 = (1 - \lambda)a$. Colour black represents component $\varphi_1(a_1, a_2)$, blue $\varphi_2(a_1, a_2)$ and magenta the sum $\varphi_1(a_1, a_2) + \varphi_2(a_1, a_2)$. The results reveal that the two components complement each other, with the sum $\varphi_1(a_1, a_2) + \varphi_2(a_1, a_2)$ matching $\varphi(a)$ for all and every $\lambda \in \left\{0, \frac{1}{8}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}, 1\right\}$. According to the input decomposition obtained, $a = a_1 + a_2$, the output splitting factor ζ is calculated from (17).

5.3 2D Classification Problem

In this 2D example, a NN classifier is decomposed into two sub-classifiers such that the first one has the left half-plane as domain and the second the right half-plane. The chosen function/data represents four semi-circles, two in each half plane as shown in Fig. 6a. The semi-circles are not compact to attain a higher difficulty to approximate the function. The classification has the following requirements: (1) whenever a small semi-circle belongs to a class, the exterior semi-circle in the same half-plane should belong to the other class; (2) whenever a small semi-circle belongs to a class, the other small semi-circle should belong to the other class. The classification was done in two consecutive stages: Stage-1) using a standard feedforward NN and the standard back-propagation algorithm. Stage-2) conceiving a multivariable network and using the Multi-back-propagation algorithm.



(a) Sub-model 1: left half-plane classifier. Black: non-classified points

(b) Sub-model 2: right half-plane classifier. Black: non-classified points

Fig. 7 Classification using the Multi-back-propagation algorithm

The initial NN divides the space into regions, labelled Classes 1 and 2 (in magenta and blue, respectively, and labelled −1 and 1). The results obtained at the first stage entirely reproduced the initial data, i.e., Fig. 6. After 10^5 iterations, the NN shows a MSE of 1.403805×10^{-4} in the classification, when using the standard back-propagation algorithm. For a null threshold of the neuronal network output, the MSE is zero.

The proposed method decomposes the original NN into components. The first component classifies only the left half-plane and the second the right half-plane. In regions not covered by each of the sub-models, the NN output components yield a null result (labelled 0) which means the absence of classification. This example aims to illustrate a classification problem into subproblems restricted to sub-regions of the classification space.

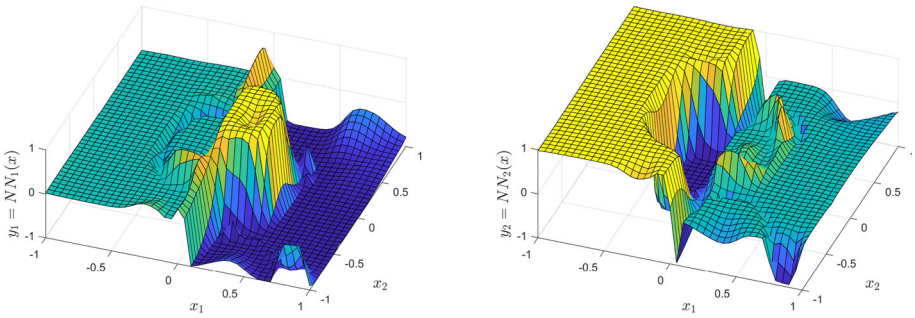
Fig. 7 shows the results obtained with the second stage. The first component of the NN classifies for $x_1 < 0$, left half-plane and it is neutral in the right half-plane—Null classification value (See Fig. 7a). The second component of the NN classifies the region $x_1 >= 0$ and it is null for the left half-plane (See Fig. 7b). Figure 8 shows the surfaces of the transfer functions for both NN components.

After 10^5 iterations and considering a learning factor of 0.2 and a MSE of 0.0038 in the last iteration, the sum of two partial classifications was obtained with a MSE $= 6.5939 \times 10^{-7}$ when compared to the original classification at the first stage. However, both show a classification error null for thesholders ± 0.5. One can observe that the sum of the two classifiers gives the same result as the standard approach.

Fig. 9a shows the transfer of information TI, as defined in (19), and the $\boldsymbol{\alpha}$ broken input vector, as defined in (17), for all neurons of the two layers of the NN. In Fig. 9b the split classifier shows a FMSE equal to 4.922010×10^{-1} and a partial MSE for the first subproblem, according to formula (18), of 0.0037.

