Stochastic One-Step Training for Feedforward Artificial Neural Networks

Hector Cano-Rocha · Raul Gonzalez-Garcia

Abstract

This paper studies the use and application of a fast method (non-iterative and instantaneous) for Feedforward Neural Networks training in which the weights of the hidden layer are assigned randomly, and the weights of the output layer are trained through a linear regression adjustment. The method solves two of the problems that are present in traditional training: training time and optimal structure. While traditional iterative training methods require long periods to train a single structure, the proposed method allows training a structure in a single step (not iterative). In this way, by scanning the number of neurons in the hidden layer, many structures are trained in a short time, and it is possible to obtain an optimal topology. A quality control criterion of the predictions is proposed based on the coefficient of determination that guarantees short times and an optimal number of hidden neurons to characterize a specific problem. The feasibility of the proposed method is tested by comparing its performance against building functions of the artificial neural networks toolbox in Matlab®, resulting superior in both approximation quality and training time. A rigorous study and analysis are performed for the regression of simulated data on two different surfaces with a specific noise and different topologies of the neural network. The resulting process time is at least 150 times shorter for proposed training than with the iterative training that Matlab uses, thus obtaining well-founded learning rules. A novel way of an amputated matrix is proposed that breaks the paradigm of the way multiple-output systems are trained and improves the quality of predictions with no detriment to training times.

Keywords Feedforward neural network · Constructive networks · Training · Cross-validation · Single-hidden layer feedforward network · Multiple responses
1 Introduction

Modeling the behavior of a system is of vital importance for its study, analysis, control, and optimization. The main objective of modeling is to find a function capable of providing a good approximation to the system’s experimental data, in addition to filtering experimental noise. An alternative for the modeling of a system is models based on neural networks. To training these models, it is necessary to have a large amount of data of the system, besides exist two problems that are present in traditional training: training time and optimal structure. Where training time is usually large due to the algorithms of optimization are slow and require long periods to train a single structure. We proposed a training method with the capacity of identified the best structure for each output variable and reducing the time training for a neural network shallow.

Artificial Neural Networks (ANNs) are models inspired by the biological neuronal systems that have been studied since the 1940s. One of the pioneering works in this branch was developed by scientists Warren S. McCulloch and Walter Pitts in 1943 [1], who generated a model capable of representing the basic functioning of a neuron. Years later, in 1958, Frank Rosenblatt [2] presented the “perceptron” (linear classifier), which is currently used as the basis for many techniques within the field of artificial intelligence (A.I.). Subsequently, research in the field of ANNs was of little interest for almost two decades, mainly due to the work presented by M. Minsky and S. Papert [3], where they exposed the perceptron’s weaknesses in trying to solve nonlinear classification problems. In addition, they presented the difficulties in adjusting the synaptic weights when the perceptron presented several levels (multi-layer perceptron).

The idea of the delta rule was first presented in 1974 by P. Werbos in his thesis work [4], but unfortunately, it went unnoticed. It was not until 1986 that Rumelhart, Hinton, and Williams [5, 6] rediscovered the delta rule for the calculation of derivatives to adjust the synaptic weights of the multilayer perceptron. Due to this work, the backpropagation learning algorithm for ANNs was proposed, thus achieving a resurgence of ANNs in the‘90 s. This algorithm is iterative and is widely used today despite having some disadvantages such as long training times and high computational cost.

ANNs have been used since the 90 s to solve a large number of problems arising in different areas of science [7–21]. Today, it is a common task to implement ANNs to solve multiple problems in various areas; this is mainly due to three factors: (1) the use of more powerful processors than those of 30 years ago; (2) the ease of storing large amounts of information, as well as the creation of large databases and; (3) the continuous development of better learning techniques for ANNs.

2 Methodology

A Feedforward Neural Network (FFNN) is depicted graphically in Fig. 1. In this net, the information flows forward: it is received in the neurons of the input layer, then processed in all the neurons of each hidden layer (it can have one or several hidden layers), and finally, it is processed in the neurons of the output layer and returns a response to the input stimulus. In any study, the number of neurons in the input and output layers is uniquely determined by the number of factors and responses to be modeled, respectively, while the number of hidden layers and neurons are generally determined heuristically by the investigator.
Stochastic One-Step Training for Feedforward Artificial...

Within the field of FFNN, the topology can be classified in two types: (1) Single-hidden Layer Feedforward Neural Network (SLFN), which is characterized by being a superficial net of simple structure with a single hidden layer, and (2) Deep Neural Network (DNN), which is characterized by being of complex structure and having multiple hidden layers. DNN’s have had a great boom in recent years due to their high abstraction capacity for pattern recognition in large amounts of data (Big Data) [22]. An example of this is the applications for in the field the computer vision, e.g., image classification [23], fine-grained image recognition [24], visual representation for image captioning [25], and generation of natural language answer through video (Video Question Answering) [26]. On the other hand, it has been considered that SLFNs do not have the same abstraction capacity as DNNs [27]; therefore, the use of SLFN has been relegated to problems of a smaller size (few inputs and few outputs). However, it has been shown that an SLFN with sufficient neurons has the ability to approximate any continuous nonlinear function [28–30]; therefore, they can model a large amount of natural and artificial phenomena which is too complex to be modeled using classical techniques.

In recent years, various strategies have been implemented to determine the optimal number of neurons that an FFNN should have, but no clear rules as to how to do so have been established. In the search for optimal topologies for SLFN, some authors have presented interesting ideas [31–35], where they evaluate the performance of the network through a selection criterion (validation models or error-linked models) as neurons are added after neuron in the hidden layer. However, conclusive results that determine the optimal structure of the SLFN for a particular task with multi-input multi-output (MIMO) problems have not yet been obtained.

### 2.1 FFNN Evaluation

The information is received in the neurons of the input layer of an FFNN. Then, each neuron from the different hidden layers and the output layer carry out a linear combination using the information vector of the neurons from the preceding layer and the vector of synaptic weights (denoted by the connection arrows), plus an offset which is then softened with an activation function. Thus, according to the schematization in Fig. 1a, the output of neuron $j$ in layer $i$ is:

![Fig. 1 Scheme and nomenclature of an ANN. a Single output FFNN structure and b sigmoidal activation function](image)

- Single output FFNN structure
- Sigmoidal activation function
\[ y_{j}^{i+1} = g \left( \theta_{j}^{i} + \sum_{k=1}^{n_{i}} y_{k}^{i} \cdot W_{k,j}^{i} \right) \text{ for } i = 1, \ldots, c - 1 \text{ and } j = 1, \ldots, n_{i+1} \]  

(1)

where \( W_{k,j}^{i} \) is the weight that connects neuron \( k \) in layer \( i \) with neuron \( j \) in layer \( i + 1 \), \( \theta_{j}^{i} \) is the offset applied to neuron \( j \) in layer \( i + 1 \), \( y_{j}^{i+1} \) is the output (response) of neuron \( j \) in layer \( i + 1 \), \( c \) is the number of layers, \( n_{i} \) is the number of neurons in layer \( i \) and \( g(\cdot) \) is the activation function.

