

# Time series forecasting by recurrent product unit neural networks

F. Fernández-Navarro<sup>1</sup> · Maria Angeles de la Cruz<sup>2</sup> · P. A. Gutiérrez<sup>3</sup> ·  
A. Castaño<sup>4</sup> · C. Hervás-Martínez<sup>3</sup>

Received: 8 August 2015 / Accepted: 12 July 2016 / Published online: 22 July 2016  
© The Natural Computing Applications Forum 2016

**Abstract** Time series forecasting (TSF) consists on estimating models to predict future values based on previously observed values of time series, and it can be applied to solve many real-world problems. TSF has been traditionally tackled by considering autoregressive neural networks (ARNNs) or recurrent neural networks (RNNs), where hidden nodes are usually configured using additive activation functions, such as sigmoidal functions. ARNNs are based on a short-term memory of the time series in the form of lagged time series values used as inputs, while RNNs include a long-term memory structure. The objective of this paper is twofold. First, it explores the potential of multiplicative nodes for ARNNs, by considering product unit (PU) activation functions, motivated by the fact that

PUs are specially useful for modelling highly correlated features, such as the lagged time series values used as inputs for ARNNs. Second, it proposes a new hybrid RNN model based on PUs, by estimating the PU outputs from the combination of a long-term reservoir and the short-term lagged time series values. A complete set of experiments with 29 data sets shows competitive performance for both model proposals, and a set of statistical tests confirms that they achieve the state of the art in TSF, with specially promising results for the proposed hybrid RNN. The experiments in this paper show that the recurrent model is very competitive for relatively large time series, where longer forecast horizons are required, while the autoregressive model is a good selection if the data set is small or if a low computational cost is needed.

✉ F. Fernández-Navarro  
i22fenaf@uco.es; fafermandez@uloyola.es

Maria Angeles de la Cruz  
Maria.Angeles.de.la.cruz@esa.int

P. A. Gutiérrez  
pagutierrez@uco.es

A. Castaño  
adiel2008@gmail.com

C. Hervás-Martínez  
chervas@uco.es

- <sup>1</sup> Department of Quantitative Methods, Universidad Loyola Andalucía, Seville, Spain
- <sup>2</sup> Advanced Concepts Team, European Space Research and Technology Centre (ESTEC), European Space Agency (ESA), Noordwijk, Netherlands
- <sup>3</sup> Department of Computer Science and Numerical Analysis, University of Córdoba, Córdoba, Spain
- <sup>4</sup> Department of Computer Science, Universidad Politécnica Salesiana, Quito, Ecuador

**Keywords** Time series forecasting · Product unit neural networks · Recurrent neural networks · Evolutionary neural networks

## 1 Introduction

Times series (TS) consist on a succession of data values chronologically sorted that belongs to a magnitude or phenomenon that has been sampled at a certain rate. An example of TS could be the evolution of the maximum daily temperature, the unemployment rate of a country or the amplitude of the seismic waves of an earthquake. Time series is present in most of the science fields like flood forecasting [1], weather forecasting [2] or energy consumption [3].

Nowadays, TS research is focused on TS analysis (TSA) and TS forecasting (TSF). The goal of TSA is to extract the main features and characteristics that describe the

underlying phenomena, while the objective of TSF is to find a function to predict the next value of the time series using its  $p$  lagged values. It is worth mentioning that the TSA is primordial to reach a good accuracy for TSF, so TSA is usually applied as a preprocessing step in TSF. Finally, TSF can be tackled by univariate or multivariate models. This paper focuses on the former type of models.

Artificial neural networks (ANNs) are a very popular machine learning (ML) tool used for TSF [4]. Feedforward neural networks (FFNNs) are the most common and simplest type of ANNs, where the information moves in a forward direction. For example, the time delay neural network (TDNN) consists on a FFNN whose inputs are the delayed values of the TS [5]. Instead, recurrent neural networks (RNNs) are based on a different architecture where the information through the system moves constituting a direct cycle [6]. This cycle can store information from previous data in the internal memory of the network, which can be useful for certain kind of applications. RNNs have shown competitive performance in several real-world problems. For example, RNNs based on the method of penalty functions were proposed to solve the bilevel linear programming problem [7]. Furthermore, a RNN approach was also recently proposed to robustly model predictive control for constrained discrete-time nonlinear systems with unmodeled dynamics affected by bounded uncertainties [8]. One example of RNN is the long short-term memory neural network (LSTMNN) [9], whose main characteristic is the capability of its nodes to remember a time series value for an arbitrary length of time. Echo state networks (ESNs) are RNNs whose architecture includes a random number of neurons whose interconnections are also randomly decided. This provides the network with a long-term memory and a competitive generalisation performance [10–12]. In this direction, the minimal complexity which is required for the construction of a competitive RNN is investigated in [12], concluding that a simple deterministically constructed cycle reservoir is comparable to the standard ESN methodology. From the analysis of ANNs in the context of TSF, it can be derived that one of the main differences between FFNNs and RNNs lies on their storage capacity. RNNs have a long-term memory because of the architecture of the model, whereas the memory of FFNNs is provided by the lagged terms at the input of the network.

On the other hand, the parameter estimation algorithm is also very important when analysing the different proposals. The more complex the structure of a neural network is, the more challenging its weight matrix estimation turns. Traditional backpropagation (BP) algorithms can result in a very high computational cost, specially when dealing with complex nonlinear error surfaces [13]. The extreme learning machine (ELM) is an example of an algorithm that

can estimate the parameters of a FFNN model efficiently [14]. It is a popular algorithm that determines the hidden layer parameters randomly and the output layer ones by using the Moore–Penrose (MP) generalised inverse [15], providing a better generalisation performance than traditional gradient-based learning algorithms for some problems.

This paper is focused on product unit neural networks (PUNNs) and its application on TSF. The basis function of the hidden neurons of PUNNs is the product unit (PU) function, where the output of the neuron is the product of their inputs raised to real-valued weights. PUNNs are an alternative to sigmoidal neural networks and are based on multiplicative nodes instead of additive ones [16]. Durbin and Rumelhart [16] empirically determined that the information capacity of a single product unit (as measured by its capacity for learning random boolean patterns) is approximately  $3N$ , compared to  $2N$  for a single summation unit, where  $N$  is the number of inputs to the units. This model has the ability to express strong interactions between input variables, providing large variations at the output from small variations at the inputs. Consequently, it has increased storage information capability and promising potential for TSF. However, they result in a highly convoluted error function, plenty of local minima. This handicap makes convenient the use of global search algorithms, such as genetic algorithms [17, 18], evolutionary algorithms [19] or swarm optimisation algorithms [20], in order to find the parameters minimising the error function. PUNNs have been widely used in classification [21] and regression problems [22], but scarcely applied to TSF, with the exception of some attempts on hydrological TSA [23, 24]. It is important to point out that, in TSF, there is an autocorrelation between the lagged values of the series. In this way, theoretically, PUNNs should constitute an appropriate model for TSF because they can easily model the interactions (correlations) between the lagged values of the time series.

The first goal of this paper is to evaluate the performance of autoregressive product unit neural networks (ARPUNNs) on TSF. The ARPUNN model should yield high performance for TSF, as it fulfils the requirements that allow the modelling of TS: ability to express the interactions between inputs and increased storage capability. However, as mentioned above, long-term memory ANNs usually obtain better results than FFNNs [25]. For this reason, a second goal of this work is to propose a hybrid ANN combining an ARPUNN with a reservoir network [26] with the objective of increasing the final storage capability. The short-term memory is provided by the different lags of the TS included in the input layer, and the long-term memory is supplied by a reservoir network included as one of the inputs of the system. The final model is called recurrent product unit neural network (RPUNN).