5.4 Surface Decomposition: Volcano

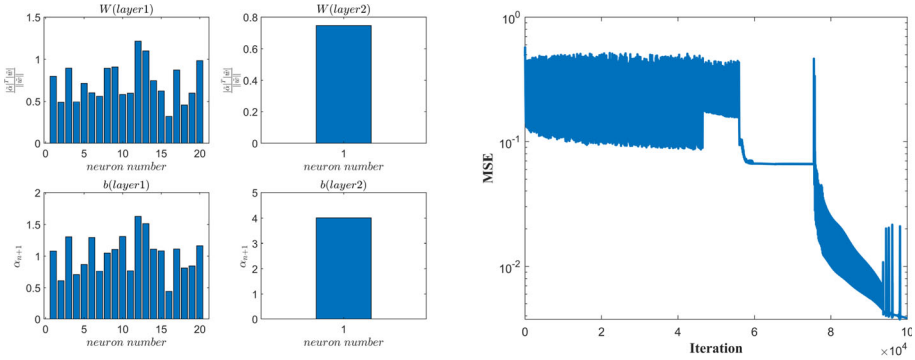
We discuss the decomposition of the identification of a surface that has the shape of a volcano. In this example a volcano surface is obtained as the result of the sum of a mountain with a crater. This is illustrated in the first row of Fig. 10 where the mountain is the first figure, the crater is the second and the volcano is the third. At Stage-1, the standard back-propagation



(a) Sub-model1: $y_j(1) = NN(x_j, \boldsymbol{w}, \boldsymbol{\alpha})$. MSE $= 1.706485e-03$

(b) Sub-model2: $y_j(2) = NN(x_j, \boldsymbol{w}, \boldsymbol{\alpha})$. MSE $= 1.760563e-03$

Fig. 8 Classification curves of the sub-models



(a) Every neurone information transfer

(b) FMSE: Multi-back-propagation

Fig. 9 Assessment of the results obtained with the splitting method applied to the classifier

technique used a NN of 4 layers, with 20 and 10 neurones in the hidden layers, respectively, and the $\tanh(a)$ as the activation function, to initialise the whole procedure.

Next, at Stage-2, we want to decompose the initial $NN\ (x_1, x_2)$, whose transfer function represents the surface of a volcano. The two components $NN_1\ (x_1, x_2)$ and $NN_2\ (x_1, x_2)$ model the mountain and the crater, respectively, using a separability criterium that is dominantly mountain-shaped for the 1st component, while the 2nd component of the NN network is decomposed with the remaining part of the NN (i.e., the crater-shaped form).

The decomposition was done using the Multi-back-propagation algorithm. The results obtained can be seen in the second row of Fig. 10, where the third figure of the row shows the reconstitution of the volcano. Figure 11 shows the transfer of information for every neurone of the two layers of the NN, using the metric defined in (17) and in (19). This example shows to be possible to decompose the transfer function of a NN as a sum of components where every component models a different aspect.

