ARTICLE IN PRESS

Performance Evaluation ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Learning in the feed-forward random neural network: A critical review

Michael Georgiopoulos [a,*], Cong Li [a], Taskin Kocak [b]

[a] School of EECS, University of Central Florida, United States

[b] Department of Computer Engineering, Bahcesehir University, Turkey

ARTICLE INFO

Article history:
Received 21 May 2010
Accepted 26 July 2010
Available online xxxx

Keywords:
Random neural network
Learning
Gradient descent
Multi-layer perceptron
Error functions
Evolutionary neural networks
ART
SVM
CART
Multi-objective optimization

ABSTRACT

The Random Neural Network (RNN) has received, since its inception in 1989, considerable attention and has been successfully used in a number of applications. In this critical review paper we focus on the feed-forward RNN model and its ability to solve classification problems. In particular, we paid special attention to the RNN literature related with learning algorithms that discover the RNN interconnection weights, suggested other potential algorithms that can be used to find the RNN interconnection weights, and compared the RNN model with other neural-network based and non-neural network based classifier models. In review, the extensive literature review and experimentation with the RNN feed-forward model provided us with the necessary guidance to introduce six critical review comments that identify some gaps in the RNN's related literature and suggest directions for future research.

1. Introduction

The random neural network (RNN) introduced by Gelenbe [1] has motivated a number of theoretical papers, as well as application papers. What is unique about the random neural network compared to other neural network models in the literature (e.g., [2]) where the activity of a neuron is either a binary variable or a continuous variable, is that each neuron is represented by its potential (which is a non-negative integer) and a neuron is considered to be in its "firing state" if its potential is positive; thus in the RNN the representation of a state of a neuron is more granular than in other neural network models in the literature. Furthermore, in the RNN, signals are transmitted from one neuron to another neuron in the form of spikes of a certain rate, which represents more closely how signals are transmitted in a biophysical neural network. Overall, the network of neurons can be modeled as a network of queues, where the state of every queue (neuron) depends on the states of the other queues (neurons) that they are connected to.

In the inaugural RNN paper [1], where the RNN model has been introduced, Gelenbe demonstrated that, under certain conditions, the probability that a neuron's potential is positive converges to a steady state value; furthermore he has shown that the vector of potentials of all the neurons in the RNN has, in its steady state, a probability mass function of a product form, and this steady state probability mass function of all the neuron potentials is the product of the marginal probability mass functions of the neuron potentials. In the same paper, Gelenbe has also illustrated the analogy of the RNN and its popular counterpart, the multi-layer perceptron (MLP) [2]. In a companion paper [3], Gelenbe showed that two classes of random neural networks (damped and balanced) have a well-defined steady state behavior. In 1993 (see [4]), Gelenbe introduced a learning algorithm for the recurrent RNN, which is a gradient descent learning procedure applied on an appropriately defined error function and whose implementation involves finding a fixed point and inverting a matrix whose dimension is equal to the number of neurons in the RNN. The learning algorithm introduced in [4] for the recurrent RNN model can

* Corresponding author.
E-mail address: michaelg@mail.ucf.edu (M. Georgiopoulos).

0166-5316/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2010.07.006

also be applied to a feed-forward RNN model. However, the learning algorithm is significantly simplified when the RNN model is of the feed-forward type compared to the recurrent type (see Section 3 for more details). A linear approximation of this learning algorithm is provided in [5], while in [6] the authors present a quadratic optimization approach for learning in the RNN, mostly suited for image texture reconstructions. A thorough survey of RNN applications is provided in [7], which covers RNN applications published until 2000. A recent RNN paper [8], published in 2010, discusses a multitude of RNN applications and covers a number of them published after 2000. A few of the major RNN applications are: In [9], a novel communications system is developed, where a RNN is used for routing network packets. Hardware implementation of this RNN-based network protocol is reported in [10]. The RNN model has been applied in various image processing problems such as texture generation [11], image segmentation [12], image enhancement and fusion [13], and image and video compression [14]. Pattern recognition and classification is another area, where we see many RNN-based applications: A target recognition system is developed in [15], an RNN is used in [16] to discriminate land mines, and a vehicle classification system is presented in [17], while a RNN is used to find defective semiconductor wafers in [18].

In [19] a RNN model with different classes (multiple classes) of signals is introduced that can be used in applications associated with the concurrent processing of different streams of information, such as color image processing. In a related paper [20], Gelenbe and Hussain, extended the learning algorithm for a single class of RNN signals to a multiple class of RNN signals and demonstrated its applicability to a color texture modeling. In 1999 [21], Gelenbe and his colleagues enhanced the RNN model [1] by considering a bipolar RNN model with positive and negative neurons and have proven that this new RNN model is a *universal function approximator.* Along the same lines Gelenbe and his colleagues demonstrated in [22] that even with a bounded number of layers a bipolar RNN is a universal function approximator. In two companion papers [23,24], Gelenbe and Timotheou introduced a variation of the RNN model, where synchronized interactions of neurons can occur, leading to synchronous firing by a large ensemble of cells. A gradient descent learning algorithm was introduced in [23] that applies to the typical RNN model, as well as to the one with synchronized interactions and its successful application to a resource allocation problem was demonstrated there. In [25] the RNN equations associated with learning are approximated to obtain a linear nonnegative least squares (NNLS) problem that can be solved optimally. The proposed algorithm is applied to a combinatorial problem emerging in disaster management.

It is a well known fact that the gradient descent (GD) procedure suffers from a slow convergence rate to a solution, and algorithms such as the original back-propagation algorithm [2], which is a gradient descent procedure applied to the error function associated with the multi-layer perceptron (MLP) neural network architecture, have been substituted by more efficient algorithms, examples of which are Delta-Bar-Delta [26], RPROP [27], Levenberg–Marquardt [28], and many others. The majority of the RNN papers that have appeared in the literature use gradient descent as the learning algorithm of choice, possibly because of its simplicity. Nevertheless, a number of researchers has also experimented with second order methods to solve the optimization problem that the RNN poses; examples of these methods are the quasi-Newton method [29] and the Levenberg–Marquardt (LM) method [30], where the superiority of these methods has been demonstrated on a limited set of problems. Although these methods have been shown to be faster than the gradient descent procedure they are also more complicated to implement (per epoch of training). A good alternative between the slow gradient descent procedure and the faster, but more complex, second order methods is RPROP [27]. RPROP has been demonstrated to outperform, in efficiency, the gradient descent procedure (sometimes orders of magnitude faster), while it keeps the needed calculations per epoch reasonable, since it only relies on first order derivatives as the gradient descent does. RPROP has also been used with the RNN neural network [31] where it was shown that it outperforms the gradient descent procedure in a problem to recognize geometrical figures. In this paper we have experimented with RPROP and Gradient Descent on a variety of classification problems and assessed the efficiency and accuracy of these two learning approaches within the context of the RNN feed-forward model for the solution of a number of classification problems.

