Potential of neural networks for maximum displacement predictions in railway beams on frictionally damped foundations

Miguel Abambres [1,1], Rita Corrêa [2], AP Costa [2], and F Simões [2]

[1]Num3ros
[2]Affiliation not available

March 19, 2024

Abstract

Since the use of finite element (FE) simulations for the dynamic analysis of railway beams on frictionally damped foundations are (i) very time consuming, and (ii) require advanced know-how and software that go beyond the available resources of typical civil engineering firms, this paper aims to demonstrate the potential of Artificial Neural Networks (ANN) to effectively predict the maximum displacements and the critical velocity in railway beams under moving loads. Four ANN-based models are proposed, one per load velocity range ([50, 175] [?] [250, 300] m/s;]175, 250[m/s) and per displacement type (upward or downward). Each model is function of two independent variables, a frictional parameter and the load velocity. Among all models and the 663 data points used, a maximum error of 5.4 % was obtained when comparing the ANN- and FE-based solutions. Whereas the latter involves an average computing time per data point of thousands of seconds, the former does not even need a millisecond. This study was an important step towards the development of more versatile (i.e., including other types of input variables) ANN-based models for the same type of problem.

Hosted file

`Software Bugs Found on March 2021.docx` available at https://authorea.com/users/594089/articles/676123-potential-of-neural-networks-for-maximum-displacement-predictions-in-railway-beams-on-frictionally-damped-foundations

Potential of neural networks for maximum displacement predictions in railway beams on frictionally damped foundations

Miguel Abambres [a], Rita Corrêa [b], António Pinto da Costa [c], Fernando Simões [c]

[a] R&D, Abambres' Lab, 1600-275 Lisboa, Portugal; abambres@netcabo.pt

[b] GRID International, Consulting Engineers, 1050-125 Lisboa, Portugal

[c] CERIS and Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisboa, Portugal

Abstract

Since the use of finite element (FE) simulations for the dynamic analysis of railway beams on frictionally damped foundations are (i) very time consuming, and (ii) require advanced know-how and software that go beyond the available resources of typical civil engineering firms, this paper aims to demonstrate the potential of Artificial Neural Networks (ANN) to effectively predict the maximum displacements and the critical velocity in railway beams under moving loads. Four ANN-based models are proposed, one per load velocity range ([50, 175] ∪ [250, 300] m/s;]175, 250[m/s) and per displacement type (upward or downward). Each model is function of two independent variables, a frictional parameter and the load velocity. Among all models and the 663 data points used, a maximum error of 5.4 % was obtained when comparing the ANN- and FE-based solutions. Whereas the latter involves an average computing time per data point of thousands of seconds, the former does not even need a millisecond. This study was an important step towards the development of more versatile (i.e., including other types of input variables) ANN-based models for the same type of problem.

Keywords: Railway Beam; Frictionally Damped Foundation; Moving Load; Maximum Displacements; Critical Velocity; Artificial Neural Networks.

1. Introduction

The study of the dynamic behavior of beams on foundations subjected to moving loads with possible applications in high-speed railway track design has been a topic of interest in the literature. In particular, the existence of a critical velocity of the load for which the beam's

oscillation amplitudes become very large has been demonstrated (Frýba 1972). The serviceability of a high-speed railway track depends on the limitation of these dynamic amplifications (Madshus and Kaynia 2000, Kaynia et al. 2000), that is, depends on the ability of its substructure to dissipate the energy transmitted by the moving loads. This substructure is composed of many stones of several sizes and shapes, interacting through surfaces that are almost always in persistent frictional contact.

Since the main mechanism governing the interaction between the infrastructure's constituents is unilateral frictional contact mechanics, a novel non-smooth foundation model, which is closer to the true frictional dissipative nature of the ballast than the viscous model, was proposed in Toscano Corrêa et al. (2018). The goal of that study was to generalize, for more realistic behaviors, the analyses in Dimitrovová and Rodrigues (2012) and Castro Jorge et al. (2015a, b) so that they could be of interest for high-speed railway engineers. Thus, a finite element (FE) program was developed within a MATLAB (The Mathworks, Inc 2017) environment to analyze the dynamic behavior of a Euler-Bernoulli beam on a foundation composed of continuous distributions of linear elastic springs and Coulomb frictional dissipators/dampers. This "Winkler-Coulomb"-type foundation is represented in Fig. 1 under a simply supported beam. It assumes that, in parallel with a linear elastic Winkler foundation, there is a reaction per unit length that, at each cross section of the beam, obeys to Coulomb's friction law: the frictional dissipators apply an instantaneous reaction per unit length $r(x,t)$ at cross section x and time instant t depending on the sign of the transverse velocity of that cross section, i.e. $r(x,t) \in -f_u \, \mathrm{Sign}(\dot{w}(x,t))$ where (i) f_u denotes the maximum force per unit length that the system of frictional dissipators of the foundation may support, (ii) $\dot{w}(x,t)$ is the

transverse velocity of the cross section, and (iii) $\text{Sign}\big(\dot{w}(x,t)\big) = \dot{w}(x,t)/|\dot{w}(x,t)|$ if $\dot{w}(x,t) \neq 0$ and $\text{Sign}\big(\dot{w}(x,t)\big) = [-1\,,+1]$ if $\dot{w}(x,t) = 0$. The expression of the reaction force is an algebraic inclusion (Glocker 2001, Studer 2009), meaning that at the instants of vanishing velocity the reaction may belong to an interval and at the instants of velocity sign change the reaction per unit length is discontinuous. This reaction is very different from the one provided by a continuous distribution of the traditional linear viscous dampers, $r(x,t) = c\,\dot{w}(x,t)$, where c is the viscous damping coefficient per unit length. In both cases the reaction opposes the velocity but, while viscous damping provides a reaction that is proportional to the local velocity itself, the frictional reaction is limited to the interval $[-f_u, +f_u]$ and is independent of the magnitude of the velocity $\dot{w}(x,t)$ (see Fig. 2 in Toscano Corrêa et al. 2018). In that study a time stepping algorithm specially designed to deal with non-smooth dynamical systems was for the first time applied to beams on distributed friction foundations and new conclusions on critical velocities, maximal displacements and dynamic amplification factors were drawn.

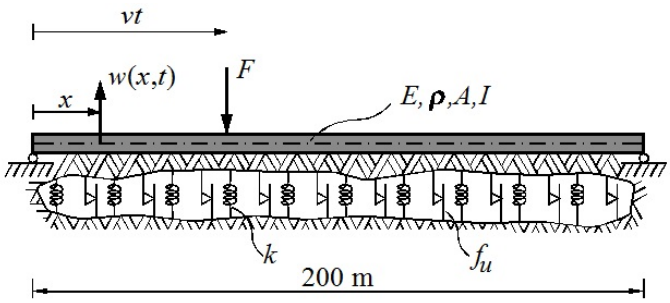

Fig. 1. Beam on a frictionally damped continuous foundation under a moving load.

Since the FE analyses in Toscano Corrêa et al. (2018) are (i) very time consuming (thus unfeasible for fast engineering estimations), and (ii) require advanced know-how and software that go beyond the available resources of typical civil engineering firms, this paper aims to

demonstrate the potential of Artificial Neural Networks (ANN) to effectively predict the maximum displacements in railway tracks on frictional foundations, as function of the frictional parameter (f_u) and load velocity (v). This is an important step towards the future development of much more versatile ANN-based analytical models for the same type of problem. The difference will be the inclusion of more independent (input) variables, such as the foundation stiffness modulus k, the applied load magnitude F, and geometrical/mechanical properties of the railway beam (see Fig. 1).

2. FE-based model and data gathering

The authors considered a horizontal simply supported linear elastic Euler-Bernoulli beam (see Fig. 1) of 200 m length, cross-sectional area and central moment of inertia respectively equal to A and I, mass density ρ and Young's modulus E. The properties of the beam are summarized in Table 1 and correspond to the ones of a UIC60 rail. Previous studies (Dimitrovová and Rodrigues 2012, Castro Jorge et al. 2015a, b) showed that a 200 m simply supported beam is a good finite length model to approximate the behavior of an infinite beam on elastic foundation with a single moving load. The beam is supposed to be connected to a fixed foundation bed by a system of linear elastic springs, with stiffness per unit length denoted by k, associated in parallel with a continuous distribution of friction dampers with a maximum force per unit length f_u. A downward concentrated force F = 83.4 kN, corresponding to half of the load per axle of a Thalys high speed train locomotive, acts on the beam moving from left to right at a constant velocity v (numerical results considered v ranging between 50 m/s and 300 m/s at intervals of 5 m/s). The motion of the beam is governed by a partial differential inclusion (eq. (2) in Toscano Corrêa et al. (2018))

that is (i) semi-discretized in space, using the finite element method, as a system of ordinary differential inclusions (eq. (5) in Toscano Corrêa et al. (2018)), and (ii) integrated in time using a special implementation of the Non-smooth Contact Dynamics Method (NSCD) developed by Moreau (1994) and Jean (1999), adapted to distributed Coulomb friction. The stiffness modulus of the foundation is k = 250 kN/m^2 and thirteen different values of the maximum force per unit length of the frictional dampers f_u (0, 1, 2, 3, 3.5, 4, 5, 6, 7, 7.5, 8, 9 and 10 kN/m) are considered. All simulations employed a 200-element uniform mesh and a time step h such that vh = 0.1 m (i.e., in each time step the load progresses 10 cm independently of its velocity v). The suitability of the mesh refinement and time step was assessed in Toscano Corrêa et al. (2018). The self-weight of the beam is not considered, as in Toscano Corrêa et al. (2018), in order to allow the comparison of results.

The maximum upward (positive w_{max}) and downward (negative w_{max}) transverse displacements of the beam under moving load are obtained as function of the frictional parameter f_u and load velocity v, using the FE program mentioned before. For each pair of f_u and v values, the computational time to obtain the corresponding upward and downward maximum displacements ranged between 4000 and 4800 seconds, when using a computer with an Intel® Core™ i5-3470 CPU @ 3.20 GHz, 8 GB of RAM, and a 64-bit Operating System.

The aforementioned data was then used to feed the neural networks analyzed in this work. After some experiments, and aiming to obtain sufficiently accurate (maximum error smaller or around 5%) models, it was decided to develop four independent ANN-based models, namely for: (i) negative w_{max} (v = [50, 175] ∪ [250, 300] m/s), (ii) negative w_{max} (v =]175, 250[m/s), (iii) positive w_{max} (v = [50, 175] ∪ [250, 300] m/s), and (iv) positive w_{max} (v =]175, 250[m/s).