The expression shown in Eq. (1) is the traditional way in which the FFNN theory is treated. However, this equation can be expressed in a matrix way if all weights and offsets are grouped between two consecutive layers in a single transformation matrix from layer \( i \) to layer \( i + 1 \), resulting in the expression:

\[ y_{\cdot}^{i+1} = g \left( \left[ \begin{array}{c} y_{1}^{i} \\ y_{2}^{i} \\ \vdots \\ y_{n_{i}}^{i} \end{array} \right]^{T} \cdot W^{i} \right) \in \mathbb{R}^{n_{i+1}} \]  

(2)

for which

\[ y_{\cdot}^{i} = \left[ \begin{array}{c} y_{1}^{i} \\ y_{2}^{i} \\ \vdots \\ y_{n_{i}}^{i} \end{array} \right] \in \mathbb{R}^{n_{i}} \]  

(3)

\[ W^{i} = \left( \begin{array}{cccc} \theta_{1}^{i} & \theta_{1}^{i} & \cdots & \theta_{n_{i+1}}^{i} \\ W_{1,1}^{i} & W_{1,2}^{i} & \cdots & W_{1,n_{i+1}}^{i} \\ W_{2,1}^{i} & W_{2,2}^{i} & \cdots & W_{2,n_{i+1}}^{i} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n_{i},1}^{i} & W_{n_{i},2}^{i} & \cdots & W_{n_{i},n_{i+1}}^{i} \end{array} \right) \in \mathbb{R}^{(n_{i+1}) \times n_{i+1}} \]  

(4)

where \( y_{\cdot}^{i} \), for \( i = 1 \), is the vector of input stimuli \((x)\) and, when \( i = c \), is the vector of responses yielded by the FFNN. The activation function \( g(\cdot) \) is applied to each element of the resulting vector in the argument. Equations (2) to (4) are key to understanding the procedure applied to the methodology outlined here. In this work, we represent the vectors with a single underlined letter and matrices with a double underline letter.

In this work, the theory is applied for an SLFN \((c = 3)\), sigmoidal activation function in the hidden layers (Fig. 1b), \( g(x) = \tanh(x) \), linear function (identity) in the layer of output, and to a single neuron at the output, \( n_{3} = 1 \). As shown in the results section, the theory can be extrapolated to multiple hidden layers \((c > 3)\), more than one neuron in the output layer \((n_{c} \geq 1)\) and other types of nonlinear activation functions in hidden layers.

### 2.2 Traditional Training

To get an ANN to perform a specific task, its architecture and the value of its parameters (weights and offsets) must be known. Part of the architecture is determined by the problem to be modeled, which establishes the number of inputs (predictor variables) and the number of network outputs (response variables). The number of hidden layers and the number of neurons in each hidden layer is, to this day, somewhat arbitrary and heuristic depending greatly on the experience and skill of the researcher.

In the context of modeling, ANNs are empirical mathematical models with parameters (weights and offsets) that must be adjusted based on the experimental data of the problem to be modeled. The adjustment of the ANN parameters is known as training, which is necessary to find the appropriate values of the synaptic weights between each neuron (weights and
offsets) so that the ANN learns to perform a specific task. In this way, the training of the ANN is carried out by formulating a non-linear regression problem in which an error function is minimized (between the experimental value and the prediction of the ANN).

Starting from a set of \( m \) experimental data in which \( n_1 \) predictor variables are studied (input matrix \( X \in \mathbb{R}^{m \times n_1} \)), a response variable (vector of targets \( T \in \mathbb{R}^{m \times 1} \)) and taking a SLFN model \( \{ n_1, n_2, 1 \} \) with sigmoidal activation function in the hidden layer and linear function in the output layer, the error function to be minimized, put into matrix form, is

\[
e(w) = SSE = \sum_{i=1}^{m} (t_i - y^3_i)^2 = \left(T - y^3\right)^T \left(T - y^3\right)
\]

where \( y^3 \) is the prediction vector of the ANN that depends on the adjustable parameters \( w \) (vector that agglomerates all the adjustable parameters, \( w = vec\{W^1, W^2\} \)). According to Eq. (2), \( y^3 \) is obtained by evaluating the SLFN as follows

\[
y^3 = \left[1 - y^2\right] \cdot W^2 \in \mathbb{R}^{m \times 1}
\]

\[
y^2 = \tanh\left[1 - X \cdot W^1\right] \in \mathbb{R}^{m \times n_2}
\]

where \( W^1 \in \mathbb{R}^{(n_1 + 1) \times n_2} \), \( W^2 \in \mathbb{R}^{n_2 + 1} \) and \( 1 \in \mathbb{R}^{m} \).

In traditional training, \( w \) is obtained by the minimization (unrestricted optimization) of Eq. (5) through an iterative procedure (i.e. conjugate gradient, Levenberg–Marquardt or other technique) in which the parameter vector \( w \) is updated recursively with

\[
w_{k+1} = w_{k} + \eta_k \cdot B_k
\]

where \( k \) is the current iteration, \( \eta_k \) is a training speed parameter and \( B_k \) is a direction vector with both elements \( \eta_k \) and \( B_k \) depending on \( w \). All iterative training methods differ in the way in which they specify these elements \([36, 37]\) and they have the disadvantage that for different initial values of the adjustment parameters can give, as a result, different solutions (different local minimums are reached). In addition, because it is iterative, it requires very long training times to reach a minimum, which is not necessarily global, resulting in one of the causes that discourage the use of ANNs.

### 2.3 Backpropagation or the Delta Rule

In the majority of ANNs iterative training methods that use Eq. (8), the derivative of \( e(w) \) is necessary for each of the adjustable parameters \([38]\). Such an algorithm, due to the calculation of derivatives, is called backpropagation and placed in a matrix manner in accordance with Eqs. (5) to (7) as follows:

The FFNN is evaluated and the delta of the output layer is calculated

\[
\delta^2 = T - y^3 \in \mathbb{R}^{m \times 1}
\]

The derivatives of the error function are calculated with respect to the parameters that connect to the last layer

\[
\frac{\partial e}{\partial W^2} = \left[1 - y^2 \right]^T \cdot \delta^2 \in \mathbb{R}^{(n_2 + 1) \times 1}
\]
The penultimate layer delta is calculated

$$
\delta_1 = g'(\left[\begin{array}{c} 1 \\ X \end{array}\right] \cdot W^1) \ast \left[\begin{array}{c} \delta_2 \\ \cdot (W^{(2)})^T \end{array}\right] \in \mathbb{R}^{m \times n_2}
$$

(11)

The derivatives of the error function are calculated with respect to the parameters that connect to the penultimate layer

$$
\frac{\partial e}{\partial W^1} = -\left[\begin{array}{c} 1 \\ X \end{array}\right]^T \cdot \delta_1 \in \mathbb{R}^{(n_1+1) \times n_2}
$$

(12)

where $g'(x) = 1 - \tanh^2(x)$ is the derivative of the activation function (Fig. 1b), the operator $\ast$ represents element-to-element multiplication between two matrices and the matrix $W^{(2)} \in \mathbb{R}^{n_2 \times 1}$ is a modification of the matrix $W^2$ to which the first row has been mutilated (row of offsets $\theta^2$).

According to Eqs. (9) to (12), the error (the delta of the output layer) spreads backward starting from the output layer to the neurons that are in the hidden layers, this being the reason for the method name. If the FFNN has more hidden layers, the steps in Eqs. (11) and (12) are repeated.