Optimization problems where the objective function to be optimized is differentiable have also been solved by techniques that are referred to as evolutionary techniques. The advantage of evolutionary techniques in solving optimization problems is that (a) they avoid the problem of getting stuck in a local minimum, since they have the ability to globally search the input space for solutions, and (b) they do not require the calculation of derivatives of the objective function, a desired characteristic in situations where this computation is a time-consuming task. Furthermore, within the context of a neural network, evolutionary techniques can be applied to optimize both the interconnection weights, as well as the structure of the neural network, a task that optimization procedures relying on derivatives are unable to solve. In the available RNN literature, we discovered one paper [32] that uses a combination of genetic algorithms and gradient descent to optimize the weights and the topology of the RNN. There the authors train (discover the weights) of a number of different RNNs, using gradient descent, and then they evolve them (optimize their topology) using genetic algorithms. They then demonstrate the advantage of this hybrid learning approach on two problems involving the recognition of alphabetic characters, and geometric figures. Two of the evolutionary approaches that solve optimization problems and have received considerable attention in the literature are the particle swarm optimization technique (PSO), introduced by Eberhart and Kennedy [33], and the Differential Evolution (DE) technique, introduced by Stork and Price [34]. After the introduction of the seminal PSO paper, numerous papers have appeared in the literature to improve the accuracy and speed of the algorithm (e.g., [35–38]). Furthermore, the PSO has been applied in a number of applications [39–41], some of which were to find the interconnection weights of neural network architectures [42–48]. In particular, in [49], the authors designed an evolutionary PSO-based feed-forward MLP (evolved weights and structure simultaneously) and illustrated, through experimentation, that it outperforms a number of other classification models in the literature. Furthermore, in [50], a new PSO approach, called multi-dimensional

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

PSO (MD-PSO) is introduced to optimize the topology of an MLP neural network, and it is shown that it compares favorably with other competitive techniques in the literature on a number of synthetic and benchmark classification problems. In the DE paper [34] the authors have provided an exhaustive comparison of the DE approach and other optimization approaches on a multitude of benchmark optimization problems. Furthermore, a combination of a DE approach and gradient descent is applied in [51] to evolve the structure and find the interconnection weights of an MLP neural network and it was shown to be more accurate and efficient than the pure gradient descent approach to find the interconnection weights and the structure of the MLP neural network.

Evolutionary techniques such as PSO and DE to solve for the interconnection weights of an RNN model have not been applied before. Due to the demonstrated success of evolutionary techniques to solve optimization problems, this is an omission that should be remedied. In this paper, we apply, for the first time, the AIW-PSO approach [38] and the DE approach [34] in solving for the interconnection weights of an RNN and we compare the solutions found by AIW-PSO and DE with the solutions obtained by the most often used gradient descent learning procedure, as well as the RPROP learning procedure [27] applied to the RNN. The measure of comparison between all these approaches is the attained accuracy of a trained RNN, and the computational complexity needed to train the RNN with each one of these methods. The comparative results aside, the evolutionary learning approaches used in this paper for the training of the RNN offer the potential of training the RNN to not only discover its interconnections weights but also determine its best structure in solving a designated problem of interest.

In this paper, we focus on the class of feed-forward RNN models and on mapping problems that are classification problems a focus that is much narrower than what is available in the RNN literature (see [7,8]). Nevertheless, the feed-forward RNN model has been utilized in a number of application papers in the existing literature, such as image and video compression [14,52–55], recognition of objects in the presence of clutter based on synthetic aperture radar images [15,56,57], automatic quantification of the quality of traffic associated with multi-media applications, such as video, speech and interactive voice [58–62], land-mine detection [16,63], wafer surface reconstruction from EM images [18], detection of malicious attacks in computer networks [64–67], design of learning agents within a simulation [68], and others. Furthermore, papers that we have already referred to earlier in the introduction use the feed-forward RNN model (such as [29,30], where different than the gradient descent learning techniques are described, as well was [21,22] where it is demonstrated that a bipolar feed-forward RNN model is a universal approximator. It is worth noting that a significant portion of the recent RNN literature deals with reinforcement learning of the RNN weights (see [69]) in the context of computer networks, referred to as Cognitive Packet Networks [9,70], and Self-Aware Networks [71]; this literature is outside the scope of this paper. It will be an omission, because of the narrower focus in this paper, to forget the impact of the Random Neural Network introduced by Gelenbe [1]. This impact is illustrated by the numerous successful RNN applications that have appeared in the literature, succinctly summarized in the RNN survey papers [7,8] and the Cognitive Packet Network Survey paper [72]. Furthermore, motivated by the neural network model in [1], Gelenbe introduced a new class of queuing networks called G-networks [73], by pioneering the notion of negative customers for work removal. G-networks provide a unifying basis for queuing and neural networks and have stimulated a number of papers in the computer networks community as the survey by Artalejo [74] has pointed out.

It is important for the reader to understand the organization of the paper. Sections 2 and 3 are background sections, where we re-introduce (in Section 2), for completion purposes, the RNN feed-forward model, while we provide (in Section 3) the learning equations needed to train the RNN with the gradient descent learning procedure, the RPROP learning procedure, the AIW-PSO learning procedure, and the DE learning procedure. The background material in Sections 2 and 3 support the experimental results presented in Sections 4 and 5. The intent of the experiments provided in Sections 4 and 5 was neither to be exhaustive, nor to be conclusive. Instead, the primary role of the experimental results in Sections 4 and 5 is to provide the appropriate justification for some of the critical review comments made in Section 5 (six critical review comments are provided) regarding our topic of interest, that is learning in the feed-forward RNN with the intent of solving classification problems. Finally, Section 6 of the paper provides a brief summary of the work conducted in this paper. It is our belief that this critical RNN review will spur additional interest within the research community about the RNN model and applications where it can be used.

2. The RNN model

The random neural network model has been extensively described in the literature in a variety of papers such as [1,3,4], among others. Here we describe only the elements of the random neural network that are needed for the purposes of this paper.

The random neural network is a collection of neurons that are interconnected with each other. The state of a neuron is represented by its potential which is a non-negative integer value. A neuron fires only if its potential is positive. A neuron u can receive exogenous signals positive or negative that are modeled as independent Poisson arrival streams of rates Λ_u, λ_u, respectively. When a neuron receives an excitatory signal (either exogenous or from other neurons in the neural network) its potential increases by one. A neuron can receive an inhibitory signal. When a neuron receives an inhibitory signal and its potential is positive its potential is decreased by one; if the neuron's potential is zero and it receives an inhibitory signal, then its potential stays at zero. When neuron u fires an excitatory spike that goes from neuron u to neuron v, then the potential of neuron u decreases by one, while the potential of neuron v increases by one. When neuron u fires an inhibitory spike that

