# Analytical Prediction of Steel Grid-Shell Stability and Dynamic Behaviors Using Neural Networks – Part 1

Miguel Abambres [1,1] and Cabello A [2]

[1]Num3ros
[2]Affiliation not available

March 19, 2024

**Abstract**

Artificial Intelligence is a cutting-edge technology expanding very quickly into every industry. It has made its way into structural engineering and it has shown its benefits in predicting structural performance as well as saving modelling and experimenting time. This paper is the first one (out of three) of a broader research where artificial intelligence was applied to the stability and dynamic analyzes of steel grid-shells. In that study, three Artificial Neural Networks (ANN) with 8 inputs were independently designed for the prediction of a single target variable, namely: (i) the critical buckling factor for uniform loading (i.e. over the entire roof), (ii) the critical buckling factor for uniform loading over half of the roof, and (iii) the fundamental frequency of the structure. This paper addresses target variable (i). The ANN simulations were based on 1098-point datasets obtained via thorough finite element analyzes.

The proposed ANN for the prediction of the critical buckling factor in steel grid-shells under uniform loading yields mean and maximum errors of 1.1% and 16.3%, respectively, for all 1098 data points. Only in 10.6% of those examples (points), the prediction error exceeds 3%.

**Hosted file**

`Software Bugs Found on March 2021.docx` available at https://authorea.com/users/594089/articles/676148-analytical-prediction-of-steel-grid-shell-stability-and-dynamic-behaviors-using-neural-networks-part-1

Abambres M, Cabello A (2020). Analytical Prediction of Steel Grid-Shell Stability and Dynamic Behaviors Using Neural Networks – Part 1, **HAL**

# Analytical Prediction of Steel Grid-Shell Stability and Dynamic Behaviors Using Neural Networks – Part 1

Miguel Abambres [a], Adrián Cabello [b]

**a** Num3ros, 1600-275 Lisbon, Portugal; amgg@mailfence.com

**b** acg.adrian@gmail.com

**Important Notes:**

1. The first author has been proposing ANN-based models in former publications, in each case designed and tested for a fairly limited amount of data (especially when empirical). Regardless the high quality of the predictions yielded by some model for the used data, the reader should not **blindly** accept that model as accurate for any other instances falling inside the input domain of the design dataset. Any analytical approximation model must undergo extensive validation before it can be taken as reliable (the more inputs, the larger the validation process). Models proposed until that stage are part of a learning process towards excellence.
2. If the reader can't find any of Abambres' papers referred as references of this work, please email the author.

## Abstract

Artificial Intelligence is a cutting-edge technology expanding very quickly into every industry. It has made its way into structural engineering and it has shown its benefits in predicting structural performance as well as saving modelling and experimenting time. This paper is the first one (out of three) of a broader research where artificial intelligence was applied to the stability and dynamic analyzes of steel grid-shells. In that study, three Artificial Neural Networks (ANN) with 8 inputs were independently designed for the prediction of a single target variable, namely: (i) the critical buckling factor for uniform loading (i.e. over the entire roof), (ii) the critical buckling factor for uniform loading over half of the roof, and (iii) the fundamental frequency of the structure. This paper addresses target variable (i). The ANN simulations were based on 1098-point datasets obtained via thorough finite element analyzes.

The proposed ANN for the prediction of the critical buckling factor in steel grid-shells under uniform loading yields mean and maximum errors of 1.1% and 16.3%, respectively, for all 1098 data points. Only in 10.6% of those examples (points), the prediction error exceeds 3%.

**Keywords:** Artificial Neural Networks, Soft Computing, Formula, Structural Engineering, Dataset, Grid-Shell Design, Lightweight Engineering, Buckling, Dynamic Analysis.

# 1. Introduction

It is not easy to trace back the origin of grid-shells but Shukhov´s diagrids are probably the most agreed starting point (Edemskaya 2016). Especially when referring to modern steel grid-shells, his roof for the Vyksa workshop can be regarded the first double-curvature lattice roof. Yet it is not until the end of the 80's, and especially the 90's, that they became really popular through the work of engineers like Schlaich and Schober (Schlaich 1996).

Grid-shells are transparent, thin and typically exhibit high structural efficiency. Their design and fabrication are high precision jobs where tolerances are tight and flexibility low (Schlaich 2009). It is for these reasons that a suitable concept design that understands its failure mode is paramount down the line.

This paper is the first one (out of three) of a broader research where artificial intelligence was applied to the stability and dynamic analyzes of steel grid-shells of paraboloid shape supported on a horizontal plane (see Fig. 1.1). In that study, three Artificial Neural Networks (ANN) with 8 inputs were independently designed for the analytical prediction of a single target variable, namely: (i) the critical (i.e. for the 1$^{st}$ mode) buckling factor for uniform loading (i.e. over the entire roof), (ii) the critical buckling factor for uniform loading over half of the roof, and (iii) the fundamental frequency of the structure. This paper provides a set of equations to obtain the **critical buckling factor** of the structure under uniform loading, where the latter is defined as the **critical load / (external load + selfweight)**. That factor provides a good indication of the stability of a structure, even though a geometrically non-linear analysis is still mandatory for the final design. The 1$^{st}$ buckling mode identified in the analysis can be global or local, whichever is the lowest. The ANN was designed for a 1098-point dataset obtained via finite element analyzes.

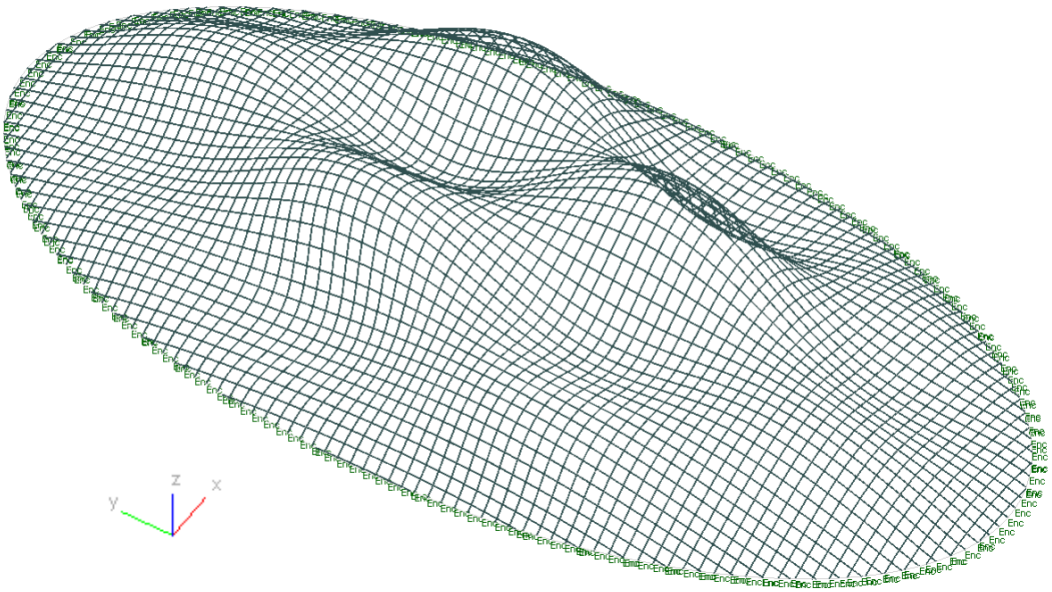

**Fig. 1.1.** 1st buckling mode of a paraboloid shell (model 1030) supported along its perimeter.

The characteristics of the finite element (FE) modelling carried out for data gathering are defined in section 2.1. The FE models meet some predetermined variables defining the scope under which the performed research is valid (section 2.2).

The state of the art regarding the stability of grid-shells was thoroughly expounded by Gioncu (1994). Most of the available analytical solutions predicting buckling of reticulated shells resort to the homogenization technique, treating the structure as a continuum shell (Dulácska and Kollár 2000, Kato 2005, Lefevre 2015). The analytical solution presented in this paper doesn't rely on this simplification. To the knowledge of the authors it hasn't been formulated yet analytical models describing the buckling or dynamic behaviors of the family of grid-shells addressed in the current study.

## 2. Data Gathering

### 2.1 Modelling techniques

A topology has been defined parametrically on Rhinoceros 3D (McNeel 2014) + Grasshopper (Rutten 2014), as illustrated in Fig. 1.2. The base geometry is a paraboloid shell obtained by means of two translational parabolas, the generatrix and

the directrix. The translational technique has the virtue of leading to flat quadrilateral surfaces that can be easily covered with planar glass panes (Pottmann 2014, Schober 2016).

Hundred and sixty (160) different paraboloids have been generated where the main dimensions $L_1$, $L_2$, and $h$, have been varied along with the spacing $s$ between beam nodes (see Fig. 1.2), where the latter remains constant in each model. The domains considered for those variations are addressed in section 2.2.

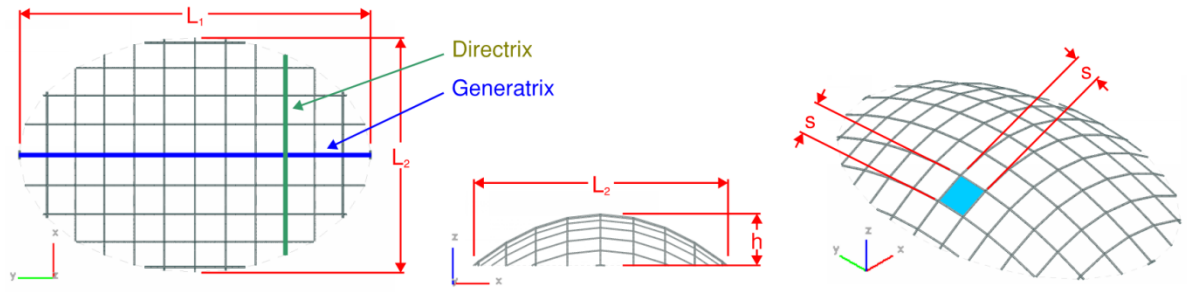


**Fig. 1.2.** Top view (left), side view (center), and perspective view (right) of the parametric topology.

The aforementioned geometries were exported to the FE package GSA (Oasys 2010), where all line segments (beams) were first transformed into beam FEs with 6 DOF (degree of freedom) nodes, and later split into 3 equal elements as result of the sensitivity study explained in section 2.3. At this stage, a Visual Basic (VB) script was run to rotate each rectangular beam (i.e., around its longitudinal axis) by an angle $\alpha$, illustrated in Fig. 1.3 as the angle between the default beam local axis $\overrightarrow{V_0}$ (following global Z) and $\overrightarrow{V_2 x V_1}$ (the cross product between $\overrightarrow{V_2}$ – vector joining transversally adjacent nodes to the beam, and $\overrightarrow{V_1}$ – vector joining beam nodes). This technique doesn´t provide the mathematically exact normal to the paraboloid surface (Makin 2006) but it leads to equal angles $\beta$ between the bar width and the supported glass panes, which is convenient engineering wise.

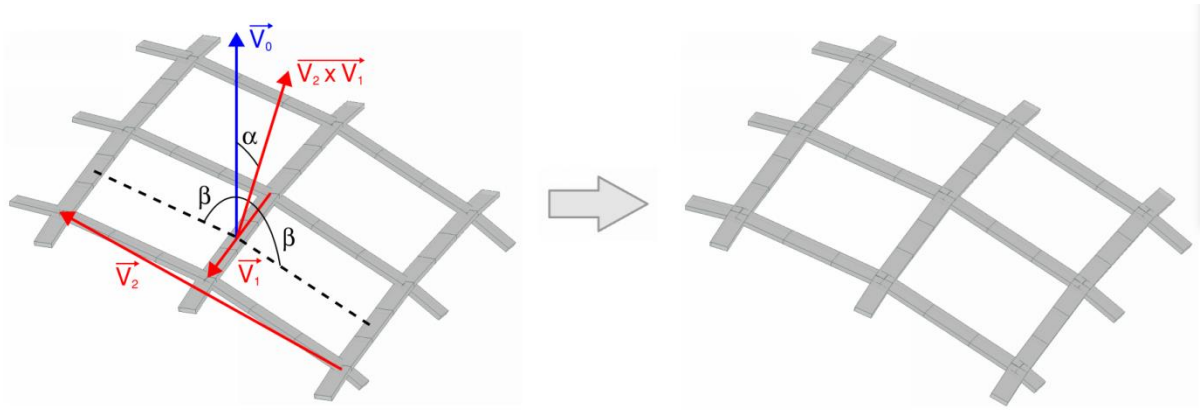


**Fig. 1.3.** Procedure carried out to rotate each beam's coordinate system in all grid-shells.

After orienting the bars, another VB script was run to generate 1098 distinct FE models from the 160 grid-shell geometries, by varying the parameters external load ***q***, bar width ***a***, bar breadth ***b***, and node stiffness $K_n$, as explained in 2.2. A last VB routine run all models and stored the relevant results in a text file.