Each model is described in sub-sections 4.1-4.4, respectively, and characterized by two independent (f_u, v) and one dependent (positive or negative w_{max}) variables. The datasets used in the development and final testing of each model are available in Authors (2018).

Table 1. Properties of the UIC60 rail (Toscano Corrêa et al. 2018).

Property	Value
Young's modulus (E)	210 GPa
Central area moment of inertia (I)	3055×10^{-8} m^4
Cross-sectional area (A)	7684×10^{-6} m^2
Density (ρ)	7800 kg/m^3

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 the theory behind is poor or absent, machine learning can be a useful tool. The world is quietly being reshaped by machine learning, the Artificial Neural Network (also referred in this manuscript as ANN or neural net) being 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, inspired

by the way the brain processes information, i.e. based on 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 modeled (Flood 2008).

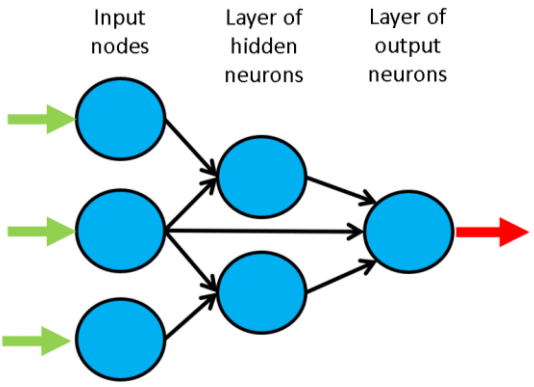

Fig. 2. Example of a feedforward neural network.

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 (as depicted later in 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 potential 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.

3.2 Learning

Each connection between two 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 'labeled' 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 labeled 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 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 modeled (Haykin 2009).

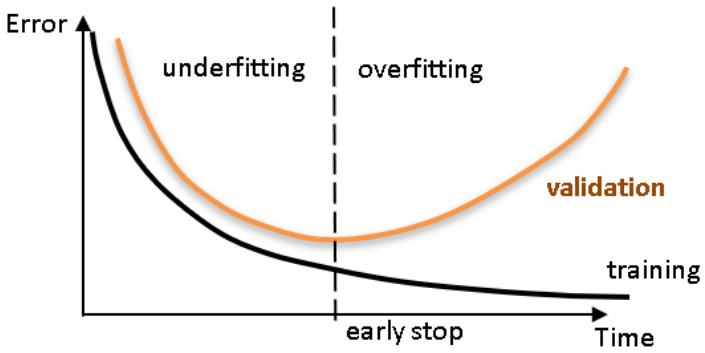

Fig. 3. Cross-validation - assessing network's generalization ability.

3.3 Implemented ANN features

The 'behavior' of any ANN depends on many 'features'. Fifteen of them were implemented 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-analyses) 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; Tables 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 Table 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.

Table 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.

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).

Table 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[φ_{min}, φ_{max}]	MLPN	1 HL	Adjacent Layers
2	-	Lin [a, b] = 0.6[φ_{min}, φ_{max}]	RBFN	2 HL	Adj Layers + In-Out
3	Hyperbolic Tang	Lin [a, b] = 0.5[φ_{min}, φ_{max}]	-	3 HL	Fully-Connected
4	-	Linear Mean Std	-	-	-
5	Bilinear	No	-	-	-
6	Compet	-	-	-	-
7	Identity	-	-	-	-

Table 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	NNC	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	-	-	-	-

3.3.3 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

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

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 modeling. The one to be removed is the one appearing less in the remaining (X, Y) pairs where $|R_{XY}| \geq 0.99$. Once a variable is selected for removal, all (X, Y) pairs 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, i.e. the 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 percentage 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) For each variable q (row) in the complete input dataset, compute its minimum and maximum values.
2) 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 'does not reach' p_t, one should add the missing amount, provided those patterns are the ones having more variables taking extreme (minimum or maximum) values.
3) 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} is not equal to the one imposed *a priori* (before step 1), which is due to the minimum required training patterns specified in step 2.

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

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

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 largest 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% [φ_{min}, φ_{max}], (ii) [a, b] = 60% [φ_{min}, φ_{max}], and (iii) [a, b] = 50% [φ_{min}, φ_{max}], being [φ_{min}, φ_{max}] the output transfer function range, and [a, b] determined to be centered within [φ_{min}, φ_{max}] and

to span the specified percentage (e.g., (b-a) = 0.7 (φ_{max} - φ_{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 only to some nodes located ahead. At this moment, it is appropriate to define the concept of partially- (PC) and fully-connected (FC) ANNs. 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 network is said to be PC (e.g., the one in Fig. 2). According to Wilamowski (2009), PC MLPNs are less powerful than MLPNs 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) labels a generic layer and '*ql'* a generic node, being q = 1,…, 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

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

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

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

where φ_l is the transfer function used for all neurons in layer l ($l = 2,\ldots, L$).

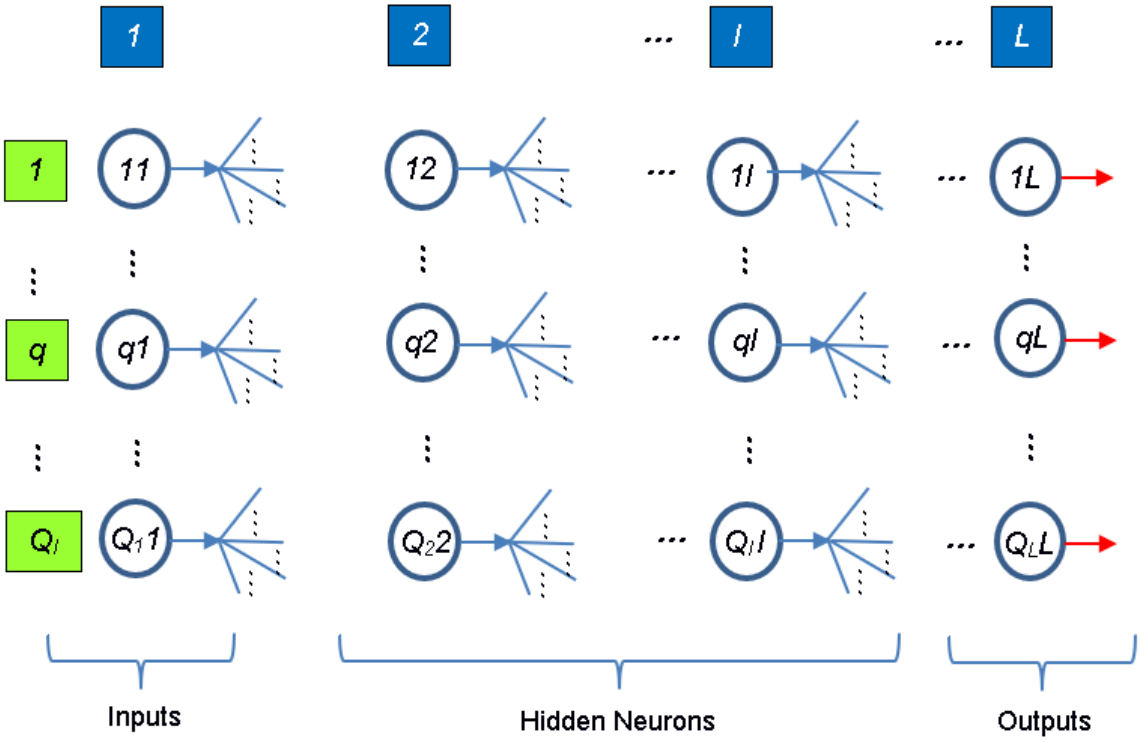

Fig. 4. Generic multi-layer feedforward network.

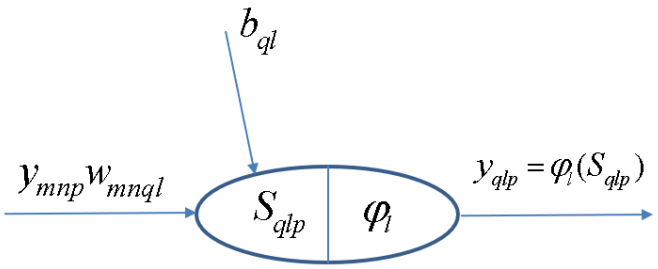

Fig. 5. Generic neuron placed anywhere in the MLPN of Fig. 4 (l = 2,…, L).

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) φ_{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 Table 4, vectors $v_{l_1 l_2 p}$ and $\xi_{l_1 l_2}$ are defined as (two versions of $v_{l_1 l_2 p}$ where implemented and the one yielding the best results was selected)

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

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

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

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.

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, i.e. 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 Table 4).

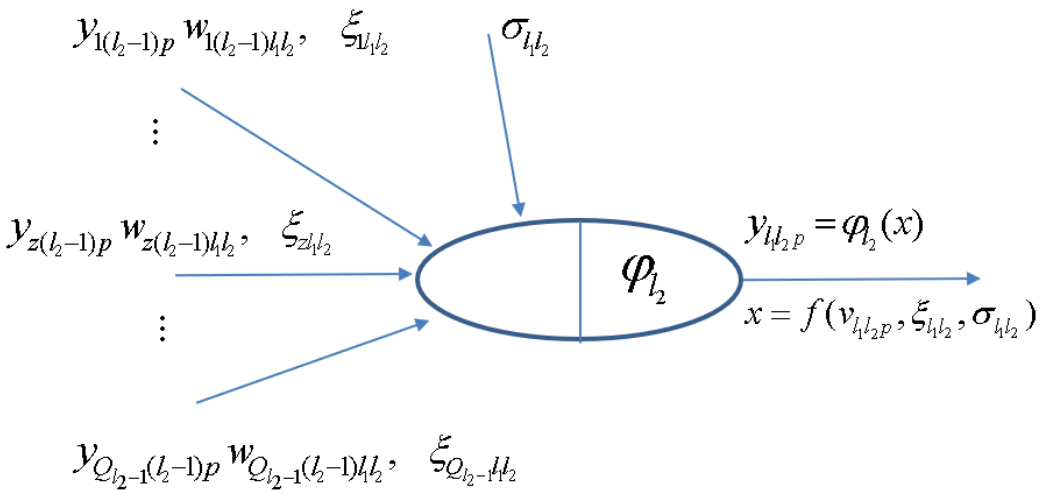

Fig. 6. Generic hidden neuron l_1l_2 placed anywhere in the RBFN of Fig. 4 ($l_2 = 2,\ldots, L\text{-}1$).