From the point of view of the learning algorithm, the complex error surface associated with PUs implies serious difficulties for searching the best parameters minimising the error function. A hybrid algorithm is proposed in this work to alleviate these difficulties. It combines the exploration abilities of global search algorithms with the exploitation ones of local search methods. The covariance matrix adaptation evolution strategy (CMA-ES) algorithm [27, 28] is used to calculate the parameter values of the hidden layer, whereas the weights of the output layer are determined by means of the MP generalised inverse. CMA-ES has been successfully applied to estimate the weights of neural networks with fixed topologies [29]. Recently, the CMA-ES was also modified to allow the simultaneous determination of topology and weights in neural networks [30]. This combination of an EA and a local search method provides us with a hybrid training of a hybrid model, able to afford the difficulties of TSF and obtain a competitive performance. Although some of the previous works also consider the combination of evolutionary algorithms and PUNNs [21, 22], both the model structure and the training algorithm considered in this paper are different and specifically adapted to TSF.

The model proposed can be seen as a generalisation of the multiplicative neuron model ANN (MNN-ANN) [31]. MNM has only one neuron in the hidden layer. Therefore, the problem of determining the number of neurons in hidden layer is automatically solved when MNM is employed [32]. Previous works have tackled the estimation of MNN-ANN parameters by cooperative random learning particle swarm optimisation (CRPSO), PSO, backpropagation algorithm and genetic algorithms [33]. The model was extended and generalised based on the concept of generalised mean of all multiplicative inputs [34]. Furthermore, the recurrent multiplicative neuron artificial neural network model (RMNM-ANN) was also proposed recently. The RMN-ANN incorporates not just AR terms but moving average terms in the model [35]. The model proposed in this paper provides a greater flexibility for modelling the complex interactions of real-world time series but also have a higher computational complexity if it is compared to the standard MNN-ANN model.

Summarising, the main contributions of this paper are the following:

- The use of PU basis functions in the field of TSF. PU basis functions were already investigated in the field of regression and classification [16, 21, 22]. However, their mathematical expression makes them specially interesting for addressing TSF problems (due to their increased storage capability and their ability to model correlations in the input space).

- A new hybrid RNN combining reservoir computing (RC) models (specifically, ESNs) and FFNs with PU basis functions, with the goal of providing to the model with a long-term memory.
- The use of the CMA-ES algorithm for the parameter estimation of the proposed models. Neural networks based on PU basis functions tend to generate complex error functions with multiple local minima. The use of this standard genetic algorithm allows the proposed models to converge to global minima.

This paper is organised as follows: Sect. 2 describes the ANN hybrid model proposed in this paper to be applied in TSF. Section 3 explains the hybrid search algorithm designed to get the parameters which optimise the error function. Sections 4 and 5 explain the experiments that were carried out and the results obtained. Finally, Sect. 6 summarises the conclusions of this work.

## 2 Models

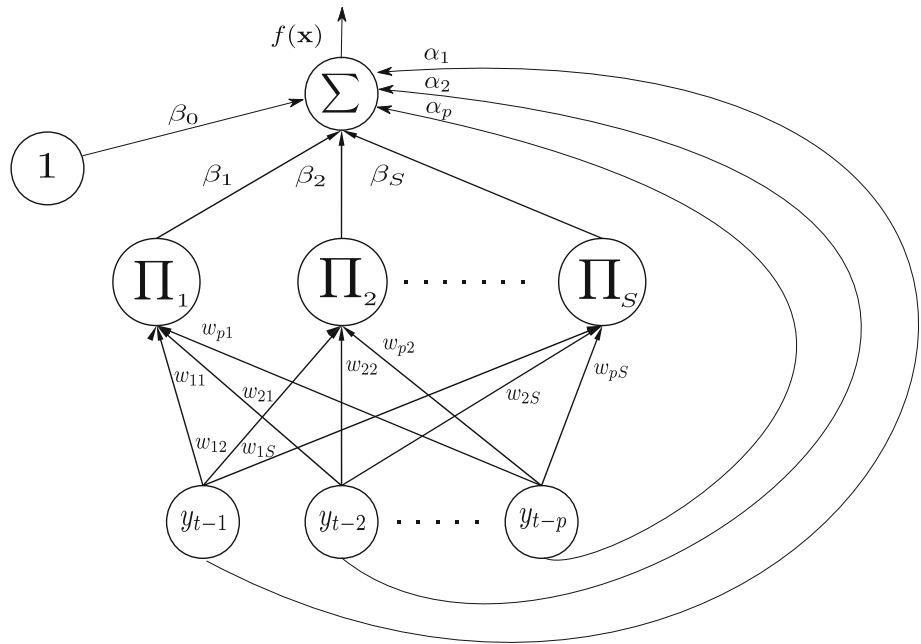
In this section, we first introduce an ARPUNN model, which is then extended by considering a reservoir to result in the RPUNN model. The models proposed addressed the TSF problem. This problem is mathematically formulated as follows. Let  $\{y_n\}_{n=1}^{N+p}$  be a TS to be predicted, where  $N + p$  values are given for training. In this way, the function  $f: \mathbb{R}^p \rightarrow \mathbb{R}$  is estimated from a training set of  $N$  patterns,  $\mathbf{D} = (\mathbf{X}, \mathbf{Y}) = \{(\mathbf{x}_n, y_{n+p})\}_{n=1}^N$  where  $\mathbf{x}_n = \{y_n, \dots, y_{n+p-2}, y_{n+p-1}\}$  is the vector of input characteristics ( $p$  past values of the TS) taking values in the space  $\Omega \subset \mathbb{R}^p$ , and the label,  $y_{n+p}$ , is the value of the TS for the  $n + p$  instant. Both models are explained in the following subsections.

### 2.1 Short memory model: autoregressive product unit neural network (ARPUNN)

This section presents the first model proposed to address the TSF problem, the so-called ARPUNN. The suggested architecture is based on considering PUs as the basis functions for the hidden layer of the network. PUNN models have the ability to express strong interactions between the input variables. The model is composed by an input, hidden and output layers. The input layer has  $p$  input units that correspond to the lagged values of the TS providing the network with a short memory. The hidden layer of the network is composed by  $S$  PUs, and the output layer contains only one neuron. A representation of model proposed is shown in Fig. 1.

The final model is linear in the basis function space together with the initial variables. A similar architecture

**Fig. 1** Architecture of the autoregressive product unit neural network (ARPUNN)



(which is usually referred to as skip-layer connections) was also considered for classification in previous works for PUs [36]. The TS value is estimated by  $\hat{y}_{n+p} = f(\mathbf{x}_n, \boldsymbol{\theta}) : \mathbb{R}^p \rightarrow \mathbb{R}$ , where the final output of the model is defined as:

$$f(\mathbf{x}_n, \boldsymbol{\theta}) = \beta_0 + \sum_{s=1}^S \beta_s B_s(\mathbf{x}_n, \mathbf{w}_s) + \sum_{k=1}^p \alpha_k y_{n+p-k}, \quad (1)$$

where  $\beta_s \in \mathbb{R}$  denotes the weight of the connection between the hidden neuron  $s$  and the output neuron ( $s = 1, 2, \dots, S$ ), leading the structure that provides the nonlinear contribution of the inputs. The  $\boldsymbol{\beta}$  vector includes all the parameters connecting the hidden with the output layer and the bias  $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \dots, \beta_S) \in \mathbb{R}^{S+1}$ . The linear contribution of the inputs is controlled by  $\alpha_k$  which is the weight of the connection between the input  $k$  and the output layer ( $k = 1, 2, \dots, p$ ). The vector  $\boldsymbol{\alpha}$  contains all the parameters connecting the input and the output layer,  $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_p) \in \mathbb{R}^p$ . Another kind of weights,  $\mathbf{w}_s \in \mathbb{R}^p$ , represents the connections of the hidden neuron  $s$  and the input layer. The  $\boldsymbol{\theta}$  vector contains the full parameter vector ( $\boldsymbol{\theta} = \{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_S, \boldsymbol{\beta}, \boldsymbol{\alpha}\}$ ). Finally,  $B_s(\mathbf{x}_n, \mathbf{w}_s) : \mathbb{R}^p \rightarrow \mathbb{R}$  represents the output of the  $s$ th PU basis function, and it is defined as:

$$B_s(\mathbf{x}_n, \mathbf{w}_s) = \prod_{i=1}^p (y_{n+p-i})^{w_{is}}, \quad s = \{1, \dots, S\}, \quad (2)$$