The set of 1098 FE models used to collect the data that fed the neural network simulations, includes a bulk of reasonably suitable designs but also covers unrealistic under/over-conservative ones, which is precisely what makes the proposed ANN-based tool useful.

The GSA software is valid to analyze grid-shell structures since it has been successfully used on numerous occasions in the past, either as a design or validation tool. Examples range from steel shells (Dini 2013, Olsson 2012), to timber shells (Kuijvenhoven 2009, Toussaint 2007) or composite material shells (du Peloux 2013, Tayeb 2015).

## 2.2 Modelling inputs

The decision about the number of inputs to consider was a trade-off between (i) the versatility of the final ANN tool, and (ii) the time needed to perform all numerical simulations for data gathering – the more input variables, the more data points (no. of

different FE models) are needed to guarantee acceptable accuracy. The following parameters were deemed unchanged in all numerical simulations:

- **Beam Material:** structural steel with linear elastic properties according to EN 1993-1-1 (2005), namely Young's modulus E = 210 GPa, shear modulus G = 81 GPa, and Poisson´s ratio $\nu$ = 0.3. The analyzed models include the self-weight of the steel considering a material density of 7.85 t/m$^3$.
- **Roofing panels:** the weight of the panels is small compared to that of the structural steel (if made of glass or polycarbonate) or even negligible (in case of ETFE). This weight is to be computed as part of the additional load ***q***.
- **Boundary conditions:** the paraboloids lay on a horizontal plane, defining an ellipse (see top view in Fig. 1.2). All nodes belonging to that plane were translationally fixed and rotationally constrained with the same bending stiffness ($K_n$) used for all grid-shell nodes.
- **Bracings:** No (cable-)bracings were applied to any grid-shell. Examples of this type of structure are the Cabot Circus in Bristol or the Joe and Rika Mansueto Library in Chicago, among others.

Tab. 1.1 shows the 8 (independent) input variables considered in all parts of this research, along with the corresponding upper and lower limit values they can take. Each of the 1098 distinct FE models corresponds to a specific combination of values taken by those variables. Tab. 1.1 also indicates the ANN input node corresponding to each variable. Further considerations about those variables read (recall Fig. 1.2):

- **Main dimensions of the paraboloid footprint** (***L₁/2*** and ***L₂/2***): the aspect ratio $1 \leq L_1/L_2 \leq 2$ is always fulfilled.
- **Height of the paraboloid** (***h***): the rise / span ratio is limited to $0.15 \leq h/L_2 \leq 0.5$. This is the range recommended by Schober (2016) for dome caps under uniform loading. For ratios below 0.14, the material usage and the risk of buckling increase considerably.
- **Beam spacing** (***s***): it is the beam spacing in both directions (or the dimension of all grid-shell planar and squared panes – see Fig. 1.2), and the values taken lay

approximately in the range observed in the long list of built projects referenced by Schober (2016).

- **External load** (***q***): It is uniformly distributed over the roof surface, with vertical direction and pointing downwards. It takes random values between 0 kPa and 3.5 kPa.
- **Beam cross-section dimensions** (***a*** and ***b***): the cross-section of all beams is rectangular and solid. Its dimensions come (roughly) from the range of values employed in the shells cited in Schober (2016).
- **Bending Stiffness of Grid-Shell Nodes** ($K_n$): Considering appropriate connection stiffness is quite important, as demonstrated by Hwang (2010) when investigating its effects on grid-shells. From the several bolted systems he studied, rotational stiffness was approximately in the domain of 30 to 130 kNm/rad. Since the present study intends to be also applicable to stiffer connections (bolted or welded), the adopted $K_n$ took values within 20 – 50000 kNm/rad, following the distribution shown in Tab. 1.2. For the sake of computational time, the rotational stiffness was not split into two variables – one for each bending axis, having assumed $K_{nXX} = K_{nYY} = K_n$.

**Tab. 1.1.** Variables and ranges of values considered in the dataset.

| INPUT VARIABLES | | ANN INPUT | VALUES | | |
|---|---|---|---|---|---|
| | | | min | max | average |
| **Grid-Shell Geometry** | $L_1/2$ (m) | 1 | 5 | 25 | 19.11 |
| | $L_2/2$ (m) | 2 | 5 | 50 | 28.86 |
| | $h$ (m) | 3 | 2 | 10 | 6.05 |
| | $s$ (m) | 4 | 0.9 | 1.5 | 1.20 |
| **External Load** | $q$ (kN/m$^2$) | 5 | 0 | 3.5 | 1.71 |
| **Beam Cross-Section** | $b$ (mm) | 6 | 30 | 200 | 114.73 |
| | $a$ (mm) | 7 | 30 | 200 | 114.39 |
| **Bending Stiffness of Grid-Shell Nodes** | $K_n$ (kNm / rad) | 8 | 20 | 49527 | 4989.15 |
| **OUTPUT VARIABLE** | | | | | |
| Critical Buckling Factor for Uniform Loading (-) | | | | | |

**Tab. 2.2.** Node bending stiffness ($K_n$) distribution for all 1098 FE models.

| Probability | | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 | 1/13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Kn* random value | From: | 20 | 50 | 80 | 120 | 200 | 320 | 510 | 810 | 1300 | 2100 | 3600 | 8000 | 20000 |
| | To: | 50 | 80 | 120 | 200 | 320 | 510 | 810 | 1300 | 2100 | 3600 | 8000 | 20000 | 50000 |

The 1098-point dataset considered in ANN simulations is available in Abambres and Cabello (2020).

## 2.3 Sensitivity studies

Two sensitivity studies were carried out prior to the 1098 FE analyzes in order to better decide which mesh density and node stiffness values to adopt in the final FE models. Three models with different geometry were used for that purpose: model 323 ($L_1$ = 50, $L_2$ = 100, $h$ = 10, $s$ = 1.3), model 1030 ($L_1$ = 50, $L_2$ = 95, $h$ = 6, $s$ = 0.9), and model 1071 ($L_1$ = 15, $L_2$ = 15, $h$ = 2, $s$ = 0.9).

The first analysis aimed to understand how sensitive the FE models were to the mesh density. The grid-shell beams (aka bars) between nodes have been subdivided into 1, 2, 3 and 4 beam FE. It was adopted $K_n$ = 990 kNm/rad for all of them. Assuming that 4 elements give the most accurate results, Tab. 1.3 (left) shows the remaining results as a percentage of the former. It turned out that any loss in accuracy when predicting the buckling factor with less than 4 subdivisions is indiscernible since it lies within the noise of the convergence. Thus, there isn't much difference in the prediction of the fundamental frequency either. Since the computational time rises with the number of FEs, the authors have opted for 3 FEs per beam.

The second study (Tab. 1.3, right) allowed to determine which node bending stiffness ($K_n$) yields results that are close enough to those obtained with fixed connections (i.e., infinite stiffness). Three subdivisions were adopted for the beam elements. It was observed that $K_n$ = 50000 kNm/rad yields differences lower than 1% when the results are compared with the fixed connection counterparts, and for that reason that was the adopted upper bound for $K_n$.

**Tab. 3.3.** Sensitivity analysis to determine the suitable (i) number of FEs in each beam (left), and (ii) $K_n$ upper bound (aiming to integrate a fixed connection scenario).

| | | Percentage of the result respect to 4 subdivisions | | | | | Percentage of the result w.r.t. the fixed connection model | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Subdivisions* → | *1* | *2* | *3* | *4* | ← *Kn [kNm/rad]* → | *50* | *200* | *1000* | *10000* | *50000* | *Fixed* |
| | | *990* | *990* | *990* | *990* | | | | | | | |
| Model 323 | 1st Buckling factor (fully loaded) | 100 | 100 | 100 | ***100*** | | 26 | 48 | 79 | 97 | 99.38 | ***100*** |
| | 1st Buckling factor (half loaded) | 100 | 100 | 100 | ***100*** | | 22 | 46 | 77 | 97 | 99.44 | ***100*** |
| | Fundamental frequency | 99.98 | 100 | 100 | ***100*** | | 32 | 52 | 79 | 97 | 99.38 | ***100*** |
| | Computational time (seconds) | 24.5 | 37.2 | 50.3 | 64.0 | | | | | | | |
| Model 1030 | 1st Buckling factor (fully loaded) | 100 | 100 | 100 | ***100*** | | 22 | 43 | 73 | 96 | 99.11 | ***100*** |
| | 1st Buckling factor (half loaded) | 100 | 100 | 100 | ***100*** | | 18 | 40 | 72 | 96 | 99.10 | ***100*** |
| | Fundamental frequency | 99.98 | 99.98 | 99.98 | ***100*** | | 29 | 47 | 74 | 96 | 99.14 | ***100*** |
| | Computational time (seconds) | 2.2 | 2.6 | 3.3 | 4.2 | | | | | | | |
| Model 1071 | 1st Buckling factor (fully loaded) | 100 | 99.98 | 99.98 | ***100*** | | 26 | 49 | 78 | 97 | 99.34 | ***100*** |
| | 1st Buckling factor (half loaded) | 99.98 | 99.98 | 100 | ***100*** | | 21 | 39 | 70 | 96 | 99.17 | ***100*** |
| | Fundamental frequency | 99.79 | 99.96 | 100 | ***100*** | | 43 | 60 | 81 | 97 | 99.34 | ***100*** |
| | Computational time (seconds) | 46.5 | 73.2 | 98.6 | 117.7 | | | | | | | |

# 3. Artificial Neural Networks

## 3.1 Introduction

Machine learning, one of the six disciplines of Artificial Intelligence (AI) without which the task of having machines acting humanly could not be accomplished**,** allows us to 'teach' computers how to perform tasks by providing examples of how they should be done (Hertzmann and Fleet 2012). When there is abundant data (also called examples or patterns) explaining a certain phenomenon, but its theory richness is poor, machine learning can be a perfect tool. The world is quietly being reshaped by machine learning, being the Artificial Neural Network (also referred in this manuscript as ANN or neural net) its (i) oldest (McCulloch and Pitts 1943) and (ii) most powerful (Hern 2016) technique. ANNs also lead the number of practical applications, virtually covering any field of knowledge (Wilamowski and Irwin 2011, Prieto et. al 2016). In its most general form, an ANN is a mathematical model designed to perform a particular task, based in the way the human brain processes information, i.e. with the help of its processing units (the neurons). ANNs have been employed to perform several types of real-world basic tasks. Concerning functional approximation, ANN-based solutions

are frequently more accurate than those provided by traditional approaches, such as multi-variate nonlinear regression, besides not requiring a good knowledge of the function shape being modelled (Flood 2008).

The general ANN structure consists of several nodes disposed in *L* vertical layers (input layer, hidden layers, and output layer) and connected between them, as depicted in Fig. 2. Associated to each node in layers 2 to *L*, also called neuron, is a linear or nonlinear transfer (also called activation) function, which receives the so-called net input and transmits an output (see Fig. 5). All ANNs implemented in this work are called feedforward, since data presented in the input layer flows in the forward direction only, i.e. every node only connects to nodes belonging to layers located at the right-hand-side of its layer, as shown in Fig. 2. ANN's computing power makes them suitable to efficiently solve small to large-scale complex problems, which can be attributed to their (i) massively parallel distributed structure and (ii) ability to learn and generalize, i.e., produce reasonably accurate outputs for inputs not used during the learning (also called training) phase.