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). Another implemented replacement 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

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

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

$$\varphi_{l_2}(s) = s\ln\left(\sqrt{s}\right), \quad s = \left\| v_{l_1 l_2 p} - \xi_{l_1 l_2} \right\|^2 \quad , \quad (20)$$

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

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

(iii) the Multiquadratic function is given by

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

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

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

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* [-Δ, Δ]**

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

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

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, Δ 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 / $\sqrt{cs}$, where *cs* is 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)

A simple and recursive approach aiming to improve ANN accuracy is called Neural Network Composite (NNC), as described in Beyer et al. (2006). In this work, a maximum of 10 extra ANNs were added to the original one, until maximum error is not improved between successive NNC solutions. Later in this manuscript, a solution given by a single neural net might be denoted as ANN, whereas the other possible solution is called NNC.

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 η_{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) percentage of 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 replacement 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 larger than 3%

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

3.4.3 Performance

In functional approximation problems, network performance is defined as the average relative error, as defined in eq. (25), among all evaluated output variables and data patterns (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. Due to paper length limit, validation results are not presented herein but they were made public online (Researcher 2018). In spite of the successful validation, several improvements have been implemented since the initial use of the software in first author's research projects.

4. Results and Proposed ANN-based Models

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 sub-analyses (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 SA 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 1st and 2nd 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 Tables 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 first SA, (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-analyses, 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. The microprocessors used for ANN simulations have the following features: OS: Win10Home 64bits, RAMs: 48.0 / 128 GB, Local Disk Memory: 1 TB, CPUs: Intel® Core™ i7 8700K @ 3.7-4.7 GHz / i9 7960X @ 2.8-4.2 GHz. The datasets used in the development and final testing of all ANN models are available in Authors (2018). In what follows, the parametric analysis results and the proposed ANN-based models are presented for each of the four problems addressed, namely: (i) negative w_{max} (v = [50, 175] ∪ [250, 300] m/s), (ii) negative w_{max} (v =]175, 250[m/s), (iii) positive w_{max} (v = [50, 175] ∪ [250, 300] m/s), and (iv) positive w_{max} (v =]175, 250[m/s). It is important to note that, in this manuscript, whenever a vector is added to a matrix, it means the former is added to all columns of the latter (valid in MATLAB).

4.1 Negative w_{max} (v = [50, 175] ∪ [250, 300] m/s)

ANN feature methods used in the best combo from each of the abovementioned nine parametric SAs are specified in Table 5 (see Tables 2-4). Table 6 shows the corresponding relevant results for those combos and the 481-point final testing dataset (which includes the ANN learning/development dataset), namely (i) maximum error, (ii) percentage of errors larger than 3%, (iii) performance (all described in sub-section 3.4, 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 Table 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. Summing up the ANN feature combinations for all parametric SAs, a total of 219 combos were run for this problem.

Table 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	5	1	1	1	1	1	3	2	3	1	3
2	1	2	6	2	5	7	1	1	1	1	3	2	5	1	3
3	1	2	6	2	5	1	1	1	1	1	3	2	3	1	3
4	1	2	6	2	5	1	2	1	1	1	3	2	3	1	3
5	1	2	6	4	5	1	3	1	1	1	3	2	3	1	3
6	1	2	6	2	5	7	4	1	1	1	3	2	3	1	3
7	1	2	6	4	5	3	5	1	1	1	3	2	3	1	3
8	1	2	6	4	5	3	5	1	1	1	3	5	3	1	3
9	1	2	6	4	5	3	5	1	3	3	3	5	3	1	3

The proposed model is the one, among the best ones from all parametric SAs, exhibiting the lowest maximum error (SA 7 – a Neural Network Composite (NNC)). 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 the following sub-sections. The proposed model is an NNC made of 7 ANNs with architecture MLPN and a distribution of nodes/layer equal to 2-8-1 for every network. Concerning connectivity, all networks are partially-connected, and the hidden and output transfer functions are all Hyperbolic Tangent (eq. (8)). All networks were trained using the LM algorithm. After design, the average NNC computing time concerning the presentation of a

single example (including data pre/postprocessing) is 3.12E-05 s; Fig. 7 depicts a simplified scheme of each NNC network. Finally, all relevant performance results concerning the proposed NNC are illustrated in sub-section 4.1.4.

Table 6. Performance results for the best design from each parametric SA and for the final testing dataset (includes the ANN learning/development dataset): ANN and NNC.

SA	ANN				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	5.6	1.0	3.7	8	7.95E-05
2	15.3	2.3	20.4	40	2.89E-05
3	5.7	1.0	3.7	8	2.88E-05
4	6.3	0.9	3.5	8	2.91E-05
5	5.7	1.0	2.7	8	3.64E-05
6	5.3	1.0	4.8	8	4.86E-05
7	6.1	1.0	4.6	8	2.75E-05
8	7.5	1.0	4.4	8	4.66E-05
9	3.2	0.6	0.2	9	4.36E-05
SA	**NNC**				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	-	-	-	-	-
6	-	-	-	-	-
7	2.7	0.4	0.0	8	3.12E-05
8	5.8	0.8	2.3	8	4.89E-05
9	-	-	-	-	-

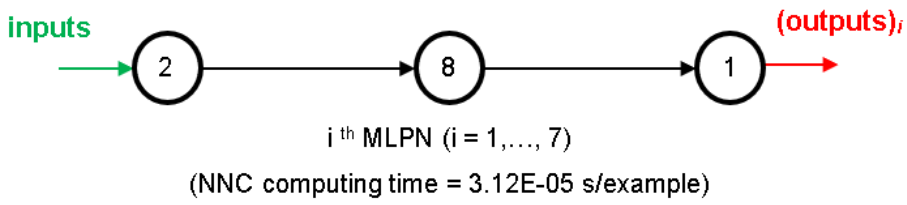

Fig. 7. Proposed NNC made of 7 partially-connected MLPNs – simplified scheme.

4.1.1 Input Data Preprocessing

For future use of the proposed NNC to simulate new data $Y_{1,sim}$ (a 2 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, 6 and 5 – see Table 2). 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 no dimensional analysis (*d.a.*) nor dimensionality reduction (*d.r.*) were carried out, 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

$$\left\{Y_{1,sim}\right\}_{n}^{after} = \left(\left\{Y_{1,sim}\right\}_{d.r}^{after} - \text{INP}(:,1)\right) ./ \text{INP}(:,2)$$

$$\text{INP} = \begin{bmatrix} 5000 & 3166.27801301405 \\ 161.527777777778 & 82.2184600671207 \end{bmatrix} \quad , \quad (27)$$

where one recalls that (i) INP(:, *j*) represents column *j* of matrix INP, and (ii) operator './' divides row *i* in the numerator by INP(*i*, 2).

4.1.2 ANN-Based Analytical Model

Once determined the preprocessed input dataset $\{Y_{1,sim}\}_n^{after}$ (a 2 x P_{sim} matrix), the next step is to present it to the proposed NNC to obtain the predicted output dataset $\{Y_{3,sim}\}_n^{after}$ (a 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), some postprocessing might be needed, as discussed in 4.1.3. Next, the mathematical representation of the proposed NNC is given, so that any user can implement it to determine $\{Y_{3,sim}\}_n^{after}$, thus contributing to diminish the generalized opinion that ANNs are 'black boxes':

$$\left\{Y_{3,sim}\right\}_n^{after} = \left\{Y_{3,sim}\right\}_{n(0)}^{after} + \sum_{i=1}^{6} \left\{Y_{3,sim}\right\}_{n(i)}^{after} \quad , \quad (28)$$

being

$$\begin{aligned} Y_{2(i)} &= \varphi_2\left(W_{1-2(i)}^T \left\{Y_{1,sim}\right\}_n^{after} + b_{2(i)}\right) \\ \left\{Y_{3,sim}\right\}_{n(i)}^{after} &= \varphi_3\left(W_{2-3(i)}^T Y_{2(i)} + b_{3(i)}\right) \end{aligned} \quad , \quad (29)$$

where $i = 0,\ldots, 6$ and

$$\varphi_2 = \varphi_3 = \varphi(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} \quad , \quad (30)$$

and arrays $W_{j\text{-}s(i)}$ and $b_{s(i)}$ are stored online in Developer (2018a), aiming to avoid an overlong article and ease model's implementation by any interested reader.

4.1.3 Output Data Postprocessing

In order to transform the output dataset obtained by the proposed NNC, $\{Y_{3,sim}\}_n^{after}$ (a 1 x P_{sim} vector), to its original format ($Y_{3,sim}$), i.e. without the effects of dimensional analysis and/or output normalization (possibly) taken in target dataset preprocessing prior training, one has

$$Y_{3,sim} = \left\{Y_{3,sim}\right\}_n^{after} \quad , \quad (31)$$

since no output normalization nor dimensional analysis were adopted in the proposed model.