### 3 Proposed Training

In this work, the ideas proposed in the literature have been adopted [39–43], where the concept of a Neural Network of Random Weights (NNRW) has been presented. Therein, it is proposed to adjust only the vector of weights of the output layer ($W^2 \in \mathbb{R}^{n_2+1}$), keeping the matrix of weights of the input layer constant ($W^1 \in \mathbb{R}^{(n_1+1) \times n_2}$) in which these parameters are specified with random values. These works propose to train the parameters of the last layer iteratively [40], which improves training speed but remains iterative and delayed. In addition, it has been proposed to adjust the parameters of the last layer using the pseudoinverse matrix [44]. The drawback in these works is that they do not provide a general methodology to determine the optimal number of neurons in the hidden layer and in some cases this problem is solved by trial and error resulting in poor FFNN.

Unlike these previous works, here it is proposed to adjust the weights towards the last layer in a single step with formal linear regression between the responses of the hidden layer and the output layer, avoiding the use of the pseudoinverse as much as possible. Here, the reasons for doing so are provided, in addition to giving clear rules regarding how to do it.

The response of an FFNN is a linear relationship with respect to the responses of the penultimate layer (last hidden layer); besides which, the parameters of the FFNN are grouped in matrices between every two consecutive layers. Thus, in an SLFN there is an array of parameters between the input layer and the hidden layer ($W^1 \in \mathbb{R}^{(n_1+1) \times n_2}$) and a matrix (a vector when you have a neuron output) of parameters between the hidden layer and the output layer ($W^2 \in \mathbb{R}^{n_2+1}$). Based on this, the SLFN parameters can be adjusted in parts: the parameters $W^2$ are adjusted by linear regression, while the parameters $W^1$ are adjusted by a non-linear regression method or by another technique. This paper proposes a procedure that finds the parameters $W^1$ instantaneously and can provide the optimal structure of SLFN with generalization capability.
To adjust the parameters $W^2$, a minimization of the error function (Eq. 5) is made by equating Eq. (10) to zero. For an SLFN with a single response, it results in:

$$-\left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T \cdot \delta^2 = 0$$ (13)

By substituting the output delta (Eq. 9) in Eq. (13) and clearing the optimized parameters, $W^{2*}$, Eq. (16) is reached:

$$\left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T \cdot \left( T - \left[\begin{array}{c} 1 \\ y^2 \end{array}\right] \cdot W^{2*}\right) = 0$$ (14)

$$\left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T \cdot \left[\begin{array}{c} 1 \\ y^2 \end{array}\right] \cdot W^{2*} = \left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T \cdot T$$ (15)

$$W^{2*} = \left( \left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T \cdot \left[\begin{array}{c} 1 \\ y^2 \end{array}\right] \right)^{-1} \cdot \left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T \cdot T$$ (16)

which is the key to the proposed method.

Observations on Eq. (16):

1. The matrix $\left( \left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T \cdot \left[\begin{array}{c} 1 \\ y^2 \end{array}\right] \right)^{-1} \cdot \left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T$ is, formally, the left inverse of $\left[\begin{array}{c} 1 \\ y^2 \end{array}\right] \in \mathbb{R}^{m \times (n_2+1)}$. It is necessary that such a matrix exists in order to obtain $W^{2*}$.

2. The necessary and sufficient condition for the left inverse to exist is that there is the inverse of the square matrix $\left[\begin{array}{c} 1 \\ y^2 \end{array}\right]^T \cdot \left[\begin{array}{c} 1 \\ y^2 \end{array}\right]$ and that happens when the columns of $\left[\begin{array}{c} 1 \\ y^2 \end{array}\right]$ are linearly independent. In a practical problem, one can move away from linear dependence if the number of lines is much greater than the number of columns, which implies that the number of experimental observations must be much greater than the number of hidden neurons, $m \gg n_2$. This implies that to train an SLFN, many experimental observations are required. In the extreme case that $\left[\begin{array}{c} 1 \\ y^2 \end{array}\right]$ has linearly dependent column vectors, the left inverse can be obtained by decomposition into singular values such as pseudoinverse.

3. The matrix $y^2$ depends on the parameters $W^1$, therefore, $W^{2*}$ also depends on $W^1$. Since in each update of $W^{2*}$ the $W^1$ is constant, then the training is done in one step and is instantaneous with Eq. 16.

4. The number of adjustable parameters is $n_2 + 1$, so at least $m = n_2 + 1$ experimental data is required. In practice, for the left inverse matrix to move away from the singularity, it is necessary that $m \gg n_2 + 1$.

5. The calculation of $W^{2*}$ is practically instantaneous compared to the time required in the iterative training methods. Thus, here it is proposed to carry out several adjustments of $W^{2*}$, making a scan in the number of neurons of the hidden layer, $n_2$, from 1 to $m - 1$.

We conjecture that when increasing $n_2$, between one to $m - 1$, the value of $e(w)$ (Eq. 5) decreases to a minimum where there is an optimal SLFN structure with a generalization capacity. In this way, $W^{1*}$ is not optimized, rather it is stochastically assigned in each calculation of $W^{2*}$ and its number of columns grows or decreases with the number of hidden neurons. The method has been called SOS (Stochastic One-Step) because the ANN trains in a single step with stochastic values of $W^1$ that are updated by moving the number of hidden neurons.
In order to fully optimize the parameters towards the hidden layer, it is necessary to solve, for $W_1^*$, Eq. 12 equal to zero:

$$\left[1 \cdot X^T\right] \cdot \delta_1^l = 0$$  \hspace{2cm} (17)

However, this results in work as challenging as the current iterative process of ANN training. Here it is shown that it is not necessary to do that.

### 3.1 Cross-Validation

In the 1990s, scientists G. Cybenko [29] and Hornik et al. [28, 30] theoretically demonstrated that an SLFN, Fig. 1a, with a finite and sufficient number of hidden neurons, has universal approximation properties. They recommend that the activation function be monotonic increasing, limited in its output value and that it be continuous. Based on this, and unlike other types of models (i.e. polynomials), ANNs can adjust the experimental noise, confusing it with the signal of interest. That is why in the training of ANN a control of the quality of the adjustment must be kept, which is accomplished using cross-validation [45].

Cross-validation is carried out by splitting the experimental data set into two: a training set and a validation set. The proportion in which these two sets should be divided is done in a heuristic manner and a proportion of 80% is recommended for training data and the complement for validation and/or testing. The training set is used to adjust the parameters, while the validation set is used to test the quality of the adjustment. The most used optimality criterion to evaluate the performance of an ANN during training is to minimize the mean squared error (MSE).

In this paper, it is proposed to continue using cross-validation to control the quality of the adjustment. Unlike most jobs in the ANN field, the coefficient of determination (Eq. 17) is used as a control parameter in training, instead of the MSE:

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$  \hspace{2cm} (17)

where $SSE$ is the sum square error that depends on the parameters of the FFNN and is obtained from Eq. (5), $SST$ is the total sum square and is a constant that represents the total variation of the response data $\left( SST = \sum_{i=1}^{m} t_i^2 - \left( \sum_{i=1}^{m} t_i \right)^2 / m \right)$. Thus, there are two values of $R^2$: one calculated with the training set and another with the validation set.