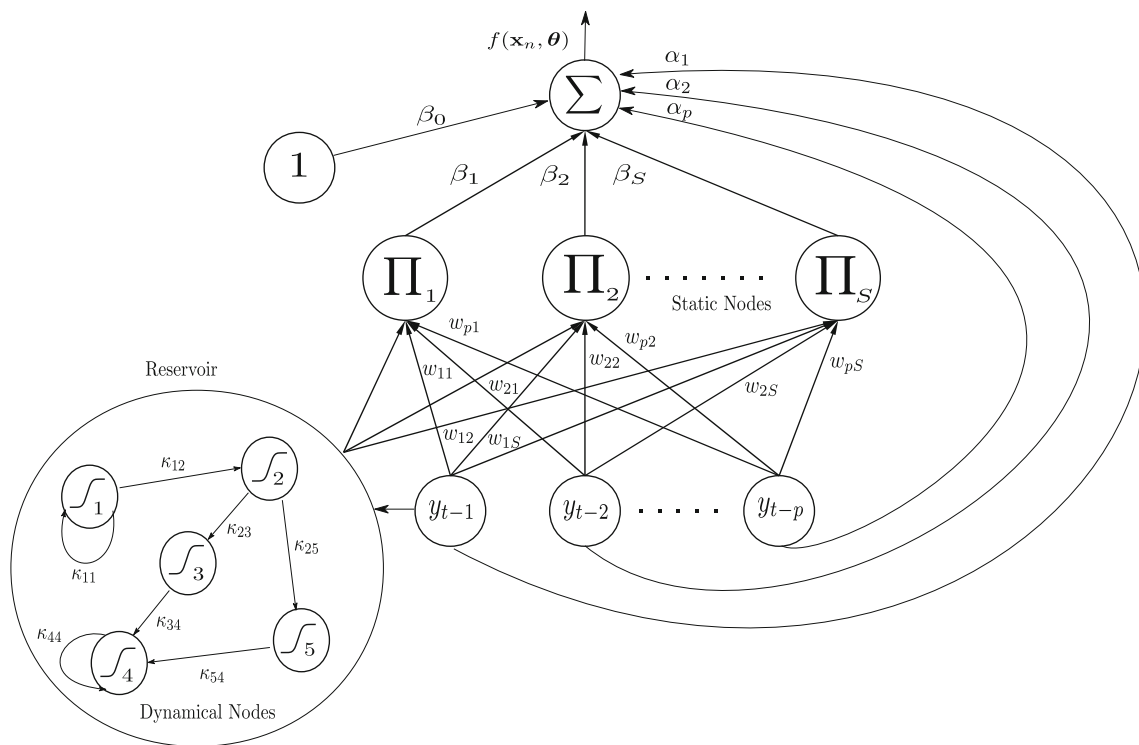
where  $w_{is} \in \mathbb{R}$  is the weight of the connection between the  $i$ th input node and the  $s$ th basis function and  $y_{n+p-i}$  denotes the  $i$ th lagged past value of the TS.

### 2.2 Long memory model: recurrent product unit neural network (RPUNN)

In this section, the long memory model is presented (called recurrent product unit neural network, RPUNN). The RPUNN model reuses the network architecture of the ARPUNN model. One aspect that should be considered on TSF is the memory or the amount of information that can be stored in the network. Traditionally, ANNs with longer memory have an enhanced performance for TSF [25]. The main difference between ARPUNN and RPUNN lies in the inclusion on a new structure as an input, a reservoir network. The reservoir network provides the whole model with long-term and dynamic memory. The structure of the RPUNN is depicted in Fig. 2.<sup>1</sup>

As can be seen, the network inherits the architecture of ARPUNN with the linear and nonlinear combination of the inputs described in the previous section. The output layer contains only one neuron, while the hidden layer of the network is composed by  $S$  neurons with the PU basis function. The input layer considered has  $p + m$  neurons that correspond to the  $p$  lagged values of the TS plus the  $m$  outputs of the reservoir network. The  $p$  lagged values provide the network with the short memory. The reservoir part is formed by a set of  $m$  nodes, and the output of each of these nodes is considered as an input to the PUs, providing

<sup>1</sup> For the sake of clarity, reservoir representation is simplified: there is a link between each reservoir node and each PU, and all reservoir nodes receive  $y_{t-1}$  time series value as input. The interconnections between reservoir nodes are random. Internal connections of the reservoir are given by  $\boldsymbol{\kappa}$ .



**Fig. 2** Architecture of the recurrent product unit neural network (RPUNN)

the whole structure with a dynamic memory. The only input considered for the reservoir is the first lagged value of the TS. The estimated TS value is defined by the final output of the model,  $\hat{y}_{n+p} = f(\mathbf{x}_n, \boldsymbol{\theta}) : \mathbb{R}^{m+p} \rightarrow \mathbb{R}$ , as follows:

$$f(\mathbf{x}_n, \boldsymbol{\theta}) = \beta_0 + \sum_{s=1}^S \beta_s B_s(\mathbf{x}_n, \boldsymbol{\psi}^{(n)}, \mathbf{w}_s) + \sum_{k=1}^p \alpha_k y_{n+p-k} \tag{3}$$

where  $\boldsymbol{\psi}^{(n)} \in \mathbb{R}^m$  is the reservoir state vector for time  $n$ , and  $\boldsymbol{\theta} = \{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_S, \boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\kappa}\}$  represents the set of the network weights, composed by the vectors  $\boldsymbol{\beta} \in \mathbb{R}^S$  and  $\boldsymbol{\alpha} \in \mathbb{R}^p$  (previously defined),  $\mathbf{w}_s \in \mathbb{R}^{m+p}$ , which represents the connections of the hidden neurons and the input layer,  $s = 1, \dots, S$ , and, finally, the matrix of the connections for the reservoir network,  $\boldsymbol{\kappa} \in \mathbb{R}^{m \times (m+2)}$ . At last,  $B_s(\mathbf{x}_n, \boldsymbol{\psi}^{(n)}, \mathbf{w}_s) : \mathbb{R}^{m+p} \rightarrow \mathbb{R}$  represents the basis function considered in the hidden layer yielding the following nonlinear output for the model:

$$B_s(\mathbf{x}_n, \boldsymbol{\psi}^{(n)}, \mathbf{w}_s) = \prod_{i=1}^p (y_{n+p-i})^{w_{is}} \prod_{j=p+1}^{p+m} (\psi_j^{(n)})^{w_{js}} \tag{4}$$

where  $s = 1, \dots, S$ ,  $\mathbf{w}_s = (w_{1s}, \dots, w_{ps}, w_{(p+1)s}, \dots, w_{(p+m)s}) \in \mathbb{R}^{m+p}$  is the hidden layer weight vector,  $w_{is} \in \mathbb{R}$  is the weight of the connection between the input neuron  $i$

and the hidden neuron  $s$ ,  $i = 1, \dots, p$ , and  $w_{(p+j)s}$  is the weight of the connection between the  $j$ th reservoir node and the hidden neuron  $s$ ,  $j = 1, \dots, m$ . Finally,  $\psi_j^{(n)}$  represents the output of the  $j$ th reservoir node at time  $n$ ,  $j = 1, \dots, m$  and the corresponding vector is  $\boldsymbol{\psi}^{(n)} = \{\psi_1^{(n)}, \dots, \psi_m^{(n)}\}$ .

The reservoir consists of a sparsely connected group of nodes, where each neuron output is randomly assigned to the input of another neuron. This allows the reservoir reproducing specific temporal patterns. All the reservoir nodes are sigmoidal nodes, as this model is more adequate in order to keep the long-term memory:

$$\begin{aligned} \psi_j^{(n)} &= R_j(\boldsymbol{\psi}^{(n-1)}, \boldsymbol{\kappa}_j) \\ &= \sigma\left(\kappa_{0j} + \sum_{i=1}^m \kappa_{ij} \psi_i^{(n-1)} + \kappa_{(m+1)j} y_{n-1}\right), \end{aligned} \tag{5}$$

where  $\sigma(x) = 1/(1 + \exp(-x))$  is the sigmoidal activation function and  $\boldsymbol{\kappa}_j$  is the vector of parameters corresponding to the  $j$ th reservoir neuron

$$\boldsymbol{\kappa}_j = \{\kappa_{0j}, \kappa_{1j}, \dots, \kappa_{mj}, \kappa_{(m+1)j}\},$$

with  $m + 2$  elements. As can be observed, self-connections are allowed. The internal structure of the reservoir is randomly fixed and kept constant during the learning process, in the same vein than it is done with ESNs [11].