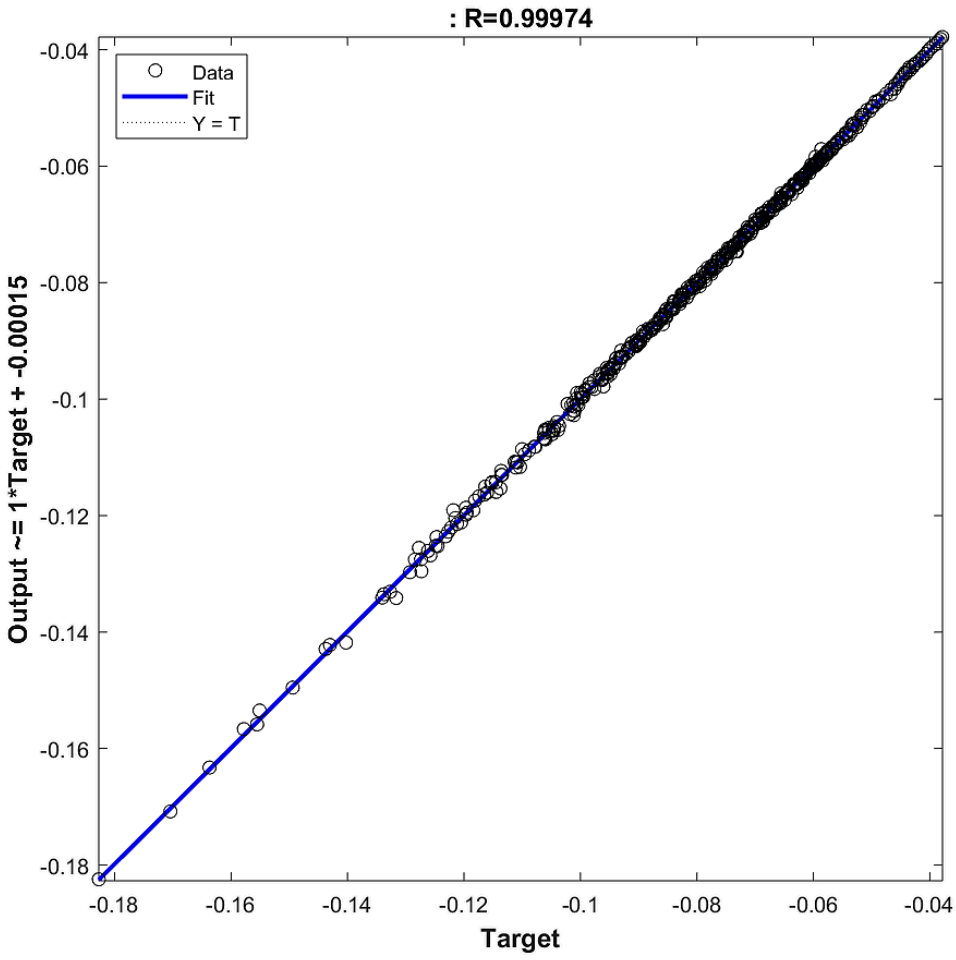

Fig. 8. Regression plot for the proposed NNC.

4.1.4 Performance Results

Finally, the results yielded by the proposed NNC for the 481-point final testing dataset (which includes the ANN learning/development counterpart), in terms of performance variables defined in sub-section 3.4, are presented in this sub-section in the form of two 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 quality is given by the Pearson Correlation Coefficient (*R*), as defined in eq. (1); and (ii) a plot (Fig. 9) indicating (for all data) the (ii_1) maximum error, (ii_2) percentage of errors larger than 3%, and (ii_3) average error (called performance).

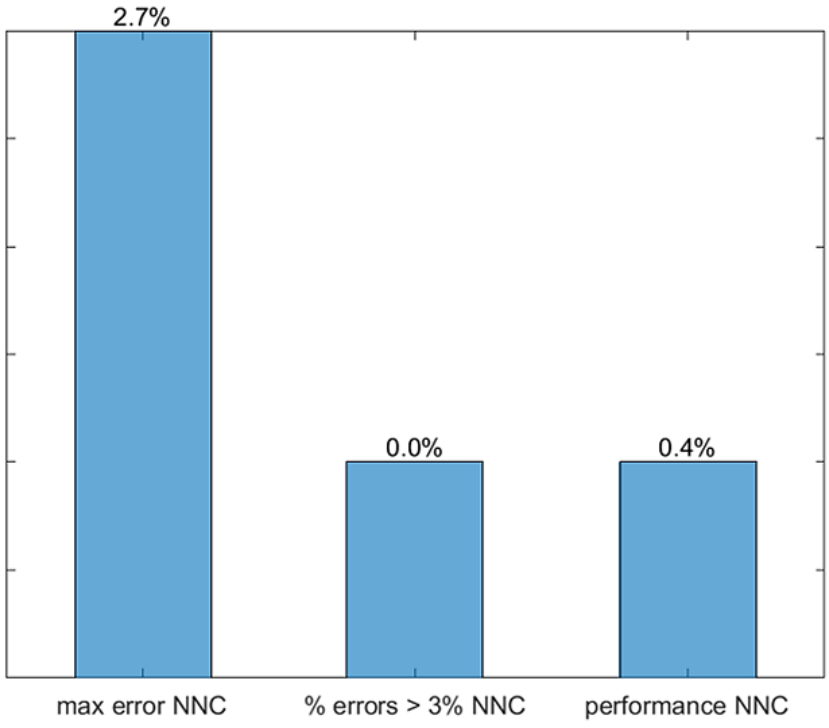

Fig. 9. Maximum and average (performance) errors for the proposed NNC.

4.2 Negative w_{max} (v =]175, 250[m/s)

ANN feature methods used in the best combo from each of the abovementioned nine parametric SAs are specified in Table 7 (numbers represent the method number as in Tables 2-4). Table 8 shows the corresponding relevant results for those combos and the 208-point final testing dataset (which includes the ANN learning/development dataset), namely (i) maximum error, (ii) percentage of errors larger than 3%, (iii) performance (all described in sub-section 3.4, 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 Table 8 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. Summing up the ANN feature combinations for all parametric SAs, a total of 204 combos were run for this problem.

The proposed model is the one, among the best ones from all parametric SAs, exhibiting the lowest maximum error (SA 9 - a Neural Network Composite (NNC)). 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 the following sub-sections. The proposed model is an NNC made of 3 ANNs with architecture RBFN and a distribution of nodes/layer given by 2-3-3-3-1 for every network. Concerning connectivity, all networks are partially-connected (see Fig. 10), and the hidden and output transfer functions are all Gaussian RBF (eq. (23)) and Hyperbolic Tangent (eq. (8)), respectively. All networks were trained using the LM algorithm. After design, the average NNC computing time concerning the presentation of a single example (including data pre/postprocessing) is 7.87E-05 s. Fig. 10

depicts a simplified scheme of each NNC network. Finally, all relevant performance results concerning the proposed NNC are illustrated in sub-section 4.2.4.

Table 7. 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	5	1	1	2	1	1	11	2	3	1	3
2	1	2	6	2	3	7	1	2	1	1	9	2	5	1	3
3	1	2	6	1	5	1	1	2	1	1	11	2	3	1	3
4	1	2	6	1	5	1	2	2	1	1	11	2	3	1	3
5	1	2	6	3	5	1	3	2	1	1	11	2	3	1	3
6	1	2	6	1	5	7	4	2	1	1	11	2	3	1	3
7	1	2	6	1	5	3	5	2	1	1	11	2	3	1	3
8	1	2	6	1	5	3	5	2	1	1	11	2	3	1	3
9	1	2	6	1	5	3	5	2	3	2	11	2	3	1	3

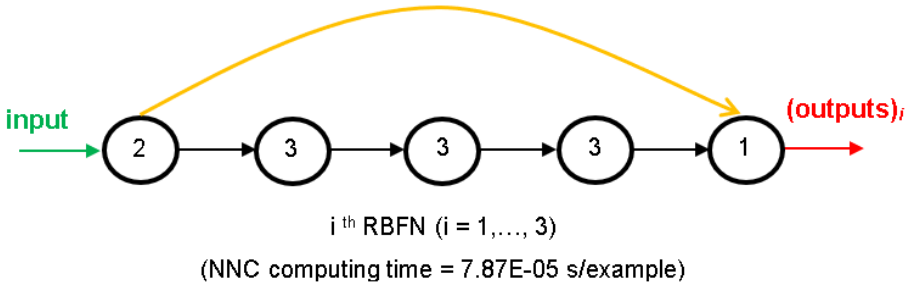

Fig. 10. Proposed NNC made of 3 partially-connected RBFNs – simplified scheme.

4.2.1 Input Data Preprocessing

For future use of the proposed NNC to simulate new data $Y_{1,sim}$ (a 2 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 is defined by the methods used for ANN features 2, 3 and 5

(respectively 2, 6 and 5 – see Table 2). Next, the necessary preprocessing to be applied to $Y_{1,sim}$ is fully described.

Table 8. Performance results for the best design from each parametric SA and for the final testing dataset (includes the ANN learning/development dataset): ANN and NNC.

SA	ANN				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	10.7	2.8	38.9	8	1.68E-04
2	102.8	16.7	69.7	50	5.50E-05
3	12.1	3.1	43.3	8	5.26E-05
4	12.0	3.1	44.7	8	4.11E-05
5	20.7	3.0	38.5	8	8.19E-05
6	22.8	3.0	40.9	8	3.84E-05
7	18.7	2.7	32.7	8	4.47E-05
8	14.2	3.0	39.4	8	5.63E-05
9	11.4	1.8	13.9	9	5.10E-05
SA	**NNC**				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	12.0	2.5	29.8	8	1.16E-04
6	8.0	1.0	6.3	8	6.63E-05
7	8.4	1.8	20.2	8	5.52E-05
8	-	-	-	-	-
9	5.2	1.1	4.8	9	7.87E-05

Dimensional Analysis and Dimensionality Reduction

Since no dimensional analysis (*d.a.*) nor dimensionality reduction (*d.r.*) were carried out, one has

$$\left\{Y_{1,sim}\right\}_{d.r.}^{after} = \left\{Y_{1,sim}\right\}_{d.a.}^{after} = Y_{1,sim} \quad . \quad (32)$$

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

$$\left\{Y_{1,sim}\right\}_{n}^{after} = \left(\left\{Y_{1,sim}\right\}_{d.r}^{after} - \text{INP}(:,1)\right) ./ \text{INP}(:,2)$$
$$\text{INP} = \begin{bmatrix} 5000 & 3171.29986868837 \\ 212.5 & 23.1146212230639 \end{bmatrix} \quad , \quad (33)$$

where one recalls that operator './' divides row i in the numerator by INP(i, 2).

