DOI: http://doi.org/10.31224/osf.io/rghpf

Potential of Neural Networks for Structural Damage Localization

Miguel Abambres [a*], Marília Marcy [b], Graciela Doz [c]

[a*] *Research & Development, Abambres' Lab, 1600-275 Lisbon, Portugal; abambres@netcabo.pt; ORCID: 0000-0003-4107-8501*

[b] *Dep of Civil and Environmental Eng, Federal Univ. of Campina Grande, Campina Grande, Brazil; ORCID: 0000-0001-5425-1751*

[c] *Dep of Civil and Environmental Eng, University of Brasília, Brasília, Brazil; ORCID: 0000-0001-6428-4012*

Abstract

Fabrication technology and structural engineering states-of-art have led to a growing use of slender structures, making them more susceptible to static and dynamic actions that may lead to some sort of damage. In this context, regular inspections and evaluations are necessary to detect and predict structural damage and establish maintenance actions able to guarantee structural safety and durability with minimal cost. However, these procedures are traditionally quite time-consuming and costly, and techniques allowing a more effective damage detection are necessary. This paper assesses the potential of Artificial Neural Network (ANN) models in the prediction of damage localization in structural members, as function of their dynamic properties – the three first natural frequencies are used. Based on 64 numerical examples from damaged (mostly) and undamaged steel channel beams, an ANN-based analytical model is proposed as a highly accurate and efficient damage localization estimator. The proposed model yielded maximum errors of 0.2 and 0.7 % concerning 64 numerical and 3 experimental data points, respectively. Due to the high-quality of results, authors' next step is the application of similar approaches to entire structures, based on much larger datasets.

Keywords: Structural Health Monitoring; Damage Localization; Steel Beams; Dynamic Properties; Natural Frequencies; Artificial Neural Networks.

1. Introduction

Fabrication technology and structural engineering states-of-art have led to a growing use of slender structures in construction industry. Those structures (or structural members) are more

DOI: http://doi.org/10.31224/osf.io/rghpf

susceptible to static and dynamic actions that may lead to damage and/or excessive vibration. In this context, regular inspections and evaluations are necessary to detect and predict structural damage and establish maintenance actions able to guarantee structural safety and durability with minimal cost. However, these procedures are traditionally quite time-consuming and costly. Thus, techniques allowing a more efficient and less resource-dependent damage detection are in high demand and will contribute to a more sustainable built environment.

In recent years, several authors (e.g., Jin et. al 2016, Onur and Abdeljaber 2015, Nguyen et. al 2015) have concluded that structural damage detection is a problem of pattern recognition, in which a classification is made as function of physical properties of a system. Within machine learning, several types of Artificial Neural Networks (ANN) (e.g. feedforward nets, self-organizing maps, learning vector quantization) can become a quite effective damage detection tool when used in conjunction with the dynamic properties of a system (e.g., Chengyin et al. 2014, Meruane and Mahu 2014) – note that nowadays is quite straight forward the accurate estimation of important dynamic properties (e.g., natural frequencies) of (possibly damaged) built structural systems (by means of accelerometers and/or other simple decices, and existing software – e.g., ARTeMIS Modal 4.0 (SVS 2018)). According to Bandara et al. (2013) and Ahmed (2016), a clear challenge concerning ANNs is the fact that they typically need structural data of both damaged and intact structures to be able to classify satisfactorily. If the structure is not considered damaged in its current state, the information regarding the damaged state will be unavailable unless detailed

DOI: http://doi.org/10.31224/osf.io/rghpf

structural models are used to generate this information, such as numerical ones based on the Finite Element Method (FEM).

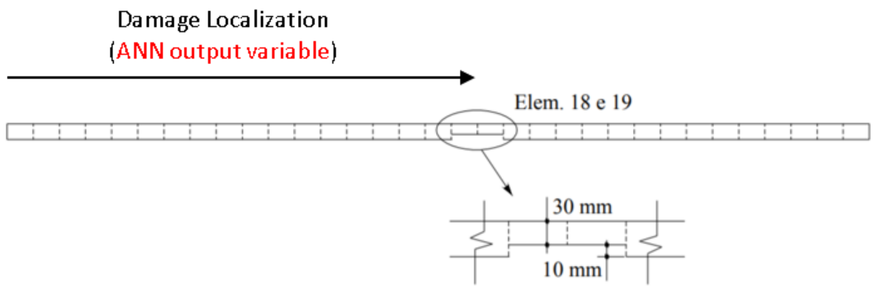
Several authors have published the application of machine learning for damage characterization in structural members (e.g., Vakil-Baghmisheh et al. 2008, Aydin and Kisi 2015, Kourehli 2015, Nazarko and Ziemianski 2017). Nonetheless, none of those studies employed exactly the same structure and input/output variables considered in this work. Moreover, the accuracy provided by those solutions are typically insufficient (maximum error for all data points > 5%) for what the authors of this paper consider to be acceptable (safe) in structural engineering practice. Thus, this paper primarily aims to assess the potential of ANN-based models in the prediction of damage localization in structural members, as function of their dynamic properties – the three first natural frequencies are used in this work. Based on numerical data from damaged (mostly) and undamaged steel channel beams, an ANN-based analytical model is proposed and tested for both numerical and experimental data. Once proved that the approach taken works well for structural members, authors' next step (in the very near future) is to apply similar procedures to entire bridge or building structures.