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi: 10.1016/j.peva.2010.07.006

goes from neuron u to neuron v then the potential of neuron u decreases by one while the potential of neuron v decreases by one, if it is positive, and stays intact if it is zero. When a neuron u fires a spike that leaves the network its potential is decreased by one. A neuron u fires a spike according to a Poisson process with rate r_u. When neuron u fires a spike it ends up being an excitatory spike for neuron v, with probability p^+_{uv}, or an inhibitory spike for neuron v with probability p^-_{uv}, or it leaves the network with probability s_u. If we define $p_{uv} = p^+_{uv} + p^-_{uv}$, then it is easy to see that

$$\sum_v p_{uv} + s_u = 1. \tag{1}$$

The above equation simply indicates that the sum of the probabilities that a firing neuron sends a positive or a negative signal to the neurons that it is connected to or to the outside world is equal to 1. Furthermore, if we define

$$w^+_{uv} = r_u p^+_{uv}, \qquad w^-_{uv} = r_u p^-_{uv} \tag{2}$$

as the positive and negative rates of signal transmissions emanating from neuron u and converging to neuron v, then these rates represent what we typically think as interconnections weights of a neural network, and are quantities that the random neural network learns through a training process.

$$w^+_{uv} = r_u p^+_{uv} = w_{uv}; \quad w_{uv} \geq 0, \qquad w^-_{uv} = r_u p^-_{uv} = |w_{uv}|; \quad w_{uv} < 0 \tag{3}$$

where in the above equations w^+_{uv}, w^-_{uv} are the RNN excitatory and inhibitory weights from neuron u to neuron v, and w_{uv} can be thought of as the MLP weights that could be positive (excitatory) or negative (inhibitory) in a multi-layer perceptron architecture.

We know that for a neuron u the following is true:

$$\sum_v [p^+_{uv} + p^-_{uv}] + s_u = 1 \tag{4}$$

which says that if node u fires it will send either an excitatory or inhibitory spike to another node in the network or will send the spike outside the network. Because of this conservation of energy principle, we can now write

$$r_u \sum_v [p^+_{uv} + p^-_{uv}] + r_u s_u = r_u \tag{5}$$

$$\sum_v [r_u p^+_{uv} + r_u p^-_{uv}] = r_u[1 - s_u] \tag{6}$$

$$\sum_v [w^+_{uv} + w^-_{uv}] = r_u[1 - s_u]. \tag{7}$$

For the case of RNN nodes u for which $s_u = 0$ (i.e., probability of node u sending a signal to the outside world is zero) the above equation becomes:

$$\sum_v [w^+_{uv} + w^-_{uv}] = r_u. \tag{8}$$

Since, in the RNN, we learn the weights w^+_{uv}, w^-_{uv}, it is clear that through this weight learning process we also learn the rates with which nodes in the RNN fire. If on the other hand $s_u \neq 0$ then we need to know s_u in order to be able to calculate r_u. The importance of r_u is that it shows up in the steady-state equations that define the outputs in the RNN neural network. The outputs of the RNN model are the steady state probabilities q_u, and represent the probability that node u has a positive potential, a long time after the RNN has been in operation.

For a random neural network which contains n neurons, we define the following quantities:

$$\lambda^+_v = \Lambda_v + \sum_u q_u r_u p^+_{uv}; \quad 1 \leq v \leq n \tag{9}$$

$$\lambda^-_v = \lambda_v + \sum_u q_u r_u p^-_{uv}; \quad 1 \leq v \leq n \tag{10}$$

where

$$q_v = \frac{\lambda^+_v}{r_v + \lambda^-_v}; \quad 1 \leq v \leq n. \tag{11}$$

In Eqs. (9)–(11) above the indices u, v range from 1 to n, where n is the number of neurons in the RNN model. It is not difficult to see that λ^+_v and λ^-_v represent the average rate of positive and negative signals to neuron v. Furthermore, we will see in the theorems that follow that the quantity q_v represents the stationary probability that the neuron v fires. The following theorems have been discussed in [1]. We include them here for completeness.

Theorem 1. *If the RNN is feed-forward the solution for the Eqs. (9) and (10), such that $\lambda^+_v \geq 0$, $\lambda^-_v \geq 0$, where $1 \leq v \leq n$, exists and is unique.*

Theorem 2. *For an random neural network with n neurons, let the vector of neuron potentials at time t be $\boldsymbol{k}(t) = (k_1(t), \ldots, k_n(t))$, and let $\boldsymbol{k} = (k_1, \ldots, k_n)$ be an vector of nonnegative integers. Then if the q_v in (11) satisfies the constraint*

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

$0 \leq q_v < 1, i = 1, \ldots, n$, *the stationary joint probability of network state* $p(\boldsymbol{k}) = \lim_{t\to\infty} p\{\boldsymbol{k}(t) = \boldsymbol{k}\}$ *is given by*

$$p(\boldsymbol{k}) = \prod_{v=1}^{n} (1 - q_v) {q_v}^{k_v} \tag{12}$$

The proof of these two theorems can be found in [1]. In this paper our focus is on the feed-forward RNN model, the equations that describe its functionality in steady-state (Eqs. (9)–(11)), and on how learning in such an RNN model is accomplished. Provided that the weights in the RNN model are fixed, the outputs (i.e., the steady probabilities q_v's) in a feed-forward RNN model with I, H, and O nodes in the input, hidden and output layer, are given by the following equations,

$$q_i^{(0)} = \frac{\lambda_i^{+(0)}}{r_i^{(0)} + \lambda_i^{-(0)}} = \frac{\Lambda_i^{(0)}}{r_i^{(0)} + \lambda_i^{(0)}} \tag{13}$$

$$q_h^{(1)} = \frac{\lambda_h^{+(1)}}{r_h^{(1)} + \lambda_h^{-(1)}} = \frac{\sum_{i=1}^{I} q_i^{(0)} w_{ih}^{+(1)}}{\sum_{0=1}^{O} [w_{ho}^{+(2)} + w_{ho}^{-(2)}] + \sum_{i=1}^{I} q_i^{(0)} w_{ih}^{-(1)}} \tag{14}$$

$$q_o^{(2)} = \frac{\lambda_o^{+(2)}}{r_o^{(2)} + \lambda_o^{-(2)}} = \frac{\sum_{h=1}^{H} q_h^{(1)} w_{ho}^{+(2)}}{\sum_{h=1}^{H} q_h^{(1)} w_{ho}^{-(2)}} \tag{15}$$

where in the above equations all the parameters involved are defined in Appendix A. In Appendix A all the parameters associated with feed-forward RNN models, that appear in Eqs. (13)–(15), as well as additional pertinent RNN notation is provided. Fig. 1, shows a feed-forward RNN model with I, H, O input, hidden and output nodes and representative interconnection weights.