4.2.2 ANN-Based Analytical Model

Once determined the preprocessed input dataset $\{Y_{1,sim}\}_n^{after}$ (a 2 x P_{sim} matrix), the next step is to present it to the proposed NNC to obtain the predicted output dataset $\{Y_{5,sim}\}_n^{after}$ (a 1 x P_{sim} vector), which will be given in the same preprocessed format of the target dataset used in learning. To convert the predicted outputs to their 'original format' (i.e., without any transformation due to normalization or dimensional analysis), some postprocessing might be needed, as described in 4.2.3. Next, the mathematical representation of the proposed NNC is given, so that any user can implement it to determine $\{Y_{5,sim}\}_n^{after}$:

$$\left\{Y_{5,sim}\right\}_{n}^{after} = \left\{Y_{5,sim}\right\}_{n(0)}^{after} + \sum_{i=1}^{2} \left\{Y_{5,sim}\right\}_{n(i)}^{after} \quad , \quad (34)$$

where (i = 0, 1, 2)

$$
\begin{aligned}
&Y_{2(i)} = \varphi_2\left(s_{(i)}\right)\\
&aux_{2(i)}(l_1, p) = \left\| \left\{Y_{1,sim}\right\}_n^{after}(:, p) - W_{1-2(i)}(:, l_1) \right\|\\
&s_{(i)} \;=\; aux_{2(i)} \text{ .x } b_{2(i)}
\end{aligned}
$$

$$
\begin{aligned}
&Y_{3(i)} = \varphi_3\left(s_{(i)}\right)\\
&aux_{3(i)}(l_1, p) = \left\| Y_{2(i)}(:, p) - W_{2-3(i)}(:, l_1) \right\|\\
&s_{(i)} \;=\; aux_{3(i)} \text{ .x } b_{3(i)}
\end{aligned}
$$

$$
\begin{aligned}
&Y_{4(i)} = \varphi_4\left(s_{(i)}\right)\\
&aux_{4(i)}(l_1, p) = \left\| Y_{3(i)}(:, p) - W_{3-4(i)}(:, l_1) \right\|\\
&s_{(i)} \;=\; aux_{4(i)} \text{ .x } b_{4(i)}
\end{aligned}
$$

$$
\left\{Y_{5,sim}\right\}_{n(i)}^{after} = \varphi_5\left(W_{1-5(i)}^T \left\{Y_{1,sim}\right\}_n^{after} + W_{4-5(i)}^T Y_{4(i)} + b_{5(i)}\right) \quad , \quad (35)
$$

and (i) p=1,…, P_{sim}, l_1=1,2, 3, (ii) operator '.x' multiplies every element in row s of the first array by element s of second array (a vector), yielding an array of the same size of the first, and (iii)

$$
\begin{aligned}
&\varphi_2 = \varphi_3 = \varphi_4 = \varphi(s) = \mathrm{e}^{-s^2}\\
&\varphi_5 = \varphi_5(s) = \frac{\mathrm{e}^s - \mathrm{e}^{-s}}{\mathrm{e}^s + \mathrm{e}^{-s}}
\end{aligned} \quad . \quad (36)
$$

Arrays $W_{j\text{-}s(i)}$ and $b_{s(i)}$ are stored online in Developer (2018b).

4.2.3 Output Data Postprocessing

In order to transform the output dataset obtained by the proposed NNC, $\{Y_{5,sim}\}_n^{after}$ (a 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, one has

$$Y_{5,sim} = \left\{Y_{5,sim}\right\}_n^{after} \quad , \quad (37)$$

since no output normalization nor dimensional analysis were adopted in the proposed model.

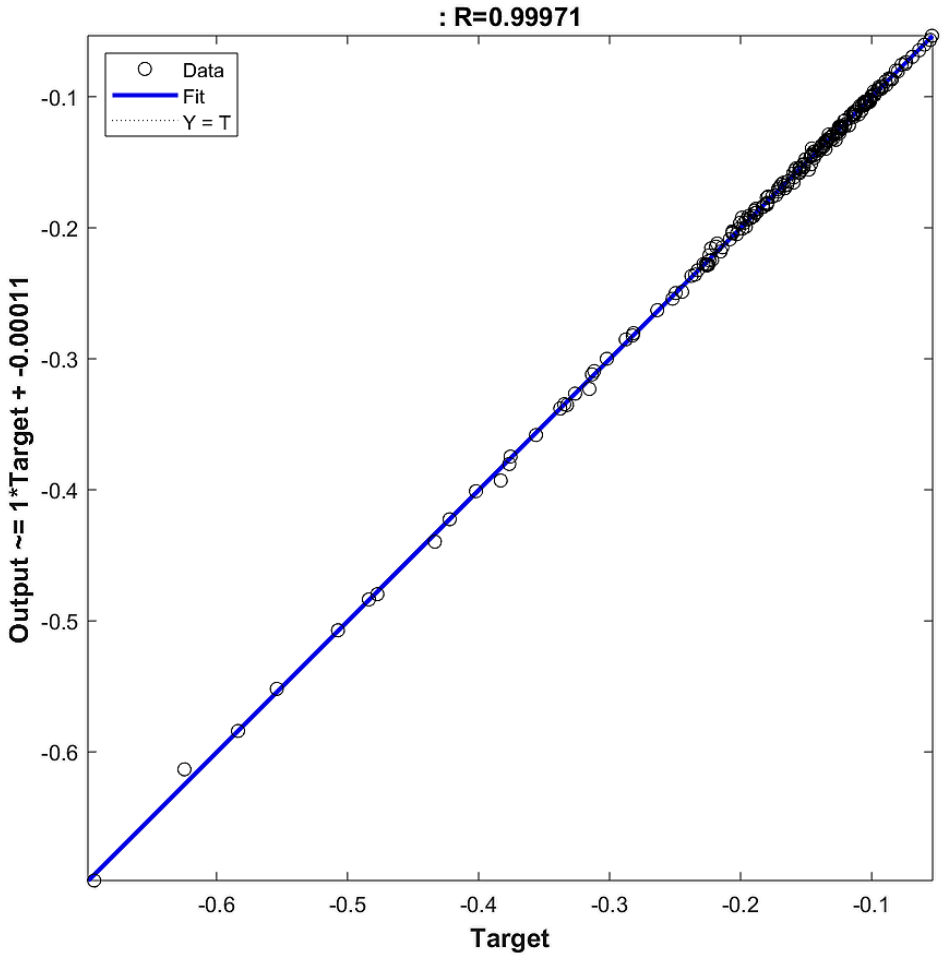

Fig. 11. Regression plot for the proposed NNC.

4.2.4 Performance Results

Finally, the results yielded by the proposed NNC for the 208-point final testing dataset (which includes the ANN learning/development counterpart), in terms of performance variables defined in sub-section 3.4, are presented in this sub-section in the form of two graphs: (i) a regression plot (Fig. 11), where network target and output data are plotted, for each data point, as *x*- and *y*-coordinates, respectively; and (ii) a plot (Fig. 12) indicating (for all data) the (ii_1) maximum error, (ii_2) percentage of errors larger than 3%, and (ii_3) average error (called performance).

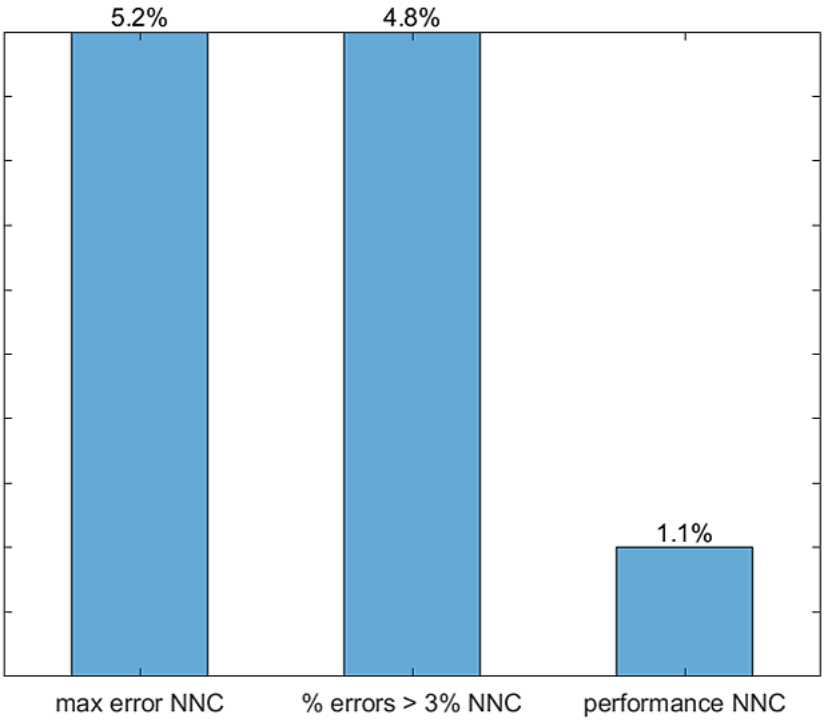

Fig. 12. Maximum and average (performance) errors for the proposed NNC.

4.3 Positive w_{max} (v = [50, 175] ∪ [250, 300] m/s)

ANN feature methods used in the best combo from each of the abovementioned nine parametric SAs are specified in Table 9 (numbers represent the method number as in Tables 2-4). Table 10 shows the corresponding relevant results for those combos and the 481-point final testing dataset (which includes the ANN learning/development dataset), namely (i) maximum error, (ii) percentage of errors larger than 3%, (iii) performance (all described in sub-section 3.4, 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 Table 10 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. Summing up the ANN feature combinations for all parametric SAs, a total of 219 combos were run for this problem.

Table 9. 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	3	1	1	1	1	1	3	2	3	1	3
2	1	2	6	2	2	7	1	2	1	1	9	2	5	1	3
3	1	2	6	3	3	1	1	1	1	1	3	2	3	1	3
4	1	2	6	3	3	1	2	1	1	1	3	2	3	1	3
5	1	2	6	1	3	1	3	1	1	1	3	2	3	1	3
6	1	2	6	4	3	7	4	1	1	1	3	2	3	1	3
7	1	2	6	4	3	7	5	1	1	1	3	2	3	1	3
8	1	2	6	4	3	7	5	1	1	1	1	2	3	1	3
9	1	2	6	4	3	7	5	1	3	3	1	2	3	1	3

The proposed model is the one, among the best ones from all parametric SAs, exhibiting the lowest maximum error (SA 9). 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 the following sub-sections. The proposed model is a single MLPN with 5 layers and a distribution of nodes/layer given by 2-3-3-3-1. Concerning connectivity, the network is fully-connected, and the hidden and output transfer functions are all Logistic (eq. (7)) and Identity (eq. (10)), 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 2.49E-05 s; Fig. 13 depicts a simplified scheme of some of network key features. Finally, all relevant performance results concerning the proposed ANN are illustrated in sub-section 4.3.4.

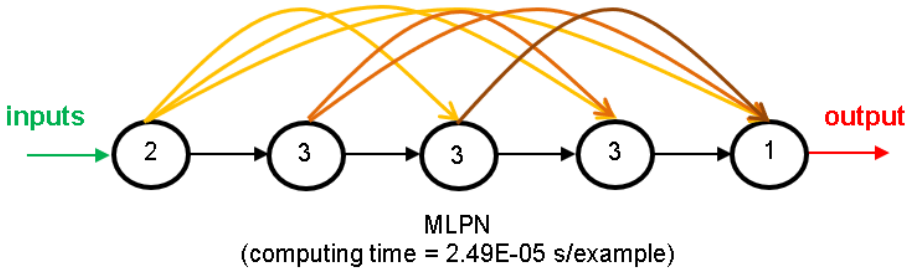

Fig. 13. Proposed 2-3-3-3-1 fully-connected MLPN– simplified scheme.