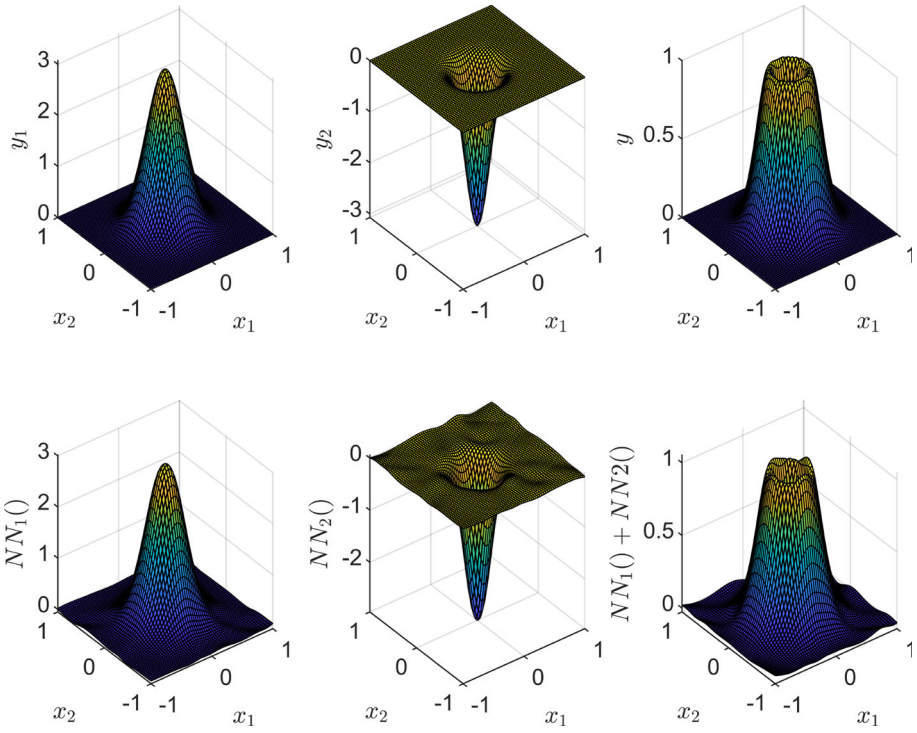
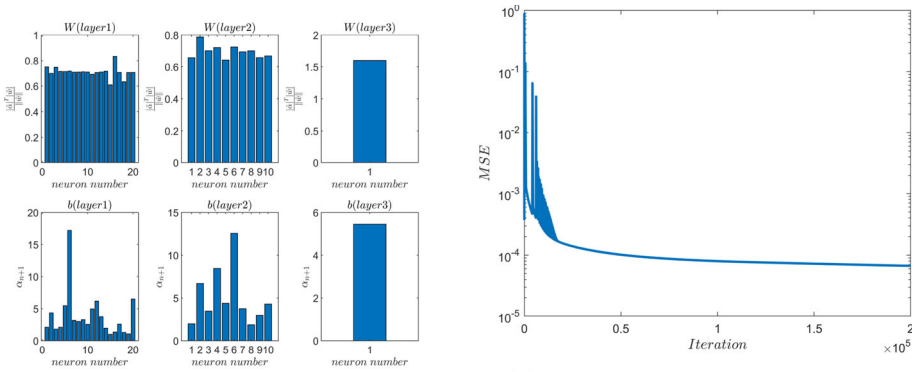


Fig. 10 Row 1: envisage the surface decomposition. Row 2: surface decomposition using the splitting approach



(a) Information transfer

(b) MSE evolution: back-propagation algorithm

Fig. 11 Assessment of the results: splitting method applied to the volcano

6 Conclusions and Future Work

This work presents a new algorithm for network training that aims to re-arrange the information of an already existing NN. To do this, the neurone is seen as a multivariable entity whose inputs, bias, and outputs are split into two or more components to optimise a separability

criterium. Fundamental to the whole procedure is to demonstrate that the activation function can be approximated by a Fourier series of m terms and that this is the best approximation of its order in the space $L^2([-T/2, T/2])$. Moreover, the components of the activation function are orthogonal between themselves, assuring full decomposition as well as reconstitution of the original whole. Thus, the transfer function is decomposed into orthogonal sub-models whose sum reconstitutes the original shape, which means that information is preserved.

The result is a new algorithm, the Multi-back-propagation method. The outcome of the Multi-back-propagation method is the splitting of the NN into disjoint components. At a first phase, the algorithm uses the standard back-propagation technique to train the NN from data. This first phase, is optional since the NN may already exist. At the second phase, the trained network is split into components. The whole procedure was outlined in Sect. 4.

In Sect. 5, it was our objective to illustrate the different features of the algorithm, that we view as an extension of the back-propagation technique applied when the decomposition of certain problems becomes possible. Thence, in Sect. 5.1, the approximation of the activation function by a truncated Fourier series is illustrated for a particular function - the hyperbolic tangent. This approximation is decomposed into orthogonal components which enables the splitting of the input and output signals and leads to a sum that recuperates the original trained NN. The orthogonal decomposition, using different splitting values, is assessed in Sect. 5.2. Moreover, the different aspects of the decomposition of the multivariable neurone have been mathematically demonstrated. In Sect. 5.3 and 5.4, the method is applied to two different problems: (1) a 2D classification problem and (2) the identification of a 3D surface. In both cases, it was possible to recuperate the original transfer function of the NN from the splitting disjoint components. The assessment examples are original and have also been constructed from scratch to accommodate the idiosyncrasies of the presented method.