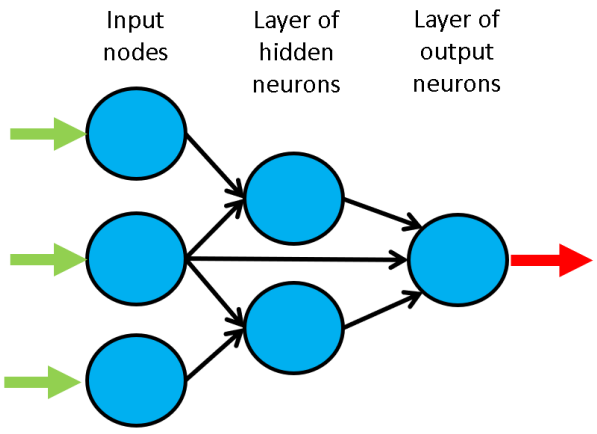


**Fig. 2.** Example of a feedforward neural network.

Further information on Artificial Neural Networks might be found in previous publications by Abambres et al. or Haykin (2009).

## 3.2 Learning

Each connection between 2 nodes is associated to a synaptic weight (real value), which, together with each neuron's bias (also a real value), are the most common types of neural net unknown parameters that will be determined through learning. Learning is nothing else than determining network unknown parameters through some algorithm in order to minimize network's performance measure, typically a function of the difference between predicted and target (desired) outputs. When ANN learning has an iterative nature, it consists of three phases: (i) training, (ii) validation, and (iii) testing. From previous knowledge, examples or data points are selected to train the neural net, grouped in the so-called training dataset. Those examples are said to be 'labelled' or 'unlabeled', whether they consist of inputs paired with their targets, or just of the inputs themselves – learning is called supervised (e.g., functional approximation, classification) or unsupervised (e.g., clustering), whether data used is labelled or unlabeled, respectively. During an iterative learning, while the training dataset is used to tune network unknowns, a process of cross-validation takes place by using a set of data completely distinct from the training counterpart (the validation dataset), so that the generalization performance of the network can be attested. Once 'optimum' network

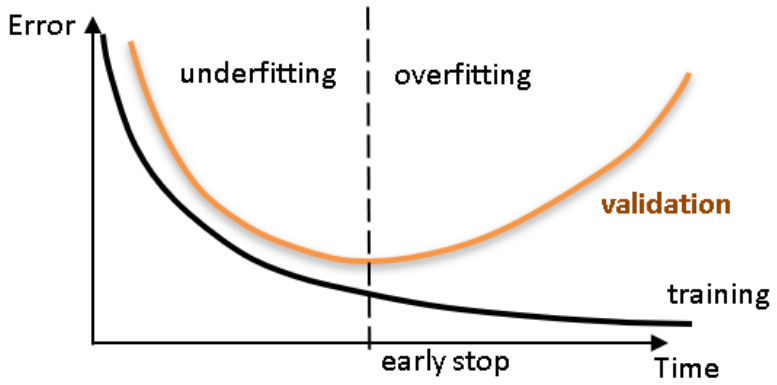


**Fig. 3.** Cross-validation - assessing network's generalization ability.

parameters are determined, typically associated to a minimum of the validation performance curve (called early stop – see Fig. 3), many authors still perform a final

assessment of model's accuracy, by presenting to it a third fully distinct dataset called 'testing'. Heuristics suggests that early stopping avoids overfitting, i.e. the loss of ANN's generalization ability. One of the causes of overfitting might be learning too many input-target examples suffering from data noise, since the network might learn some of its features, which do not belong to the underlying function being modelled (Haykin 2009).

## 3.3 Implemented ANN features

The 'behavior' of any ANN depends on many 'features', having been implemented 15 ANN features in this work (including data pre/post processing ones). For those features, it is important to bear in mind that no ANN guarantees good approximations via extrapolation (either in functional approximation or classification problems), i.e. the implemented ANNs should not be applied outside the input variable ranges used for network training. Since there are no objective rules dictating which method per feature guarantees the best network performance for a specific problem, an extensive parametric analysis (composed of nine parametric sub-analyzes) was carried out to find 'the optimum' net design. A description of all implemented methods, selected from state of art literature on ANNs (including both traditional and promising modern techniques), is presented next – Tabs. 2-4 present all features and methods per feature. The whole work was coded in MATLAB (The Mathworks, Inc. 2017), making use of its neural network toolbox when dealing with popular learning algorithms (1-3 in Tab. 4). Each parametric sub-analysis (SA) consists of running all feasible combinations (also called 'combos') of pre-selected methods for each ANN feature, in order to get performance results for each designed net, thus allowing the selection of the best ANN according to a certain criterion. The best network in each parametric SA is the one exhibiting the smallest average relative error (called performance) for all learning data.

It is worth highlighting that, in this manuscript, whenever a vector is added to a matrix, it means the former is to be added to all columns of the latter (valid in MATLAB).

**Tab. 2.** Implemented ANN features (F) 1-5.

| FEATURE METHOD | F1 | F2 | F3 | F4 | F5 |
|---|---|---|---|---|---|
| | Qualitative Var Represent | Dimensional Analysis | Input Dimensionality Reduction | % Train-Valid-Test | Input Normalization |
| **1** | Boolean Vectors | Yes | Linear Correlation | 80-10-10 | Linear Max Abs |
| **2** | Eq Spaced in ]0,1] | No | Auto-Encoder | 70-15-15 | Linear [0, 1] |
| **3** | - | - | - | 60-20-20 | Linear [-1, 1] |
| **4** | - | - | Ortho Rand Proj | 50-25-25 | Nonlinear |
| **5** | - | - | Sparse Rand Proj | - | Lin Mean Std |
| **6** | - | - | No | - | No |

### 3.3.1 Qualitative Variable Representation (feature 1)

A qualitative variable taking *n* distinct 'values' (usually called classes) can be represented in any of the following formats: one variable taking *n* equally spaced values in ]0,1], or 1-of-*n* encoding (boolean vectors – e.g., *n*=3: [1 0 0] represents class 1, [0 1 0] represents class 2, and [0 0 1] represents class 3). After transformation, qualitative variables are placed at the end of the corresponding (input or output) dataset, in the same original order.

**Tab. 3.** Implemented ANN features (F) 6-10.

| FEATURE METHOD | F6 | F7 | F8 | F9 | F10 |
|---|---|---|---|---|---|
| | Output Transfer | Output Normalization | Net Architecture | Hidden Layers | Connectivity |
| **1** | Logistic | Lin [a, b] = 0.7[$\varphi_{min}$, $\varphi_{max}$] | MLPN | 1 HL | Adjacent Layers |
| **2** | - | Lin [a, b] = 0.6[$\varphi_{min}$, $\varphi_{max}$] | RBFN | 2 HL | Adj Layers + In-Out |
| **3** | Hyperbolic Tang | Lin [a, b] = 0.5[$\varphi_{min}$, $\varphi_{max}$] | - | 3 HL | Fully-Connected |
| **4** | - | Linear Mean Std | - | - | - |
| **5** | Bilinear | No | - | - | - |
| **6** | Compet | - | - | - | - |
| **7** | Identity | - | - | - | - |

**Tab. 4.** Implemented ANN features (F) 11-15.

| FEATURE METHOD | F11 | F12 | F13 | F14 | F15 |
|---|---|---|---|---|---|
| | Hidden Transfer | Parameter Initialization | Learning Algorithm | Performance Improvement | Training Mode |
| **1** | Logistic | Midpoint (W) + Rands (b) | BP | - | Batch |
| **2** | Identity-Logistic | Rands | BPA | - | Mini-Batch |
| **3** | Hyperbolic Tang | Randnc (W) + Rands (b) | LM | - | Online |
| **4** | Bipolar | Randnr (W) + Rands (b) | ELM | - | - |
| **5** | Bilinear | Randsmall | mb ELM | - | - |
| **6** | Positive Sat Linear | Rand [-Δ, Δ] | I-ELM | - | - |
| **7** | Sinusoid | SVD | CI-ELM | - | - |
| **8** | Thin-Plate Spline | MB SVD | - | - | - |
| **9** | Gaussian | - | - | - | - |
| **10** | Multiquadratic | - | - | - | - |
| **11** | Radbas | - | - | - | - |

### 3.3.2 Dimensional Analysis (feature 2)

The most widely used form of dimensional analysis is the Buckingham's π-theorem, which was implemented in this work as described in Bhaskar and Nigam (1990).

### 3.3.3 Input Dimensionality Reduction (feature 3)

When designing any ANN, it is crucial for its accuracy that the input variables are independent and relevant to the problem (Gholizadeh et al. 2011, Kasun et al. 2016). There are two types of dimensionality reduction, namely (i) feature selection (a subset of the original set of input variables is used), and (ii) feature extraction (transformation of initial variables into a smaller set). In this work, dimensionality reduction is never performed when the number of input variables is less than six. The implemented methods are described next.

***Linear Correlation***

In this feature selection method, all possible pairs of input variables are assessed with respect to their linear dependence, by means of the Pearson correlation

coefficient $R_{XY}$, where $X$ and $Y$ denote any two distinct input variables. For a set of $n$ data points ($x_i$, $y_i$), the Pearson correlation is defined by

$$R_{XY} = \frac{\sum_{i=1}^{n}\left(x_i-\bar{x}\right)\left(y_i-\bar{y}\right)}{\sqrt{\sum_{i=1}^{n}\left(x_i-\bar{x}\right)^2\sum_{i=1}^{n}\left(y_i-\bar{y}\right)^2}} = \frac{Cov(X,Y)}{\sqrt{Var(X)\,Var(Y)}} \quad , \quad (1)$$

where (i) $Var(X)$ and $Cov(X, Y)$ are the variance of $X$ and covariance of $X$ and $Y$, respectively, and (ii) $\bar{x}$ and $\bar{y}$ are the mean values of each variable. In this work, cases where $|R_{XY}| \geq 0.99$ indicate that one of the variables in the pair must be removed from the ANN modelling. The one to be removed is the one appearing less in the remaining pairs $(X, Y)$ where $|R_{XY}| \geq 0.99$. Once a variable is selected for removal, all pairs $(X, Y)$ involving it must be disregarded in the subsequent steps for variable removal.

***Auto-Encoder***

This feature extraction technique uses itself a 3-layer feedforward ANN called auto-encoder (AE). After training, the hidden layer output ($y_{2p}$) for the presentation of each problem's input pattern ($y_{1p}$) is a compressed vector ($Q_2$ x 1) that can be used to replace the original input layer by a (much) smaller one, thus reducing the size of the ANN model. In this work, $Q_2$=*round*($Q_1$/2) was adopted, being *round* a function that rounds the argument to the nearest integer. The implemented AE was trained using the 'trainAutoencoder(…)' function from MATLAB's neural net toolbox. In order to select the best AE, 40 AEs were simulated, and their performance compared by means of the performance variable defined in sub-section 3.4. Each AE considered distinct (random) initialization parameters, half of the models used the 'logsig' hidden transfer functions, and the other half used the 'satlin' counterpart, being the identity function the common option for the output activation. In each AE, the maximum number of epochs – number of times the whole training dataset is presented to the network during learning, was defined (regardless the amount of data) by

$$\max epochs = \begin{cases} 3000, Q_1 > 8 \\ 1500, Q_1 \leq 8 \end{cases} \quad . \quad (2)$$

Concerning the learning algorithm used for all AEs, no $L_2$ weight regularization was employed, which was the only default specification not adopted in 'trainAutoencoder(…)'.

***Orthogonal and Sparse Random Projections***

This is another feature extraction technique aiming to reduce the dimension of input data $Y_1$ ($Q_1$ x $P$) while retaining the Euclidean distance between data points in the new feature space. This is attained by projecting all data along the (i) orthogonal or (ii) sparse random matrix $A$ ($Q_1$ x $Q_2$, $Q_2 < Q_1$), as described by Kasun et al. (2016).