Table 10. Performance results for the best design from each parametric SA and for the final testing dataset (includes the ANN learning/development dataset): ANN and NNC.

SA	ANN				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	7.4	0.5	7.1	8	2.58E-05
2	8.2	0.7	8.5	60	3.43E-05
3	6.7	0.6	7.7	8	4.52E-05
4	6.9	0.5	7.5	8	2.70E-05
5	7.0	0.5	7.3	8	2.62E-05
6	6.2	0.5	8.3	8	2.65E-05
7	6.7	0.5	6.7	8	3.72E-05
8	6.4	0.5	7.5	8	2.91E-05
9	3.7	0.2	0.8	9	2.49E-05
SA	**NNC**				
	Max Error (%)	Performance	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)

	All Data (%)				
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	6.9	0.4	5.8	8	2.87E-05
5	-	-	-	-	-
6	3.7	0.2	0.2	8	3.29E-05
7	6.4	0.5	6.7	8	4.16E-05
8	6.2	0.5	7.1	8	2.98E-05
9	-	-	-	-	-

4.3.1 Input Data Preprocessing

For future use of the proposed ANN to simulate new data $Y_{1,sim}$ (a 2 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, 6 and 3 – see Table 2). In what follows, the necessary preprocessing to be applied to $Y_{1,sim}$ is fully described.

Dimensional Analysis and Dimensionality Reduction

Since no dimensional analysis (*d.a.*) nor dimensionality reduction (*d.r.*) were carried out, one has

$$\left\{Y_{1,sim}\right\}_{d.r.}^{after} = \left\{Y_{1,sim}\right\}_{d.a.}^{after} = Y_{1,sim} \quad . \quad (38)$$

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

$$\begin{aligned}
&\left\{Y_{1,sim}\right\}_n^{after} = \text{INP}(:,1) + rab.\text{x}\left(\left\{Y_{1,sim}\right\}_{d.r}^{after} - \text{INP}(:,3)\right)./den \\
&\text{INP} = \begin{bmatrix} -1 & 1 & 0 & 10000 \\ -1 & 1 & 50 & 300 \end{bmatrix} \\
&rab = \text{INP}(:,2) - \text{INP}(:,1) \\
&den = \text{INP}(:,4) - \text{INP}(:,3)
\end{aligned} \quad , \quad (39)$$

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 './').

4.3.2 ANN-Based Analytical Model

Once determined the preprocessed input dataset $\{Y_{1,sim}\}_n^{after}$ (a 2 x P_{sim} matrix), the next step is to present it to the proposed ANN to obtain the predicted output dataset $\{Y_{5,sim}\}_n^{after}$ (a 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), some postprocessing might be needed, as described in 4.3.3. Next, the mathematical representation of the proposed ANN is given, so that any user can implement it to determine $\{Y_{5,sim}\}_n^{after}$:

$$\begin{aligned}
Y_2 &= \varphi_2\left(W_{1-2}^T\left\{Y_{1,sim}\right\}_n^{after} + b_2\right) \\
Y_3 &= \varphi_3\left(W_{1-3}^T\left\{Y_{1,sim}\right\}_n^{after} + W_{2-3}^T Y_2 + b_3\right) \\
Y_4 &= \varphi_4\left(W_{1-4}^T\left\{Y_{1,sim}\right\}_n^{after} + W_{2-4}^T Y_2 + W_{3-4}^T Y_3 + b_4\right) \\
\left\{Y_{5,sim}\right\}_n^{after} &= \varphi_5\left(W_{1-5}^T\left\{Y_{1,sim}\right\}_n^{after} + W_{2-5}^T Y_2 + W_{3-5}^T Y_3 + W_{4-5}^T Y_4 + b_5\right)
\end{aligned} \quad , \quad (40)$$

where

$$\varphi_2 = \varphi_3 = \varphi_4 = \varphi(s) = \frac{1}{1+e^{-s}}$$
$$\varphi_5 = \varphi_5(s) = s \quad . \quad (41)$$

Arrays $W_{j\text{-}s}$ and b_s can be found online in Developer (2018c).

4.3.3 Output Data Postprocessing

In order to transform the output dataset obtained by the proposed ANN, $\{Y_{5,sim}\}_n^{after}$ (a 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, one has

$$Y_{5,sim} = \left\{Y_{5,sim}\right\}_n^{after} \quad , \quad (42)$$

since no output normalization nor dimensional analysis were adopted in the proposed model.

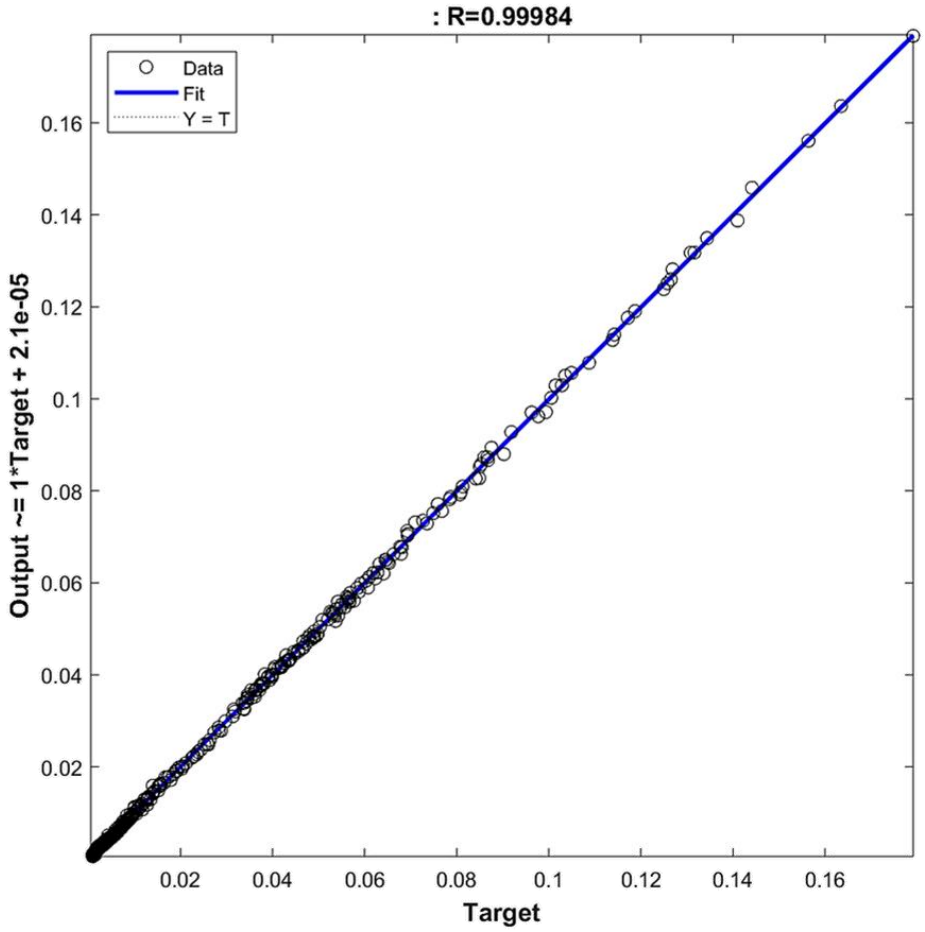

Fig. 14. Regression plot for the proposed ANN.

4.3.4 Performance Results

Finally, the results yielded by the proposed ANN for the 481-point final testing dataset (which includes the ANN learning/development counterpart), in terms of performance variables defined in sub-section 3.4, are presented in this sub-section in the form of two graphs: (i) a regression plot (Fig. 14), where network target and output data are plotted, for each data point, as *x*- and *y*- coordinates, respectively; and (ii) a plot (Fig. 15) indicating (for all data) the (ii_1) maximum error, (ii_2) percentage of errors larger than 3%, and (ii_3) average error (called performance).

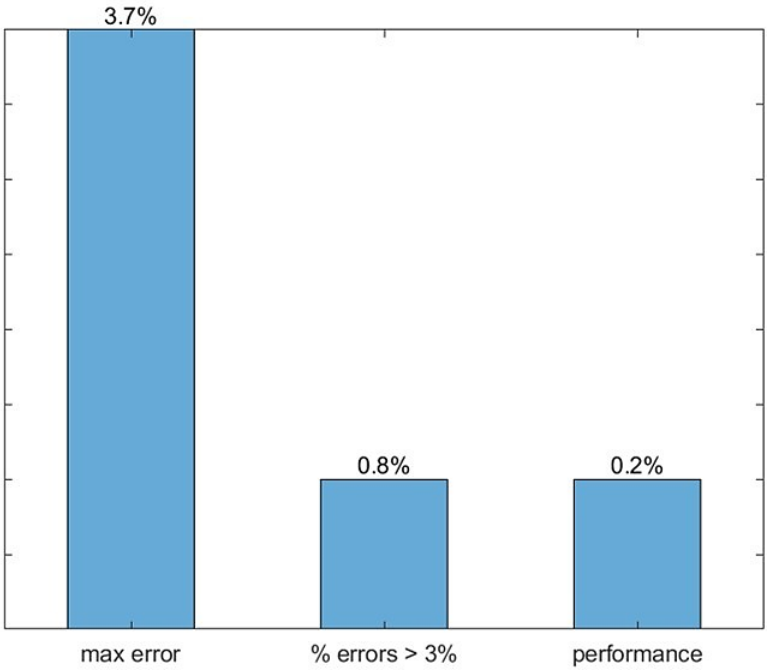

Fig. 15. Maximum and average (performance) errors for the proposed ANN.

4.4 Positive w_{max} (v =]175, 250[m/s)

ANN feature methods used in the best combo from each of the abovementioned nine parametric SAs are specified in Table 11 (numbers represent the method number as in Tables 2-4). Table 12 shows the corresponding relevant results for those combos and the 208-point final testing dataset (which includes the ANN learning/development dataset), namely (i) maximum error, (ii) percentage of errors larger than 3%, (iii) performance (all described in sub-section 3.4, 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 Table 12 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. Summing up the ANN feature combinations for all parametric SAs, a total of 219 combos were run for this problem.