Additionally, to control the excessive inclusion of parameters to the model, as is done in linear regression, the adjusted determination coefficient, $R_{adj}^2$, which punishes the inclusion of parameters is used:

$$R_{adj}^2 = 1 - \tau \cdot (1 - R^2)$$  \hspace{2cm} (18)

where $\tau$ is the control that punishes the inclusion of parameters and for both the training set and the validation set, for practical reasons, here it is taken as:

$$\tau = \frac{m_t - 1}{m_t - n_2 - 1}$$  \hspace{2cm} (19)

where $m_t$ is the number of experimental data used in the training set. There are also two values of $R_{adj}^2$: one calculated with the training set and another with the validation set.

These parameters are chosen because they represent standardized measures in [0, 1]. $R^2$ is a measure related to $SSE$ and $R_{adj}^2$ is a measure related to $MSE$. Now the question is...
Fig. 2 Surfaces used to obtain simulated data in SLFN training. a Surface of the Gaussian function, and b surface of the function similar to an egg carton

which of all these parameters serves as a criterion for choosing the number of neurons in the hidden layer, $n_2$? The answer is given in the results section.

3.2 Study Data

Here we work with simulated data to which an amount of experimental noise is added (20% of its amplitude in the working region). Simulated data have been chosen in order to have control over the study of the quality of the approximations since they can be compared with real data. The selected surfaces are presented in the following figure:

The first is a Gaussian surface that has a bulge in the center (Fig. 2a), and the second is a function that has several valleys and ridges similar to an egg carton (Fig. 2b). These functions have been chosen as a standard case to test the performance and generalization of an SLFN. This gives a better perspective of the generalization of an SLFN because the level of noise added to the data can be controlled, which is very difficult with real data. The objective is to verify that the network trained by the proposed method can reconstruct the original surfaces and filter out the added noise, resulting in a high capacity for generalization.

4 Results

The experimental results performed with the SOS training method are presented. The performance of the SOS method is measured using the simulated data from the surfaces shown in Fig. 2. All results are obtained with the structure SLFN { 2, $n_2$, 1 }, using an activation function $\tanh(x)$ in the hidden layer and identity activation function in the output layer. All experiments were carried out in the Matlab R2015a® environment, which runs on an HP ProBook 4540 s laptop with Intel (R) Core (TM) i5-3210 M processor, CPU @ 2.50 GHz, 8 GB of RAM, and a Windows 7 to 64-bit operating system.
To show the benefits of the SOS method, simulated data from the surfaces of Fig. 2 were used. The training data consisted of 580 randomly sampled points within the same region of independent variables \((x, y)\) shown in Fig. 2. These values form the matrix of inputs \(X \in \mathbb{R}^{580 \times 2}\) in Eqs. (7) and (16). The value of the response variable consisted of the value of the function, \(z(x, y)\), plus an experimental noise of 20\% with respect to the amplitude of the corresponding surface of Fig. 2. These values form the target vector \(T \in \mathbb{R}^{580 \times 1}\) in Eq. 16. The set of 580 experimental points was split into two groups: one for training and one for validation. The training set consisted of \(m_t = 435\) experimental points (75\%) and the validation set consisted of \(m_v = 145\) experimental points. For SLFN training, all experimental data, both input and response variables, were scaled in the range \([-1, +1]\).

First, to demonstrate the speed, accuracy and generalization capacity of the SOS method, a comparison was made against the iterative training method incorporated into the Matlab® environment (trainlm). In both cases, an SLFN with structure \(\{2, 55, 1\}\) was chosen and the adjustable parameters of the network were randomized in the range \([-1, +1]\). The results of the reconstructed surfaces are shown in Fig. 3.

A different quality of reconstruction provided by each method is observed. The surfaces trained by SOS (Fig. 3a and c, left side) have a greater smoothness compared to surfaces iteratively trained by the native Matlab method (Fig. 3b and d, right side). This result demonstrates that the SOS training method provides results that can compete, in quality, with traditional iterative methods. Note that this result is not a stroke of luck since in the reconstruction of both surfaces, the experimental error has been adequately filtered. This is shown in the

**Fig. 3** Reconstruction of the surfaces by the SLFN. Surfaces obtained with SOS training (a and c) and surfaces obtained with Matlab training (trainlm) (b and d). Each SLFN has a topology \(\{2, 55, 1\}\)
Table 1: ANN's performance parameters in surface reconstruction. $R^2$ for training data set.

| Function | SOS Time (s) | $R^2$ | Matlab (trainlm) Time (s) | $R^2$ |
|----------|--------------|-------|---------------------------|-------|
| Fig. 2a  | 0.0019       | 0.8990| 0.2842                    | 0.9225|
| Fig. 2b  | 0.0021       | 0.9306| 0.3165                    | 0.9669|

graphs (lower right corner) of the response predicted by the network against the experimental response, where it is observed that the noise has been adequately filtered. It is also observed that the SLFN trained by SOS has generalization capacity since the validation set data (red crosses) have been properly estimated and the experimental error has been filtered.

Table 1 shows a comparison of training times and the coefficients of determination of the training set for both methods (SOS and Matlab) on both surfaces. The processing time is at least 150 times shorter for SOS training than with the iterative training that Matlab uses. It should be noted that the codes used by Matlab are highly optimized, while the codes used in the SOS method are not. Matlab adjustments have higher coefficient determination values; however, this does not mean that the predictions are better quality, as seen in the reconstruction of the surfaces in Fig. 3a and b. A very high value of $R^2$ of the training set could indicate that there is overfitting.

Thus, the SOS method, compared to the iterative method, is practically instantaneous, provides very acceptable quality results, and results with generalization capabilities are achieved in a single step.

4.2 Effect of Hidden Neurons on FFNN Performance

By having a practically instantaneous training method, studies can now be done in which the number of neurons in the hidden layer is varied and the quality of the SLFN performance is observed through the coefficient of determination ($R^2$). The number of hidden neurons can vary, from 1 to $m - 1$, either by sequential scanning or sporadic scanning (increments greater than 1). This type of study was difficult to carry out with iterative training due to slow training.

Figure 4 shows the evolution of the coefficient of determination as a function of the number of neurons in the hidden layer. The evolution is shown only up to 150 neurons, since with a very large number of neurons, the quality of the reconstruction begins to deteriorate, as can be seen on the right side of Fig. 4a.

As shown in Fig. 4, the coefficients of determination vary when the number of hidden neurons increases though in general the following is observed:

1. The coefficient of determination of the training set, $R^2_{\text{train}}$, in the blue square curve is always increasing and approaches 1 when the number of hidden neurons tends to $m_t - 1$. This parameter does not provide a valid criterion for obtaining the optimal structure of the SLFN.

2. The adjusted determination coefficient of the training set, $R^2_{\text{adj.train}}$, in the curve of cyan-colored points (triangle down) begins to increase rapidly until it reaches a maximum and then decreases smoothly but in the end, it decreases rapidly. For the cases shown, these maximum values are reached at 201 and 214 hidden neurons respectively, $N^*_{\text{adj.train}}$. This parameter could provide a valid criterion for obtaining the optimal structure of the
SLFN, but it is discarded because they are relatively high values that are in the overfitting zone.