### 3 Parameter estimation

This section discusses the training algorithm proposed to fit ARPUNN and RPUNN parameters. As stated above, PUNNs exhibit a highly convoluted error surface, which can easily make the training algorithm get stuck in local minima and, in consequence, avoid that the optimum parameters are obtained. In general, this can be overcome by using global search algorithms, but instead they can be slow to reach the global optimum. The method considered in this work focuses in obtaining a trade-off between both extremes, which is achieved by a hybrid algorithm. The parameter set to be optimised in the ARPUNN model is

$$\theta = \{\beta, \alpha, \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_S\},$$

which is composed by the set of weights of the hidden layer nodes ( $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_S$ ) and the set of weights of the output layer,  $\beta$  and  $\alpha$ . In the case of the RPUNN, it is also required to estimate the values of the parameters included in the vector  $\kappa$ , i.e. the weights of the reservoir interconnections.

The beginning of the algorithm involves the CMA-ES method as a global optimisation procedure [27]. CMA-ES is an evolutionary algorithm for difficult nonlinear non-convex optimisation problems in continuous domain. The evolution strategy defined in this algorithm is based on the use of a covariance matrix that represents the pairwise dependencies between the candidate values of the variables to be optimised. The distribution of the covariance matrix is updated by means of the covariance matrix adaptation method that attempts to learn a second-order model of the cost function similar to the optimisation made in the quasi-Newton methods [37]. The CMA-ES has several invariance properties and does not require a complex parameter tuning. In this paper, the uncertainty is undertaken as proposed in [38] and a subtractive update of the covariance matrix is done as in [28]. Another consideration is to adapt only the diagonal of the covariance matrix for a number of initial iterations, as stated in [39], leading to a faster learning. The upper and lower bounds of the parameters are handled as proposed in [38]. The standard deviation considered in the initialisation stage is set to 0.3. For both ARPUNN and RPUNN models, the target parameters under optimisation by the CMA-ES algorithm are the weights from the input layer to the hidden layer  $\{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_S\}$ . The hybrid algorithm starts by randomly generating the values for these weights. Although the rest of the weights are needed to obtain the cost function, they can be analytically calculated by using the MP generalised inverse, as done in the ELM [15]. This process has to be performed on each iteration of the CMA-ES algorithm and for each individual of the population. Let  $\phi = (\beta_1, \dots, \beta_S, \alpha_1, \dots, \alpha_p)^T$

denote the weights of the links connecting hidden and output layers. The calculation of  $\phi$  can be done by taking into account that the system is linear if the basis function space is considered. In this way, the nonlinear system can be converted into a linear system:

$$\mathbf{Y} = \mathbf{H}\phi, \quad (6)$$

where  $\mathbf{H} = \{h_{ij}\}$  ( $i = 1, \dots, N$  and  $j = 1, \dots, S + p$ ) represents the hidden and input layers output matrix: for  $j = 1, \dots, S$ ,  $h_{ij} = B_j(\mathbf{x}_i, \mathbf{w}_j)$  (for the ARPUNN model) or  $h_{ij} = B_j(\mathbf{x}_i, \psi^{(i)}, \mathbf{w}_j)$  (for the RPUNN model); for  $j = S + 1, \dots, S + p$ ,  $h_{ij} = y_{i+p-j}$ . Finally, the determination of  $\phi$  can be obtained by finding the least-square solution of the equation:

$$\hat{\phi} = \mathbf{H}^\dagger \mathbf{Y} \quad (7)$$

where  $\mathbf{H}^\dagger$  is the MP generalised inverse of the matrix  $\mathbf{H}$ . The solution provided by this method is unique, and it has the smallest norm within all least-square solutions. In addition, it obtains a high generalisation performance that decreases the time required to learn the sequence as states [40].

The parameters of the reservoir for the RPUNN model are randomly fixed before starting the CMA-ES optimisation and then kept constant for the whole evolution, given that, otherwise, the computational cost and the complexity of the final model would increase significantly. Sparsity is achieved by randomly setting to 0 a percentage (in our case,  $\sim 90\%$ ) of the weights for the connections between reservoir nodes (i.e.  $\kappa_{ij} = 0$ , for some randomly selected  $i$  and  $j$  values,  $i = 1, \dots, m$ ,  $j = 1, \dots, m$ ). The reservoir is composed of 30 sigmoidal nodes that are not considered in the value computed as number of hidden neurons (NHNs) of the experimental part. The spectral radius  $\alpha$  is set to 0.5. The weights of the reservoir are randomly initialised from a uniform distribution over  $[-1, 1]$ . After this initialisation, the weights of the reservoir are normalised to a spectral radius  $\alpha < 1$ , by scaling them using the factor  $\alpha/|\lambda_{\max}|$ , where  $\lambda_{\max}$  is the largest eigenvalue of the vector of weights (as suggested in [41]). Finally, isolated reservoir neurons are not allowed in the model to guarantee the consistency of the network.

### 4 Experiments

In order to analyse the performance of the proposed methods, an experimental study was carried out. The TS data selected, the metrics considered to evaluate the performance of the models and the algorithms used for comparison purposes are described in the following subsections.

#### 4.1 Data sets selected

The time series used for the experimental set-up belongs to the NNGC1, Acont, B1dat, D1dat and Edat forecasting competitions.<sup>2</sup> These data sets were selected based on the TSF task proposed in [42]. A total of 29 time series available in the KEEL-data set repository<sup>3</sup> [43] have been considered.

The NNGC1 data sets contain transportation data, including highway traffic, traffic data of cars in tunnels, traffic at automatic payment systems on highways, traffic of individuals on subway systems, domestic aircraft flights, shipping imports, border crossings, pipeline flows and rail transportation that are sampled with a weekly, daily or hourly frequency. Specifically, 24 time series belonging to this set have been used. In the case of the Acont data set, only one time series has been used that contains a laser univariate time record of a single observed quantity, measured in a physics laboratory experiment. On the other hand, B1dat is a multivariate data set recorded from a patient in the sleep laboratory of the Beth Israel Hospital, in Boston, Massachusetts. The lines in the original data set file are spaced by 0.5 s. The data set contains three physiological measurements: the first is the heart rate, the second is the chest volume (respiration force), and the third is the blood oxygen concentration (measured by ear oximetry). The Edat is a set of measurements of the light curve (time variation of the intensity) of the variable white dwarf star PG1159-035 during March 1989, and it was recorded by the Whole Earth Telescope (a coordinated group of telescopes distributed around the Earth that permits the continuous observation of an astronomical object). Finally, the Ddat is a time series data set generated synthetically by a computer. Table 1 describes the main features of the time series used in the experiments.

Following the procedures used in [42], the augmented Dickey–Fuller [44] test has been applied to the series in order to consider only stationary series. In addition, linearity test has been performed during the building procedure in order to exclude data sets that exhibit linear properties. The lag set was adjusted using an uniform embedding with a constant time lag through adjacent input channels. Specifically, four lagged values were considered. Although other alternative embeddings could be considered [45], this decision was taken to simplify the experimentation.