Table 11. 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	5	1	1	1	1	1	3	2	3	1	3
2	1	2	6	2	5	7	1	2	1	1	9	2	5	1	3
3	1	2	6	1	5	1	1	1	1	1	3	2	3	1	3
4	1	2	6	2	5	1	2	1	1	1	3	2	3	1	3
5	1	2	6	3	5	1	3	1	1	1	3	2	3	1	3
6	1	2	6	1	5	7	4	1	1	1	3	2	3	1	3
7	1	2	6	1	5	7	5	1	1	1	3	2	3	1	3
8	1	2	6	1	5	7	5	1	1	1	1	2	3	1	3
9	1	2	6	1	5	7	5	1	3	3	1	2	3	1	3

The proposed model is the one, among the best ones from all parametric SAs, exhibiting the lowest maximum error (SA 9 - a Neural Network Composite (NNC)). 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 the following sub-sections. The proposed model is an NNC made of 4 ANNs with architecture MLPN and a distribution of nodes/layer given by 2-3-3-3-1 for every network. Concerning connectivity, all networks are fully-connected, and the hidden and output transfer functions are all Logistic (eq. (7)) and Identity (eq. (10)), respectively. All networks were trained using the LM algorithm. After design, the average NNC computing time concerning the presentation of a single example (including data pre/postprocessing) is 4.08E-05 s; Fig. 16 depicts a simplified scheme of each NNC network. Finally, all relevant performance results concerning the proposed NNC are illustrated in sub-section 4.4.4.

Table 12. Performance results for the best design from each parametric SA and for the final testing dataset (includes the ANN learning/development dataset): ANN and NNC.

SA	ANN				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	20.3	2.5	31.3	8	1.16E-04
2	135.1	10.3	61.5	50	4.17E-05
3	20.3	2.5	31.3	8	4.16E-05
4	23.7	2.6	30.8	8	4.23E-05
5	19.4	2.6	29.8	8	4.16E-05
6	20.3	2.9	35.1	8	3.50E-05
7	18.3	2.7	32.7	8	3.66E-05
8	21.3	2.7	34.1	8	5.32E-05
9	8.8	0.8	5.8	9	3.55E-05
SA	**NNC**				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	-	-	-	-	-
6	18.9	0.9	5.8	8	4.13E-05
7	19.1	1.2	10.6	8	4.16E-05
8	18.4	1.5	17.3	8	5.70E-05
9	5.4	0.4	1.9	9	4.08E-05

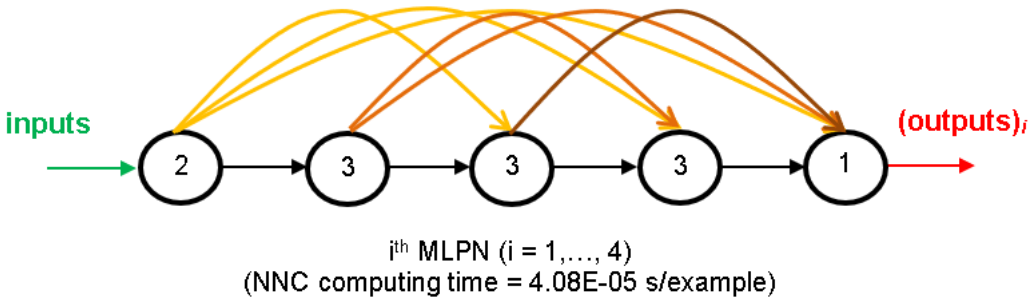

Fig. 16. Proposed NNC made of 4 fully-connected MLPNs – simplified scheme.

4.4.1 Input Data Preprocessing

For future use of the proposed NNC to simulate new data $Y_{1,sim}$ (a 2 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, 6 and 5 – see Table 2). Next, the necessary preprocessing to be applied to $Y_{1,sim}$ is fully described.

Dimensional Analysis and Dimensionality Reduction

Since no dimensional analysis (*d.a.*) nor dimensionality reduction (*d.r.*) were carried out, one has

$$\left\{Y_{1,sim}\right\}_{d.r.}^{after} = \left\{Y_{1,sim}\right\}_{d.a.}^{after} = Y_{1,sim} \quad . \quad (43)$$

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

$$\left\{Y_{1,sim}\right\}_{n}^{after} = \left(\left\{Y_{1,sim}\right\}_{d.r}^{after} - \text{INP}(:,1)\right) ./ \text{INP}(:,2)$$
$$\text{INP} = \begin{bmatrix} 5000 & 3171.90409844877 \\ 214.666666666667 & 22.2385441413878 \end{bmatrix} \quad , \quad (44)$$

where one recalls that operator './' divides row i in the numerator by INP(i, 2).

4.4.2 ANN-Based Analytical Model

Once determined the preprocessed input dataset $\{Y_{1,sim}\}_n^{after}$ (a 2 x P_{sim} matrix), the next step is to present it to the proposed NNC to obtain the predicted output dataset $\{Y_{5,sim}\}_n^{after}$ (a 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), some postprocessing might be needed, as described in 4.4.3. Next, the mathematical representation of the proposed NNC is given, so that any user can implement it to determine $\{Y_{5,sim}\}_n^{after}$:

$$\left\{Y_{5,sim}\right\}_n^{after} = \left\{Y_{5,sim}\right\}_{n(0)}^{after} + \sum_{i=1}^{3} \left\{Y_{5,sim}\right\}_{n(i)}^{after} \quad , \quad (45)$$

being

$$\begin{aligned}
Y_{2(i)} &= \varphi_2\left(W_{1-2(i)}^T \left\{Y_{1,sim}\right\}_n^{after} + b_{2(i)}\right) \\
Y_{3(i)} &= \varphi_3\left(W_{1-3(i)}^T \left\{Y_{1,sim}\right\}_n^{after} + W_{2-3(i)}^T Y_{2(i)} + b_{3(i)}\right) \\
Y_{4(i)} &= \varphi_4\left(W_{1-4(i)}^T \left\{Y_{1,sim}\right\}_n^{after} + W_{2-4(i)}^T Y_{2(i)} + W_{3-4(i)}^T Y_{3(i)} + b_{4(i)}\right) \\
\left\{Y_{5,sim}\right\}_{n(i)}^{after} &= \varphi_5\left(W_{1-5(i)}^T \left\{Y_{1,sim}\right\}_n^{after} + W_{2-5(i)}^T Y_{2(i)} + W_{3-5(i)}^T Y_{3(i)} + W_{4-5(i)}^T Y_{4(i)} + b_{5(i)}\right)
\end{aligned} \quad , \quad (46)$$

where $i = 0,\ldots, 3$ and

$$\begin{aligned}
\varphi_2 = \varphi_3 = \varphi_4 = \varphi(s) &= \frac{1}{1+e^{-s}} \\
\varphi_5 = \varphi_5(s) &= s
\end{aligned} \quad . \quad (47)$$

Arrays $W_{j\text{-}s(i)}$ and $b_{s(i)}$ are stored online in Developer (2018d).

4.4.3 Output Data Postprocessing

In order to transform the output dataset obtained by the proposed NNC, $\{Y_{5,sim}\}_n^{after}$ (a 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, one has

$$Y_{5,sim} = \left\{ Y_{5,sim} \right\}_{n}^{after} \quad , \quad (48)$$

since no output normalization nor dimensional analysis were adopted in the proposed model.

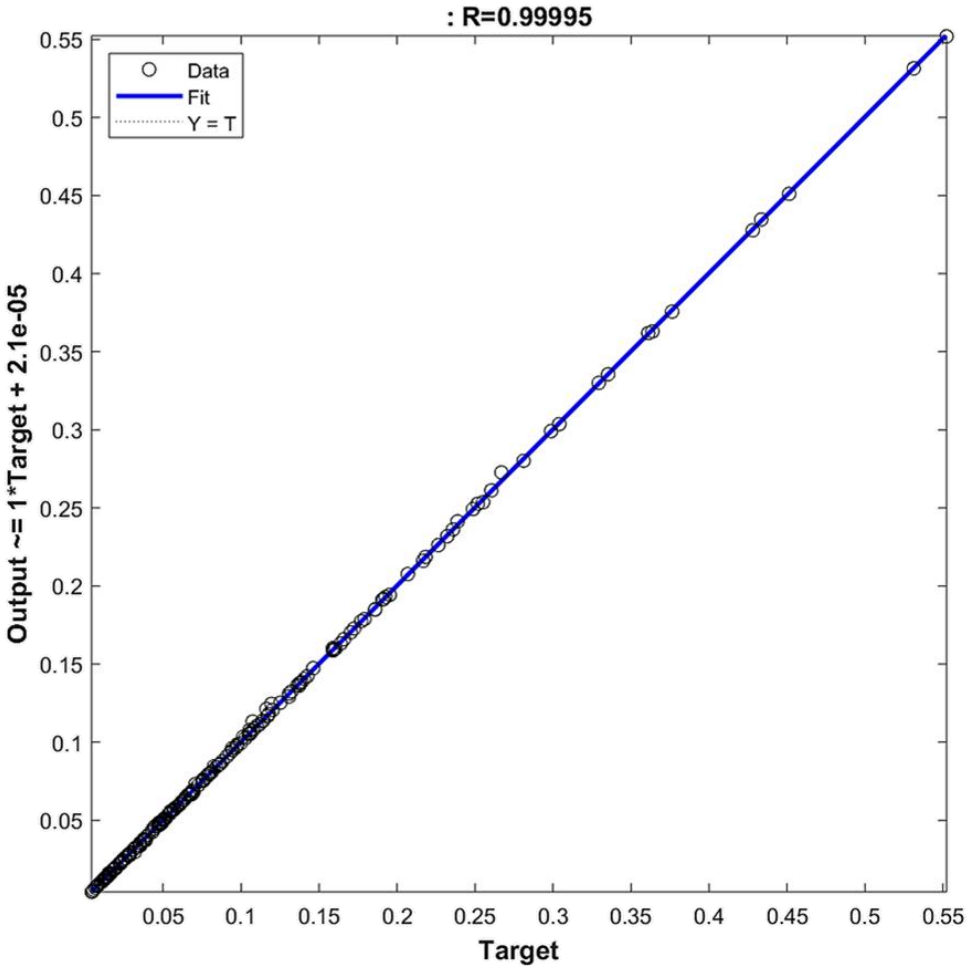

Fig. 17. Regression plot for the proposed NNC.