3. The coefficient of determination of the validation set, $R^2_{val}$, in the curve of red-colored points (triangle up) begins to increase rapidly until reaching a maximum then decreases smoothly, but eventually decreases rapidly with somewhat erratic behavior. For the cases shown, these maximum values are reached in 78 and 91 hidden neurons respectively, $N^*_{val}$. This parameter could provide a valid criterion for obtaining the optimal structure of the SLFN.

4. The adjusted determination coefficient of the validation set, $R^2_{adj, val}$, in the magenta dot curve starts increasing rapidly until reaching a maximum, then decreases smoothly, but in the end, it decreases rapidly and somewhat erratically. For the cases shown, these maximum values are reached in 71 and 91 hidden neurons respectively, $N^*_{adj, val}$. This parameter could provide a valid criterion for obtaining the optimal structure of the SLFN.

5. The coefficient of determination of the actual data, $R^2_{real}$, in the green asterisk curve is an additional parameter that has been used here to compare the quality of the prediction with the real data. It is obtained according to Eq. (17), only that the value of the sum of squares
of the error, \( SSE \), is calculated between the differences of the real value without noise and the value predicted by the network. It should be mentioned that, in a practical case, the actual value without noise is unknown. However, in this theoretical study, where simulated data is used, it is known and can be used to determine the parameter that best qualifies the evolution of the SLFN. This parameter starts increasing rapidly until it reaches a maximum, after which it decreases smoothly but in the end, it decreases rapidly and somewhat erratically. Where there is a maximum value in \( R_{val}^2 \) the best structure of the SLFN is reached, \( n_2^* = N_{real}^* \). For the cases shown, these maximum values are reached at 71 and 90 hidden neurons respectively, \( N_{real}^* \).

6. In general, it has been observed that the maximums are related as follows: \( N_{adj,train}^* \geq N_{val}^* \). The question here is which of the parameters \( N_{adj,train}^*, N_{val}^*, \) or \( N_{adj,val}^* \) is closest to \( N_{real}^* \) and provides the optimal SLFN structure? The following section presents a statistical study to answer this question.

### 4.3 Appropriate Criteria to Select the Best Network

It is known that the optimal value of neurons in the hidden layer should be as close to \( N_{real}^* \) as it is where the SLFN predictions best approximate the real (unknown) surface. From the results obtained in Fig. 4, it can be seen that both \( N_{adj,val}^* \) and \( N_{val}^* \) are close to \( N_{real}^* \), but are found sooner or later. The \( N_{adj,train}^* \) is not an option because it is well above \( N_{real}^* \). The question is which of \( N_{adj,val}^* \) or \( N_{val}^* \) is closest to \( N_{real}^* \) and represents the best criterion for choosing the optimal number of neurons in the hidden layer?

In order to determine the best criteria to obtain an optimal structure of the SLFN, a series of 20 simulations were carried out under different conditions. The type of surface, the amount of data, the proportion used for the training set, and the proportion of noise with respect to the amplitude of the signal were modified. In each simulation, the number of neurons where there is a maximum in the coefficients of determination was determined: validation, \( N_{adj,val}^* \), adjusted validation, \( N_{adj,val}^* \), and real, \( N_{real}^* \). The results of the 20 simulations are shown in Table 2.

It is observed that on average \( N_{adj,val}^* \leq N_{val}^* \leq N_{real}^* \) (52.30 < 55.10 < 58.25); however, these data have a certain degree of dispersion, and although they appear different, they can be statistically equal. To verify the similarity of this information, the results of \( N_{adj,val}^* \) and \( N_{val}^* \) are contrasted against \( N_{real}^* \) with \( t\)-student tests for the means of two paired samples. It has been found that \( N_{adj,val}^* \) is statistically different from \( N_{real}^* \) \( (p = 0.0034) \) and that \( N_{val}^* \) is statistically the same as \( N_{real}^* \) \( (p = 0.1019) \).

In this way, it is concluded that in practice the optimal number of neurons in the hidden layer is equal to that in which there is a maximum in \( R_{val}^2 \), that is, \( n_2^* = N_{val}^* \). In general, it has been found that any number of hidden neurons, \( n_2 \), between \( N_{adj,val}^* \) and before \( N_{adj,train}^* \) gives good results.

This learning method, as it is NOT iterative, requires very short training times and reaches a global minimum where the network has the ability to generalize. Thus, we already have a general training methodology with well-defined criteria to determine the optimal number of neurons in the hidden layer, which solves the inconvenience of the different local minima and the long training times, resulting in a great incentive for the use of SLFN.

### 4.4 Effect of Under- and Overtraining

Based on the determination coefficient of the validation set in Fig. 4 (curve of the red-colored triangle up), it is observed that there are very bad values of \( R_{val}^2 \) many neurons before and
Table 2  Experimental scenario with 20 simulations under different training conditions of an SLFN of structure \( \{2, n^*_2, 1\} \)

| Case | Surface | # Data | % Train | % Noise | \( N^*_\text{val} \) | \( N^*_\text{adj, val} \) | \( N^*_\text{real} \) |
|------|---------|--------|---------|---------|-----------------|-----------------|-----------------|
| 1    | b       | 1000   | 90      | 5.00    | 56              | 56              | 52              |
| 2    | a       | 200    | 50      | 30.00   | 34              | 34              | 58              |
| 3    | b       | 1000   | 50      | 5.00    | 45              | 45              | 46              |
| 4    | a       | 1000   | 90      | 5.00    | 89              | 89              | 90              |
| 5    | a       | 1000   | 50      | 30.00   | 65              | 54              | 71              |
| 6    | b       | 1000   | 70      | 17.50   | 55              | 55              | 57              |
| 7    | b       | 600    | 70      | 30.00   | 48              | 35              | 41              |
| 8    | a       | 600    | 50      | 5.00    | 85              | 80              | 83              |
| 9    | b       | 400    | 70      | 17.50   | 50              | 50              | 57              |
| 10   | a       | 600    | 70      | 23.75   | 69              | 69              | 69              |
| 11   | b       | 200    | 50      | 30.00   | 46              | 46              | 46              |
| 12   | a       | 200    | 90      | 30.00   | 56              | 29              | 46              |
| 13   | b       | 600    | 70      | 17.50   | 60              | 60              | 60              |
| 14   | a       | 600    | 70      | 17.50   | 57              | 57              | 67              |
| 15   | b       | 600    | 70      | 17.50   | 35              | 35              | 34              |
| 16   | b       | 600    | 70      | 17.50   | 59              | 59              | 54              |
| 17   | a       | 600    | 70      | 17.50   | 59              | 59              | 63              |
| 18   | a       | 200    | 90      | 30.00   | 21              | 21              | 38              |
| 19   | a       | 600    | 50      | 5.00    | 82              | 82              | 90              |
| 20   | b       | 200    | 90      | 30.00   | 31              | 31              | 43              |

after the optimal value, \( N^*_\text{val} \). This is because before reaching \( N^*_\text{val} \), the SLFN does not have enough neurons in the hidden layer that capture the general topology of the response, giving rise to an underfitting effect. On the other hand, long after reaching \( N^*_\text{val} \), the SLFN has a surplus number of neurons in the hidden layer that are able to capture the general topology of the response along with the experimental error, giving rise to an overfitting effect. At this stage, the network does not differentiate between the original data and the experimental noise.