3. Learning in the RNN

The type of problems that we focus on this paper is classification problems. In particular, we assume that a training collection of input/output pairs, $\{\Lambda(p), \lambda(p); d(p)\}, 1 \leq p \leq PT$ is available to us, and the objective is to map the input vector $(\Lambda(p), \lambda(p))$ to its corresponding desired output vector $d(p)$, for all $p : 1 \leq p \leq PT$. The parameter PT corresponds to the number of input/output pairs in the training set. The pth input output pair from the training collection that is presented to the RNN neural network is designated by $(\Lambda^{(0)}(p), \lambda^{(0)}(p))$. We assume that $\Lambda^{(0)}(p), \lambda^{(0)}(p)$ are vectors of dimensionality I, and their ith component is designated as $\Lambda_i^{(0)}(p), \lambda_i^{(0)}(p)$, respectively. We also assume that $d^{(2)}(p)$ is a O-dimensional vector, and its corresponding oth component is designated by $d_o^{(2)}(p)$. We therefore define an error function for each input/output pair from the training collection, as follows:

$$E(p) = \frac{1}{2} \sum_{\tilde{o}=1}^{O} \left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p) \right]^2. \tag{16}$$

The total error function (for all input/output pairs) is defined below.

$$E = \sum_{p=1}^{PT} E(p) = \frac{1}{2} \sum_{p=1}^{PT} \sum_{\tilde{o}=1}^{O} \left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p) \right]^2. \tag{17}$$

In [4], a learning algorithm has been developed to minimize this error function by updating (learning) the network's interconnection weights. This algorithm is a gradient descent procedure applied to the total error function, where the independent parameters are the RNN's interconnection weights. In its most general form (when all interconnections weights are allowable in the RNN) this learning algorithm involves finding a fixed point and inverting an $n \times n$ matrix, where n is the number of neurons in the RNN. It turns out that for a feed-forward RNN structure we do not have to solve a fixed point equation (the outputs of an RNN feed-forward structure of Fig. 1 can be computed through the set of algebraic equations (13)–(15)). It also turns out that for a feed-forward RNN structure we do not have to invert the $n \times n$ matrix (for reasons explained later in Section 3.1) and the changes for the RNN interconnections weights can also be computed through a set of algebraic equations, provided in Section 3.1.

As we have mentioned earlier, there are other learning approaches to find the minimum of an error function, such as the RPROP [27], AIW-PSO [38] and DE [34]. In particular, the RPROP method takes advantage of non-alternating or alternating

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

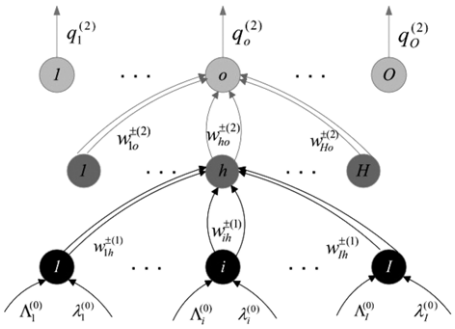

Fig. 1. A feed-forward RNN architecture, with I input nodes, H hidden nodes and O output nodes. All the inputs, outputs and representative interconnection weights are depicted in the figure.

signs of the gradients to accelerate or decelerate the descent to the solution, while the PSO and DE approaches start with a population of solutions (guesses) and they update this population using evolutionary techniques to improve the guessed solutions. In the following sections the details of each one of these approaches, such as GD (Gradient Descent), RPROP, PSO, and DE will be delineated.

3.1. Gradient descent (GD) approach

Fig. 1 shows a feed-forward RNN with I input nodes, H hidden nodes, and O output nodes. Let us denote as $w_{ih}^{+(1)}$, $w_{ih}^{-(1)}$, the excitatory and inhibitory interconnection weights from node i in the input layer (layer 0) to node h in the hidden layer (layer 1). Let us also denote as $w_{ho}^{+(2)}$, $w_{ho}^{-(2)}$, the positive and negative interconnection weights from node h in the hidden layer to node o in the output layer (layer 2). Then, it is easy to see that.

$$\frac{\partial E(p)}{\partial w_{ho}^{+(2)}} = \frac{\partial}{\partial w_{ho}^{+(2)}}\left\{\frac{1}{2}\sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]^2\right\} = \sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]\frac{\partial q_{\tilde{o}}^{(2)}(p)}{\partial w_{ho}^{+(2)}}. \tag{18}$$

After a number of algebraic manipulations we can write Eq. (18) in terms of quantities that are computable. More specifically, it can be shown that

$$\frac{\partial E(p)}{\partial w_{ho}^{+(2)}} = \sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]\left[-\frac{q_h^{(1)}(p)}{D_h^{(1)}(p)\,D_{\tilde{o}}^{(2)}(p)}\left(w_{h\tilde{o}}^{+(2)} - w_{h\tilde{o}}^{-(2)}q_{\tilde{o}}^{(2)}(p)\right) + \frac{q_h^{(1)}(p)}{D_{\tilde{o}}^{(2)}(p)}\mathbf{1}\left[\tilde{o}=o\right]\right]. \tag{19}$$

In a similar fashion,

$$\frac{\partial E(p)}{\partial w_{ho}^{-(2)}} = \frac{\partial}{\partial w_{ho}^{-(2)}}\left\{\frac{1}{2}\sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]^2\right\} = \sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]\frac{\partial q_{\tilde{o}}^{(2)}(p)}{\partial w_{ho}^{-(2)}} \tag{20}$$

which after a number of algebraic manipulations becomes

$$\frac{\partial E(p)}{\partial w_{ho}^{-(2)}} = \sum_{\tilde{o}=1}^{O}\left\{\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]\left[-\frac{q_h^{(1)}(p)}{D_h^{(1)}(p)\,D_{\tilde{o}}^{(2)}(p)}\left(w_{h\tilde{o}}^{+(2)} - w_{h\tilde{o}}^{-(2)}q_{\tilde{o}}^{(2)}(p)\right) - \frac{q_h^{(1)}(p)\,q_{\tilde{o}}^{(2)}(p)}{D_{\tilde{o}}^{(2)}(p)}\mathbf{1}[\tilde{o}=o]\right]\right\}. \tag{21}$$

Furthermore,

$$\frac{\partial E(p)}{\partial w_{ih}^{+(1)}} = \frac{\partial}{\partial w_{ih}^{+(1)}}\left\{\frac{1}{2}\sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]^2\right\} = \sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]\frac{\partial q_{\tilde{o}}^{(2)}(p)}{\partial w_{ih}^{+(1)}} \tag{22}$$

or after a number of algebraic manipulations

$$\frac{\partial E(p)}{\partial w_{ih}^{+(1)}} = \sum_{\tilde{o}=1}^{O}\left\{\frac{\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]\sum_{\tilde{h}=1}^{H}\left[-\frac{q_i^{(0)}}{D_i^{(0)}(p)D_{\tilde{h}}^{(1)}(p)}\left(w_{i\tilde{h}}^{+(1)} - w_{i\tilde{h}}^{-(1)}q_{\tilde{h}}^{(1)}(p)\right) + \frac{q_i^{(0)}(p)}{D_{\tilde{h}}^{(1)}(p)}\mathbf{1}\left[\tilde{h}=h\right]\right]\left(w_{\tilde{h}\tilde{o}}^{+(2)} - w_{\tilde{h}\tilde{o}}^{-(2)}q_{\tilde{o}}^{(2)}(p)\right)}{D_{\tilde{o}}^{(2)}(p)}\right\}. \tag{23}$$