### 3.3.4 Training, Validation and Testing Datasets (feature 4)

Four distributions of data (methods) were implemented, namely $p_t$-$p_v$-$p_{tt}$ = {80-10-10, 70-15-15, 60-20-20, 50-25-25}, where $p_t$-$p_v$-$p_{tt}$ represent the amount of training, validation and testing examples as % of all learning data ($P$), respectively. Aiming to divide learning data into training, validation and testing subsets according to a predefined distribution $p_t$-$p_v$-$p_{tt}$, the following algorithm was implemented (all variables are involved in these steps, including qualitative ones after converted to numeric – see 3.3.1):

1) Reduce $p_t$-$p_v$-$p_{tt}$ values by 10 units each.
2) For each variable $q$ (row) in the complete input dataset, compute its minimum and maximum values.
3) Select all patterns (if some) from the learning dataset where each variable takes either its minimum or maximum value. Those patterns must be included in the training dataset, regardless what $p_t$ is. However, if the number of patterns is lower than the rounding of $p_t$ * $P$/100, more patterns should be added to the training set in the following way:
    a. Compute the number of patterns (L$p_t$) that need to be added to the initially selected training patterns to equal round($p_t$ * $P$/100).
    b. Randomly select 10.000 combinations of L$p_t$ patterns from all those not included in the training set defined prior a).

c. For each combination/scenario in b), add those L$p_t$ patterns to the set of training patterns defined prior a), and label all remaining learning patterns as “validation+testing”.
d. For each scenario in c), and for each pattern labeled as “validation+testing”, check if that pattern has at least one input variable that takes a value not taken by any pattern in the training set. If it hasn´t, then that pattern should be moved to the training set.
e. Among all 10.000 scenarios of training and “validation+testing” subsets addressed in b) till d), the “winner” should be the one guaranteeing the amount of training data ($P_{t*}$) closest to round($p_t$ * $P$/100).
f. If the winning training set selected in e) guarantees | $P_{t*}$ / $P$ - $p_t$ | ≤ 0.2, then that becomes the training data to be taken for simulation. Otherwise, the training data should be selected according to step 2 in subsection 3.3.4 of Abambres et al. (2018).

4) Increase $p_t$-$p_v$-$p_{tt}$ values by 10 units each (to re-obtain the original input values – recall step 1).
5) In order to select the validation patterns, randomly select $p_v$ / ($p_v$ + $p_{tt}$) of those patterns not belonging to the previously defined training dataset. The remainder defines the testing dataset.

It might happen that the actual distribution $p_t$-$p_v$-$p_{tt}$ to be used in the simulation is not equal to the one imposed *a priori* (before step 1).

### 3.3.5 Input Normalization (feature 5)

The progress of training can be impaired if training data defines a region that is relatively narrow in some dimensions and elongated in others, which can be alleviated by normalizing each input variable across all data patterns. The implemented techniques are the following:

***Linear Max Abs***

Lachtermacher and Fuller (1995) proposed a simple normalization technique given by

$$\{Y_1\}_n(i,:)= \frac{Y_1(i,:)}{\max\left\{\left|Y_1(i,:)\right|\right\}} \quad , \quad (3)$$

where $\{Y_1\}_n(i,:)$ and $Y_1(i,:)$ are the normalized and non-normalized values of the $i^{th}$ input variable for all learning patterns, respectively. Notation ':' in the column index, indicate the selection of all columns (learning patterns).

***Linear [0, 1] and [-1, 1]***

A linear transformation for each input variable ($i$), mapping values in $Y_1$(i,:) from [$a$*, $b$*]=[min($Y_1$(i,:)), max($Y_1$(i,:))] to a generic range [$a$, $b$], is obtained from

$$\{Y_1\}_n(i,:)= a+\frac{\left(Y_1(i,:)-a^*\right)}{\left(b^*-a^*\right)}(b-a) \quad . \quad (4)$$

Ranges [$a$, $b$]=[0, 1] and [$a$, $b$]=[-1, 1] were considered.

***Nonlinear***

Proposed by Pu and Mesbahi (2006), although in the context of output normalization, the only nonlinear normalization method implemented for input data reads

$$\{Y_1\}_n(i,j)= sign\left(Y_1(i,j)\right)\sqrt{\frac{\left|Y_1(i,j)\right|}{10^t}}+C(i) \quad , \quad (5)$$

where (i) $Y_1(i, j)$ is the non-normalized value of input variable $i$ for pattern $j$, (ii) $t$ is the number of digits in the integer part of $Y_1(i, j)$, (iii) *sign*(…) yields the sign of the argument, and (iv) $C$(i) is the average of two values concerning variable $i$, $C_1$(i) and $C_2$(i), where the former leads to a minimum normalized value of 0.2 for all patterns, and the latter leads to a maximum normalized value of 0.8 for all patterns.

***Linear Mean Std***

Tohidi and Sharifi (2014) proposed the following technique

$$\{Y_1\}_n(i,:) = \frac{Y_1(i,:) - \mu_{Y_1(i,:)}}{\sigma_{Y_1(i,:)}} \quad , \quad (6)$$

where $\mu_{Y_1(i,:)}$ and $\sigma_{Y_1(i,:)}$ are the mean and standard deviation of all non-normalized values (all patterns) stored by variable *i*.

### 3.3.6 Output Transfer Functions (feature 6)

***Logistic***

The most usual form of transfer functions is called Sigmoid. An example is the logistic function given by

$$\varphi(s) = \frac{1}{1+e^{-s}} \quad . \quad (7)$$

***Hyperbolic Tang***

The Hyperbolic Tangent function is also of sigmoid type, being defined as

$$\varphi(s) = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}} \quad . \quad (8)$$

***Bilinear***

The implemented Bilinear function is defined as

$$\varphi(s) = \begin{cases} s, & s \geq 0 \\ 0, & s < 0 \end{cases} \quad . \quad (9)$$

***Identity***

The Identity activation is often employed in output neurons, reading

$$\varphi(s) = s \quad . \quad (10)$$

### 3.3.7 Output Normalization (feature 7)

Normalization can also be applied to the output variables so that, for instance, the amplitude of the solution surface at each variable is the same. Otherwise, training may tend to focus (at least in the earlier stages) on the solution surface with the greatest amplitude (Flood and Kartam 1994a). Normalization ranges not including the zero value might be a useful alternative since convergence issues may arise due to the presence of many small (close to zero) target values (Mukherjee et al. 1996). Four normalization methods were implemented. The first three follow eq. (4), where (i) [a, b] = 70% [$\varphi_{min}$, $\varphi_{max}$], (ii) [a, b] = 60% [$\varphi_{min}$, $\varphi_{max}$], and (iii) [a, b] = 50% [$\varphi_{min}$, $\varphi_{max}$], being [$\varphi_{min}$, $\varphi_{max}$] the output transfer function range, and [a, b] determined to be centered within [$\varphi_{min}$, $\varphi_{max}$] and to span the specified % (e.g., (b-a) = 0.7 ($\varphi_{max}$ - $\varphi_{min}$)). Whenever the output transfer functions are unbounded (Bilinear and Identity), it was considered [a, b] = [0, 1] and [a, b] = [-1, 1], respectively. The fourth normalization method implemented is the one described by eq. (6).

### 3.3.8 Network Architecture (feature 8)

***Multi-Layer Perceptron Network (MLPN)***

This is a feedforward ANN exhibiting at least one hidden layer. Fig. 2 depicts a 3-2-1 MLPN (3 input nodes, 2 hidden neurons and 1 output neuron), where units in each layer link to nodes located ahead only. The network is said to be partially-connected (PC) since no connections across layers are allowed (between the source and output layers, in this case). At this moment, it is appropriate to define the concept of fully-connected (FC) ANN. Although traditionally, the network shown in Fig. 2 would be called FC, in this work a FC feedforward network is characterized by having each node connected to every node in a different layer placed forward – any other type of feedforward network is said to be PC. According to Wilamowski (2009), PC MLPNs are less powerful than MLPN where connections across layers are allowed, which usually lead to smaller networks (less neurons).

Fig. 4 represents a generic MLFN composed of $L$ layers, where $l$ ($l$ = 1,…, $L$) is a generic layer and '*ql*' a generic node, being $q = 1,\ldots, Q_l$ its position in layer $l$ (1 is reserved to the top node). Fig. 5 represents the model of a generic neuron ($l$ = 2,…, $L$), where (i) $p$ represents the data pattern presented to the network, (ii) subscripts $m$ = 1,…, $Q_n$ and $n$ = 1,…, $l$-1 are summation indexes representing all possible nodes connecting to neuron '*ql*' (recall Fig. 4), (iii) $b_{ql}$ is neuron's bias, and (iv) $w_{mnql}$ represents the synaptic weight connecting units '*mn*' and '*ql*'. Neuron's net input for the presentation of pattern $p$ ($S_{qlp}$) is defined as

$$S_{qlp} = y_{mnp}\, w_{mnql} + b_{ql}, \qquad y_{mnp}\, w_{mnql} \equiv \sum_{m=1}^{Q_n} \sum_{n=1}^{l-1} y_{mnp}\, w_{mnql} \qquad , \quad (11)$$

where $y_{m1p}$ is the value of the $m^{th}$ network input concerning example $p$. The output of a generic neuron can then be written as ($l$ = 2,…, $L$)

$$y_{qlp} = \varphi_l(S_{qlp}) \qquad , \quad (12)$$

where $\varphi_l$ is the transfer function used for all neurons in layer $l$.

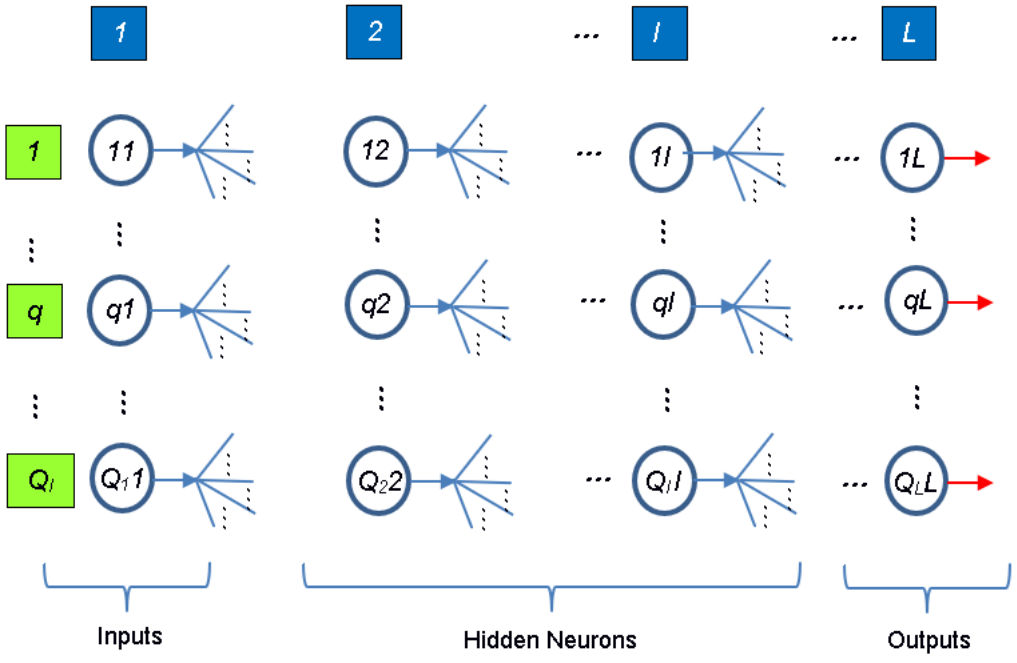


**Fig. 4.** Generic multi-layer feedforward network.

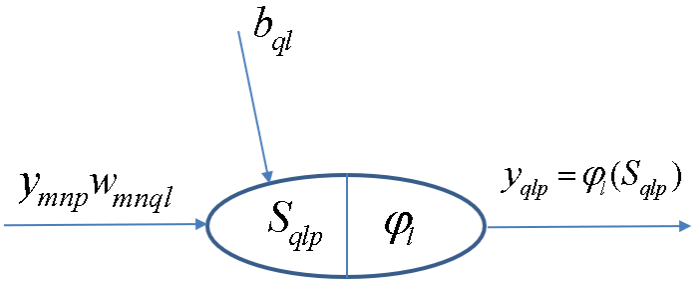


**Fig. 5.** Generic neuron placed anywhere in the MLPN of Fig. 4 ($l$ = 2,…, $L$).

***Radial-Basis Function Network (RBFN)***

Although having similar topologies, RBFN and MLPN behave very differently due to distinct hidden neuron models – unlike the MLPN, RBFN have hidden neurons behaving differently than output neurons. According to Xie et al. (2011), RBFN (i) are specially recommended in functional approximation problems when the function surface exhibits regular peaks and valleys, and (ii) perform more robustly than MLPN when dealing with noisy input data. Although traditional RBFN have 3 layers, a generic multi-hidden layer (see Fig. 4) RBFN is allowed in this work, being the generic hidden neuron's model concerning node '$l_1l_2$' ($l_1$ = 1,…,$Q_{l2}$, $l_2$ = 2,…, $L$-1) presented in Fig. 6. In this model, (i) $v_{l_1l_2p}$ and $\xi_{l_1l_2}$ (called RBF center) are vectors of the same size ($\xi_{zl_1l_2}$ denotes de z component of vector $\xi_{l_1l_2}$, and it is a network unknown), being the former associated to the presentation of data pattern $p$, (ii) $\sigma_{l_1l_2}$ is called RBF width (a positive scalar) and also belongs, along with synaptic weights and RBF centers, to the set of network unknowns to be determined through learning, (iii) $\varphi_{l_2}$ is the user-defined radial basis (transfer) function (RBF), described in eqs. (20)-(23), and (iv) $y_{l_1l_2p}$ is neuron's output when pattern $p$ is presented to the network. In ANNs not involving learning algorithms 1-3 in Tab. 4, vectors $v_{l_1l_2p}$ and $\xi_{l_1l_2}$ are defined as (two versions of $v_{l_1l_2p}$ where implemented and the one yielding the best results was selected)