We would like to mention that although the issue of information separability criteria has not been addressed in this work, we present two examples to assess the herein presented method. In the first example (Sect. 5.3), the classifier is to unfold into two sub-classifiers which are applicable to the two distinct half-planes of the classification domain. In the second example (Sect. 5.4), the separability criterion used favours the sub-model with a mountain shape and leaving the remaining information to the other sub-model.

From our research, we find this method very promising, specifically in separating different aspects of a problem. We also understand that a few issues still need further research and we would like to delve deeper into them. Namely, we would like to do some more work to strengthen the algorithm. For instance, in the example of Sect. 5.1, the decomposition of $\tanh(a)$ has been done using the same $\boldsymbol{\alpha}$ for all the harmonics of the Fourier series. However, different values of $\boldsymbol{\alpha}$ can be used to calculate the different terms.

The issue of information separability criteria should be addressed. Also, some different orthogonal functions can be used to approximate the activation function. An example that splits the NN information into more than two parts has to be constructed. Moreover, different metrics can be used to assess the transfer of information between neurones. Paramount is also to apply the splitting multi-algorithm to some problems with real added value.

Acknowledgements Work financed by FCT - Fundação para a Ciência e a Tecnologia under project: (i) UIDB/04033/2020 for the first author. (ii) UIDB/00048/2020 for the second author.
The authors would like to thank the reviewers and the editor in chief of the Neural Processing Letters for their careful reading and helpful comments and suggestions.

Funding Open access funding provided by FCT|FCCN (b-on).

Declarations

Conflict of interest Both authors certify that they have no affiliations with or involvement in any organisation or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

A Mathematical Proofs of Subsection 2

A.1 Proof of Theorem 2.1

Write the Fourier Series of m terms for $\Phi(a)$, where $\hat{\Phi}_n$ are the Fourier coefficients: $S_m(a) = \sum_{n=-\infty}^{\infty} \hat{\Phi}_n e^{2\pi i n a/T}$ with $\hat{\Phi}_n = \frac{1}{T}\int_{-T/2}^{T/2} e^{-2\pi i n a/T}\,\Phi(a)da$. As $\Phi(a)$ is assumed to be known, $\hat{\Phi}_0 = \frac{1}{T}\int_{-T/2}^{T/2} \Phi(a)da$ is the average value of the function in the period and coefficients $\hat{\Phi}_n$ are complex: $\hat{\Phi}_n = \frac{1}{2}(a_n - ib_n)$ and $\hat{\Phi}_{-n} = \frac{1}{2}(a_n + ib_n)$, $n = 1, \ldots, m$. Hence:

$$\begin{aligned} S_m(a) &= \sum_{n=-m}^{m} \hat{\Phi}_n e^{2\pi i n a/T} \\ &= \frac{\hat{\Phi}_0}{2} + \sum_{n=1}^{m} \hat{\Phi}_n e^{2\pi i n a/T} + \sum_{n=-1}^{-m} \hat{\Phi}_n e^{2\pi i n a/T} \\ &= \frac{\hat{\Phi}_0}{2} + \sum_{n=1}^{m} \left(a_n \cos 2\pi n a/T + b_n \sin 2\pi n a/T\right). \end{aligned}$$

Also, if we assume the signal Φ to be real, then $\hat{\Phi}_{-n} = \overline{\hat{\Phi}_n} = \hat{\Phi}_n$. When $n = 0$, $c_0 = \frac{a_0}{2}$ appears twice. We make the change of variable: $\omega_0 = 2\pi/T$, define $\omega_n = n\omega_0$, and next write the Fourier Series in this new variable: $S_m(a) = \frac{\hat{\Phi}_0}{2} + \sum_{n=1}^{m}\left(a_n \cos n\omega_0 a + b_n \sin n\omega_0 a\right)$. Furthermore, if Φ is an odd function (and real), then the Fourier coefficients are also odd. Hence:

$$S_m(a) = \sum_{n=1}^{m} b_n \sin(n\omega_0 a) = \sum_{n=1}^{m} b_n \sin(\omega_n a)$$

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} \sin(\omega_n a)\, \Phi(a) da.$$

Without loss of generality, we can redefine the Fourier coefficients for an odd function $\Phi(a)$ as $\hat{\Phi}_n := b_n$. Equivalently:

$$\Phi(a) = \sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n a) + \sum_{n=m+1}^{\infty} \hat{\Phi}_n \sin(\omega_n a) \tag{22}$$

with $\hat{\Phi}_n = \frac{2}{T} \int_{-T/2}^{T/2} \sin(\omega_n a)\, \Phi(a) da.$

Moreover, as Φ is symmetric in the positions $2k\frac{T}{4}$, $k \neq 0$, then only the odd harmonics of series (22) will be considered. That is, $\omega_n = (2n-1)\omega_0$.