4.4.4 Performance Results

Finally, the results yielded by the proposed NNC for the 208-point final testing dataset (which includes the ANN learning/development counterpart), in terms of performance variables defined in sub-section 3.4, are presented in this sub-section in the form of two graphs: (i) a regression

plot (Fig. 17), where network target and output data are plotted, for each data point, as *x*- and *y*-coordinates, respectively; and (ii) a plot (Fig. 18) indicating (for all data) the (ii_1) maximum error, (ii_2) percentage of errors larger than 3%, and (ii_3) average error (called performance).

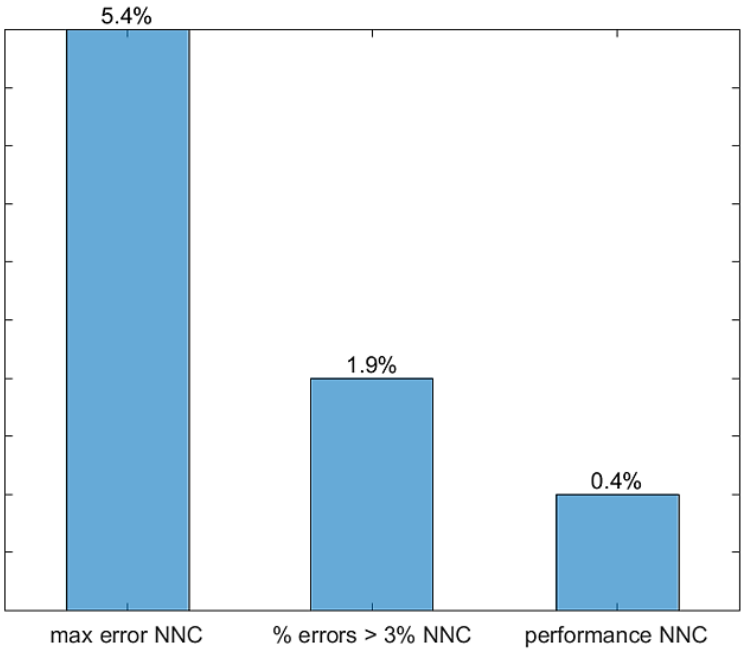

Fig. 18. Maximum and average (performance) errors for the proposed NNC.

5. Critical velocities and maximum displacements predictions

Eleven pairs of curves were obtained as output of the ANN-based models described in sub-sections 4.1-4.4. Each pair presents the maximum negative (downward) and positive (upward) displacement predictions as function of load velocity (from 50 to 300 m/s in intervals of 5 m/s) for different values of the maximal distributed friction force f_u, as depicted in Fig. 19 (two plots are presented for the sake of legibility). Note that the classic Winkler foundation case corresponds to the frictionless case ($f_u = 0$). Comparing the homologous curves in Fig.19 and

Fig. 8(a) in Toscano et al. (2018), a very good visual agreement between the ANN-based predictions and the results of the mechanical model is observed. For a precise comparison, the obtained target (FE-based) and output (ANN-based) values can be found in Authors (2018).

The set of curves shows that the increase of the maximum frictional force per unit length (f_u) leads, as expected, to the reduction of the displacement peaks. The existence of a critical velocity, that is, a velocity that induces the beam's highest displacements, is also clear in Fig. 19. It is observed that, for small values of f_u, the value of the critical velocity is just slightly affected, whereas for larger frictional forces that value clearly rises.

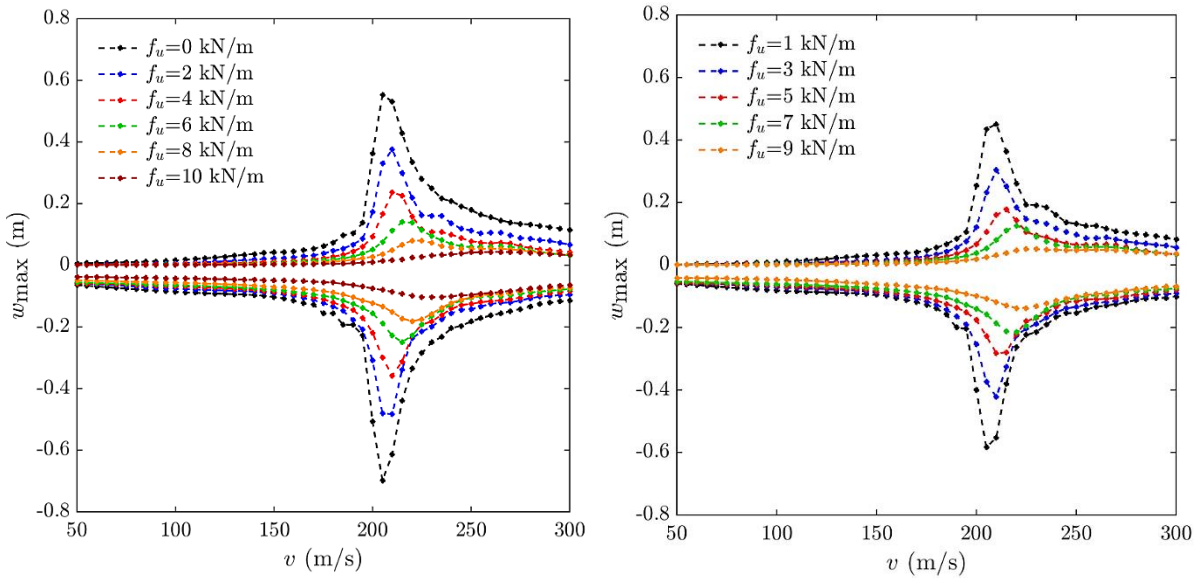

Fig. 19. Maximum upward and downward ANN-based displacements for a frictionally damped beam as function of the load velocity: (a) f_u = 0, 2, 4, 6, 8 and 10 kN/m, (b) f_u = 1, 3, 5, 7 and 9 kN/m.

5. Discussion

In future publications it will be guaranteed that the validation and testing data subsets will be composed only by points where at least one variable (does not have to be the same for all)

takes a value not taken in the training subset by that same variable. Based on very recent empirical conclusions by Abambres, the author believes it will lead to more robust ANN-based analytical models concerning their generalization ability (i.e. prediction accuracy for any data point within the variable ranges of the design data).

6. Final Remarks

This paper demonstrated the potential of Artificial Neural Networks (ANN) to effectively predict the maximum displacements and the critical velocities in railway beams under moving loads. Four ANN-based models were proposed, one per load velocity range ([50, 175] ∪ [250, 300] m/s;]175, 250[m/s) and per displacement type (upward or downward). Each model is function of two independent variables, a frictional parameter and the load velocity. Among all models and the 663 data points used, a maximum error of 5.4 % was obtained when comparing the ANN- and FE-based solutions. Whereas the latter involves an average computing time per data point of thousands of seconds, the former does not even need a millisecond. More versatile ANN-based analytical models for the same type of problem may follow from this study by including more independent variables, such as the foundation stiffness modulus, the applied load magnitude, and the geometrical/mechanical properties of the railway beam.

Regardless the high quality of the predictions yielded by the proposed model, the reader should not **blindly** accept it 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 meanwhile are part of a learning process towards excellence.

Acknowledgements

There are no conflicts of interest to disclose.

Author Contributions

Sections 1 and 2: Rita Corrêa, António Pinto da Costa, Fernando Simões; Sections 3, 4 and 5: Miguel Abambres; Section 6 and final review: All authors.

References

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.

Authors (2018). ANN development + final testing datasets [Data set], downloadable.

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.

Castro Jorge P, Pinto da Costa A, Simões FMF (2015). Finite element dynamic analysis of finite beams on a bilinear foundation under a moving load. Journal of Sound and Vibration, 346(June):328–44, doi: 10.1016/j.jsv.2014.12.044.

Castro Jorge P, Simões FMF, Pinto da Costa A (2015). Dynamics of beams on non-uniform nonlinear foundations subjected to moving loads. Computers and Structures, 148(February):26–34, doi: 10.1016/j.compstruc.2014.11.002.

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.

Developer (2018a). Negative wmax (v = [50, 175] ∪ [250, 300] m/s) [Data set], downloadable

Developer (2018b). Negative wmax (v =]175, 250[m/s) [Data set], downloadable

Developer (2018c). Positive wmax (v = [50, 175] ∪ [250, 300] m/s) [Data set], downloadable

Developer (2018d). Positive wmax (v =]175, 250[m/s) [Data set], downloadable

Dimitrovová Z, Rodrigues AFS (2012). Critical velocity of a uniformly moving load. Advances in Engineering Software, 50(August):44–56, doi: 10.1016/j.advengsoft.2012.02.011.

Flood I (2008). Towards the next generation of artificial neural networks for civil engineering. Advanced Engineering Informatics, 228(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.

Frýba L (1972). Vibration of solids and structures under moving loads. Groningen: Noordhoff International Publishing.

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.

Glocker C (2001). Set-Values force laws. Lecture notes in applied and computational mechanics. Berlin, Heidelberg: Springer, ISBN 978-3-540-41436-0.

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.

Jean M (1999). The non-smooth contact dynamics method. Computer Methods in Applied Mechanics and Engineering, 177(3-4):235–57, doi: 10.1016/S0045-7825(98)00383-1.

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.

Kaynia A, Madshus C, Zackrisson P (2000). Ground vibration from high-speed trains: pre- diction and countermeasure. Journal of Geotechnical and Geoenvironmental Engineering, 126(6):531–7.

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

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.

Madshus C, Kaynia A (2000). High-speed railway lines on soft ground: dynamic behaviour at critical train speed. Journal of Sound and Vibration, 231(3):689–701, doi: 10.1006/jsvi.1999.2647.

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

Moreau J (1994). Some numerical methods in multibody dynamics: application to granular materials. European Journal of Mechanics A/Solids, 13(4):94–114.

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.

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, doi:10.6084/m9.figshare.6962873.

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

Studer C (2009). Numerics of unilateral contacts and friction – Lecture notes in applied and computational mechanics. Berlin, Heidelberg: Springer, ISBN 978-3-642-01099-6.

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.

Toscano Corrêa R, Pinto da Costa A, Simões FMF (2018). Finite element modelling of a rail resting on a Winkler-Coulomb foundation and subjected to a moving concentrated load. International Journal of Mechanical Sciences, 140(May):432-45, doi: 10.1016/j.ijmecsci.2018.03.022.

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, 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