2. Data Gathering

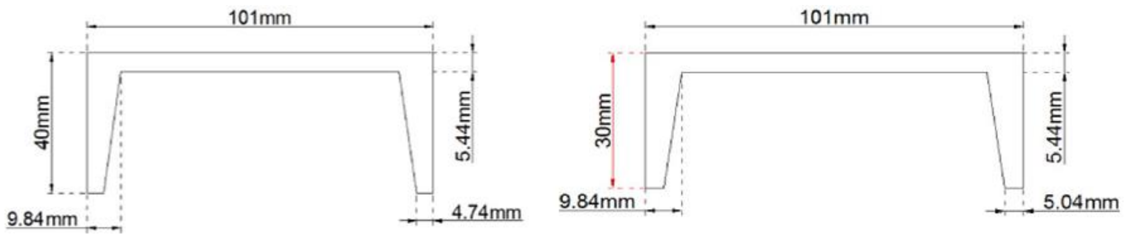
Inspired by the experimental research of Brasiliano (2005), who assessed the effect of structural damage on natural (free vibration) frequency values, the data used for the present investigation concerns damaged (mostly) and undamaged ASTM A36 steel channel beams (U101.6x4.67,

DOI: http://doi.org/10.31224/osf.io/rghpf

Gerdau 2018) with a length of 2.155 m and free-free boundary conditions. Sixty-four distinct beams (also called examples or data points in this manuscript) were simulated in ANSYS (ANSYS, Inc 2018) FEA software to obtain a 3-input and 1-output dataset for ANN design. The three first natural frequencies (Hz) of the beam are the input (independent) variables – see Tab. 1, whereas the damage location is the output (dependent) variable. The latter is given by the longitudinal distance (m) from beam's edge to the mid-point of local cross-section reduction that defines the damage (see Fig. 1(a)). For the 13 undamaged beams, the damage location adopted is non-null, randomly taken below 0.005 m, an approach typically providing better ANN-based approximations, according to authors' experience.



(a)

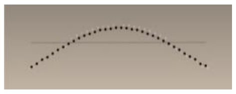
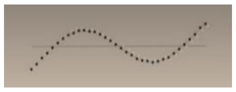
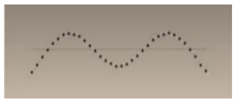


(b)

Fig. 1. Damaged beam tested by Brasiliano (2005): (a) experimental layout and damage location details, and (b) undamaged and damaged cross-sections.

DOI: http://doi.org/10.31224/osf.io/rghpf

Tab. 1. Three first natural frequencies (ANN input variables): numerical vs. test (Brasiliano 2005) results.

Vibration Mode	Undamaged Beam			Damaged Beam		
	Test (Hz)	FEA (Hz)	Error FEA vs Test (%)	Test (Hz)	FEA (Hz)	Error FEA vs Test (%)
	43.66	42.54	2.6	39.59	41.14	3.9
	120.11	117.04	2.6	117.31	117.19	0.1
	235.01	229.00	2.6	222.88	222.22	0.3

Timoshenko beam FEs of type BEAM188 (ANSYS, Inc 2018), characterized by six degrees of freedom per node, were employed in all numerical models. For validation purposes, the first two models were used to predict the three first natural frequencies of two beams tested by Brasiliano (2005) (also reported in Marcy et al. 2014). These beams are characterized by the material and geometrical properties mentioned before, being one undamaged/intact and the other not. The latter was divided into 33 equal longitudinal elements and a 10 mm reduction of its cross-section (shortening of both flanges) was performed in elements 18 and 19, as illustrated in Fig. 1. Tab. 1 presents the validation results in terms of natural frequencies, as well as the corresponding numerical modal shapes. The maximum error of 3.9 % indicates the suitability of the FE model for the present study. Once validated the numerical model, 50 other damage scenarios were simulated, varying damage extent and/or location.

The last 12 models were made without damage but under different temperatures from -5 to 40 degrees Celsius. Considering a room temperature of 22 °C, distinct Young moduli were adopted as proposed

DOI: http://doi.org/10.31224/osf.io/rghpf

by Callister and Rethwish (2009). The dataset used in ANN design can be found online in Authors (2018a). Next section provides all details concerning the ANN formulation, analyses and results.

3. Artificial Neural Networks

3.1 Introduction

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

DOI: http://doi.org/10.31224/osf.io/rghpf

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

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

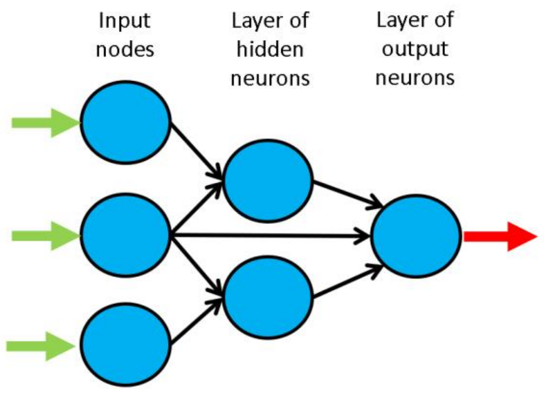


Fig. 2. Example of a feedforward neural network.

DOI: http://doi.org/10.31224/osf.io/rghpf

3.2 Learning

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

DOI: http://doi.org/10.31224/osf.io/rghpf

learning too many input-target examples suffering from data noise, since the network might learn some of its features, which do not belong to the underlying function being modelled (Haykin 2009).

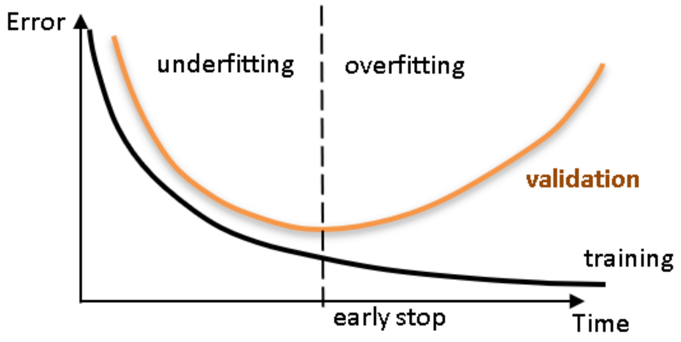


Fig. 3. Cross-validation – assessing network's generalization ability.

3.3 Implemented ANN features

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

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

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.