A.2 Proof of Lemma 2.1

Recalling the Parseval-Rayleigh's identitity, that says that we can compute the energy of the signal by adding up the energies of the individual harmonics:

$$\frac{1}{T} \int_{-T/2}^{T/2} |\Phi(a)|^2 da = \sum_{n=1}^{\infty} \hat{\Phi}_n^2. \tag{23}$$

For Φ defined in $L^2\left([2\underline{a}, 2\overline{a}]\right)$, we have

$$\begin{aligned}
\int_{-T/2}^{T/2} |\Phi(a)|^2 dx = \|\Phi(a)\|^2 &= \langle \Phi(a), \Phi(a) \rangle \\
&= \langle S_m(a) + R_m(a), S_m(a) + R_m(a) \rangle \\
&= \langle S_m(a), S_m(a) \rangle + 2\langle S_m(a), R_m(a) \rangle + \langle R_m(a), R_m(a) \rangle \\
&\qquad \text{By orthogonality} \\
&= \langle S_m(a), S_m(a) \rangle + \langle R_m(a), R_m(a) \rangle \\
&\qquad \text{Again by orthogonality} \\
&= \|S_m(a)\|^2 + \|R_m(a)\|^2 \\
&= \int_{-T/2}^{T/2} |S_m(a)|^2 da + \int_{-T/2}^{T/2} |R_m(a)|^2 da
\end{aligned}$$

Trivially, follows:

$$\frac{1}{T} \int_{-T/2}^{T/2} |\Phi(a)|^2 da = \sum_{n=1}^{\infty} \hat{\Phi}_n^2 \Leftrightarrow$$

$$\Leftrightarrow \frac{1}{T} \left(\int_{-T/2}^{T/2} |S_m(a)|^2 da + \int_{-T/2}^{T/2} |R_m(a)|^2 da \right) = \sum_{n=1}^{m} \hat{\Phi}_n^2 + \sum_{n=m+1}^{\infty} \hat{\Phi}_n^2$$

$$\Longrightarrow \frac{1}{T}\int_{-T/2}^{T/2}|R_m(a)|^2 da = \sum_{n=m+1}^{\infty}\hat{\Phi}_n^2 \tag{24}$$

$$\text{and} \quad \frac{1}{T}\int_{-T/2}^{T/2}|S_m(a)|^2 da = \sum_{n=1}^{m}\hat{\Phi}_n^2 \tag{25}$$

A.3 Proof of Lemma 2.2

Start in (6) and partition the integral in three different intervals: $\left[-\frac{T}{2}, -\frac{T}{4}\right], \left[-\frac{T}{4}, \frac{T}{4}\right], \left[\frac{T}{4}, \frac{T}{2}\right]$:

$$\begin{aligned}\hat{\Phi}_n &= \frac{2}{T}\int_{-T/2}^{T/2}\sin(\omega_n a)\,\Phi(a)da \\ &= \frac{2}{T}\int_{-T/2}^{-T/4}\sin(\omega_n a)\,\Phi(a)da + \frac{2}{T}\int_{-T/4}^{T/4}\sin(\omega_n a)\,\Phi(a)da + \frac{2}{T}\int_{T/4}^{T/2}\sin(\omega_n a)\,\Phi(a)da\end{aligned}$$

In (2) consider $2\underline{a} = -T/2$, $\underline{a} = -T/4$, $\overline{a} = T/4$ and $2\overline{a} = T/2$ and use the definition of Φ according to the correct interval:

$$\begin{aligned}&= \frac{2}{T}\int_{-T/2}^{-T/4}\sin(\omega_n a)\,\varphi\left(-a-\frac{T}{2}\right)da + \frac{2}{T}\int_{-T/4}^{T/4}\sin(\omega_n a)\,\varphi(a)da \\ &\quad + \frac{2}{T}\int_{T/4}^{T/2}\sin(\omega_n a)\,\varphi\left(-a+\frac{T}{2}\right)da\end{aligned} \tag{26}$$