The effect of the number of hidden neurons on the predicted response is shown in Fig. 5. Figures 5a and b show the reconstruction of the surfaces by the network trained with an optimal number of neurons (78 and 91 respectively for each surface). Figures 5c and d show the reconstruction with an amount of neurons well below the optimum (25 neurons per surface). Figures 5e and f show the reconstruction with an amount of neurons well above the optimum (130 for each surface). In Figs. 5a and b, it is observed that the reconstruction is excellent because there is an optimal number of neurons in the hidden layer. In Figs. 5c and d, a poor reconstruction of the surfaces is observed because there are not enough neurons in the hidden layer of the SLFN; however, although the structure is not enough, the network tries to capture the overall shape from the surface. In Figs. 5e and f a poor reconstruction of the surfaces is observed because there is an excessive number of neurons in the hidden layer, and its prediction has involuntarily captured the experimental error. The latter can be corroborated using the graphs of predictions against reals where the training points (in blue)
Fig. 5 Reconstructed surfaces that show the effect that the number of hidden neurons has on prediction quality. An SLFN with topology \{2, n_2,1\} was used where \(n_2\) is: a 78, b 91, c 25, d 25, e 130 and f 130

are very well placed on the 45° line, but the validation crosses (in red) begin to leave the 45° line. Note that in all cases the SOS training filters the data noise in the first instance.

After analyzing the effect of excessive numbers of neurons in the hidden layer, it can be seen that in Fig. 3b there is a corrugated effect on the surface, which is due more to the excess of neurons used in the training than to poor training by the iterative method used in Matlab. Thus, this corrugated effect is due to the excess of neurons in the hidden layer and is independent of the training method. This corrugated effect, due to excess hidden neurons, had not been shown in previous ANNs work.
4.5 Expanding Training to Multiple Responses

When there are several output responses, the iterative ANN training with the delta rule (backpropagation) is obtained by minimizing the total error function (Eq. 20) which is the sum of the error functions of each response (Eq. 5):

$$
e(w) = SSE = \sum_{i=1}^{m} \sum_{j=1}^{n_3} (t_{i,j} - y_{i,j}^3)^2 = trace\left(\left( T - \hat{y}^3 \right)^T \cdot \left( T - \hat{y}^3 \right)\right)$$

where $trace(\cdot)$ is the trace function of the matrix (sum of the elements of the main diagonal); therefore, Eq. (6) changes to matrix form as:

$$y^3 = \left[1 \ y^2\right] \cdot W^2 \in \mathbb{R}^{m \times n_3}$$

where $W^2 \in \mathbb{R}^{(n_2+1) \times n_3}$. All other equations remain the same. Equation (20) poses a total error function that is the contribution of all responses and, being minimized, assigns the same importance to each response. This can be considered wrong since not all responses have the same degree of structural complexity. An ANN trained in this way can provide little importance to a structurally complex response and much importance to a structurally smooth response, resulting in a poor ANN.

We find that each response has a different degree of importance that depends primarily on the topology of the response and therefore needs a different number of hidden neurons. A complex response with several ridges and valleys requires a network with a greater number of hidden neurons than a soft response with a single crest and/or valley. In this way, it is proposed to train a network and measure the quality of the prediction independently in each response. This had never been proposed before because iterative methods are very time-consuming, and if that were added during the ANN training for each response, then it would be a very reckless proposal. However, in the context of SOS training, this is a very appropriate proposal since consumes additional time only to calculate the $R^2_{val}$ for each response. In this way, Eq. (16), due to multiple responses, is transformed into

$$W^{2*} = \left( \left[ 1 \ y^2 \right]^T \cdot \left[ 1 \ y^2 \right] \right)^{-1} \cdot \left[ 1 \ y^2 \right]^T \cdot T \in \mathbb{R}^{(n_2+1) \times n_3}$$

where each column vector of matrix $W^{2*}$ corresponds to the parameters of each response variable as if they had been trained separately. As can be seen in Eq. (22), for a given number of hidden neurons training does not take more time than would be used for a single response. To determine the optimal number of neurons for each response, start with a neuron in the hidden layer. Parameters $W^1$ are assigned stochastically and using Eq. (22), the optimized $W^{2*}$ are obtained. A scan is done by moving the hidden neurons sequentially one at a time or in larger increments from 1 to $m_t - 1$, and in each step the $R^2_{val}$ is calculated for each response. With each neuron that is added, the $W^1$ matrix also grows in a column and the $W^{2*}$ matrix grows in a row. However, a stochastic column vector is added to the matrix $W^1$ (the preceding columns are kept unchanged), while the matrix $W^{2*}$ is completely renewed from Eq. (22). At the end of the scan, there is the number of hidden neurons where each response separately has the maximum value of $R^2_{val}$. These $N^*_val$ values represent the neurons necessary for that response, and the final SLFN is reconstructed with these values. For example, if the first response requires 30 neurons and the second requires 50 neurons, then $W^1$ is stochastically assigned with 30 columns and using Eq. (16) the column vector $W^{2*,1} \in \mathbb{R}^{31 \times 1}$ is calculated.
using the data from the first response and then increased with 20 zeros at the end to complete the rows of $W^{2*1} \in \mathbb{R}^{51 \times 1}$. Then 20 stochastic column vectors are added to the end of the matrix $W^1$ and using Eq. (16), the column vector $W^{2*2} \in \mathbb{R}^{51 \times 1}$ is calculated using the data from the second response. In the end, the augmented matrix $W^2 = \begin{bmatrix} W^{2*1} & W^{2*2} \end{bmatrix} \in \mathbb{R}^{51 \times 2}$ is reconstructed and together with $W^1$ they form the optimized structure of an SLFN. When we simulated multiple responses using the surfaces studied here (Fig. 2), we obtained practically the same results as presented in Figs. 5a and b.

Moreover, we have applied the SOS procedure to the 60,000 data from the MNIST database (Modified National Institute of Standards and Technology) [46], and have achieved good results. We used 36,000 randomly chosen data for the training set and the rest for validation; an SLFN $\{784, 7451, 10\}$ was obtained, which only fails in 2.72% of the 10,000 data in the test set. The significant results from the training are shown in Table 3 below. Each digit (0–9) presents a different number of $N_{opt}$ (optimal number of active hidden nodes), e.g., the digit 1 exhibits the highest number of active nodes. In contrast, digit 9 has the fewest number of active nodes, and it could be said that digit 1 presents a more significant difficulty compared to digit 9 to be classified in an adequate way because it requires more hidden nodes, and this translates to having a model with more parameters involved. Furthermore, it is hypothesized that manipulating and preprocessing the dataset can improve the quality of the prediction, without having to use proprietary DNN techniques (filters).