DOI: http://doi.org/10.31224/osf.io/rghpf

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

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

3.3.2 Dimensional Analysis (feature 2)

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

3.3.3 Input Dimensionality Reduction (feature 3)

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

DOI: http://doi.org/10.31224/osf.io/rghpf

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), R_{XY} is defined by

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

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

Auto-Encoder

This feature extraction technique uses itself a 3-layer feedforward ANN called auto-encoder (AE). After training, the hidden layer output (y_{2p}) for the presentation of each problem's input pattern (y_{1p}) is a compressed vector (Q_2 x 1) that can be used to replace the original input layer by a (much)

DOI: http://doi.org/10.31224/osf.io/rghpf

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

FEATURE METHOD	F6	F7	F8	F9	F10
	Output Transfer	Output Normalization	Net Architecture	Hidden Layers	Connectivity
1	Logistic	Lin [a, b] = 0.7[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	-	-	-	-

smaller one, thus reducing the size of the ANN model. In this work, Q_2=*round*(Q_1/2) was adopted, being *round* a function that rounds the argument to the nearest integer. The implemented AE was trained using the 'trainAutoencoder(…)' function from MATLAB's neural net toolbox. In order to select the best AE, 40 AEs were simulated, and their performance compared by means of the performance variable defined in sub-section 3.4. Each AE considered distinct (random) initialization parameters, half of the models used the 'logsig' hidden transfer functions, and the other half used the 'satlin' counterpart, being the identity function the common option for the output activation. In each AE, the maximum number of epochs – number of times the whole training dataset is presented to the network during learning, was defined (regardless the amount of data) by

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

DOI: http://doi.org/10.31224/osf.io/rghpf

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(…)'.

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

FEATURE METHOD	F11	F12	F13	F14	F15
	Hidden Transfer	Parameter Initialization	Learning Algorithm	Performance Improvement	Training Mode
1	Logistic	Midpoint (W) + Rands (b)	BP	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	-	-	-	-

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).

DOI: http://doi.org/10.31224/osf.io/rghpf

3.3.4 Training, Validation and Testing Datasets (feature 4)

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

1) 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, providing 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.

DOI: http://doi.org/10.31224/osf.io/rghpf

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.

DOI: http://doi.org/10.31224/osf.io/rghpf

Nonlinear

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

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

where (i) $Y_1(i, j)$ is the non-normalized value of input variable i for pattern j, (ii) t is the number of digits in the integer part of $Y_1(i, j)$, (iii) $sign(\ldots)$ 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.

DOI: http://doi.org/10.31224/osf.io/rghpf

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

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

Hyperbolic Tang

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

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

Bilinear

The implemented Bilinear function is defined as

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

Identity

The Identity activation is often employed in output neurons, reading

$$\varphi(s) = s \quad . \tag{10}$$

DOI: http://doi.org/10.31224/osf.io/rghpf

3.3.7 Output Normalization (feature 7)

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

3.3.8 Network Architecture (feature 8)

Multi-Layer Perceptron Network (MLPN)

This is a feedforward ANN exhibiting at least one hidden layer. Fig. 2 depicts a 3-2-1 MLPN (3 input nodes, 2 hidden neurons and 1 output neuron), where units in each layer link 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

DOI: http://doi.org/10.31224/osf.io/rghpf

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 MLPN where connections across layers are allowed, which usually lead to smaller networks (less neurons).

Fig. 4 represents a generic MLFN composed of *L* layers, where *l* (*l* = 1,…, *L*) is a generic layer and "*ql*" a generic node, being *q = 1,…, 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 (*l* = 2,…, *L*)

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

where φ_l is the transfer function used for all neurons in layer *l*.

DOI: http://doi.org/10.31224/osf.io/rghpf

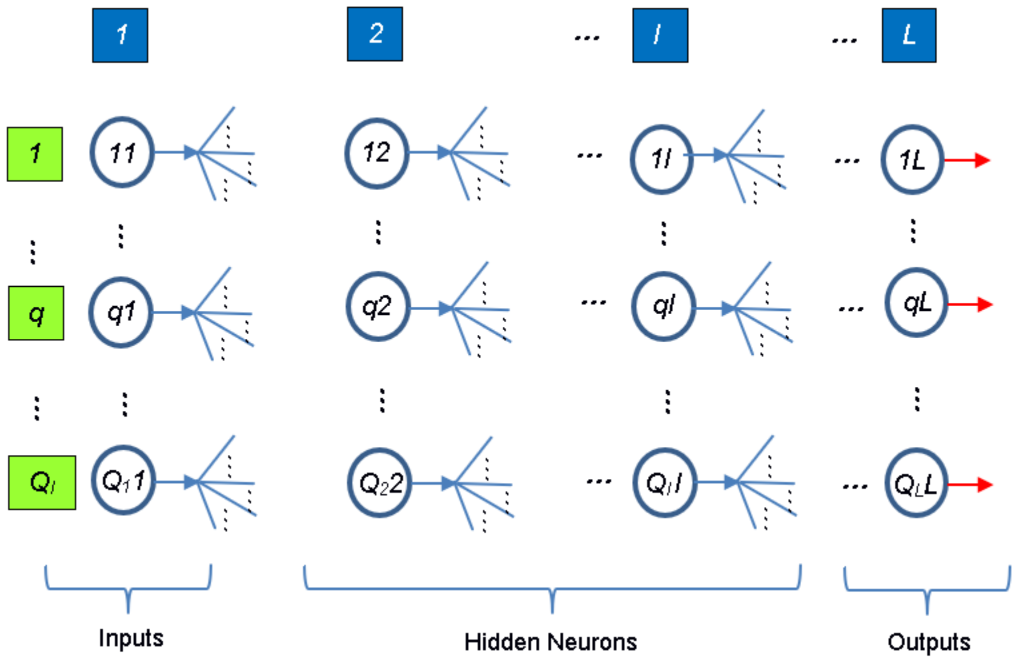


Fig. 4. Generic multi-layer feedforward network.

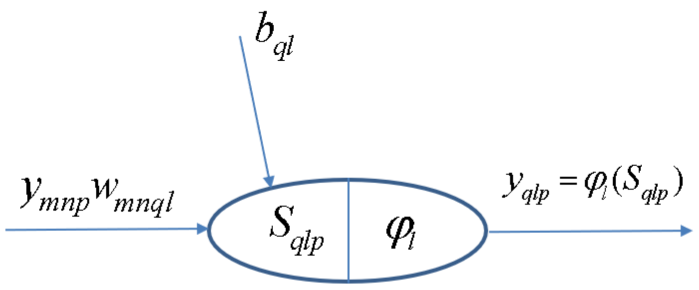


Fig. 5. Generic neuron placed anywhere in the MLPN of Fig. 4 (l = 2,…, L).