Finally, the data sets have been preprocessed to adapt the inputs to the mathematical characteristics of the PU-based models: input variables have been scaled in the rank

**Table 1** Characteristics of the benchmark data sets, alphabetically ordered

| Data set        | #Attributes | #Patterns |
|-----------------|-------------|-----------|
| Acont_1_2000    | 4           | 1995      |
| B1dat_1_2000    | 4           | 1995      |
| B1dat_2_2000    | 4           | 1995      |
| D1dat_1_2000    | 4           | 1995      |
| Edat_1_1661     | 4           | 1655      |
| NNGC1_D1_V1_002 | 4           | 1175      |
| NNGC1_D1_V1_003 | 4           | 430       |
| NNGC1_D1_V1_004 | 4           | 545       |
| NNGC1_D1_V1_005 | 4           | 430       |
| NNGC1_D1_V1_006 | 4           | 610       |
| NNGC1_D1_V1_007 | 4           | 610       |
| NNGC1_D1_V1_008 | 4           | 540       |
| NNGC1_D1_V1_009 | 4           | 540       |
| NNGC1_D1_V1_010 | 4           | 585       |
| NNGC1_E1_V1_001 | 4           | 370       |
| NNGC1_E1_V1_008 | 4           | 740       |
| NNGC1_E1_V1_009 | 4           | 740       |
| NNGC1_E1_V1_010 | 4           | 650       |
| NNGC1_F1_V1_001 | 4           | 965       |
| NNGC1_F1_V1_002 | 4           | 1020      |
| NNGC1_F1_V1_003 | 4           | 1735      |
| NNGC1_F1_V1_004 | 4           | 1735      |
| NNGC1_F1_V1_005 | 4           | 1735      |
| NNGC1_F1_V1_006 | 4           | 1735      |
| NNGC1_F1_V1_007 | 4           | 895       |
| NNGC1_F1_V1_008 | 4           | 895       |
| NNGC1_F1_V1_009 | 4           | 895       |
| NNGC1_F1_V1_010 | 4           | 895       |
| NNGC1_F1_V1_011 | 4           | 895       |

[0.1, 0.9].<sup>4</sup> The experimental design was conducted using a fivefold cross-validation, with 10 repetitions per each fold.

#### 4.2 Metrics considered for evaluation

The metrics considered in this paper are the mean absolute percentage error (MAPE) (in the generalisation set, MAPE<sub>G</sub>) and the number of hidden nodes (NHN). Given that all the models consider fully connected neurons, NHN is a measure of the size of the neural network. Neural networks are very sensitive to this value (generally, large networks require a longer processing time [46]).

<sup>2</sup> Available at <http://www.neural-forecasting-competition.com>.

<sup>3</sup> Which can be found at <http://sci2s.ugr.es/keel/timeseries.php>.

<sup>4</sup> Scaling the input data to positive values is required to avoid having complex numbers as output of the basis function. Additionally, the scaling considered also avoids having inputs equal to zero or one.

### 4.3 Algorithms selected for comparison purposes

In order to evaluate the performance of the RPUNN and ARPUNN models, they have been compared to some of the most promising neural networks models for TSF. Aiming to outline different characteristics of the methods, the compared methods have been grouped in two sets. The main objective behind the first set of models is comparing ARPUNN and RPUNN methods to baseline algorithms. This set is composed by the following methods:

- The minimum complexity echo state network (MCESN) [12]. This model is constructed deterministically unlike the standard ESNs. The architecture used is the simple cycle reservoir (SCR) one due to its competitive accuracy as reported in [12].
- The nonlinear autoregressive neural network (NARNN) proposed in [47].
- The standard echo state network (ESN) [26].
- The extreme learning machine method (ELM) [15].

The second set of models is selected with the purpose of analysing the performance of PU basis functions for TSF. Due to this, the two models proposed are compared to ANN models trained with the same algorithm, but considering other basis functions. The models employed in this set are:

- The nonlinear autoregressive radial basis function neural network (NARRBFNN).
- The nonlinear autoregressive sigmoidal neural network (NARSIGNN).

All the hyperparameters considered in this paper were estimated by a nested fivefold cross-validation procedure. The most important hyperparameter was the NHN, and the corresponding range of possible values considered for model selection depends on the model in the following manner:

- In the case of the NARNN, NARRBFNN, NARSIGNN, ARPUNN and RPUNN algorithms, the experiment was carried out using neural networks with 5, 10, 15 and 20 hidden nodes.
- The MCESN, ESN and ELM algorithms require a higher number of hidden neurons that can supply the network with sufficiently informative random projections [12, 15]. In this case, neural networks with 10, 20, 50, 100, 150, 200 and 300 hidden nodes were considered.

In order to control the overfitting of the models developed in this paper, three approaches have been implemented:

- The output weights of the models were determined through the MP pseudoinverse, ensuring that these parameters have the smallest norm within all least-square solutions.

- As previously discussed, the NHN was determined through a fivefold cross-validation procedure over the training set.
- Different boundaries for the weights connecting the input and the hidden layer were considered:
  - The hidden layer weights of the ARPUNN and RPUNN algorithms are initialised randomly within the interval  $[-5, 5]$ , which is also the boundary considered for their input weights.
  - In the case of the NARRBFNN algorithm, the boundary established for the centroids of the RBFs, as well as the limits for input variable normalisation, were  $[0.1, 0.9]$ .
  - The weights of NARSIGNN and ELM algorithms were randomly initialised in the interval  $[0, 1]$ .

The range considered was determined after preliminary experiments and an analysis of the characteristic of the basis functions of each model. Furthermore, these bounds were already tested in the literature for PUNN models in function approximation and classification problems [36, 48].

Once the test prediction with a given model has been done, some considerations have been considered in order to provide a fair comparison between the different tested algorithms. The architecture of the ESN and MCESN methods requires several cycles until a proper predicted value can be obtained. The first cycles of the model are used to accommodate for a washout of the arbitrary (random or zero) initial reservoir state needed at time 1. For this reason, part of the first samples of the data set need to be discarded. In the experiments undertaken, a discard rate of the 25 % of the first samples of each data set has been considered. This constraint has been applied to every data set independently of the method used for TSF, in order to assure equitable MAPE values.

## 5 Results

For all of the 29 data series, models were trained, predictions were made on the test set, and the MAPE and NHN were computed. A ranking has been established ( $\bar{R}_{\text{MAPE}}$  and  $\bar{R}_{\text{NHN}}$ ) for each method in each data set depending on the value obtained for MAPE and NHN ( $R = 1$  stands for the best performing method and  $R = 8$  for the worst one). The complete table of results has been included in a separated document.<sup>5</sup> Analysing the results obtained for the MAPE from a descriptive point of view, it can be seen that the proposed models in this study achieved a considerable competitive performance. The ARPUNN method obtains

<sup>5</sup> <http://www.uco.es/ayrna/datasets/MAPEresultsNCAA>.



the best results in 2 out of the 29 data sets considered, while the improved version of the ARPUNN, the RPUNN, achieves the best results in 14 out of the 29 data sets tested. With regard to the NHN metric, it is worth mentioning the competitive results achieved by the ARPUNN model. This model yielded the second best ranking for the NHN metric, being outperformed only by MCESN.

Additionally, Table 2 also reports the averaged results over all the series for the methods compared (including the averaged value for the metric and the averaged ranking). As shown in Table 2, the RPUNN model yielded the best mean in MAPE ( $\overline{\text{MAPE}} = 0.1345$  and  $\overline{R}_{\text{MAPE}_G} = 2.2758$ ) followed by the ARPUNN model ( $\overline{\text{MAPE}} = 0.1400$  and  $\overline{R}_{\text{MAPE}_G} = 3.4827$ ). The minimum  $\overline{\text{NHN}}$  is obtained by the MCESN model with a mean of 11.17 followed by the ESN model with a mean of 12.06. In terms of NHN ranking, the best results are obtained by the MCESN model with a 2.86 mean position, followed by the ARPUNN model with a 3.17 mean position. The RPUNN model leads to a  $\overline{\text{NHN}}$  of 15.10 and a  $\overline{R}_{\text{NHN}}$  of 4.79. After the examination of the experimental results,<sup>6</sup> we can conclude that the RPUNN model seems to obtain lower error if the data set to be forecasted is relatively large, because longer forecast horizons are required (note that the best results for RPUNN are obtained when large data sets are evaluated). However, its computational cost is higher than that of ARPUNN (see the end of this section). On the contrary, the ARPUNN model is more precise if the data set is relatively small (shorter memory is required), and it should be selected when the competitive computational cost is a must. A boxplot of the results obtained for the ranking of the  $\text{MAPE}_G$  and NHN is shown in Fig. 3 where it can be appreciated the performance above mentioned.