### 4.6 Final Procedure for SOS training

The proposed SOS procedure for training and determining the optimal architecture of an SLFN with the structure $\{n_1, n_2, n_3\}$, is as follows:

1. Experimental data variables are scaled in the range $[-1, +1]$.
2. An initial number of hidden neurons is assigned, $n_2 \geq 1$.
3. The $W^1$ parameters are initialized randomly in a suitable range such that the left inverse matrix of $\begin{bmatrix} 1 & y^2 \end{bmatrix}$ moving away from containing linearly dependent columns. Naturally, the parameters are initialized in a range of $[-1, +1]$, but that does not guarantee the above. We have found that good results are achieved in the independence of the columns when the amplitude of the weights is chosen in a range which is inversely proportional to the number of entries, $n_1$, but this is only a preliminary result.
4. The neuron outputs of the hidden layer of the training set are calculated from Eq. 7:
   \[
   y^2 = \tanh\left( \begin{bmatrix} X_{train} & 1 \end{bmatrix} \cdot W^1 \right) \in \mathbb{R}^{m \times n_2}
   \]
5. The parameters optimized to the last layer of the training set are calculated from Eq. (22):
   \[
   W^{2*} = \left( \begin{bmatrix} 1 & y^2 \end{bmatrix}^T \cdot \begin{bmatrix} 1 & y^2 \end{bmatrix} \right)^{-1} \cdot \begin{bmatrix} 1 & y^2 \end{bmatrix}^T \cdot T_{train}
   \]
6. The coefficient of determination of the validation set for each response is calculated from Eq. (17):
   \[
   R^2_{val} = 1 - \frac{SSE_{val}}{SST_{val}}
   \]
Table 3 Results of an SLFN \([784, 7451, 10]\) trained with the SOS method using 36,000 data from the MNIST database

| Digit | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $R^2_{train}$ | 0.9506 | 0.9578 | 0.9108 | 0.8931 | 0.9112 | 0.8926 | 0.9456 | 0.9021 | 0.8787 | 0.8466 |
| $R^2_{adj.train}$ | 0.9361 | 0.9438 | 0.8871 | 0.8642 | 0.8827 | 0.8629 | 0.9285 | 0.8796 | 0.8445 | 0.8122 |
| $R^2_{val}$ | 0.9029 | 0.9231 | 0.8297 | 0.7984 | 0.8341 | 0.8020 | 0.8823 | 0.8444 | 0.7709 | 0.7605 |
| $R^2_{adj.val}$ | 0.8744 | 0.8977 | 0.7844 | 0.7437 | 0.7807 | 0.7473 | 0.8451 | 0.8087 | 0.7063 | 0.7067 |
| $N_{opt}$ | 6801 | 7451 | 6301 | 6401 | 7301 | 6501 | 7201 | 5601 | 6601 | 5501 |
| Fails | 8 | 10 | 40 | 26 | 26 | 24 | 20 | 44 | 31 | 43 |
| Total cases | 980 | 1135 | 1032 | 1010 | 982 | 892 | 958 | 1028 | 974 | 1009 |
The $SSE_{val}$ is obtained when the network is evaluated using the validation set by adding the elements from the main diagonal of $\left( T_{val} - y^3 \right)^T \cdot \left( T_{val} - y^3 \right)$.

7. Save the pairs $\{ n_2, R^2_{val} \}$ from each of the responses.
8. Increase the number of neurons in the hidden layer, $n_2$, and repeat the process from step 3. The number of hidden neurons can be moved from 1 to $m_t - 1$. The change can be done sequentially one by one or by blocks, but it is necessary to increase it.
9. Find the optimal number of neurons in the hidden layer, $n_2 = N_{val}^*$ that corresponds to the highest value of $R^2_{val}$ for each of the responses.
10. If there is more than one response at the end of the scan, reconstruct the amputated matrix, $w^{2*}$, as indicated in the previous section.

When there are several hidden layers, all the parameters between them are assigned stochastically and only the weights of the last layer are calculated using Eq. (22). Different strategies can be applied to move the number of neurons in each hidden layer; the most common is to move them in the same way in each layer.

5 Conclusions

Finally, a training algorithm has been presented with well-defined criteria to determine the optimal number of neurons in the hidden layer, which solves the inconvenience of the different local minimums and the long training times, resulting in a very good incentive for the use of ANN. For a large number of problems, the method can be implemented on any personal computer, as long as it has the processing capacity to evaluate the inverse matrix that appears in Eq. (22), which becomes tedious when you have a very large data set.

The SOS method, compared to the iterative method, is practically instantaneous, yields very acceptable quality results and results with generalization capabilities are achieved in a single step. It has been found that the optimal number of neurons in the hidden layer is equal to that in which there is a maximum in $R^2_{val}$, for a particular output response. In general, it has been found that any number of hidden neurons, above $N_{adj,val}^*$ and before $N_{adj,train}^*$ gives good results. Besides, even when the number of neurons is above or below the optimal number required, the SOS method always tends to generalize (filters the noise).

The number of neurons in the hidden layer can be searched by scanning from 1 to before number of data with increments of one or greater than one. For problems with multiple responses, the scan provides the optimal number of neurons required for each response (according to its structural complexity) by generating an amputated matrix of weights towards the responses without detriment to the rapidity of the method.

The SOS method yields an optimal number of neurons in the hidden layer, which is greater than that which would be found using a traditional iterative method. However, with the iterative method, it is not practical to implement a scan search. The excessive number of neurons in the hidden layer causes a corrugated prediction quality and is independent of the training method. This corrugated effect, due to excess hidden neurons, had not been shown in previous ANN work. To avoid the linear dependence of the column vectors in the response matrix of the hidden layer, it is recommended to assign the random weights to the hidden layer in a range of amplitude inversely proportional to the number of entries in the network.

Acknowledgements The authors thank Conacyt and Prodep for the support received through the PCIQ and CAC36 of the FCQ-UASLP. H. Cano thank CONACYT fellowship 586482.
Compliance with Ethical Standards