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 "$l_1 l_2$" ($l_1 = 1,\ldots,Q_{l2}$, $l_2 = 2,\ldots, L\text{-}1$) presented in Fig. 6. In this model, (i) $v_{l_1 l_2 p}$ and $\xi_{l_1 l_2}$ (called RBF center) are vectors of the same size ($\xi_{z l_1 l_2}$ denotes de z component of vector $\xi_{l_1 l_2}$, and it is a network unknown), being the former associated to the presentation of data pattern p, (ii) $\sigma_{l_1 l_2}$ 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) $y_{l_1 l_2 p}$ is neuron's output when pattern p is presented to the network. In ANNs not involving learning algorithms 1-3 in Tab. 4, vectors $v_{l_1 l_2 p}$ 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)

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

or

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

and

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

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

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

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.

DOI: http://doi.org/10.31224/osf.io/rghpf

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

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

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

3.3.10 Connectivity (feature 10)

For this ANN feature, three methods were implemented, namely (i) adjacent layers – only connections between adjacent layers are made possible, (ii) adjacent layers + input-output – only connections between (ii_1) adjacent and (ii_2) input and output layers are allowed, and (iii) fully-connected (all possible feedforward connections).

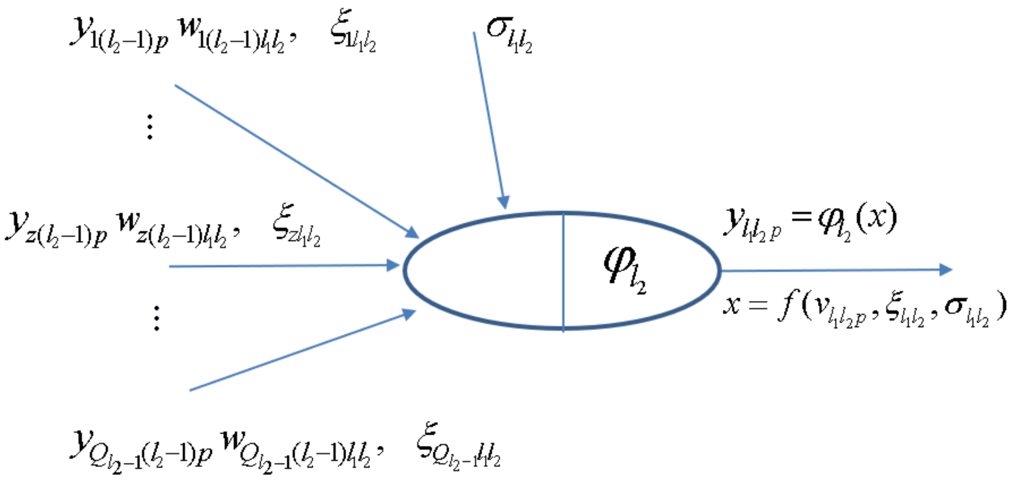

$y_{1(l_2-1)p}\ w_{1(l_2-1)l_1l_2},\ \xi_{1l_1l_2}$
$\sigma_{l_1l_2}$
$y_{z(l_2-1)p}\ w_{z(l_2-1)l_1l_2},\ \xi_{zl_1l_2}$
φ_{l_2}
$y_{l_1l_2p} = \varphi_{l_2}(x)$
$x = f(v_{l_1l_2p}, \xi_{l_1l_2}, \sigma_{l_1l_2})$
$y_{Q_{l_2-1}(l_2-1)p}\ w_{Q_{l_2-1}(l_2-1)l_1l_2},\ \xi_{Q_{l_2-1}l_1l_2}$

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

DOI: http://doi.org/10.31224/osf.io/rghpf

3.3.11 Hidden Transfer Functions (feature 11)

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

Identity-Logistic

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

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

Bipolar

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

DOI: http://doi.org/10.31224/osf.io/rghpf

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

Positive Saturating Linear

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

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

Sinusoid

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

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

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

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

DOI: http://doi.org/10.31224/osf.io/rghpf

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

$$\varphi_{l_2}(s) = e^{-0.5s}, \quad s = \left\| v_{l_1 l_2 p} - \xi_{l_1 l_2} \right\|^2 \Big/ \sigma_{l_1 l_2}^2 \qquad , \quad (21)$$

(iii) the Multiquadratic function is given by

$$\varphi_{l_2}(s) = \sqrt{s}, \quad s = \left\| v_{l_1 l_2 p} - \xi_{l_1 l_2} \right\|^2 + \sigma_{l_1 l_2}^2 \qquad , \quad (22)$$

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

$$\varphi_{l_2}(s) = e^{-s^2}, \quad s = \left\| v_{l_1 l_2 p} - \xi_{l_1 l_2} \right\| \sigma_{l_1 l_2} \qquad , \quad (23)$$

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 [-lim, lim]

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

DOI: http://doi.org/10.31224/osf.io/rghpf

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

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, *lim* is always taken as 0.5.

SVD

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

Mini-Batch SVD

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

3.3.13 Learning Algorithm (feature 13)

The most popular learning algorithm is called error back-propagation (BP), a first-order gradient method. Second-order gradient methods are known to have higher training speed and accuracy

DOI: http://doi.org/10.31224/osf.io/rghpf

(Wilamowski 2011). The most employed is called Levenberg-Marquardt (LM). All these traditional schemes were implemented using MATLAB toolbox (The Mathworks, Inc 2017).

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

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

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

(ii) Minimum performance gradient = 0.

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

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

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

DOI: http://doi.org/10.31224/osf.io/rghpf

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

DOI: http://doi.org/10.31224/osf.io/rghpf

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) % 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

DOI: http://doi.org/10.31224/osf.io/rghpf

associated to target values close to zero. Even so, this trick may still lead to (relatively) large solution errors while groundbreaking results are depicted as regression plots (target vs. predicted outputs).

3.4.1 Maximum Error

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

3.4.2 Percentage of Errors > 3%

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

3.4.3 Performance

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

3.5 Software Validation

Several benchmark datasets/functions were used to validate the developed software, involving low- to high-dimensional problems and small to large volumes of data. Due to paper length limit, validation results are not presented herein but they were made public online (Researcher 2018).