The significance of the experimental results was assessed by using nonparametric statistical tests. Following the methodology recommended by Demsar [49] for this type of multiple comparisons, we have used Friedman, Nemenyi and Holm statistical tests. Friedman test checks whether there are significant differences in the results, while Nemenyi is used to detect which of all the comparable pairs are significantly different. Holm test can be applied when a control method is considered and corrects the statistics for multiple comparisons. A detailed description of these tests can be found in Zar’s book [50] and in [51–53]. This kind of statistical validation has been extensively applied in time series forecasting [42, 54–56]. The *pre-hoc* Friedman test has been performed with the ranking obtained in the MAPE and NHN of the best models as test variables. The test shows that the effect of the method used for forecasting is statistically significant at a significance level of 10 %, as

**Table 2** Summary of results for MAPE and NHN as the test variables

|          | MAPE                       |                                | NHN                     |                             |
|----------|----------------------------|--------------------------------|-------------------------|-----------------------------|
|          | $\overline{\text{MAPE}}_G$ | $\overline{R}_{\text{MAPE}_G}$ | $\overline{\text{NHN}}$ | $\overline{R}_{\text{NHN}}$ |
| MCESN    | 0.1624                     | 4.82                           | <b>11.17</b>            | <b>2.86</b>                 |
| MCESN    | 0.1624                     | 4.82                           | <b>11.17</b>            | <b>2.86</b>                 |
| NARNN    | 0.1398                     | 3.55                           | 14.37                   | 4.15                        |
| NARRBFNN | 0.2257                     | 7.41                           | 14.82                   | 4.24                        |
| ESN      | 0.1626                     | 5.17                           | <b>12.06</b>            | 3.43                        |
| ELM      | 6.1255                     | 4.89                           | 163.69                  | 7.65                        |
| NARSIGNN | 0.1493                     | 4.3793                         | 16.58                   | 5.68                        |
| ARPUNN   | <i>0.1400</i>              | <i>3.48</i>                    | <i>12.65</i>            | <i>3.17</i>                 |
| RPUNN    | <b>0.1325</b>              | <b>2.27</b>                    | 15.10                   | 4.79                        |

The best result is in bold face and the second one in italics

the confidence interval is  $C_0 = (0, F_{0.10} = 1.809)$  and the *F*-distribution statistical values are  $F^* = 19.47 \notin C_0$  for NHN and  $F^* = 17.40 \notin C_0$  for MAPE. Therefore, null hypothesis is rejected stating that all algorithms perform equally in mean ranking.

According to the previous rejection results, the Nemenyi *post hoc* test has been used in order to compare all the models to each other. The Nemenyi test analyses the performance of different models considering that a model is significantly different if its mean rank differs by at least the critical difference (CD) defined by:

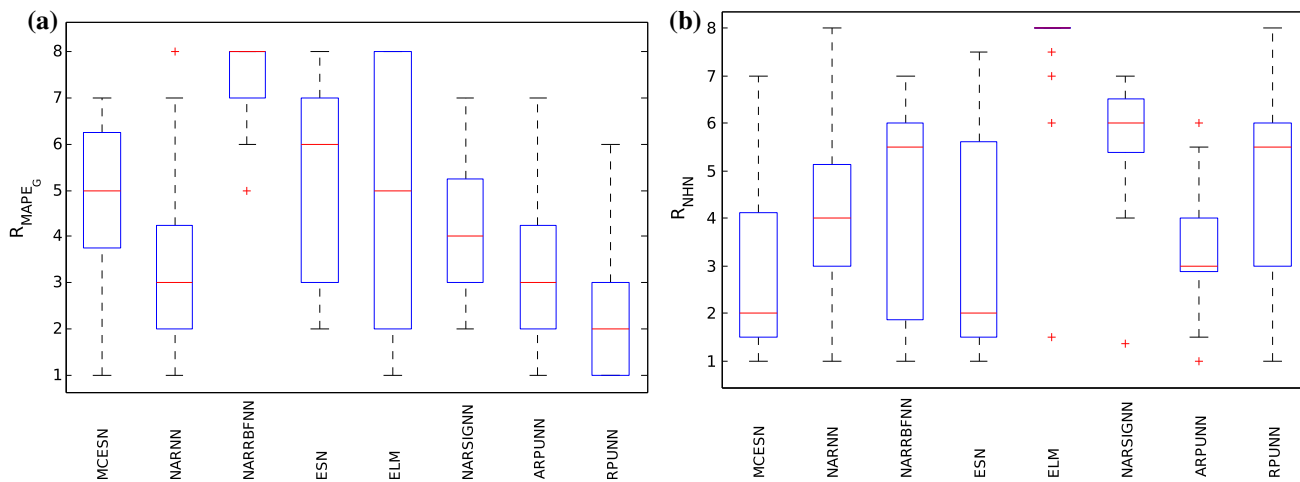
$$CD = q \sqrt{\frac{K(K+1)}{6D}} \tag{8}$$

where *K* and *D* are the number of models and data sets used and *q* is derived from the studentised range statistic divided by  $\sqrt{2}$ . The results of the Nemenyi test for  $\alpha = 0.10$  are shown in Fig. 4, where the CD is shown for MAPE and NHN, and the mean ranking of each algorithm is represented in the scale. When the mean rankings of two algorithms differ more than the CD, then significant differences can be assessed.

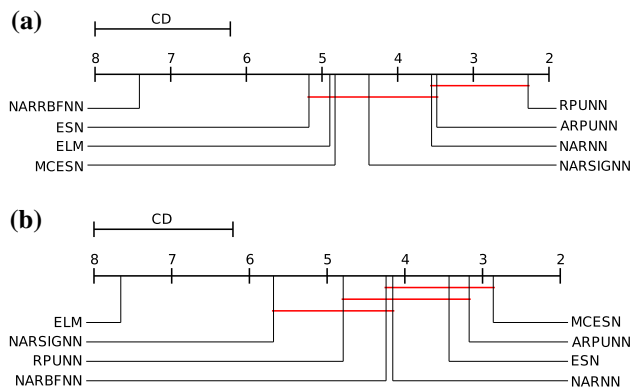
The results of the Nemenyi test for  $\alpha = 0.10$  in the case of the MAPE metric show that the RPUNN model is significantly better than the state-of-the-art models considered in our experiments except ARPUNN and NARNN. Regarding the NHN parameter, there are no significant differences between the models that present the best results: MCESN, ARPUNN, ESN, NARNN and NARRBFNN. The rest of the models present a lower performance due to their complex architecture that requires a higher NHN.

Generally, the results of the Nemenyi test comparing all models to each other in a *post hoc* are not as sensitive as the approach of comparing all the models to a given one, that is used as a control method. This is the philosophy

<sup>6</sup> <http://www.uco.es/ayrna/datasets/MAPEresultsNCAA>.



**Fig. 3** Boxplot for the average ranking of  $MAPE_G$  (MAPE over the generalisation set) and  $NHN$  over the 29 data sets. **a** Test variable:  $R_{MAPE_G}$ , **b** Test Variable:  $R_{NHN}$



**Fig. 4** Ranking test diagrams for the mean generalisation MAPE and  $NHN$  ( $\alpha = 0.10$ ). **a** Nemenyi CD diagram comparing the generalization  $MAPE_{mean}$  rankings of the different methods, **b** Nemenyi CD diagram comparing  $NHN$  mean rankings of the different methods

followed by the Holm test presented now. The results of the Holm test are available in Table 3. The control methods used for the  $MAPE_G$  and  $NHN$  measures are  $RPUNN$  and  $MCESN$ , as they have the best ranking performances, respectively.

The results given by the Holm test show slightly different results than those obtained with the Nemenyi test. In the case of the Holm test, it can be appreciated that the  $RPUNN$  model performs significantly better (in  $MAPE$ ) than the rest of the models. Regarding the  $NHN$  metric, the  $MCESN$  method shows a significant lower complexity than the  $RPUNN$ ,  $NARSIGNN$  and  $ELM$  methods.