Conflict of interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Bioph 5:115–133. https://doi.org/10.1007/BF02478259
2. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65:386–408. https://doi.org/10.1037/h0042519
3. Minsky ML, Papert SA (1969) Perceptrons: an introduction to computational geometry. MIT Press, Cambridge
4. Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. Harvard University, Cambridge
5. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536. https://doi.org/10.1038/323533a0
6. Rumelhart DE, Hinton GE, Williams RJ (1985) Learning internal representations by error propagation
7. Brunton SL, Kutz JN (2019) Data-driven science and engineering. Cambridge University Press, Cambridge
8. Herrera-Hernández EC, Ocampo-Perez R, Aguilar-Madera CG, Flores-Cano JV (2018) Adsorption-diffusion model with neural network-based equilibrium relationship. Desalin WATER Treat 132:42–51. https://doi.org/10.5004/dwt.2018.23030
9. González-García R, Rico-Martínez R, Wolf W et al (1998) Caracterización de sistemas no-lineales mediante redes neuronales: estudio de una reacción electroquímica. Ing Quim 30:173–184
10. Hou Z-Y, Dai Q, Wu X-Q, Chen G-T (1997) Artificial neural network aided design of catalyst for propane ammoxidation. Appl Catal A Gen 161:183–190. https://doi.org/10.1016/S0926-860X(97)00063-X
11. Baughman DR, Liu YA (1995) Artificial neural networks in bioprocessing and chemical engineering. Elsevier, Amsterdam
12. Himmelblau DM (2000) Applications of artificial neural networks in chemical engineering. Korean J Chem Eng 17:373–392. https://doi.org/10.1007/BF02706848
13. Gutiérrez Urueta GL, Colorado Garrido D, Hernandez JA et al (2019) Performance estimation and optimization of an adiabatic H2 O-Libr absorption system using artificial neural networks. Ing Invest y Tecnol 20:1–13. https://doi.org/10.22201/I.25940732e.2019.20n1.007
14. Karri RR, Sahu JN (2018) Modeling and optimization by particle swarm embedded neural network for adsorption of zinc (II) by palm kernel shell based activated carbon from aqueous environment. J Environ Manage 206:178–191. https://doi.org/10.1016/j.jenvman.2017.10.026
15. Goldsmith BR, Esterhuizen J, Liu J-X et al (2018) Machine learning for heterogeneous catalyst design and discovery. AIChE J 64:2311–2323. https://doi.org/10.1002/aic.16198
16. Fernandez M, Barron H, Barnard AS (2017) Artificial neural network analysis of the catalytic efficiency of platinum nanoparticles. RSC Adv 7:48962–48971. https://doi.org/10.1039/C7RA06622H
17. Rosales-Colunga LM, González-García R, De León Rodríguez A (2010) Estimation of hydrogen production in genetically modified E. coli fermentations using an artificial neural network. Int J Hydrogen Energy 35(13186–13192. https://doi.org/10.1016/j.ijhydene.2010.08.137
18. González-Grimaldo RA, Cuevas-Tello JC (2008) Analysis of time series with artificial neural networks. In: 2008 Seventh Mexican International conference on artificial intelligence. IEEE, pp 131–137
19. Serra JM, Corma A, Argente E et al (2003) Neural networks for modelling of kinetic reaction data applicable to catalyst scale up and process control and optimisation in the frame of combinatorial catalysis. Appl Catal A Gen 254:133–145. https://doi.org/10.1016/S0926-860X(03)00283-7
20. González-García R, Rico-Martínez R, Wolf W et al (2001) Characterization of a two-parameter mixed-mode electrochemical behavior regime using neural networks. Phys D Nonlinear Phenom 151:27–43. https://doi.org/10.1016/S0167-2789(00)00219-0
21. González-García R, Rico-Martínez R, Kevrekidis IG (1998) Identification of distributed parameter systems: a neural net based approach. Comput Chem Eng 22:S965–S968. https://doi.org/10.1016/S0098-1354(98)00191-4
22. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313:504–507. https://doi.org/10.1126/science.1127647
23. Uzair M, Shafait F, Ghanem B, Mian A (2018) Representation learning with deep extreme learning machines for efficient image set classification. Neural Comput Appl 30:1211–1223. https://doi.org/10.1007/s00521-016-2758-x

24. Yu J, Tan M, Zhang H et al (2019) Hierarchical deep click feature prediction for fine-grained image recognition. IEEE Trans Pattern Anal Mach Intell 8828:1–1. https://doi.org/10.1109/TPAMI.2019.2932058

25. Yu J, Li J, Yu Z, Huang Q (2019) Multimodal transformer with multi-view visual representation for image captioning. IEEE Trans Circuits Syst Video Technol 14:1. https://doi.org/10.1109/TCSVT.2019.2947482

26. Yu T, Yu J, Yu Z, Tao D (2020) Compositional attention networks with two-stream fusion for video question answering. IEEE Trans Image Process 29:1204–1218. https://doi.org/10.1109/TIP.2019.2940677

27. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117. https://doi.org/10.1016/j.neunet.2014.09.003

28. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2:359–366. https://doi.org/10.1016/0893-6080(89)90020-8

29. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals, Syst 2:303–314. https://doi.org/10.1007/BF02551274

30. Hornik K (1991) Approximation capabilities of multilayer feedforward networks. Neural Netw 4:251–257. https://doi.org/10.1016/0893-6080(91)90009-T

31. Liang Nan-Ying, Huang Guang-Bin, Saratchandran P, Sundararajan N (2006) A fast and accurate online sequential learning algorithm for feedforward networks. IEEE Trans Neural Netw 17:1411–1423. https://doi.org/10.1109/TNN.2006.880583

32. Huang G-B, Chen L, Siew C-K (2006) Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans Neural Netw 17:879–892. https://doi.org/10.1109/TNN.2006.875977

33. Lan Y, Soh YC, Huang G-B (2010) Constructive hidden nodes selection of extreme learning machine for regression. Neurocomputing 73:3191–3199. https://doi.org/10.1016/j.neucom.2010.05.022

34. Kapanova KG, Dimov I, Sellier JM (2018) A genetic approach to automatic neural network architecture optimization. Neural Comput Appl 29:1481–1492. https://doi.org/10.1007/s00521-016-2510-6

35. Wang R, Xie H, Feng J et al (2019) Multi-criteria decision making based architecture selection for single-hidden layer feedforward neural networks. Int J Mach Learn Cybern 10:655–666. https://doi.org/10.1007/s13042-017-0746-9

36. Hagan MT, Menhaj MB (1994) Training feedforward networks with the Marquardt algorithm. IEEE Trans Neural Networks 5:989–993. https://doi.org/10.1109/72.329697

37. He S, Sepehri N, Unbehauen R (2001) Modifying weights layer-by-layer with levenberg-marquardt backpropagation algorithm. Intell Autom Soft Comput 7:233–247. https://doi.org/10.1080/10798587.2000.10750281

38. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. Parallel Distrib Process Explor Microstruct Cognit 1:318–362

39. Guang-Bin Huang, Qiu-Yu Zhu, Chee-Kheong Siew (2004) Extreme learning machine: a new learning scheme of feedforward neural networks. In: 2004 IEEE international joint conference on neural networks (IEEE Cat. No.04CH37541). IEEE, pp 985–990

40. Anderson PG, Ge M, Raghavendra S et al (1998) Using quasirandom weights in neural networks. Intell Autom Soft Comput 4:61–71. https://doi.org/10.1080/10798587.1998.10750722

41. Pao Y-H, Park G-H, Sobajic DJ (1994) Learning and generalization characteristics of the random vector functional-link net. Neurocomputing 6:163–180. https://doi.org/10.1016/0925-2312(94)90053-1

42. Schmidt WF, Kraaijveld MA, Duin RPW (1992) Feedforward neural networks with random weights. In: Proceedings, 11th IAPR international conference on pattern recognition. Vol II. Conference B: pattern recognition methodology and systems. IEEE Comput. Soc. Press, pp I–4

43. Cao W, Wang X, Ming Z, Gao J (2018) A review on neural networks with random weights. Neurocomputing 275:278–287. https://doi.org/10.1016/j.neucom.2017.08.040

44. Huang G-B, Zhu Q-Y, Siew C-K (2006) Extreme learning machine: theory and applications. Neurocomputing 70:489–501. https://doi.org/10.1016/j.neucom.2005.12.126

45. Stone M (1974) Cross-validatory choice and assessment of statistical predictions. J R Stat Soc Ser B 36:111–133. https://doi.org/10.1111/j.2517-6161.1974.tb00994.x

46. LeCun Y, Cortes C, Burges CJC “MNIST handwritten digit database”. http://yann.lecun.com/exdb/mnist/

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.