3.6 Parametric Analysis Results

Aiming to reduce the computing time by cutting in the number of combos to be run – note that all features combined lead to hundreds of millions of combos, the whole parametric simulation was divided into nine parametric SAs, where in each one feature 7 only takes a single value. This measure aims to make the performance ranking of all combos within each “small” analysis more “reliable”, since results used for comparison are based on target and output datasets as used in ANN training and yielded by the designed network, respectively (they are free of any postprocessing that eliminates output normalization effects on relative error values). Whereas (i) the 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 Tabs. 2-4) – SA 1 involved learning algorithms 1-3 and SA 2 involved the ELM-based counterpart, (ii) the 3rd – 7th SAs combined all possible methods from features 3, 4, 6 and 7, and concerning all other features, adopted the methods integrating the best combination from the aforementioned 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.

DOI: http://doi.org/10.31224/osf.io/rghpf

ANN feature methods used in the best combo from each of the abovementioned nine parametric sub-analyses, are specified in Tab. 5 (the numbers represent the method number as in Tabs 2-4). Tab. 6 shows the corresponding relevant results for those combos, namely (i) maximum error, (ii) % errors > 3%, (iii) performance (all described in section 3, and evaluated for all learning data), (iv) total number of hidden nodes in the model, and (v) average computing time per example (including data pre- and post-processing). All results shown in Tab. 6 are based on target and output datasets computed in their original format, i.e. free of any transformations due to output normalization and/or dimensional analysis. The microprocessor used in this work has the following features: OS: Win10 Home-64bits, RAM: 8GB, Local Disk Memory: 128GB, CPU: Intel® Core™ i5 6200U (dual-core) @ 2.30 GHz.

Tab. 5. ANN features (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	1	1	1	1	1	1	3	2	3	1	3
2	1	2	6	2	3	7	1	2	1	1	9	2	7	1	3
3	1	2	6	3	1	1	1	1	1	1	3	2	3	1	3
4	1	2	6	2	1	1	2	1	1	1	3	2	3	1	3
5	1	2	6	1	1	1	3	1	1	1	3	2	3	1	3
6	1	2	6	2	1	7	4	1	1	1	3	2	3	1	3
7	1	2	6	4	1	7	5	1	1	1	3	2	3	1	3
8	1	2	6	4	1	7	5	1	1	1	3	2	3	1	3
9	1	2	6	4	1	7	5	1	3	3	3	2	3	1	3

DOI: http://doi.org/10.31224/osf.io/rghpf

Overall, to obtain satisfactory results, 219 ANN feature combinations were run in the parametric analysis of this problem. In 3.7, the best ANN-based model obtained is proposed to efficiently and effectively solve the real-world problem addressed. In sub-section 3.7.4, the performance results of the proposed ANN are also based on target and output datasets computed in their original format.

Tab. 6. Performance results for the best design from each parametric sub-analysis: (a) ANN, (b) NNC.

SA	ANN				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	46.2	2.7	23.4	12	4.40 E -03
2	1598.9	99.2	90.6	43	2.58 E -04
3	45.5	2.5	21.9	12	1.77 E -03
4	141.2	8.7	34.4	12	3.23 E -04
5	10.1	1.6	17.2	12	1.74 E -03
6	253.4	8.9	31.3	12	3.04 E -03
7	12.4	1.2	9.4	12	7.53 E -04
8	108.8	8.4	31.3	12	1.44 E -03
9	0.2	0.0	0.0	12	1.34 E -03

(a)

SA	NNC				
	Max Error (%)	Performance All Data (%)	Errors > 3% (%)	Total Hidden Nodes	Running Time / Data Point (s)
1	14.1	1.7	20.3	12	4.56 E -03
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	-	-	-	-	-
6	253.3	8.4	28.1	12	5.22 E -03
7	9.4	0.6	4.7	12	1.65 E -03
8	108.7	8.1	28.1	12	3.79 E -03
9	-	-	-	-	-

(b)

DOI: http://doi.org/10.31224/osf.io/rghpf

3.7 Proposed ANN-Based Model

The proposed ANN is the one, among the ones simulated during the parametric analysis, exhibiting the lowest maximum error. In this case, that model was yielded by SA 9 and is characterized by the ANN feature methods {1, 2, 6, 4, 1, 7, 5, 1, 3, 3, 3, 2, 3, 1, 3} in Tabs. 2-4. Aiming to allow implementation of this model by any user, all variables/equations required for (i) data preprocessing, (ii) ANN simulation, and (iii) data postprocessing, are presented in 3.7.1-3.7.3, respectively. The proposed ANN is a MLPN with 5 layers and a distribution of nodes/layer given by 3-4-4-4-1. Concerning connectivity, the network is fully-connected (across layer connections allowed), and the hidden and output transfer functions are all Hyperbolic Tangent and Identity, respectively. The network was trained using the LM algorithm (1500 epochs). After design, the network computing time concerning the presentation of a single example (including data pre/postprocessing) is 1.34x10^{-3} s – Fig. 7 depicts a simplified scheme of some of network key features. Lastly, all relevant performance results concerning the proposed ANN are illustrated in 3.7.4.

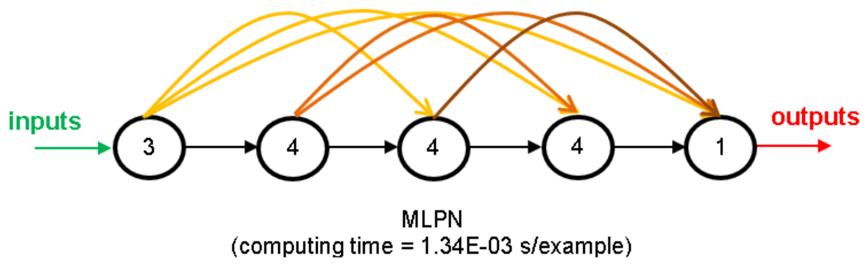


Fig. 7. Proposed 3-4-4-4-1 fully-connected MLPN – simplified scheme.

DOI: http://doi.org/10.31224/osf.io/rghpf

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

3.7.1 Input Data Preprocessing

For future use of the proposed ANN to simulate new data $Y_{1,sim}$ (3 x P_{sim} vector) 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 1 – see Tab. 2), which should be applied after all (eventual) qualitative variables in the input dataset are converted to numerical (using feature 1's method). Next, the necessary preprocessing to be applied to $Y_{1,sim}$, concerning features 2, 3 and 5, is fully described.