$$v_{l_1 l_2 p} = \left[ y_{1(l_2-1)p}\, w_{1(l_2-1)l_1 l_2} \quad \cdots \quad y_{z(l_2-1)p}\, w_{z(l_2-1)l_1 l_2} \quad \cdots \quad y_{Q_{l_2-1}(l_2-1)p}\, w_{Q_{l_2-1}(l_2-1)l_1 l_2} \right]$$

or

$$v_{l_1 l_2 p} = \left[ y_{1(l_2-1)p} \quad \cdots \quad y_{z(l_2-1)p} \quad \cdots \quad y_{Q_{l_2-1}(l_2-1)p} \right] \quad , (13)$$

and

$$\xi_{l_1 l_2} = \left[ \xi_{1 l_1 l_2} \quad \cdots \quad \xi_{z l_1 l_2} \quad \cdots \quad \xi_{Q_{l_2-1} l_1 l_2} \right]$$

whereas the RBFNs implemented through MATLAB neural net toolbox (involving learning algorithms 1-3 in Tab. 4) are based on the following definitions

$$\begin{aligned} v_{l_1 l_2 p} &= \left[ y_{1(l_2-1)p} \quad \cdots \quad y_{z(l_2-1)p} \quad \cdots \quad y_{Q_{l_2-1}(l_2-1)p} \right] \\ \xi_{l_1 l_2} &= \left[ w_{1(l_2-1)l_1 l_2} \quad \cdots \quad w_{z(l_2-1)l_1 l_2} \quad \cdots \quad w_{Q_{l_2-1}(l_2-1)l_1 l_2} \right] \end{aligned} \quad . \quad (14)$$

Lastly, according to the implementation carried out for initialization purposes (described in 3.3.12), (i) RBF center vectors per hidden layer (one per hidden neuron) are initialized as integrated in a matrix (termed RBF center matrix) having the same size of a weight matrix linking the previous layer to that specific hidden layer, and (ii) RBF widths (one per hidden neuron) are initialized as integrated in a vector (called RBF width vector) with the same size of a hypothetic bias vector.

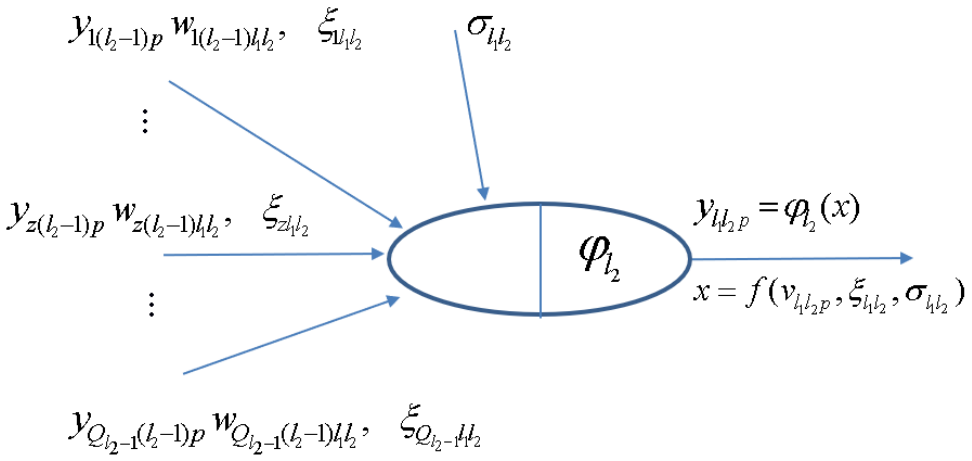


**Fig. 6.** Generic hidden neuron $l_1 l_2$ placed anywhere in the RBFN of Fig. 4 ($l_2$ = 2,…, $L$-1).

### 3.3.9 Hidden Nodes (feature 9)

Inspired by several heuristics found in the literature for the determination of a suitable number of hidden neurons in a single hidden layer net (Aymerich and Serra 1998, Rafiq et al. 2001, Xu and Chen 2008), each value in *hntest*, defined in eq. (15), was tested in this work as the total number of hidden nodes in the model, ie the sum of nodes in all hidden layers (initially defined with the same number of neurons). The number yielding the smallest performance measure for all patterns (as defined in 3.4, with outputs and targets not postprocessed), is adopted as the best solution. The aforementioned *hntest* is defined by

$$
\begin{aligned}
&incr = [4,\ 4,\ 4,\ 10,\ 10,\ 10,\ 10] \\
&minimum = [1,\ 1,\ 1,\ 10,\ 10,\ 10,\ 10] \\
&max_1 = \min\left(\text{round}\left(\max\left(2Q_1+Q_L,\ 4Q_1,\ \sqrt{\frac{P}{Q_1 \ln(P)}}\right)\right),\ 1500\right) \\
&max_2 = \max\left(\min\left(\text{round}\left(0.1P\right),1500\right),\ 300\right) \\
&maximum = [max_1,\ max_1,\ max_1,\ max_2,\ max_2,\ max_2,\ max_2] \\
&hntest = minimum(\mathrm{F}_{13}) : incr(\mathrm{F}_{13}) : maximum(\mathrm{F}_{13})
\end{aligned}
\quad , \quad (15)
$$

where (i) $Q_1$ and $Q_L$ are the number of input and output nodes, respectively, (ii) $P$ and $P_t$ are the number of learning and training patterns, respectively, and (iii) $F_{13}$ is the number of feature 13's method (see Tab. 4).

### 3.3.10 Connectivity (feature 10)

For this ANN feature, three methods were implemented, namely (i) adjacent layers – only connections between adjacent layers are made possible, (ii) adjacent layers +

input-output – only connections between ($ii_1$) adjacent and ($ii_2$) input and output layers are allowed, and (iii) fully-connected (all possible feedforward connections).

### 3.3.11 Hidden Transfer Functions (feature 11)

Besides functions (i) Logistic – eq. (7), (ii) Hyperbolic Tangent – eq. (8), and (iii) Bilinear – eq. (9), defined in 3.3.6, the ones defined next were also implemented as hidden transfer functions. During software validation it was observed that some hidden node outputs could be infinite or NaN (not-a-number in MATLAB – e.g., 0/0=Inf/Inf=NaN), due to numerical issues concerning some hidden transfer functions and/or their calculated input. In those cases, it was decided to convert infinite to unitary values and NaNs to zero (the only exception was the bipolar sigmoid function, where NaNs were converted to -1). Other implemented trick was to convert possible Gaussian function's NaN inputs to zero.

***Identity-Logistic***

In Gunaratnam and Gero (1994), issues associated with flat spots at the extremes of a sigmoid function were eliminated by adding a linear function to the latter, reading

$$\varphi(s) = \frac{1}{1+e^{-s}} + s \quad . \quad (16)$$

***Bipolar***

The so-called bipolar sigmoid activation function mentioned in Lefik and Schrefler (2003), ranging in [-1, 1], reads

$$\varphi(s) = \frac{1-e^{-s}}{1+e^{-s}} \quad . \quad (17)$$

***Positive Saturating Linear***

In MATLAB neural net toolbox, the so-called Positive Saturating Linear transfer function, ranging in [0, 1], is defined as

$$\varphi(s)=\begin{cases}1, & s\geq 1\\ s, & 0<s<1\\ 0, & s\leq 0\end{cases} \quad . \quad (18)$$

***Sinusoid***

Concerning less popular transfer functions, reference is made in Bai et al. (2014) to the sinusoid, which in this work was implemented as

$$\varphi(s)=\sin\left(\frac{\pi}{2}s\right) \quad . \quad (19)$$

***Radial Basis Functions (RBF)***

Although Gaussian activation often exhibits desirable properties as a RBF, several authors (e.g., Schwenker et al. 2001) have suggested several alternatives. Following nomenclature used in 3.3.8, (i) the Thin-Plate Spline function is defined by

$$\varphi_{l_2}\left(s\right)=s\ln\left(\sqrt{s}\right), \quad s=\left\|v_{l_1l_2p}-\xi_{l_1l_2}\right\|^2 \quad , \quad (20)$$

(ii) the next function is employed as Gaussian-type function when learning algorithms 4-7 are used (see Tab. 4)

$$\varphi_{l_2}\left(s\right)=e^{-0.5s}, \quad s=\left\|v_{l_1l_2p}-\xi_{l_1l_2}\right\|^2\Big/\sigma_{l_1l_2}^2 \quad , \quad (21)$$

(iii) the Multiquadratic function is given by

$$\varphi_{l_2}\left(s\right)=\sqrt{s}, \quad s=\left\|v_{l_1l_2p}-\xi_{l_1l_2}\right\|^2+\sigma_{l_1l_2}^2 \quad , \quad (22)$$

and (iv) the Gaussian-type function (called ‘radbas’ in MATLAB toolbox) used by RBFNs trained with learning algorithms 1-3 (see Tab. 4), is defined by

$$\varphi_{l_2}\left(s\right)=e^{-s^2}, \quad s=\left\|v_{l_1l_2p}-\xi_{l_1l_2}\right\|\sigma_{l_1l_2} \quad , \quad (23)$$

where || … || denotes the Euclidean distance in all functions.

### 3.3.12 Parameter Initialization (feature 12)

The initialization of (i) weight matrices ($Q_a$ x $Q_b$, being $Q_a$ and $Q_b$ node numbers in layers *a* and *b* being connected, respectively), (ii) bias vectors ($Q_b$ x 1), (iii) RBF center matrices ($Q_{c-1}$ x $Q_c$, being *c* the hidden layer that matrix refers to), and (iv) RBF width vectors ($Q_c$ x 1), are independent and in most cases randomly generated. For each ANN design carried out in the context of each parametric analysis combo, and whenever the parameter initialization method is not the 'Mini-Batch SVD', ten distinct simulations varying (due to their random nature) initialization values are carried out, in order to find the best solution. The implemented initialization methods are described next.

***Midpoint, Rands, Randnc, Randnr, Randsmall***

These are all MATLAB built-in functions. *Midpoint* is used to initialize weight and RBF center matrices only (not vectors). All columns of the initialized matrix are equal, being each entry equal to the midpoint of the (training) output range leaving the corresponding initial layer node – recall that in weight matrices, columns represent each node in the final layer being connected, whereas rows represent each node in the initial layer counterpart. *Rands* generates random numbers with uniform distribution in [-1, 1]. *Randnc* (only used to initialize matrices) generates random numbers with uniform distribution in [-1, 1], and normalizes each array column to 1 (unitary Euclidean norm). *Randnr* (only used to initialize matrices) generates random numbers with uniform distribution in [-1, 1], and normalizes each array row to 1 (unitary Euclidean norm). *Randsmall* generates random numbers with uniform distribution in [-0.1, 0.1].

***Rand [-lim, lim]***

This function is based on the proposal in Waszczyszyn (1999), and generates random numbers with uniform distribution in [-*lim*, *lim*], being *lim* layer-dependent and defined by

$$lim = \begin{cases} Q_b^{\,1/Q_a}, b < L \\ 0.5 \quad , b = L \end{cases} , \quad (24)$$

where *a* and *b* refer to the initial and final layers integrating the matrix being initialized, and *L* is the total number of layers in the network. In the case of a bias or RBF width vector, *lim* is always taken as 0.5.

***SVD***

Although Deng et al. (2016) proposed this method for a 3-layer network, it was implemented in this work regardless the number of hidden layers.