First we calculate:

$$\begin{aligned}\int_{-T/2}^{-T/4}\sin(\omega_n a)\,\varphi\left(-a-\frac{T}{2}\right)dx &= \int_{-T/2}^{-T/4}\sin(\omega_n a)\,\varphi\left(-(a+\frac{T}{2})\right)da \\ &= \int_{-T/2}^{-T/4} -\sin(\omega_n a)\,\varphi\left(a+\frac{T}{2}\right)da \\ &= -\int_{-T/4}^{-T/2} -\sin(\omega_n a)\,\varphi\left(a+\frac{T}{2}\right)da \\ &= \int_{-T/4}^{-T/2}\sin(\omega_n a)\,\varphi\left(a+\frac{T}{2}\right)da\end{aligned}$$

To arrive at this result, we took into account that sine is an odd function and use the properties of the Riemann integral.

Now, changing the variable: $y = -a \implies dy = -da$

$$
\begin{aligned}
&= \int_{T/4}^{T/2} \sin\left(\omega_n(-y)\right)\ \varphi\left(-y+\frac{T}{2}\right)(-dy), \text{ knowing that sine is an odd function} \\
&= \int_{T/4}^{T/2} -\sin(\omega_n y)\,\varphi\left(-y+\frac{T}{2}\right)(-dy) \\
&= \int_{T/4}^{T/2} \sin(\omega_n y)\,\varphi\left(-y+\frac{T}{2}\right)dy
\end{aligned}
$$

Substituting this result back in (26), we get:

$$
\begin{aligned}
\hat{\Phi}_n &= \frac{4}{T}\int_{T/4}^{T/2} \sin(\omega_n a)\,\varphi\left(-a+\frac{T}{2}\right)da + \frac{2}{T}\int_{-T/4}^{T/4} \sin(\omega_n a)\,\varphi(a)da \\
&= \frac{4}{T}\int_{T/4}^{T/2} \sin(\omega_n a)\,\varphi\left(-a+\frac{T}{2}\right)da + \frac{4}{T}\int_{0}^{T/4} \sin(\omega_n a)\,\varphi(a)da \qquad (27)
\end{aligned}
$$

Next, we proof that: $\int_{T/4}^{T/2} \sin(\omega_n a)\,\varphi\left(-a+\frac{T}{2}\right)da = \int_{-T/4}^{0} \sin(\omega_n a)\,\varphi(a)da$. Changing the variable: $z = a - \frac{T}{2}$, then:

$$
\begin{aligned}
\int_{T/4}^{T/2} \sin(\omega_n a)\,\varphi\left(-a+\frac{T}{2}\right)da &= \int_{-T/4}^{0} \sin\left(\omega_n\left(z+\frac{T}{2}\right)\right)\varphi(-z)dz \qquad \varphi \text{ is odd} \\
&= \int_{-T/4}^{0} \sin\left(\omega_n z + \omega_n\frac{T}{2}\right)\left(-\varphi(z)\right)dz \\
&= \int_{-T/4}^{0} \sin\left(\omega_n z + n\pi\right)\ \left(-\varphi(z)\right)dz \\
&= \int_{-T/4}^{0} \left(-\sin\left(\omega_n z\right)\right)\ \left(-\varphi(z)\right)dz \\
&= \int_{-T/4}^{0} \sin\left(\omega_n z\right)\ \varphi(z)dz,
\end{aligned}
$$

since $\omega_n = n\omega_0$, with $\omega_0 = \frac{2\pi}{T} = \frac{2\pi}{2\pi}$ then $\omega_n \frac{T}{2} = n\omega_0 \frac{T}{2} = n\frac{2\pi}{T}\frac{T}{2} = n\pi$.
Back to (27):

$$\hat{\Phi}_n = \frac{4}{T}\left(\int_{-T/4}^{0} \sin(\omega_n a)\,\varphi\,(a)\,da + \int_{0}^{T/4} \sin(\omega_n a)\,\varphi(a)da\right)$$
$$= 2\left(\underbrace{\frac{2}{T}\int_{-T/4}^{T/4} \sin\omega_n a\,\varphi(a)da}_{:=\hat{\varphi}_n}\right).$$