Dimensional Analysis and Dimensionality Reduction

Since neither dimensional analysis (*d.a.*) nor dimensionality reduction (*d.r.*) were carried out,

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

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

DOI: http://doi.org/10.31224/osf.io/rghpf

$$\left\{Y_{1,sim}\right\}_{n}^{after} = \begin{bmatrix} 0.0222 \\ 0.0082 \\ 0.0042 \end{bmatrix} .\text{x}\ Y_{1,sim} \quad , \quad (27)$$

where ".x" multiplies component *i* in the l.h.s vector by all components in row *i* of $Y_{1,sim}$.

3.7.2 ANN-Based Analytical Model

Once determined the preprocessed input dataset $\{Y_{1,sim}\}_{n}^{after}$ (3 x P_{sim} matrix), the next step is to present it to the proposed ANN to obtain the predicted output dataset $\{Y_{5,sim}\}_{n}^{after}$ (1 x P_{sim} vector), which will be given in the same preprocessed format of the target dataset used in learning. In order to convert the predicted outputs to their "original format" (i.e., without any transformation due to normalization or dimensional analysis – the only transformation visible will be the (eventual) qualitative variables written in their numeric representation), some postprocessing is needed, as described in detail in 3.7.3. Next, the mathematical representation of the proposed ANN is given, so that any user can implement it to determine $\{Y_{5,sim}\}_{n}^{after}$, thus eliminating all rumors that ANNs are "black boxes".

DOI: http://doi.org/10.31224/osf.io/rghpf

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

where

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

$$
W_{1-2} = \begin{bmatrix} 5.92014666514410 & -8.42287477536242 & 6.12188206788266 & 16.2938196162533 \\ -2.56058135230626 & 1.89089789773532 & 15.4065109454804 & -16.7015058877463 \\ -2.60190678017837 & 19.1276198184410 & 2.25430196289090 & 6.16075643046659 \end{bmatrix}
$$

$$
b_2 = \begin{bmatrix} 3.12956390068096 \\ -12.1226968878702 \\ -23.0312462334084 \\ -5.37192667684186 \end{bmatrix} \quad , \ (30)
$$

DOI: http://doi.org/10.31224/osf.io/rghpf

$$W_{1-3} = \begin{bmatrix} 0.414738619705944 & 1.32893635513501 & 4.14403332041229 & -1.74907381715892 \\ 1.18092219635388 & 2.26733319392057 & -0.780958164772676 & 4.06317925046579 \\ 2.00787682675325 & -4.26930673182629 & -4.04682458597197 & 2.31298898175833 \end{bmatrix}$$

$$W_{2-3} = \begin{bmatrix} -2.63455533785155 & 1.42443597315850 & -0.992635855262673 & -1.44068011428239 \\ -3.40557389137861 & -0.378443916473808 & -2.95088552349602 & -1.29760727903780 \\ 10.4283984813359 & -7.59151007212646 & 8.65915039538676 & -2.08761536504187 \\ -3.60024250992970 & -22.6288310553954 & 18.6518904980882 & -7.02100976668633 \end{bmatrix} \quad , \quad (31)$$

$$b_3 = \begin{bmatrix} -3.19599142132871 \\ 0.578891626317490 \\ 0.593077244491166 \\ -1.81765959400271 \end{bmatrix}$$

$$W_{1-4} = \begin{bmatrix} -10.1446983162026 & 0.587033479365037 & 1.10810082763965 & -0.719586110812137 \\ 2.92157866250819 & 1.06902882835772 & 2.02229814934862 & -1.04151405318611 \\ -6.88857125074414 & 0.633560737751578 & 3.32191629973955 & -1.37693811199200 \end{bmatrix}$$

$$W_{2-4} = \begin{bmatrix} 7.79122079835436 & 0.939498926219798 & 0.994920549794745 & -1.44588737852196 \\ 7.72109214777274 & 3.69400872747076 & 7.99494671486522 & -0.330044349509708 \\ 11.0635679581952 & -1.42326742691051 & 1.20946817714587 & 1.09672903767770 \\ -8.46335994709181 & -2.52477307911999 & -2.68232307734636 & -1.46464483126179 \end{bmatrix}$$

$$W_{3-4} = \begin{bmatrix} -3.94857119009544 & 0.773568727455620 & 33.2244475474305 & -1.94259316527410 \\ 10.4006204162642 & 6.76962264847463 & 15.3006148123197 & -0.493929406811997 \\ 11.4879411133621 & -2.66775810317843 & -25.6404239680795 & 6.39085101284729 \\ -16.4530465325905 & 0.239469158882274 & 1.35702273276274 & -0.544559259202252 \end{bmatrix}$$

$$b_4 = \begin{bmatrix} 9.21240069987668 \\ 0.165758464719451 \\ 1.44718146443782 \\ -1.77437690033072 \end{bmatrix} \quad , \quad (32)$$

DOI: http://doi.org/10.31224/osf.io/rghpf

$$W_{1-5}=\begin{bmatrix}2.41916145661974\\1.38165874874808\\2.23320041672227\end{bmatrix},\quad W_{2-5}=\begin{bmatrix}2.73625025655459\\-0.914148067927541\\-2.83722897348479\\0.199079605256181\end{bmatrix},\quad W_{3-5}=\begin{bmatrix}4.09625222832201\\-4.44884606844407\\-1.58753838697364\\8.92516458734120\end{bmatrix}$$

$$W_{4-5}=\begin{bmatrix}12.3854353599432\\14.1529826905679\\-1.07394444351335\\12.9119477184451\end{bmatrix},\quad b_5=1.93363683792993 \quad . \qquad (33)$$

Vectors and matrices presented in eqs. (30)-(33) can also be found in Authors (2018b), aiming to ease their implementation by any interested reader.