***Mini-Batch SVD***

Based on Deng et al. (2016), this scheme is an alternative version of the former SVD. Now, training data is split into min{$Q_b$, $P_t$} chunks (or subsets) of equal size $P_{ti}$ = max{*floor*($P_t$ / $Q_b$), 1} – *floor* rounds the argument to the previous integer (whenever it is decimal) or yields the argument itself, being each chunk aimed to derive $Q_{bi}$ = 1 hidden node.

### 3.3.13 Learning Algorithm (feature 13)

The most popular learning algorithm is called error back-propagation (BP), a first-order gradient method. Second-order gradient methods are known to have higher training speed and accuracy (Wilamowski 2011). The most employed is called Levenberg-Marquardt (LM). All these traditional schemes were implemented using MATLAB toolbox (The Mathworks, Inc 2017).

***Back-Propagation (BP, BPA), Levenberg-Marquardt (LM)***

Two types of BP schemes were implemented, one with constant learning rate (BP) –'traingd' in MATLAB, and another with iteration-dependent rate, named BP with adaptive learning rate (BPA) – 'traingda' in MATLAB. The learning parameters set different than their default values are:

(i) Learning Rate = 0.01 / $cs^{0.5}$, being cs the chunk size, as defined in 3.3.15.

(ii) Minimum performance gradient = 0.

Concerning the LM scheme – 'trainlm' in MATLAB, the only learning parameter set different than its default value was the abovementioned (ii).

***Extreme Learning Machine (ELM, mb ELM, I-ELM, CI-ELM)***

Besides these traditional learning schemes, iterative and time-consuming by nature, four versions of a recent, powerful and non-iterative learning algorithm, called Extreme Learning Machine (ELM), were implemented (unlike initially proposed by the authors of ELM, connections across layers were allowed in this work), namely: (batch) ELM (Huang et al. 2006a), Mini-Batch ELM (mb ELM) (Liang et al. 2006), Incremental ELM (I-ELM) (Huang et al. 2006b), Convex Incremental ELM (CI-ELM) (Huang and Chen 2007).

### 3.3.14 Performance Improvement (feature 14)

None implemented.

### 3.3.15 Training Mode (feature 15)

Depending on the relative amount of training patterns, with respect to the whole training dataset, that is presented to the network in each iteration of the learning process, several types of training modes can be used, namely (i) batch or (ii) mini-batch. Whereas in the batch mode all training patterns are presented (called an epoch) to the network in each iteration, in the mini-batch counterpart the training dataset is split into several data chunks (or subsets) and in each iteration a single and new chunk is presented to the network, until (eventually) all chunks have been presented. Learning involving iterative schemes (e.g., BP- or LM-based) might require many epochs until an 'optimum' design is found. The particular case of having a mini-batch mode where all chunks are composed by a single (distinct) training pattern (number of data chunks = $P_t$ , chunk size = 1), is called online or sequential mode. Wilson and Martinez (2003) suggested that if one wants to use mini-batch training with the same stability as online training, a rough estimate of the suitable learning rate to be used in learning algorithms such as the BP, is $\eta_{\text{online}}/\sqrt{cs}$, where $cs$ is the chunk size and $\eta_{\text{online}}$ is

the online learning rate – their proposal was adopted in this work. Based on the proposal of Liang et al. (2006), the constant chunk size (*cs*) adopted for all chunks in mini-batch mode reads $cs = \min\{mean(hn) + 50, P_t\}$, being *hn* a vector storing the number of hidden nodes in each hidden layer in the beginning of training, and *mean*(*hn*) the average of all values in *hn*.

## 3.4 Network Performance Assessment

Several types of results were computed to assess network outputs, namely (i) maximum error, (ii) % errors greater than 3%, and (iii) performance, which are defined next. All abovementioned errors are relative errors (expressed in %) based on the following definition, concerning a single output variable and data pattern,

$$e_{qp} = 100 \left| \frac{d_{qp} - y_{qLp}}{d_{qp}} \right| \quad , \quad (25)$$

where (i) $d_{qp}$ is the $q^{th}$ desired (or target) output when pattern $p$ within iteration $i$ ($p$=1,…, $P_i$) is presented to the network, and (ii) $y_{qLp}$ is net's $q^{th}$ output for the same data pattern. Moreover, denominator in eq. (25) is replaced by 1 whenever $|d_{qp}| < 0.05$ – $d_{qp}$ in the nominator keeps its real value. This exception to eq. (25) aims to reduce the apparent negative effect of large relative errors associated to target values close to zero. Even so, this trick may still lead to (relatively) large solution errors while groundbreaking results are depicted as regression plots (target vs. predicted outputs).

### 3.4.1 Maximum Error

This variable measures the maximum relative error, as defined by eq. (25), among all output variables and learning patterns.

### 3.4.2 Percentage of Errors > 3%

This variable measures the percentage of relative errors, as defined by eq. (25), among all output variables and learning patterns, that are greater than 3%.

### 3.4.3 Performance

In functional approximation problems, network performance is defined as the average relative error, as defined in eq. (25), among all output variables and data patterns being evaluated (e.g., training, all data).

## 3.5 Software Validation

Several benchmark datasets/functions were used to validate the developed software, involving low- to high-dimensional problems and small to large volumes of data. Validation results are not presented herein but they were made public in Researcher (2018). Moreover, several papers involving the successful application of this software have already been published by Abambres and his co-workers.

## 3.6 Parametric Analysis Results

Aiming to reduce the computing time by cutting in the number of combos to be run – note that all features combined lead to hundreds of millions of combos, the whole parametric simulation was divided into nine parametric SAs, where in each one feature 7 only takes a single value. This measure aims to make the performance ranking of all combos within each 'small' analysis more 'reliable', since results used for comparison are based on target and output datasets as used in ANN training and yielded by the designed network, respectively (they are free of any postprocessing that eliminates output normalization effects on relative error values). Whereas (i) the 1$^{st}$ and 2$^{nd}$ SAs aimed to select the best methods from features 1, 2, 5, 8 and 13 (all combined), while adopting a single popular method for each of the remaining features ($F_3$: 6, $F_4$: 2, $F_6$: {1 or 7}, $F_7$: 1, $F_9$: 1, $F_{10}$: 1, $F_{11}$: {3, 9 or 11}, $F_{12}$: 2, $F_{14}$: 1, $F_{15}$: 1 – see Tabs. 2-4) – SA 1 involved learning

algorithms 1-3 and SA 2 involved the ELM-based counterpart, (ii) the 3rd – 7th SAs combined all possible methods from features 3, 4, 6 and 7, and concerning all other features, adopted the methods integrating the best combination from the aforementioned SAs 1-2, (iii) the 8th SA combined all possible methods from features 11, 12 and 14, and concerning all other features, adopted the methods integrating the best combination (results compared after postprocessing) among the previous five sub-analyzes, and lastly (iv) the 9th SA combined all possible methods from features 9, 10 and 15, and concerning all other features, adopted the methods integrating the best combination from the previous analysis. Summing up the ANN feature combinations for all parametric SAs, a total of 475 combos were run for this work (note that this value is much lower than the total number of ANNs simulated).

ANN feature methods used in the best combo from each of the abovementioned nine parametric sub-analyzes, are specified in Tab. 5 (the numbers represent the method number as in Tabs 2-4). Tab. 6 shows the corresponding relevant results for those combos, namely (i) maximum error, (ii) % errors > 3%, (iii) performance (all described in section 3, and evaluated for all learning data), (iv) total number of hidden nodes in the model, and (v) average computing time per example (including data pre- and post-processing). All results shown in Tab. 6 are based on target and output datasets computed in their original format, i.e. free of any transformations due to output normalization and/or dimensional analysis. The microprocessor used in this work has the following features: OS: Win10Home 64bits, RAM: 128 GB, Local Disk Memory: 1 TB, CPU: Intel® Core™ i9 7960X @ 2.80-4.20 GHz.

**Tab. 5.** ANN feature (F) methods used in the best combo from each parametric sub-analysis (SA).

| SA | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | F11 | F12 | F13 | F14 | F15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 2 | 6 | 2 | 2 | 7 | 1 | 1 | 1 | 1 | 3 | 2 | 3 | 1 | 3 |
| **2** | 1 | 2 | 6 | 2 | 2 | 7 | 1 | 2 | 1 | 1 | 9 | 2 | 4 | 1 | 3 |
| **3** | 1 | 2 | 6 | 4 | 2 | 3 | 1 | 1 | 1 | 1 | 3 | 2 | 3 | 1 | 3 |
| **4** | 1 | 2 | 6 | 1 | 2 | 3 | 2 | 1 | 1 | 1 | 3 | 2 | 3 | 1 | 3 |
| **5** | 1 | 2 | 1 | 2 | 2 | 1 | 3 | 1 | 1 | 1 | 3 | 2 | 3 | 1 | 3 |
| **6** | 1 | 2 | 6 | 1 | 2 | 7 | 4 | 1 | 1 | 1 | 3 | 2 | 3 | 1 | 3 |
| **7** | 1 | 2 | 1 | 2 | 2 | 7 | 5 | 1 | 1 | 1 | 3 | 2 | 3 | 1 | 3 |
| **8** | 1 | 2 | 1 | 2 | 2 | 7 | 5 | 1 | 1 | 1 | 3 | 5 | 3 | 1 | 3 |
| **9** | 1 | 2 | 1 | 2 | 2 | 7 | 5 | 1 | 3 | 3 | 3 | 5 | 3 | 1 | 3 |

## 3.7 Proposed ANN-Based Model

The proposed model is the one, among the best ones from all parametric SAs, exhibiting the lowest maximum error (SA 9). That model is characterized by the ANN feature methods {1, 2, 1, 2, 2, 7, 5, 1, 3, 3, 3, 5, 3, 1, 3} in Tabs. 2-4. Aiming to allow implementation of this model by any user, all variables/equations required for (i) data preprocessing, (ii) ANN simulation, and (iii) data postprocessing, are presented in 3.7.1-3.7.3, respectively.

**Tab. 6.** Performance results for the best design from each parametric sub-analysis.

| SA | ANN | | | | |
|---|---|---|---|---|---|
| | Max Error (%) | Performance All Data (%) | Errors > 3% (%) | Total Hidden Nodes | Running Time / Data Point (s) |
| **1** | 209.4 | 11.2 | 64.7 | 32 | 9.95E-05 |
| **2** | 4148.6 | 177.4 | 94.7 | 250 | 1.21E-04 |
| **3** | 248.4 | 10.5 | 59.1 | 32 | 9.68E-05 |
| **4** | 202.4 | 11.9 | 60.0 | 32 | 1.02E-04 |
| **5** | 241.7 | 11.7 | 59.1 | 32 | 1.09E-04 |
| **6** | 219.5 | 11.4 | 59.1 | 32 | 9.73E-05 |
| **7** | 246.3 | 12.7 | 65.8 | 32 | 1.05E-04 |
| **8** | 247.0 | 14.2 | 66.3 | 32 | 1.02E-04 |
| **9** | 16.3 | 1.1 | 10.6 | 33 | 1.06E-04 |

The proposed model is a single MLPN with 5 layers and a distribution of nodes/layer of 8-11-11-11-1. Concerning connectivity, the network is fully-connected, and the hidden and output transfer functions are all Hyperbolic Tangent and Identity, respectively. The network was trained using the LM algorithm (1500 epochs). After design, the average network computing time concerning the presentation of a single example (including data pre/postprocessing) is $1.06 \times 10^{-4}$ s – Fig. 7 depicts a simplified scheme of some of network key features. Lastly, all relevant performance results concerning the proposed ANN are illustrated in 3.7.4. The obtained ANN solution for every data point can be found in Abambres and Cabello (2020), making it possible to compute the exact (with all decimal figures) approximation errors.

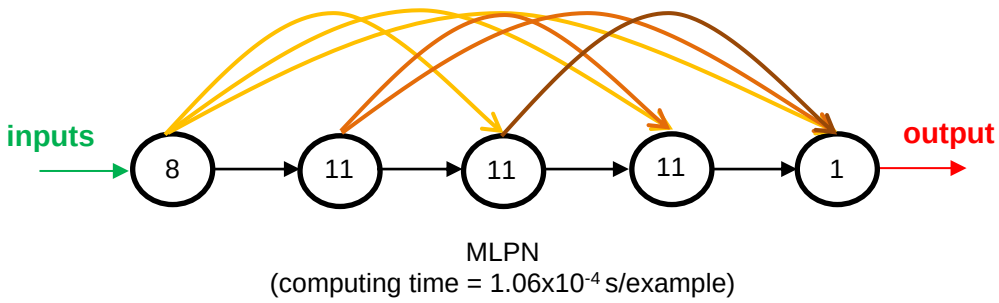


**Fig. 7.** Proposed 8-11-11-11-1 fully-connected MLPN – simplified scheme.

It is worth recalling that, in this manuscript, whenever a vector is added to a matrix, it means the former is to be added to all columns of the latter (valid in MATLAB).