B Mathematical Proofs of Sect. 3

B.1 Proof of Theorem 3.1

Assume

$$\begin{aligned}
a_1 &= \sum_{i=1}^{n+1} \tau_i w_i x_i(1)\\
&= \sum_{i=1}^{n+1} \tau_i w_i \beta_i(1) x_i = \boldsymbol{w}^T \mathrm{diag}(\boldsymbol{\tau})\mathrm{diag}(\boldsymbol{\beta}))\boldsymbol{x}\\
&= \boldsymbol{w}^T \mathrm{diag}(\boldsymbol{\tau} \odot \boldsymbol{\beta})\boldsymbol{x}\\
a_2 &= a - a_1 = \sum_{i=1}^{n+1} w_i x_i - \sum_{i=1}^{n+1} \tau_i w_i x_i(1)\\
&= \sum_{i=1}^{n+1} w_i\,(x_i(1) + x_i(2)) - \sum_{i=1}^{n+1} \tau_i w_i x_i(1)\\
&= \sum_{i=1}^{n+1} w_i x_i(2) + \sum_{i=1}^{n+1} (1-\tau_i)\, w_i x_i(1)\\
&= \boldsymbol{w}^T\,((I - \mathrm{diag}(\boldsymbol{\tau}))\,X_1 + X_2)\\
&= \boldsymbol{w}^T\,(\boldsymbol{x} - \mathrm{diag}(\boldsymbol{\tau})X_1)\\
&= \boldsymbol{w}^T\,(I - \mathrm{diag}(\boldsymbol{\tau} \odot \boldsymbol{\beta}))\,\boldsymbol{x}
\end{aligned}$$

And we have for every i, $a_1 + a_2 = \sum_{i=1}^{n+1} (\tau_i w_i \beta_i(1) x_i + w_i(1 - \tau_i \beta_i) x_i) = w_i x_i$.

B.2 Proof of Theorem 3.2

Recalling the approximation of φ by the Fourier m-Series:

$$\varphi(a) \approx \sum_{n=1}^{m} \hat{\Phi}_n \sin(\omega_n a) \Leftrightarrow$$

$$
\begin{aligned}
\Leftrightarrow \varphi\left(a_1+a_2\right) &\approx \sum_{n=1}^{m} \hat{\Phi}_n \sin\left(\omega_n\left(a_1+a_2\right)\right) \Leftrightarrow \\
&\approx \sum_{n=1}^{m} \hat{\Phi}_n \left(\sin\left(\omega_n a_1\right)\cos\left(\omega_n a_2\right)+\cos\left(\omega_n a_1\right)\sin\left(\omega_n a_2\right)\right) \Leftrightarrow \\
&\approx \sum_{n=1}^{m} \hat{\Phi}_n \sin\left(\omega_n a_1\right)\cos\left(\omega_n a_2\right) + \sum_{n=1}^{m} \hat{\Phi}_n \cos\left(\omega_n a_1\right)\sin\left(\omega_n a_2\right) \\
&\approx \varphi_1\left(a_1, a_2\right) + \varphi_2\left(a_1, a_2\right)
\end{aligned}
$$

B.3 Proof of Theorem 3.3

We know that

$$
\int_{-T/2}^{T/2} \sin n\omega_0 t \cos m\omega_0 t dt = 0, \forall n, m. \tag{28}
$$

We want to proof that

$$
\begin{aligned}
&\int_{-T/2}^{T/2}\int_{-T/2}^{T/2} \varphi_1\left(a_1, a_2\right)\varphi_2\left(a_1, a_2\right) da_1 da_2 \\
&= \int_{-T/2}^{T/2} \sum_{n=1}^{m} \hat{\Phi}_n \sin\left(\omega_n a_1\right)\cos\left(\omega_n a_2\right) \sum_{j=1}^{m} \hat{\Phi}_j \cos\left(\omega_j a_1\right)\sin\left(\omega_j a_2\right) da_1 da_2 \\
&= \sum_{n=1}^{m}\sum_{j=1}^{m} \hat{\Phi}_n \hat{\Phi}_j \int_{-T/2}^{T/2}\int_{-T/2}^{T/2} \sin\left(\omega_n a_1\right)\cos\left(\omega_n a_2\right)\cos\left(\omega_j a_1\right)\sin\left(\omega_j a_2\right) da_1 da_2 \\
&= \sum_{n=1}^{m}\sum_{j=1}^{m} \hat{\Phi}_n \hat{\Phi}_j \int_{-T/2}^{T/2} \underbrace{\left(\int_{-T/2}^{T/2} \sin\left(\omega_n a_1\right)\cos\left(\omega_j a_1\right) da_1\right)}_{=0} \cos\left(\omega_n a_2\right)\sin\left(\omega_j a_2\right) da_2 \\
&= 0.
\end{aligned}
$$