3.7.3 Output Data Postprocessing

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

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

Once obtained $\{Y_{5,sim}\}_n{}^{after}$, the following relations hold for its transformation to its non-normalized format $\{Y_{5,sim}\}_{d.a.}^{after}$ (just after the dimensional analysis stage), and for latter's transformation to its original format $Y_{5,sim}$ (with no influence of preprocessing)

$$Y_{5,sim} = \left\{ Y_{5,sim} \right\}_{d.a.}^{after} = \left\{ Y_{5,sim} \right\}_{n}^{after} \quad , \quad (34)$$

since no output normalization nor dimensional analysis were carried out. Moreover, since no negative output values are physically possible for the problem addressed herein, the ANN prediction should be defined as

$$Y_{5,sim} = \max \left\{ Y_{5,sim}, 0 \right\} \quad , \quad (35)$$

meaning that no structural damage exists whenever the output yielded by eq. (34) is negative.

3.7.4 Performance Results

Results yielded by the proposed ANN can be found either (i) online in Authors (2018a), where the target and ANN output values are provided together with the corresponding input dataset, or (ii) in terms of performance variables defined in sub-section 3.4, as presented next in the form of several graphs: (ii$_1$) a regression plot (Fig. 8) where network target and output data are plotted, for each data point, as *x*- and *y*- coordinates respectively – a measure of linear correlation is given by the Pearson Correlation Coefficient (*R*), as defined in eq. (1); (ii$_2$) a performance plot (Fig. 9), where performance values are displayed for several datasets; and (ii$_3$) an error plot (Fig. 10), where values concern the maximum error and the % of errors greater than 3%, for all data. It´s worth highlighting that all graphical results just mentioned 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.

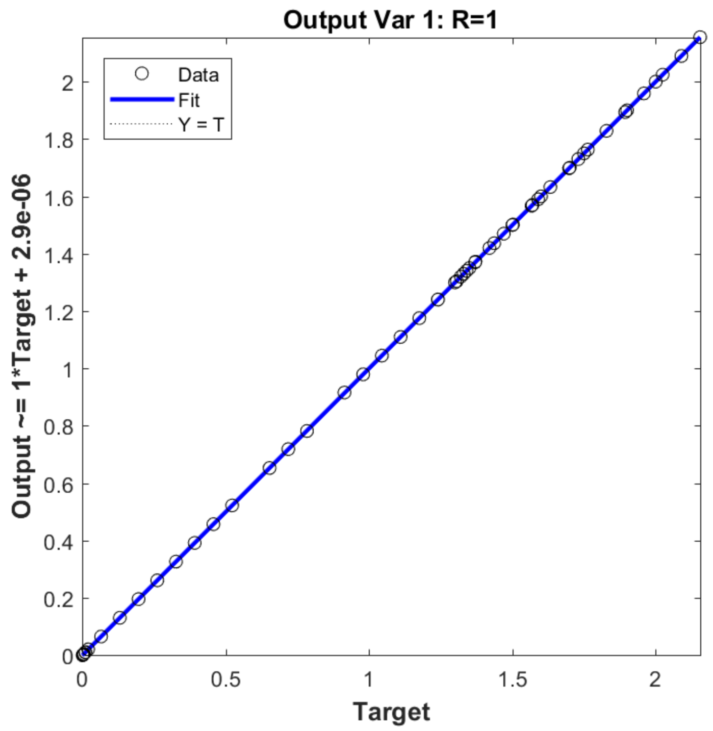


Fig. 8. Regression plot for the proposed ANN (see output variable in Fig. 1(a)).

3.8 Further Testing: Prediction of Experimental Results

Aiming to test the proposed analytical model to the prediction of experimental results, three test results taken from Marcy et.al (2014) were considered, as shown in Tab. 7. Only tests I and III

DOI: http://doi.org/10.31224/osf.io/rghpf

regard damaged members. The errors (smaller than 1%) displayed in Tab. 7 attest once again the capability of the proposed ANN-based analytical model.

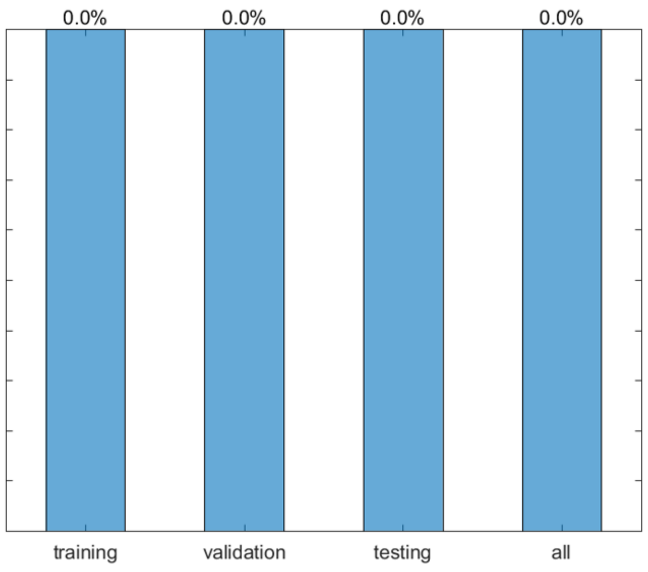


Fig. 9. Performance plot for the proposed ANN.

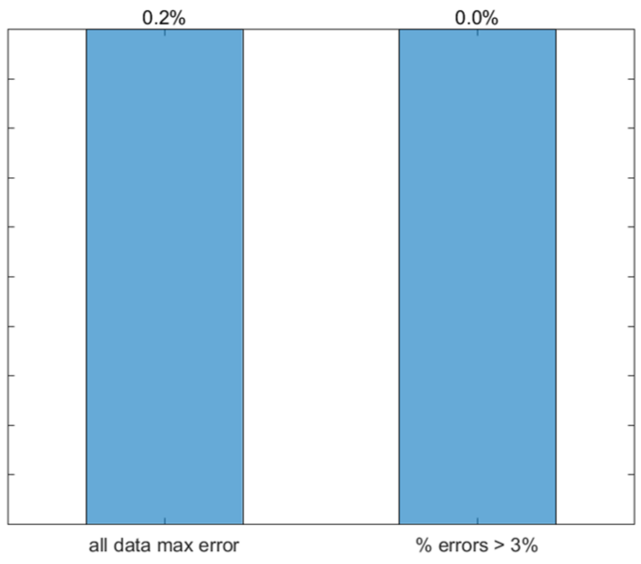


Fig. 10. Error plot for the proposed ANN.