### 3.7.1 Input Data Preprocessing

For future use of the proposed ANN to simulate new data $Y_{1,sim}$ (8 x $P_{sim}$ matrix) concerning $P_{sim}$ patterns, the same data preprocessing (if any) performed before training must be applied to the input dataset. That preprocessing is defined by the methods used for ANN features 2, 3 and 5 (respectively 2, 1 and 2 – see Tab. 2), which should be applied after all (eventual) qualitative variables in the input dataset are converted to numerical (using feature 1's method). Next, the necessary preprocessing to be applied to $Y_{1,sim}$, concerning features 2, 3 and 5, is fully described.

***Dimensional Analysis and Dimensionality Reduction***

Since dimensional analysis (*d.a.*) was not carried out, and the dimensionality reduction (*d.r.*) tentative hasn´t yielded any result according to the described in 3.3.3 (linear correlation), one has

$$\left\{Y_{1,sim}\right\}_{d.r.}^{after} = \left\{Y_{1,sim}\right\}_{d.a.}^{after} = Y_{1,sim} \quad . \quad (26)$$

***Input Normalization***

After input normalization, the new input dataset $\{Y_{1,sim}\}_n^{after}$ is defined as function of the previously determined $\{Y_{1,sim}\}_{d.r}^{after}$, and they have the same size, reading

$$\left\{Y_{1,sim}\right\}_{n}^{after} = \mathrm{INP}(:,1) + rab.\mathrm{x}\left(\left\{Y_{1,sim}\right\}_{d.r}^{after} - \mathrm{INP}(:,3)\right)./den$$

$$\mathrm{INP} = \begin{bmatrix} 0 & 1 & 5 & 25 \\ 0 & 1 & 5 & 50 \\ 0 & 1 & 2 & 10 \\ 0 & 1 & 0.9 & 1.5 \\ 0 & 1 & 0 & 3.504 \\ 0 & 1 & 30 & 200 \\ 0 & 1 & 30 & 200 \\ 0 & 1 & 20 & 49527 \end{bmatrix}$$

$$\begin{aligned} rab &= \mathrm{INP}(:,2) - \mathrm{INP}(:,1) \\ den &= \mathrm{INP}(:,4) - \mathrm{INP}(:,3) \end{aligned} \quad , \quad (27)$$

where one recalls that operator '.x' multiplies component *i* in vector *rab* by all components in row *i* of subsequent term (analogous definition holds for './').

### 3.7.2 ANN-Based Analytical Model

Once determined the preprocessed input dataset $\{Y_{1,sim}\}_n^{after}$ (8 x $P_{sim}$ matrix), the next step is to present it to the proposed ANN to obtain the predicted output dataset $\{Y_{5,sim}\}_n^{after}$ (1 x $P_{sim}$ vector), which will be given in the same preprocessed format of the target dataset used in learning. In order to convert the predicted outputs to their 'original format' (i.e., without any transformation due to normalization or dimensional analysis – the only transformation visible will be the (eventual) qualitative variables written in their numeric representation), some postprocessing is needed, as described in detail in 3.7.3. Next, the mathematical representation of the proposed ANN is given, so that any user

can implement it to determine $\{Y_{5,sim}\}_n^{after}$, thus eliminating all rumors that ANNs are 'black boxes'.

$$\begin{aligned}
Y_2 &= \varphi_2\left(W_{1-2}^T\left\{Y_{1,sim}\right\}_n^{after} + b_2\right)\\
Y_3 &= \varphi_3\left(W_{1-3}^T\left\{Y_{1,sim}\right\}_n^{after} + W_{2-3}^T Y_2 + b_3\right)\\
Y_4 &= \varphi_4\left(W_{1-4}^T\left\{Y_{1,sim}\right\}_n^{after} + W_{2-4}^T Y_2 + W_{3-4}^T Y_3 + b_4\right)\\
\left\{Y_{5,sim}\right\}_n^{after} &= \varphi_5\left(W_{1-5}^T\left\{Y_{1,sim}\right\}_n^{after} + W_{2-5}^T Y_2 + W_{3-5}^T Y_3 + W_{4-5}^T Y_4 + b_5\right)
\end{aligned} \quad , (28)$$

where

$$\begin{aligned}
&\varphi_2 = \varphi_3 = \varphi_4 = \varphi(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}\\
&\varphi_5 = \varphi_5(s) = s
\end{aligned} \quad . (29)$$

Arrays $W_{j\text{-}s}$ and $b_s$ are stored online in Abambres (2020), aiming to avoid an overlong article and ease model's implementation by any interested reader.

### 3.7.3 Output Data Postprocessing

In order to transform the output dataset obtained by the proposed ANN, $\{Y_{5,sim}\}_n^{after}$ (1 x $P_{sim}$ vector), to its original format ($Y_{5,sim}$), i.e. without the effects of dimensional analysis and/or output normalization (possibly) taken in target dataset preprocessing prior training, the postprocessing addressed next must be performed.

#### *Non-normalized (just after dimensional analysis) and Original formats*

Once obtained $\{Y_{5,sim}\}_n^{after}$, the following relations hold for its transformation to its non-normalized format $\{Y_{5,sim}\}_{d.a.}^{after}$, i.e. just after the dimensional analysis stage, and its original format.

$$Y_{5,sim} = \left\{ Y_{5,sim} \right\}_{d.a.}^{after} = \left\{ Y_{5,sim} \right\}_{n}^{after} \quad , \quad (30)$$

since neither output normalization nor dimensional analysis were carried out.

### 3.7.4 Performance Results

Finally, results yielded by the proposed ANN, in terms of performance variables defined in sub-section 3.4, are presented in this section in the form of several graphs: (i) a regression plot (Fig. 8), where network target and output data are plotted, for each data point, as *x*- and *y*- coordinates respectively – a measure of linear correlation is given by the Pearson Correlation Coefficient (*R*), as defined in eq. (1); (ii) a performance plot (Fig. 9), where performance (average error) values are displayed for several learning datasets; and (iii) an error plot (Fig. 10) for functional approximation problems, where values concern all data ($iii_1$) maximum error and ($iii_2$) % of errors greater than 3%.

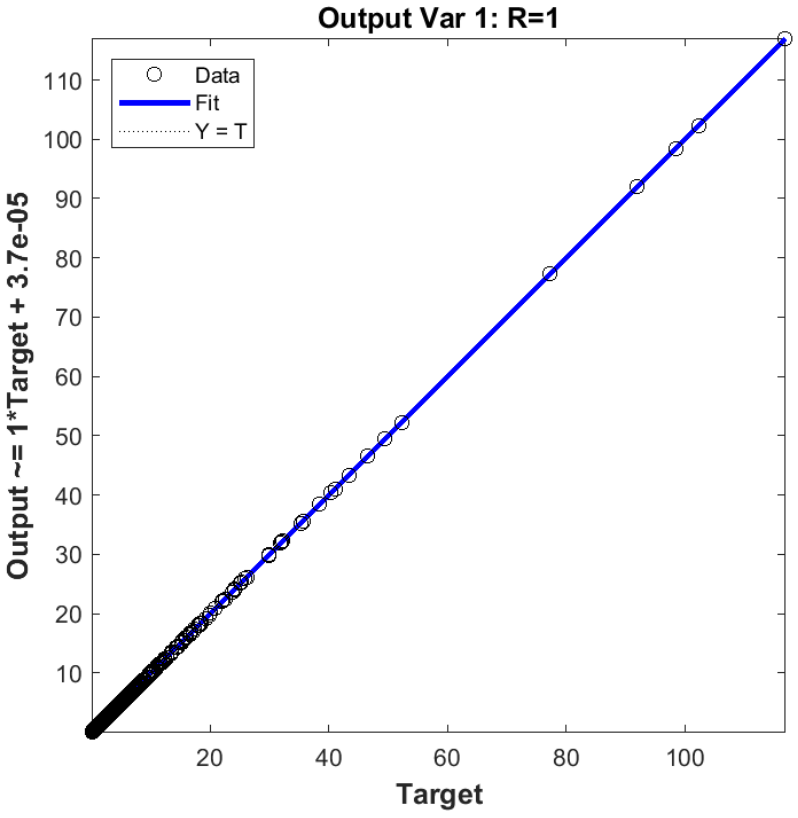


**Fig. 8.** Regression plot for the proposed ANN.

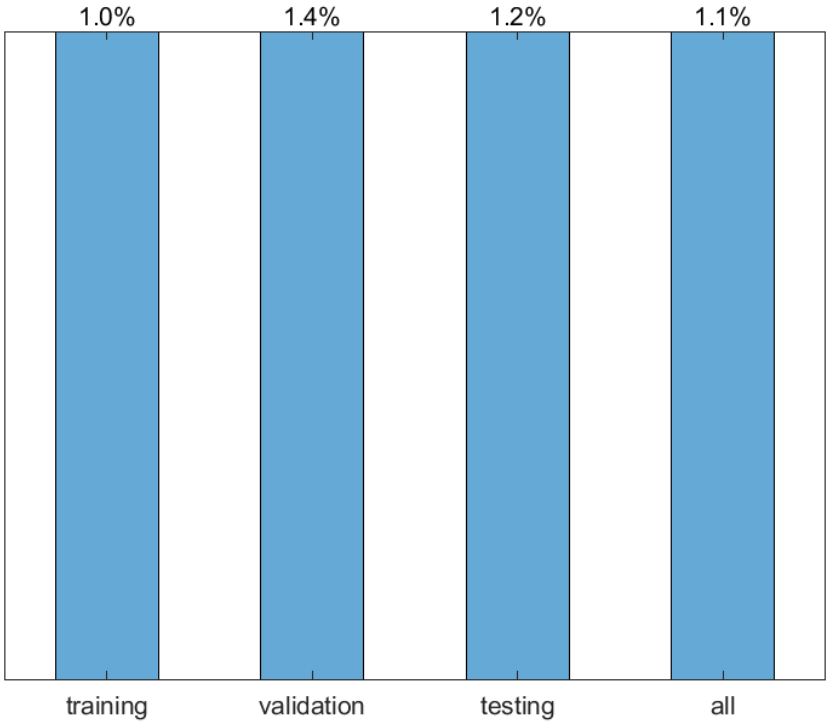


**Fig. 9.** Performance plot (mean errors) for the proposed ANN.

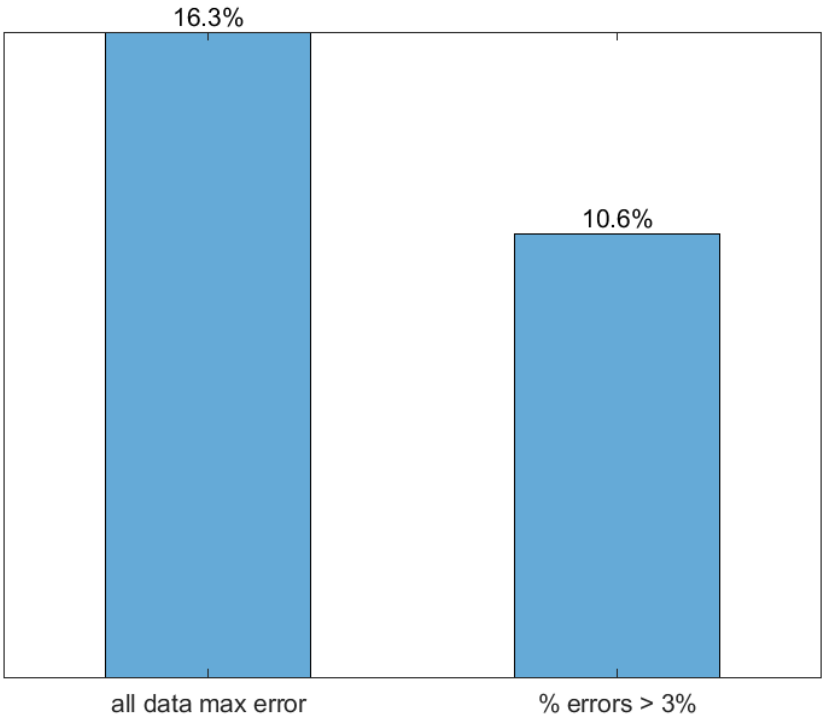


**Fig. 10.** Error plot for the proposed ANN.

## 4. Conclusions

Any engineering structure must comply with the appropriate strength, stability and serviceability criteria defined by design standards. Since grid-shells are slender structures highly prone to buckling, understanding how they perform in terms of stability is a critical aspect of their design. This paper presents an application of Artificial Intelligence to predict the onset of elastic buckling on steel grid-shells of paraboloid shape when subjected to uniform vertical loading.