Also,

$$\frac{\partial E(p)}{\partial w_{ih}^{-(1)}} = \frac{\partial}{\partial w_{ih}^{-(1)}}\left\{\frac{1}{2}\sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]^2\right\} = \sum_{\tilde{o}=1}^{O}\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right]\frac{\partial q_{\tilde{o}}^{(2)}(p)}{\partial w_{ih}^{-(1)}} \tag{24}$$

and eventually,

$$\frac{\partial E(p)}{\partial w_{ih}^{-(1)}} = \sum_{\tilde{o}=1}^{O} \left\{ \frac{\left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p)\right] \sum_{\tilde{h}=1}^{H} \left[-\frac{q_i^{(0)}}{D_i^{(0)}(p) D_{\tilde{h}}^{(1)}(p)} \left(w_{i\tilde{h}}^{+(1)} - w_{i\tilde{h}}^{-(1)} q_{\tilde{h}}^{(1)}(p) \right) - \frac{q_i^{(0)}(p) q_{\tilde{h}}^{(1)}(p)}{D_{\tilde{h}}^{(1)}(p)} \mathbf{1}[\tilde{h} = h] \right] \left(w_{\tilde{h}\tilde{o}}^{+(2)} - w_{\tilde{h}\tilde{o}}^{-(2)} q_{\tilde{o}}^{(2)}(p) \right)}{D_{\tilde{o}}^{(2)}(p)} \right\}. \quad (25)$$

Eqs. (19), (21), (23) and (25) provide us with expressions that are computable and define the partial derivatives of the pth error function with respect to the independent parameters (i.e., the positive and negative interconnection weights from input to hidden and from hidden to output layers of the RNN). In a batch gradient descent (BGD) procedure approach, we can then modify the RNN interconnection weights as follows:

$$\triangle w_{ho}^{+(2)} = -\eta \sum_{p=1}^{PT} \frac{\partial E(p)}{\partial w_{ho}^{+(2)}} \quad (26)$$

$$\triangle w_{ho}^{-(2)} = -\eta \sum_{p=1}^{PT} \frac{\partial E(p)}{\partial w_{ho}^{-(2)}} \quad (27)$$

$$\triangle w_{ih}^{+(1)} = -\eta \sum_{p=1}^{PT} \frac{\partial E(p)}{\partial w_{ih}^{+(1)}} \quad (28)$$

$$\triangle w_{ih}^{-(1)} = -\eta \sum_{p=1}^{PT} \frac{\partial E(p)}{\partial w_{ih}^{-(1)}} \quad (29)$$

where in the above equations the terms in the summations are provided by Eqs. (19), (21), (23) and (25).

3.2. The RPROP approach

RPROP has been used successfully [27] in training multi-layer perceptron neural networks, where it was shown that it can outperform the typical gradient descent learning procedure on a number of benchmark problems. RPROP takes advantage of the signs of the gradient in order to increase the learning rate for the descent in cases where consecutive gradients are of the same sign, and in order to decrease the learning rate for the descent when consecutive signs of the gradient are alternating. Furthermore, RPROP backtracks when two consecutive signs of the gradient are alternating. Consequently for every interconnection weight in the RNN, RPROP has a different learning rate, whose value depends on the sequence of the signs of the derivatives with respect to this weight that have been encountered by the algorithm. The functionality of the RPROP depends on a number of parameters (e.g., factor with which to increase or decrease the learning rate (η^+, η^-), a minimum and a maximum change of the weight ($\Delta_{\min}$, $\Delta_{\max}$) that is allowed) but good default values for these parameters have been found in the literature (see [27]) that work well for a variety of problems. Hence, RPROP requires the computation of the derivatives of the objective function with respect to each RNN weight (see Eqs. (26)–(29)) as GD does. The details of the RPROP learning algorithm are provided in [27].

3.3. The AIW-PSO approach

In the AIW-PSO approach [38] we start with a population of initial solutions (guesses) for the RNN interconnection weights, called particles, and then these guesses are improved using a conscience factor that takes into consideration prior best guesses of a particle, and a social factor that takes into consideration the best guess of any particle in the population. Let us denote

$$X_m^t = [X_{m1}^t, X_{m2}^t, \ldots, X_{md}^t, \ldots, X_{mD}^t] \quad (30)$$

the vector with components all the positive and negative interconnection weight values of the RNN corresponding to particle m at generation t. In particular, X_{md}^t is one of the positive or negative interconnection weights of the RNN, at generation t. Note that D is equal to the number of positive and negative interconnection weights in a feed-forward RNN with I input nodes, H hidden nodes, and O output nodes, that is $D = 2(I \cdot H + H \cdot O)$. The PSO equations that update this particle's values (i.e., the RNN interconnection weights) from generation t to generation $t+1$ are given below,

$$V_{md}^{t+1} = w_{md}^t V_{md}^t + c_1 rand() \left(P_{md}^t - X_{md}^t\right) + c_2 rand() \left(P_{gd}^t - X_{md}^t\right) \quad (31)$$

$$X_{md}^{t+1} = X_{md}^t + V_{md}^{t+1} \quad (32)$$

where in the above equations P_m^t is the vector corresponding to the best fit mth particle, up to generation t, while P_g^t is the best fit particle amongst all particles, up to generation t. The fitness of a particle is determined by the value of the error E (see Eq. (17)) that this particle attains; a particle is more fit, the smaller the value of E is that it attains. In order to find the error corresponding to a particle, we have to calculate the q values for every output node and every input pattern applied at

the input of the RNN. Eq. (31) calculates the velocity of particle m of dimension d. Eq. (31) has three terms. The first term is the exploration term that allows the particle to visit portions of the input space that it has not visited before. The constant w_{md}^t involved in the first term, referred to as the inertia weight, controls of how much exploration a particle undergoes. A large inertia weight facilitates exploration but it takes the particle long time to converge, while a small inertia weight forces the particle to converge fast, but sometimes to local minima. The second and the third term in Eq. (31) are the ones that exploit prior good positions of the input space that the particle has visited; c_1 and c_2 are the conscience and social factors that move a particle's position towards the personal best of a particle and the global best of all the particles.

Typically, the conscience and social constants are chosen in the literature to be equal to 2.0, while the inertia weight in the PSO variation that we consider in this paper (in [38]) is determined by the following two equations. The version of the PSO that updates the inertia weight according to the following two equations is referred to as AIW-PSO (Adaptive Inertia Weight-PSO),

$$w_{md}^t = 1 - \frac{1}{1 + \exp(-\alpha \cdot ISA_{md}^t)} \tag{33}$$

$$ISA_{md}^t = \frac{\left|X_{md}^t - P_{md}^t\right|}{\left|P_{md}^t - P_{gd}^t\right| + \varepsilon} \tag{34}$$

where α is a positive constant in the interval (0, 1], and ε is a small positive constant to guarantee that the denominator in (34) is never zero. Hence, in AIW-PSO, each one of the population of particles updates its position from one generation to the next until a meaningful stopping criterion is met. The details of the AIW-PSO algorithm are provided in Appendix B and in [38].