Finally, the computational time of the proposed methods is analysed and compared to that of the algorithms already presented in the experimental section. The computational time recorded included the whole training process: cross-validation, training and test. This time is shown in Table 4. The number of hyperparameters of each method is decisive

for the final time spent in running the algorithms, given that they have to be adjusted using a time-consuming cross-validation process. It is clear that the methods proposed in this paper ( $NARRBFNN$ ,  $NARSIGNN$ ,  $ARPUNN$  and  $RPUNN$ ) have a higher computational cost, which is mainly due to the application of the CMA-ES evolutionary algorithm. From the three autoregressive models ( $NARRBFNN$ ,  $NARSIGNN$  and  $ARPUNN$ ), the lowest time is obtained with sigmoidal nodes, followed by PUs. The exponentiation of PUs implies a higher computational time than sigmoidal basis functions, while lower than that of RBFs. On the other hand, the use of the reservoir ( $RPUNN$ ) increases the computational time with respect to  $RPUNN$ .

The application of EAs is a controversial topic in ANN literature. evolutionary algorithms (EAs) and, in general, global search algorithms are computationally expensive especially when compared to local search algorithms, but the evolution of weights enables ANNs to avoid local minima in different data sets without human intervention or multiple repetitions [57], usually leading to better accuracy. In any case,  $ARPUNN$  and  $RPUNN$  are intended for environments where the training time is not a critical feature. Searching in a space with very few restrictions allows more solutions to be explored, but it is also more time-consuming. The improvement of accuracy obtained for the different data sets justifies the computational time.

### 6 Conclusions

This paper proposes two new models of ANNs based on the use of PUs as a basis function for time series forecasting (TSF). The interest on the use of PUs arises from its ability to express strong interactions between input variables, a

**Table 3** Results of the Holm test considering the MAPE<sub>G</sub> and NHN as the test variables

| Algorithm                  | <i>z</i> -statistic | <i>p</i> value | $\alpha'_{\text{Holm}}$ |
|----------------------------|---------------------|----------------|-------------------------|
| MAPE: RPUNN control method |                     |                |                         |
| NARRBFNN <sub>o</sub>      | 7.9872              | 0.0000         | 0.0142                  |
| ESN <sub>o</sub>           | 4.5028              | 1.0E−5         | 0.0166                  |
| ELM <sub>o</sub>           | 4.0740              | 5.0E−5         | 0.0200                  |
| MCESN <sub>o</sub>         | 3.9668              | 7.0E−5         | 0.0250                  |
| NARSIGNN <sub>o</sub>      | 3.2699              | 0.0010         | 0.0333                  |
| NARNN <sub>o</sub>         | 1.9834              | 0.0473         | 0.0500                  |
| ARPUNN <sub>o</sub>        | 1.8762              | 0.060          | 0.1000                  |
| NHN: MCESN control method  |                     |                |                         |
| ELM <sub>o</sub>           | 7.411               | 0.0000         | 0.0142                  |
| NARSIGNN <sub>o</sub>      | 4.3956              | 1E−5           | 0.0166                  |
| RPUNN <sub>o</sub>         | 3.0014              | 0.0026         | 0.0200                  |
| NARBFNN                    | 2.1442              | 0.0320         | 0.0250                  |
| NARNN                      | 2.0102              | 0.0444         | 0.0333                  |
| ESN                        | 0.8844              | 0.3764         | 0.0500                  |
| ARPUNN                     | 0.4824              | 0.6294         | 0.1000                  |

o: Statistical differences were found

**Table 4** Average of computational time results in seconds (cross-validation, training and test) for the 29 data sets considered and all the methods

| Average computational time |          |
|----------------------------|----------|
| MCESN                      | 0.1945   |
| NARNN                      | 0.0412   |
| NARRBFNN                   | 177.9317 |
| ESN                        | 0.2763   |
| ELM                        | 0.0040   |
| NARSIGNN                   | 13.7543  |
| ARPUNN                     | 89.4619  |
| RPUNN                      | 146.7219 |

feature truly important in TSF where there is autocorrelation between the lagged values of the time series (TS). Two models of PUNNs have been implemented, the ARPUNN and the RPUNN, which consists on an enhanced version of the ARPUNN. The architecture of ARPUNN considers a short-term memory provided by the lagged values of the TS, whereas the RPUNN model includes an additional set of inputs supplied by a reservoir network which provides a long-term memory to the model. The parameters of the models were determined by a hybrid learning algorithm that combines a global and a local search methods (the CMA-ES algorithm and the use of the MP generalised inverse, respectively).

TS data sets available in the community have been used as benchmark test sets. Five baseline algorithms from the state-of-the-art TSF literature (NARNN, NARRBFNN, ESN, ELM and NARSIGNN) and one advanced recurrent

neural network model have been used for comparison and the model performance has been evaluated using the mean absolute percentage error (MAPE) and the number of hidden nodes (NHN) measures. Finally, nonparametric statistical tests have been performed to validate the results and the models have been compared also in terms of computational efficiency. The results show that the introduced models present a very good performance in terms of MAPE and NHN, at the cost of a higher computational cost.

Some suggestions for future research are the following: to use the proposed algorithms on other real-world applications, such as detection of tipping points [58] or stock market forecasting,<sup>7</sup> to investigate the use of other advanced basis functions such as the generalised radial basis function [59] in the place of the product unit basis functions, and to analyse the impact of regularisation procedures in the models proposed.

## References

1. Furquim G, Pessin G, Façal B, Mendiondo E, Ueyama J (2015) Improving the accuracy of a flood forecasting model by means of machine learning and chaos theory: a case study involving a real wireless sensor network deployment in brazil. In: Neural computing and applications, pp 1–13. doi:[10.1007/s00521-015-1930-z](https://doi.org/10.1007/s00521-015-1930-z)
2. Arroyo J, Maté C (2009) Forecasting histogram time series with *k*-nearest neighbours methods. *Int J Forecast* 25(1):192–207
3. Arriandiaga A, Portillo E, Sánchez J, Cabanes I, Pombo I (2015) A new approach for dynamic modelling of energy consumption in the grinding process using recurrent neural networks. In: Neural computing and applications, pp 1–16. doi:[10.1007/s00521-015-1957-1](https://doi.org/10.1007/s00521-015-1957-1)
4. Hansen J, Nelson R (1997) Neural networks and traditional time series methods: a synergistic combination in state economic forecasts. *IEEE Trans Neural Netw* 8(4):863–873
5. Sitte R, Sitte J (2000) Analysis of the predictive ability of time delay neural networks applied to the S&P 500 time series. *IEEE Trans Syst Man Cybern Part C Appl Rev* 30(4):568–572
6. Connor J, Martin R, Atlas L (1994) Recurrent neural networks and robust time series prediction. *IEEE Trans Neural Netw* 5(2):240–254
7. He X, Li C, Huang T, Li C, Huang J (2014) A recurrent neural network for solving bilevel linear programming problem. *IEEE Trans Neural Netw Learn Syst* 25(4):824–830
8. Yan Z, Wang J (2014) Robust model predictive control of non-linear systems with unmodeled dynamics and bounded uncertainties based on neural networks. *IEEE Trans Neural Netw Learn Syst* 25(3):457–469
9. Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural Comput* 9(8):1735–1780

<sup>7</sup> In these kind of problems small variations in the inputs could produce large changes in the output of the TS. This situation could be modelled with the product units basis functions (as they are potential basis functions).