DOI: http://doi.org/10.31224/osf.io/rghpf

Tab. 7. ANN performance in the prediction of 3 test results.

TEST	**Freq. 1** (Hz)	**Freq. 2** (Hz)	**Freq. 3** (Hz)	**Real Damage Location** (m)	**ANN-based Damage Location** (m)	**Error (%)** ANN vs Real
I	40.142	117.454	221.441	1.2407	1.2367	0.3
II	42.523	118.771	231.677	No Damage	-0.23 ↔ 0	0
III	39.590	117.305	221.143	1.306	1.315	0.7

4. Conclusions

This paper primarily aimed to assess the potential of Artificial Neural Network (ANN) models in the prediction of damage localization in structural members, as function of their dynamic properties – the three first natural frequencies were used. Based on 64 numerical examples from damaged (mostly) and undamaged steel channel beams, an ANN-based analytical model was proposed as a highly accurate and efficient damage localization estimator. The proposed model yielded maximum errors of 0.2 and 0.7 % concerning 64 numerical and 3 experimental data points, respectively.

Since it was proved that the approach taken works well for structural members, authors' next step (in the very near future) is to apply similar procedures to entire bridge or building structures, this time based on much larger datasets in order to provide an analytical solution with high credibility concerning its generalization capability, i.e. its capacity of giving good results for a large amount of examples (i) within the ranges considered for the input variables, and (ii) not considered during ANN development.

DOI: http://doi.org/10.31224/osf.io/rghpf

Acknowledgements

The 2nd and 3rd authors wish to acknowledge the support given by the Brazilian National Council for Scientific and Technological Development (CNPq) and the University of Brasília.

There are no conflicts of interest to disclose.

References

Ahmed MS (2016). Damage Detection in Reinforced Concrete Square Slabs Using Modal Analysis and Artificial Neural Network. PhD thesis, Nottingham Trent University, Nottingham, UK.

ANSYS, Inc. (2018). ANSYS® – Academic Research Mechanical, Release 18.1, Canonsburg, PA, USA.

Authors (2018a). data_set_ANN + results [Data set]. Zenodo, http://doi.org/10.5281/zenodo.1463849

Authors (2018b). W_b_arrays [Data set]. Zenodo, http://doi.org/10.5281/zenodo.1469120

Aydin K, Kisi O (2015). Damage Diagnosis in Beam-Like Structures by Artificial Neural Networks. Journal of Civil Engineering and Management, 21(5), 591–604, doi:10.3846/13923730.2014.890663

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.

Bandara RP, Chan THT, Thambiratnam DP (2013). The Three Stage Artificial Neural Network Method for Damage Assessment of Building Structures. Australian Journal of Structural Engineering, 14 (1), 13-25.

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.

Brasiliano A (2005). Identificação de Sistemas e Atualização de Modelos Numéricos com Vistas à Avaliação (in Portuguese). PhD thesis, Technology Faculty, University of Brasilia (UnB), Brasília, Brazil.

Callister WD, Rethwisch DG (2009). Materials Science and Engineering: An Introduction (8th ed). John Wiley & Sons, Versailles, USA.

Chengyin L, Wu X, Wu N, Liu C (2014). Structural Damage Identification Based on Rough Sets and Artificial Neural Network, The Scientific World Journal, 2014(ID 193284), 1-9, doi: 10.1155/2014/193284

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.

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.

Gerdau (2018). Perfil U Gerdau. [online] Available at https://www.gerdau.com/br/pt/produtos/perfil-u-gerdau#ad-image-0 [Accessed 15 Oct. 2018].

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.

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.

Jin C, Jang S, Sun X, Li J, Christenson R (2016). Damage detection of a highway bridge under severe temperature changes using extended Kalman filter trained neural network. Journal of Civil Structural Health Monitoring, 6(3), 545 - 560.

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.

Kourehli SS (2015). Damage Assessment in Structures Using Incomplete Modal Data and Artificial Neural Network. International Journal of Structural Stability and Dynamics, 15(06), 1450087-1-17, doi:10.1142/s0219455414500874

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 Mech and Eng, 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.

Marcy M, Brasiliano A, da Silva G, Doz, G (2014). Locating damages in beams with artificial neural network. Int. J. of Lifecycle Performance Engineering, 1(4), 398-413.

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

DOI: http://doi.org/10.31224/osf.io/rghpf

Meruane V, Mahu J (2014). Real-Time Structural Damage Assessment Using Artificial Neural Networks and Antiresonant Frequencies. Shock and Vibration, 2014 (ID 653279), 1-14, doi: 10.1155/2014/653279

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.

Nazarko P, Ziemianski L (2017). Application of artificial neural networks in the damage identification of structural elements. Computer Assisted Methods in Engineering and Science, 18(3), 175–189, Available at: http://cames.ippt.gov.pl/index.php/cames/article/view/113 (accessed on Nov 2nd 2018).

Nguyen VV, Dackermann U, Li J, Makki Alamdari M, Mustapha S, Runcie P, Ye L (2015). Damage Identification of a Concrete Arch Beam Based on Frequency Response Functions and Artificial Neural Networks. Electronic Journal of Structural Engineering, 14(1), 75-84.

Onur Avci, P. O., & Abdeljaber, A. O. (2016). Self-Organizing Maps for Structural Damage Detection: A Novel Unsupervised Vibration-Based Algorithm. Journal of Performance of Constructed Facilities, 30(3), 1-11.

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, Neurocomp., 214(Nov), 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: http://doi.org/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.

Structural Vibration Solutions A/S (SVS) (2018). ARTeMIS Modal 4.0®, Aalborg, Denmark.

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.

Vakil-Baghmisheh M-T, Peimani M, Sadeghi MH, Ettefagh MM (2008). Crack detection in beam-like structures using genetic algorithms. Applied Soft Computing, 8(2), 1150–1160. doi:10.1016/j.asoc.2007.10.003.

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, IEEE (eds), Auburn Univ., Auburn, AL.

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.

DOI: http://doi.org/10.31224/osf.io/rghpf
© **2018** by Abambres M, Marcy M, Doz G (**CC BY 4.0**)

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

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