3.4. The DE approach

In the Differential Evolution (DE) approach [34] we also start with an initial population of solutions for the RNN interconnection weights, and then this set of solutions is mutated and crossed-over to obtain better solutions. The DE approach uses two parameters, called a mutation constant, designated as F, and a cross-over constant, designated as CR. Every solution of the population is first mutated by making it equal to the sum of another solution of random index (different than the index of the solution which is mutated), plus an F portion of the differences of two other solutions of random indices (different than the index of the solution which is mutated, and the index of the solution to which this difference of solutions is added). Then, the mutated solution is crossed over (component wise) with the old solution (using the cross-over probability CR) to produce a cross-over solution, which substitutes the old solution that we started with, if and only if, the cross-over solution has a better fitness value (in our case a smaller E value) than the old solution; otherwise the old solution is copied in the new generation. This process of mutating and crossing over solutions from one generation to the next is continued until a meaningful stopping criterion is met. The details of the DE approach are provided in Appendix C and in [34].

3.5. Computational complexity of the BGD, RPROP, AIW-PSO and DE

3.5.1. The complexity of the gradient descent procedure and RPROP for the RNN

In the paper by Gelenbe [4], titled *Learning in the Recurrent Random Neural Network,* the RNN learning procedure is introduced, which involves calculation of the q values for the RNN (requires finding the fixed point of a set of equations of the form shown in Eqs. (9)–(11)), as well as calculation of $\frac{\partial q}{\partial w}$ terms needed for the computation of the $\frac{\partial E}{\partial w}$ terms. The q and $\frac{\partial q}{\partial w}$ values are needed by the gradient descent procedure to calculate the changes of the RNN weights. It is not difficult to see that for the feed-forward RNN model the calculation of the q's (see Eqs. (13)–(15) which are the specialized version of Eqs. (9)–(11) applicable for the feed-forward RNN model) does not require a fixed point calculation. Instead Eqs. (13) through (15) can be used directly for the calculation of the RNN outputs, starting from the nodes in the input layer, then moving to the nodes of the hidden layer and eventually dealing with the nodes of the output layer. For the recurrent RNN model, Gelenbe illustrated in [4] that the inversion of a matrix $I - W$ is needed, where W is an $n \times n$ matrix with entries dependent of the excitatory and inhibitory RNN interconnection weights. It also not difficult to see that for a feed-forward RNN model, like the one that we are focusing on in this paper, the matrix W is upper triangular which avoids the need for the $I - W$'s inversion, a significant simplification. Therefore, if we use the general learning equations for the calculation $\frac{\partial q}{\partial w}$ terms in the recurrent RNN model, as provided in [4], but we take advantage of the upper triangular nature of W, we end up with the learning equations that we have supplied in Section 3.1 of the paper. It is straightforward, now that we know the form of the learning equations for the feed-forward RNN model (Eqs. (13)–(15), (19), (21), (23), (25)), to provide an evaluation of the computational complexity involved in calculating the change of the RNN weights for the feed-forward RNN model.

To compute $q_i^{(0)}$; $1 \le i \le I$, we need $O(I \cdot H)$ calculation (see Eq. (13)). To compute $q_h^{(1)}$; $1 \le h \le H$, we need $O(I \cdot H + H \cdot O)$ computations (see Eq. (14)). To compute $q_o^{(2)}$; $1 \le o \le O$, we need $O(H \cdot O)$ computations (see Eq. (15)). To compute $\triangle w_{ho}^{+(2)}$, $\triangle w_{ho}^{-(2)}$ we need $O(H \cdot O)$ computations (see Eqs. (19) and (21)). To compute $w_{ih}^{+(1)}$, $\triangle w_{ih}^{-(1)}$ we need $O(I \cdot H \cdot O)$ computations (see Eqs. (23) and (25)). These computations need to be performed for every input/output pair,

Table 1
Characteristics of the datasets used in the RNN experiments. The attributes of a dataset determine the number of input nodes, used in the RNN. The number of classes of every dataset used is 2, and as such we need only one RNN output node, whose desired output is zero for one class, and one for the other class.

Dataset	Total instances	Training instances	Test instances	Attributes	Classes	Majority class (%)
G05	5500	500	5000	2	2	50
G15	5500	500	5000	2	2	50
G25	5500	500	5000	2	2	50
G40	5500	500	5000	2	2	50
C00	5500	500	5000	2	2	50
C05	5500	500	5000	2	2	50
C15	5500	500	5000	2	2	50
C25	5500	500	5000	2	2	50
IRIS	5300	500	4800	2	2	50
Bupa	245	100	145	6	2	56
Pima	382	150	182	7	2	67
Magic	5256	501	4755	10	2	65

and as a result the complexity of one epoch of the gradient descent procedure in RNN is $O(PT \cdot I \cdot H \cdot O)$. Since RPROP relies on first order derivatives for the computation of the weight changes RPROP's complexity is also $O(PT \cdot I \cdot H \cdot O)$.

3.5.2. The complexity of the AIW-PSO and DE for the RNN

AIW-PSO and DE start with a population S of solutions and they evolve these solutions from one generation to the next. During this evolution process the error function E needs to be calculated for every member of the population. To calculate E we need to compute $q_i^{(0)}$, $q_h^{(1)}$, $q_o^{(2)}$ (see Eqs. (13)–(15)). Therefore we need $O(I \cdot H + H \cdot O)$ calculations. These calculations need to be performed for every input/output pair, and hence E's calculation requires $O(PT \cdot [I \cdot H + H \cdot O])$ calculations. Furthermore, these calculations need to be computed for S solutions (sets of RNN interconnection weights), resulting in a total cost for an epoch (generation) of the AIW-PSO or DE approach equal to $O(S \cdot PT \cdot [I \cdot H + H \cdot O])$. Quite often, the population size of the PSO or DE approaches is chosen to be equal to a small scalar quantity times the dimensionality of the parameter space, which is equal to I, and consequently the total cost of an epoch of the AIW-PSO and DE approach is $O(PT \cdot I^2 \cdot H + PT \cdot I \cdot H \cdot O) \cong O(PT \cdot I^2 \cdot H)$.

4. Experiments with the RNN

We have experimented with 12 datasets (see Table 1), of which 8 are simulated databases and 4 are real datasets. Table 1 provides a description of these datasets including number of training data, test data, number of attributes (equal to the number of input nodes of the RNN model), number of classes, and the percentage of data in the majority class.

Each dataset was randomly divided into two subsets; training, and testing. The simulated databases include 4 Gaussian databases: G05, G15, G25, and G40. These are 2-dimensional databases with 2-classes and 5%, 15%, 25%, and 40% overlap, between the classes. The overlap in these Gaussian datasets implies that if the optimal (Bayes) classifier were to be used the classification error attained (optimal error) would be equal to the overlap percentage. The dataset denoted by C00 is the benchmark one circle in a square problem, 2-dimensional, two class classification problem. The probability of finding a data point within a circle or inside the square of the circle is equal to 1/2. This dataset is considered when there is no noise in the data (C00), and when there is 5%, (C05), 15% (C15), and 25% (C25) noise in the data. In this problem, noise in the data means that the labels of a portion of the data-points (designated by the noise percentage) are switched. The optimal classifier for the C00, C05, C15, and C25 datasets will result in 100%, 95%, 85%, and 75% accuracy, respectively. The rest of the datasets (IRIS, Bupa, PIMA, Magic) were obtained from the UCI repository (see [75]), where more information about these datasets can be found. In particular, the IRIS dataset was slightly modified from the one provided in the UCI repository (we eliminated one class that is completely linearly separable from the other two classes and we added some artificial data in order to increase the size of the data in this dataset).