10. Jaeger H (2002) Adaptive nonlinear system identification with echo state networks. In: *Advances in neural information processing systems*, pp 593–600
11. Gallicchio C, Micheli A (2011) Architectural and markovian factors of echo state networks. *Neural Netw* 24(5):440–456
12. Rodan A, Tino P (2011) Minimum complexity echo state network. *IEEE Trans Neural Netw* 22(1):131–144
13. Hecht-Nielsen R (1989) Theory of the backpropagation neural network. In: *International joint conference on neural networks 1989 IJCNN*. IEEE, pp 593–605
14. Pan F, Zhang H, Xia M (2009) A hybrid time-series forecasting model using extreme learning machines. In: *Second international conference on Intelligent Computation Technology and Automation, ICICTA '09*, vol 1, pp 933–936
15. Huang G-B, Zhou H, Ding X, Zhang R (2012) Extreme learning machine for regression and multiclass classification. *IEEE Trans Syst Man Cybern Part B Cybern* 42(2):513–529
16. Durbin R, Rumelhart D (1989) Products units: a computationally powerful and biologically plausible extension to backpropagation networks. *Neural Comput* 1(1):133–142
17. Goldberg DE et al (1989) *Genetic algorithms in search, optimization, and machine learning*, vol 412. Addison-Wesley, Reading Menlo Park
18. Li P, Tan Z, Yan L, Deng K (2011) Time series prediction of mining subsidence based on genetic algorithm neural network. In: *2011 international symposium on computer science and society (ISCCS)*, pp 83–86
19. Luque C, Ferran J, Vinuela P (2007) Time series forecasting by means of evolutionary algorithms. In: *IEEE international 2007 parallel and distributed processing symposium, IPDPS 2007*, pp 1–7
20. Cai X, Zhang N, Venayagamoorthy G, Wunsch D (2004) Time series prediction with recurrent neural networks using a hybrid PSO–EA algorithm. In: *2004 IEEE international joint conference on neural networks, 2004. Proceedings*, vol 2, pp 1647–1652
21. Martínez-Estudillo FJ, Hervás-Martínez C, Gutiérrez PA, Martínez-Estudillo AC (2008) Evolutionary product-unit neural networks classifiers. *Neurocomputing* 72(1–3):548–561
22. Martínez-Estudillo AC, Martínez-Estudillo FJ, Hervás-Martínez C, García-Pedrajas N (2006) Evolutionary product unit based neural networks for regression. *Neural Netw* 19(4):477–486
23. Dulakshi AWJ, Karunasingha SK, Li WK (2011) Evolutionary product unit based neural networks for hydrological time series analysis. *J Hydroinf* 13(4):825–841
24. Piotrowski AP, Napiorkowski JJ (2012) Product-units neural networks for catchment runoff forecasting. *Adv Water Resour* 49:97–113
25. Sundermeyer M, Oparin I, Gauvain J-L, Freire B, Schluter R, Ney H (2013) Comparison of feedforward and recurrent neural network language models. In: *2013 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, pp 8430–8434
26. Lukoševičius M, Jaeger H (2009) Reservoir computing approaches to recurrent neural network training. *Comput Sci Rev* 3(3):127–149
27. Hansen N (2006) The CMA evolution strategy: a comparing review. In: *Towards a new evolutionary computation. Studies in fuzziness and soft computing*, vol 192. Springer, Berlin, pp 75–102
28. Jastrebski G, Arnold D (2006) Improving evolution strategies through active covariance matrix adaptation. In: *IEEE congress on 2006 evolutionary computation, CEC 2006*, pp 2814–2821
29. Heidrich-Meisner V, Igel C (2009) Neuroevolution strategies for episodic reinforcement learning. *J Algorithms* 64(4):152–168
30. Moriguchi H, Honiden S (2012) CMA-TWEANN: efficient optimization of neural networks via self-adaptation and seamless augmentation. In: *Proceedings of the 14th annual conference on genetic and evolutionary computation*. ACM, pp 903–910
31. Gundogdu O, Egrioglu E, Aladag C, Yolcu U (2015) Multiplicative neuron model artificial neural network based on gaussian activation function. *Neural Comput Appl* 27:927–935
32. Yadav R, Kalra P, John J (2007) Time series prediction with single multiplicative neuron model. *Appl Soft Comput* 7(4):1157–1163 (Soft computing for time series prediction)
33. Zhao L, Yang Y (2009) PSO-based single multiplicative neuron model for time series prediction. *Expert Syst Appl* 36(2):2805–2812 (Part 2)
34. Attia M, Sallam E, Fahmy M (Aug 2012) A proposed generalized mean single multiplicative neuron model. In: *2012 IEEE international conference on intelligent computer communication and processing (ICCP)*, pp 73–78
35. Egrioglu E, Yolcu U, Aladag C, Bas E (2015) Recurrent multiplicative neuron model artificial neural network for non-linear time series forecasting. *Neural Process Lett* 41(2):249–258
36. Gutiérrez PA, Segovia-Vargas MJ, Salcedo-Sanz S, Hervás-Martínez C, Sanchis A, Portilla-Figueras JA, Fernández-Navarro F (2010) Hybridizing logistic regression with product unit and rbf networks for accurate detection and prediction of banking crises. *Omega* 38(5):333–344
37. Saini L, Soni M (2002) Artificial neural network based peak load forecasting using Levenberg–Marquardt and quasi-Newton methods. *IEE Proc Gener Transm Distrib* 149(5):578–584
38. Hansen N, Niederberger ASP, Guzzella L, Koumoutsakos P (2009) A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Trans Evolut Comput* 13(1):180–197
39. Ros R, Hansen N (2008) A simple modification in CMA-ES achieving linear time and space complexity. In: *Proceedings of the 10th international conference on parallel problem solving from nature: PPSN X*. Springer, pp 296–305
40. Huang G-B, Zhu Q-Y, Siew C-K (2004) Extreme learning machine: a new learning scheme of feedforward neural networks. In: *2004 IEEE international joint conference on neural networks, 2004. Proceedings*, vol 2, pp 985–990
41. Ozturk MC, Xu D, Príncipe JC (2007) Analysis and design of echo state networks. *Neural Comput* 19(1):111–138
42. Bergmeir C, Triguero I, Molina D, Aznarte J, Benitez J (2012) Time series modeling and forecasting using memetic algorithms for regimen-switching models. *IEEE Trans Neural Netw Learn Syst* 23(11):1841–1847
43. Alcalá-Fdez J, Fernández A, Luengo J, Derrac J, García S (2011) Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Mult Valued Logic Soft Comput* 17(2–3):255–287
44. Said SE, Dickey DA (1984) Testing for unit roots in autoregressive-moving average models of unknown order. *Biometrika* 71(3):599–607
45. Ragulskis M, Lukoseviciute K (2009) Non-uniform attractor embedding for time series forecasting by fuzzy inference systems. *Neurocomputing* 72(10):2618–2626
46. Crone S, Dhawan R (2007) Forecasting seasonal time series with neural networks: a sensitivity analysis of architecture parameters. In: *International joint conference on neural networks, IJCNN 2007*, pp 2099–2104
47. Chow TWS, Leung C (1996) Nonlinear autoregressive integrated neural network model for short-term load forecasting. *IEE Proc Gener Transm Distrib* 143(5):500–506
48. Redel-Macías MD, Fernández-Navarro F, Gutiérrez PA, Cubero-Atienza AJ, Hervás-Martínez C (2013) Ensembles of evolutionary product unit or RBF neural networks for the identification of sound for pass-by noise test in vehicles. *Neurocomputing* 109:56–65

49. Demsar J (2006) Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res* 7:1–30
50. Zar JH et al (1999) *Biostatistical analysis*: Pearson Education India. Prentice Hall City, New Jersey
51. Friedman M (1940) A comparison of alternative tests of significance for the problem of m rankings. *Ann Math Stat* 11(1):86–92
52. Dunn OJ (1961) Multiple comparisons among means. *J Am Stat Assoc* 56(293):52–64
53. Hochberg Y, Tamhane AC (1987) *Multiple comparison procedures*. Wiley, New York
54. Aznarte JL, Alcalá-Fdez J, Arauzo-Azofra A, Benítez JM (2012) Financial time series forecasting with a bio-inspired fuzzy model. *Expert Syst Appl* 39(16):12302–12309
55. Adhikari R, Agrawal R (2012) Forecasting strong seasonal time series with artificial neural networks. *J Sci Ind Res* 71(10):657
56. Rocha T, Paredes S, de Carvalho P, Henriques J (2013) An effective wavelet strategy for the trend prediction of physiological time series with application to phealth systems. In: 35th annual international conference of the IEEE engineering in medicine and biology society (EMBC) (2013). IEEE, pp 6788–6791
57. Yao X (1999) Evolving artificial neural networks. *Proc IEEE* 87(9):1423–1447
58. Nikolaou A, Gutiérrez PA, Durán A, Dicaire I, Fernández-Navarro F, Hervás-Martínez C (2015) Detection of early warning signals in paleoclimate data using a genetic time series segmentation algorithm. *Clim Dyn* 44(7–8):1919–1933
59. Fernández-Navarro F, Hervás-Martínez C, Gutiérrez PA (2013) Generalised gaussian radial basis function neural networks. *Soft Comput* 17(3):519–533