References

1. Baughman DR, Liu YA (1995) 2—Fundamental and practical aspects of neural computing. In: Baughman DR, Liu YA (eds) Neural networks in bioprocessing and chemical engineering. Academic Press, Boston, pp 21–109
2. Bishop CM (1995) Neural networks for pattern recognition. Oxford University Press Inc, USA
3. Kevin LP, Keller PE (2005) Artificial neural networks: an introduction. SPIE Press - Technology & Engineering, Bellingham, Washington, USA
4. Urban S, van der Smagt P (2016) A neural transfer function for a smooth and differentiable transition between additive and multiplicative interactions. arXiv: 1503.05724
5. Kim J, Park Y, Kim G, Hwang SJ (2017) SplitNet: ksemantically split deep networks for parameter reduction and model parallelization. In: Precup D, Teh YW (eds.) Proceedings of the 34th international conference on machine learning, vol 70, pp 1866–1874. PMLR, Sydney NSW Australia

6. Wołczyk M, Tabor J, Śmieja M, Maszke S (2019) Biologically-inspired spatial neural networks. arXiv:1910:02776
7. Hosseini E Malek, Hajabdollahi M, Karimi N, Samavi S, Shirani S (2020) Splitting convolutional neural network structures for efficient inference. arXiv: 2002.03302
8. Henriksen P, Lomuscio A (2021) Deepsplit: an efficient splitting method for neural network verification via indirect effect analysis. In: Zhou Z-H (ed.) Proceedings of the thirtieth international joint conference on artificial intelligence, IJCAI-21, Montreal, Canada, pp 2549–2555
9. Wynne-Jones M (1993) Node splitting: a constructive algorithm for feedforward neural networks. Neural Comput Appl 1(1):17–22
10. Adamu A, Maul T, Bargiela A (2013) On training neural networks with transfer function diversity. In: Proceedings of the international conference on computational intelligence and information technology, CIIT '13, Mumbai, India
11. Duch W, Jankowski N (2001) Transfer functions: hidden possibilities for better neural networks. In: Brown D, Green S (eds.) Proceedings of the ESANN 2001, 9th European symposium on artificial neural networks, Bruges, Belgium, pp 25–27 (2001). ACM
12. Himanshu S (2019) Activation functions : sigmoid, tanh, ReLU, Leaky ReLU, PReLU, ELU, Threshold ReLU and Softmax basics for neural networks and deep learning. https://himanshuxd.medium.com/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e
13. Cohen MA, Tan CO (2012) A polynomial approximation for arbitrary functions. Appl Math. Lett. 25(11):1947–1952
14. Nebioglu B, Iliev AI (2023) Higher order orthogonal polynomials as activation functions in artificial neural networks. Serdica J. Comput. 17(1):1–16
15. Katznelson Y (2004) An introduction to harmonic analysis, 3rd edn. Cambridge University Press, Cambridge, UK
16. Osgood PB (2014) Lecture Notes for EE 261 the fourier transform and its applications. Create Space Independent Publishing Platform, Stanford University, CA, USA
17. Bishop CM (2006) Pattern recognition and machine learning. Springer, Cambridge, UK
18. Rumelhart DE, Hilton GE, Williams RJ (1986) Learning internal representations by error propagation. In: Rumelhart DE, JLM, Group PR (eds) Parallel distributed processing: explorations in the microstructure of cognition, volume 1: foundations, pp 318–362. MIT Press. Reprinted in Anderson and Rosenfeld (1988), Cambridge, MA (1986)
19. Salgado PAC (2023) Multi-Back-Propagation algorithm—the code. https://meocloud.pt/link/3c3b7720-8108-4ea9-9e8d-6a9db05d0df8/MATLAB/ (Accessed 21 July 2023)