In all the experiments conducted we performed limited experimentation, to find a good number of nodes in the hidden layer of the RNN, that is a large enough network to attain a reasonable performance on the training set. For all these datasets, using five nodes in the hidden layer of a feed-forward RNN provided good results for the network's performance on the training set. We also used the training set to find good learning rates for the Batch Gradient Descent (BGD) procedure. For all the other learning procedures we use the default learning parameter values that were reported in the respective papers ([27] for RPROP, [38] for AIW-PSO and [34] for DE).

In our experiments we have examined the performance (accuracy) of the resulting RNN classifier on the test data after the RNN classifier has been trained with the training data using four different learning procedures: (a) Batch Gradient Descent (BGD), (b) RPROP, (c) AIW-PSO, and (d) DE. For each one of these learning procedures the criterion of convergence was: (i) Average Sum of the Squared Error attained a value smaller than a threshold, or (ii) The maximum number of epochs of training has been reached, or (iii) The standard deviation of the average sum of the squared error attained for a fixed number of epochs is smaller than a threshold; the training was considered completed (i.e., learning procedure has converged to a solution) if one of the above three stopping criteria were satisfied. For the BGD and RPROP procedures we experimented

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

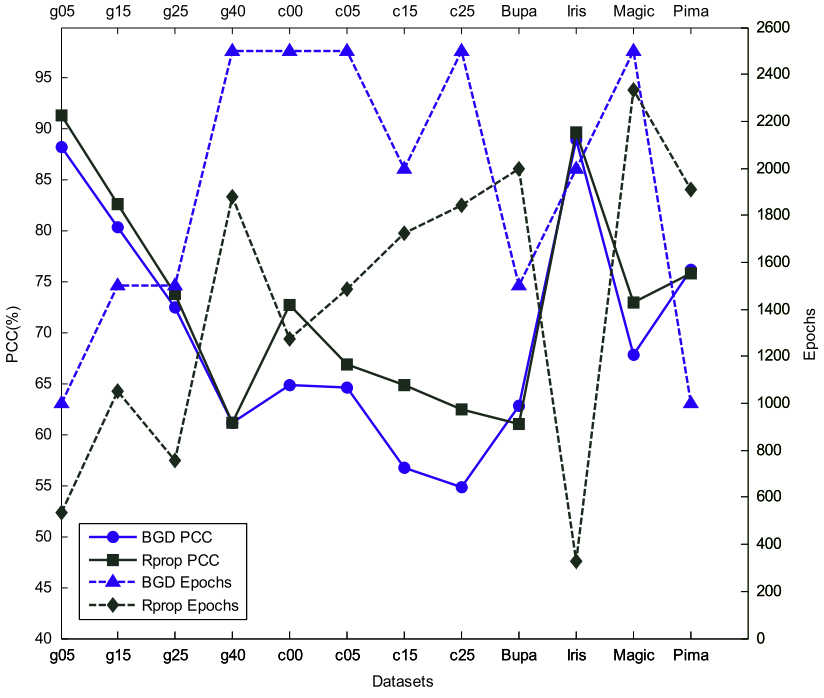

Fig. 2. Average percentage of correct classification (*PCC*), and average number of epochs (*Epochs*) needed until the end of the training for RPROP and BGD learning procedures and for each of the Gaussian, Circle in the Square, IRIS, Bupa, Pima, Magic datasets; the average is considered over 20 runs. The experiments were run for a maximum number of 500, 1000, 1500, 2000 and 2500 epochs of training or until the average sum of the squared errors did not improve significantly over a fixed number of epochs.

with 5 different values of the maximum number of epochs allowed for training, which were 500, 1000, 1500, 2000, and 2500. For the AIW-PSO, and DE procedures we experimented with 5 different values of the maximum number of epochs allowed in training, which were 100, 200, 300, 400, and 500. The difference between the maximum number of epochs that we experimented with BDG and RPROP, versus AIW-PSO and DE was a result of the fact that it typically took the BGD and RPROP learning procedures longer to converge to a solution. For each one of the datasets considered, we run the learning procedure (BGD, RPROP, AIW-PSO, and DE) 20 different times (starting from different initial sets of weights for the RNN) and stored the number of epochs (*Epochs*) needed for the completion of training, the error (*Error*) attained at the end of training, as well as the percentage of correct classification of the trained RNN on the test data (*PCC*).

In Fig. 2 we show the average *Epochs*, and *PCC* attained by BGD and RPROP on the test data for each one of the aforementioned datasets. From Fig. 2 it is clear that RPROP learns the problem in a smaller number of epochs. One can argue that if BGD were left to run for a much larger number of epochs it would eventually match RPROP's performance, but the obvious conclusion extracted from the results of Fig. 2 is that if you have a fixed computational budget to provide a solution, RPROP is the clear choice, since it provides a better generalization performance for a specified maximum number of epochs allowed for training (e.g. for the C00 dataset RPROP attains a 7.8% better generalization in half as many epochs as BGD does).

In Fig. 3, we show the average *Epochs*, and *PCC* attained by RPROP and AIW-PSO for each one of the aforementioned datasets. In Fig. 4, we show the average *Epochs*, and *PCC* attained by AIW-PSO and DE for each one of the aforementioned datasets. The averages are over 20 experiments started with different initial weight values.

Some observations are in order from the entries of Figs. 3 and 4.

- RPROP takes more epochs to converge to a solution compared to the AIW-PSO and DE (Figs. 3 and 4). The computational complexity of an RPROP epoch and an AIW-PSO or DE epoch was calculated in Section 3 and depends on whether I (number of input nodes) or O (number of output nodes) is larger for an RNN; when I is larger than O (the most frequent case) the AIW-PSO, and DE epochs are more computationally complex, otherwise the RPROP epoch is more computationally complex.
- The AIW-PSO attains (in most instances) higher generalization performance than RPROP for a fixed number of epochs allowed (Fig. 3). This difference for some of the datasets is statistically significant, considering that in these datasets the test set used is a few thousand patterns.
- The AIW-PSO approach attains (in most instances) higher generalization performance than DE for a fixed number of epochs allowed (Fig. 4). This difference for some of the datasets is statistically significant, considering that in these datasets the test set used is few thousand patterns. The computational effort involved in an epoch of AIW-PSO and DE is similar.