The proposed Artificial Neural Network (ANN) yields mean and maximum errors of 1.1% and 16.3%, respectively, for all 1098 data points (i.e., FE models). Only in 10.6% of those points the prediction error exceeds 3%. The analytical formulation corresponding to the proposed ANN is thoroughly described. This is a hands-on tool enabling any user to obtain the buckling factor of any grid-shell belonging to the family (and domain) of those described herein. Using an ANN to obtain such set of predictive formulas is a novel approach not relying on any homogenization of the structure, thus avoiding denaturing its discretized condition.

## Author Contributions

Abambres was in charge of section 3 (Artificial Neural Networks), and Cabello of all remaining sections. Both authors equally contributed to the Conclusions section.

## References

Abambres M (2020). W and b arrays, URL.

Abambres M, Marcy M, Doz G (2018). Potential of Neural Networks for Structural Damage Localization, hal-02074844v2

Abambres M, Cabello A (2020). Dataset + Tabled Results, URL.

Anderson D, Hines EL, Arthur SJ, Eiap EL (1997). Application of Artificial Neural Networks to the Prediction of Minor Axis Steel Connections, Computers & Structures, 63(4), 685-692.

Aymerich F, Serra M (1998). Prediction of fatigue strength of composite laminates by means of neural networks, Key Eng. Materials, 144(September), 231–240.

Bai Z, Huang G, Wang D, Wang H, Westover M (2014). Sparse extreme learning machine for classification, IEEE Transactions on Cybernetics, 44(10), 1858–70.

Beyer W, Liebscher M, Beer M, Graf W (2006). Neural Network Based Response Surface Methods - A Comparative Study, 5th German LS-DYNA Forum, October 2006, 29-38, Ulm.

Bhaskar R, Nigam A (1990). Qualitative physics using dimensional analysis, Artificial Intelligence, 45(1-2), 111–73.

Deng W-Y, Bai, Z., Huang, G.-B. and Zheng, Q.-H. (2016). A fast SVD-Hidden-nodes based extreme learning machine for large-scale data Analytics, Neural Networks, 77(May), 14–28.

Dini M, Estrada G, Froli M, Baldassini N. (2013). Form-finding and buckling optimisation of gridshells using genetic algorithms. Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium.

Dulácska E, Kollár L (2000) Buckling analysis of reticulated shells. International Journal of Space Structures. Vol 15, 3-4.

Edemskaya E, Agkathidis A. (2016). Rethinking Complexity: Vladimir Shukhov's Steel Lattice Structures. Journal of the International Association for Shell and Spatial Structures. Vol. 57

EN 1993-1-1 (2005) Eurocode 3: Design of steel structures - Part 1-1: General rules and rules for buildings. European Committee for Standardization.

Flood I (2008). Towards the next generation of artificial neural networks for civil engineering, Advanced Engineering Informatics, 22(1), 4-14.

Flood I, Kartam N (1994a). Neural Networks in Civil Engineering: I-Principals and Understanding, Journal of Computing in Civil Engineering, 8(2), 131-148.

Gholizadeh S, Pirmoz A, Attarnejad R (2011). Assessment of load carrying capacity of castellated steel beams by neural networks, Journal of Constructional Steel Research, 67(5), 770–779.

Gioncu V (1994). Buckling of reticulated shells: State-of-the-art, International Journal of Space Structures. Vol 10, 1.

Gunaratnam DJ, Gero JS (1994). Effect of representation on the performance of neural networks in structural engineering applications, Computer-Aided Civil and Infrastructure Engineering, 9(2), 97–108.

Haykin SS (2009). Neural networks and learning machines, Prentice Hall/Pearson, New York.

Hern A (2016). Google says machine learning is the future. So I tried it myself. Available at: www.theguardian.com/technology/2016/jun/28/all (Accessed: 2 November 2016).

Hertzmann A, Fleet D (2012). Machine Learning and Data Mining, Lecture Notes CSC 411/D11, Computer Science Department, University of Toronto, Canada.

Huang G, Chen L, Siew C (2006b). Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE transactions on neural networks, 17(4), 879–92.

Huang G-B, Chen L (2007). Convex incremental extreme learning machine, Neurocomputing, 70(16–18), 3056–3062.

Huang G-B, Zhu Q-Y, Siew C-K (2006a). Extreme learning machine: Theory and applications, Neurocomputing, 70(1-3), 489-501.

Hwang K J. (2010). Advanced Investigations of Grid Spatial Structures. Considering various connection systems. PhD thesis. ITKE, University of Stuttgart.

Kasun LLC, Yang Y, Huang G-B, Zhang Z (2016). Dimension reduction with extreme learning machine, IEEE Transactions on Image Processing, 25(8), 3906–18.

Kato S, Yamauchi Y, Ueki T, Okuhira K (2005). Buckling load of elliptic paraboloidal single layer reticulated roofs with simple supports under uniform load. International Journal of Space Structures. Vol 20, 4.

Kuijvenhoven M. (2009). A design method for timber grid shells. MSc thesis. Department of Structural Mechanics. Delft University of Technology.

Lachtermacher G, Fuller JD (1995). Backpropagation in time-series forecasting, Journal of Forecasting 14(4), 381–393.

Lefevre B, Douthe C, Baverel O (2015) Buckling of elastic gridshells. Journal of the International Association of Shell and Spatial Structures. Vol 56, 153-171.

Lefik M, Schrefler BA (2003). Artificial neural network as an incremental non-linear constitutive model for a finite element code, Computer Methods in Applied Mechanics and Engineering, 192(28–30), 3265–3283.

Liang N, Huang G, Saratchandran P, Sundararajan N (2006). A fast and accurate online Sequential learning algorithm for Feedforward networks, IEEE Transactions on Neural Networks, 17(6), 1411–23.

Makin T. (2006). Timber Gridshell structures. Final Project. Oxford University, Department of Engineering Science

McCulloch WS, Pitts W (1943). A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, 5(4), 115–133.

McNeel (2014) Rhinoceros v.5, USA.

Mukherjee A, Deshpande JM, Anmala J (1996), Prediction of buckling load of columns using artificial neural networks, Journal of Structural Engineering, 122(11), 1385–7.

Oasys Ltd (2010) General Structural Analysis GSA v.8.5 Manual, London.

Olsson J. (2012). Form finding and size optimisation. Implementation of beam elements and size optimization in real time form finding using dynamic relaxation. MSc thesis. Chalmers University of Technology. Department of Applied Mechanics.

Peloux (du) L, Baverel O, Caron J-F, Tayeb F. (2013). From shape to shell: a design tool to materialize freeform shapes using gridshell structures. Rethinking Prototyping, Design Modelling Symposium.

Pottmann H, Eigensatz M, Vaxman A, Wallner J. (2014) Architectural Geometry. Computers & Graphics. Vol 47.

Prieto A, Prieto B, Ortigosa EM, Ros E, Pelayo F, Ortega J, Rojas I (2016). Neural networks: An overview of early research, current frameworks and new challenges, Neurocomputing, 214(November), 242-268.

Pu Y, Mesbahi E (2006). Application of artificial neural networks to evaluation of ultimate strength of steel panels, Engineering Structures, 28(8), 1190–1196.

Rafiq M, Bugmann G, Easterbrook D (2001). Neural network design for engineering applications, Computers & Structures, 79(17), 1541–1552.

Researcher, The (2018). "Annsoftwarevalidation-report.pdf", figshare, 10.6084/m9.figshare.6962873.

Rutten D (2014) Grasshopper v. 0.9.0076, USA.

Schlaich J, Schober H. (1996). Glass-Covered Grid-shells. Structural Engineering International 2/96 Aesthetics in Structural Engineering.

Schlaich M, Burkhardt U, Irisarri L, Goñi J (2009). Palacio de Comunicaciones – a single layer glass grid-shell over the courtyard of the future town hall of Madrid. Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium.

Schober H. (2016). Transparent Shells. Form, Topology, Structure. Ernst & Sohn

Schwenker F, Kestler H, Palm G (2001). Three learning phases for radial-basis-function networks, Neural networks, 14(4-5), 439–58.

Tayeb F, (2015). Simulation numérique du comportement mécanique non linéaire de gridshells composés de poutres élancées en matériaux composites et de sections quelconques. PhD Thesis. École Nationale des Ponts et Chaussées.

The Mathworks, Inc (2017). MATLAB R2017a, User's Guide, Natick, USA.

Tohidi S, Sharifi Y (2014). Inelastic lateral-torsional buckling capacity of corroded web opening steel beams using artificial neural networks, The IES Journal Part A: Civil & Structural Eng, 8(1), 24–40.

Toussaint M. (2007). A Design Tool for Timber Gridshells. Delft University of Technology Department of Structural and Building Engineering.

Waszczyszyn Z (1999). Neural Networks in the Analysis and Design of Structures, CISM Courses and Lectures No. 404, Springer, Wien, New York.

Wilamowski BM (2009). Neural Network Architectures and Learning algorithms, IEEE Industrial Electronics Magazine, 3(4), 56-63.

Wilamowski BM (2011). How to not get frustrated with neural networks, 2011 IEEE International Conference on Industrial Technology (ICIT), 14-16 March 2011, IEEE (eds), Auburn University, Auburn, AL, USA.

Wilamowski BM, Irwin JD (2011). The industrial electronics handbook: Intelligent Systems, CRC Press, Boca Raton.

Wilson DR, Martinez TR (2003). The general inefficiency of batch training for gradient descent learning, Neural Networks, 16(10), 1429–1451.

Xie T, Yu H, Wilamowski B (2011). Comparison between traditional neural networks and radial basis function networks, 2011 IEEE International Symposium on Industrial Electronics (ISIE), IEEE(eds), 27-30 June 2011, Gdansk University of Technology Gdansk, Poland, 1194–99.

Xu S, Chen L (2008). Novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining, In: International Conference on Information Technology and Applications (ICITA), Cairns (Australia), 23–26 June 2008, pp 683–686.

## Software bugs in previous papers – sources and solutions

On **March 2021**, Abambres found some bugs on his own ANN software, which had been used to yield ANN results for several papers published until that date, including this one. A not-so-severe bug was found in the definition of the 8th parametric sub-analysis, since it was not defined exactly as it was described in section “Parametric Analysis Results”. The critical bugs were found in non-ELM (non-Extreme Learning Machine) learning algorithms for "mini-batch" and "online" training modes, and unintendedly caused the “% Train - Valid - Test” ANN feature to become “100 - 0 - 0” during neural net design, thus affecting the generalization potential of the proposed ANN. The bug sources were the following:

- "trainlm", "traingd" and "traingda" training functions do not allow incremental (online or mini-batch) training when using MATLAB neural network toolbox – regardless the data format employed in *train(…)* arguments, only batch training will be used.
- if *net.divideMode* is not specified when using MATLAB neural net toolbox, the default value for feedforward ANNs is *'sample'*. However, the implemented data division (train / valid / test) for incremental training uses timestep (instead of sample) indexes, which means the assignment *net.divideMode = 'time'* was missing right before *net.divideFcn = 'divideind'*.

In order to overcome the above cited issues, Abambres first explored MATLAB’s training functions that are said to allow incremental training, but found several limitations (reported in here https://archive.ph/r6Yw3 and here https://archive.ph/HsaC6). Thus, Abambres’ final decision was to redefine the ANN parametric analysis in order to (i) remove all incremental non-ELM learning algorithms, and (ii) increase the ELM-based simulations while improving their cross-validation.

Miguel Abambres

## Final Note

For personal and professional reasons, I have decided not to write/publish (pro-bono) the new version of this paper for the results obtained with the debugged ANN software, even though the former were pretty satisfactory.

Miguel Abambres