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

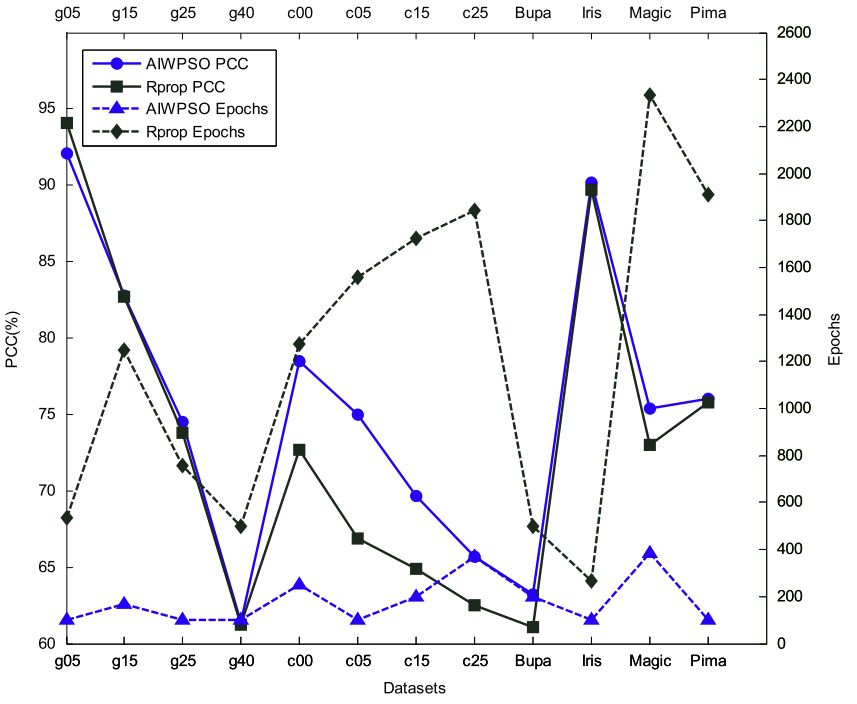

Fig. 3. Average percentage of correct classification (*PCC*), and average number of epochs (*Epochs*) needed until the end of training for RPROP and AIW-PSO learning procedures and for each of the Gaussian, Circle in the Square, IRIS, Bupa, Pima and Magic datasets; the average is considered over 20 runs. The experiments were run for a maximum number of 500, 1000, 1500, 2000 and 2500 epochs of training for the RPROP method, and for a maximum number of 100, 200, 300, 400, 500 epochs of training for the PSO method, or until the average sum of the squared errors did not improve significantly over a fixed number of epochs.

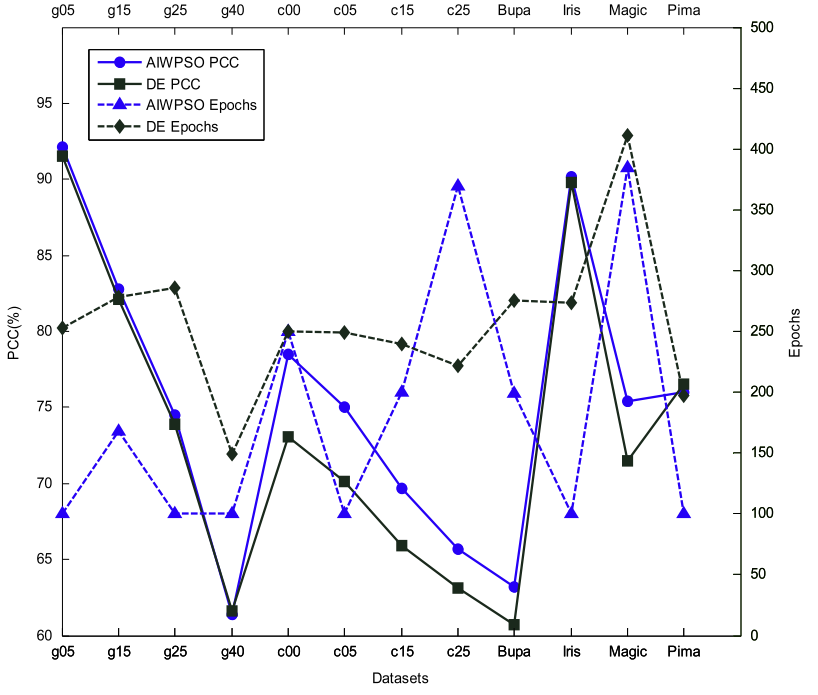

Fig. 4. Average percentage of correct classification (*PCC*), and average number of epochs (*Epochs*) needed until the end of training for AIW-PSO and DE learning procedures and for each of the Gaussian, Circle in the Square, IRIS, Bupa, Pima and Magic datasets; the average is considered over 20 runs. The experiments were run for a maximum number of 100, 200, 300, 400, 500 epochs of training for the PSO and DE methods, or until the average sum of the squared errors did not improve significantly over a fixed number of epochs.

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

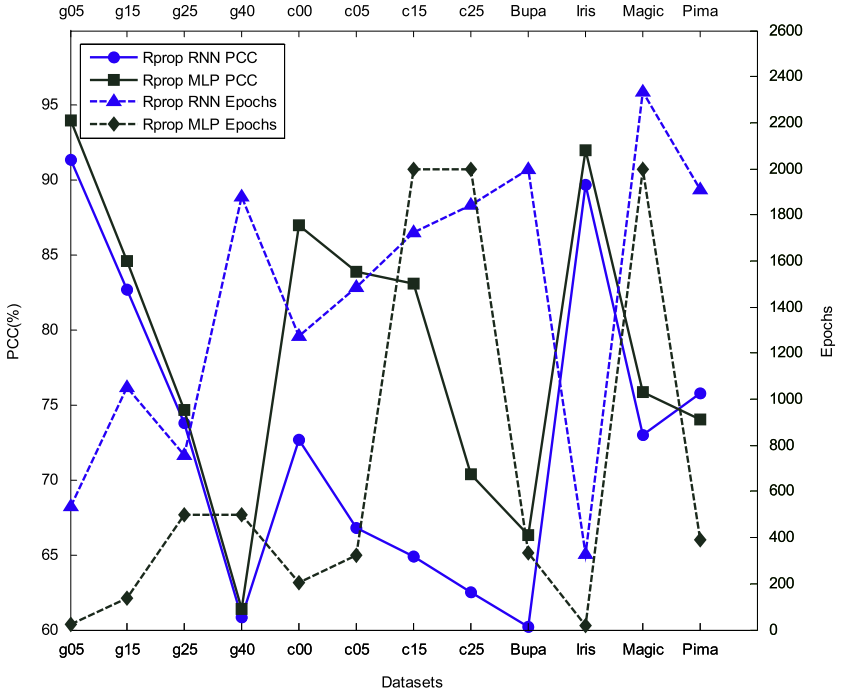

Fig. 5. Average percentage of correct classification (*PCC*), and average number of epochs (*Epochs*) until the end of training for RPROP RNN, RPROP MLP learning procedures and for each of the Gaussian, Circle in the Square, IRIS, Bupa, Pima and Magic datasets; the average is considered over 20 runs. The experiments were run for a maximum number of 500, 1000, 1500, 2000 and 2500 epochs of training for the RPROP method, applied to the RNN or MLP model, or until the average sum of the squared errors did not improve significantly over a fixed number of epochs.

5. Discussion: critical review

5.1. Other classifier models (MLP, ART, SVMs, Decision Trees)

MLP: We make the assumption here that the reader is familiar with the multi-layer perceptron (MLP) neural network architecture, as well as the back-propagation learning algorithm (see [2]), which is the gradient descent procedure applied to a similar sum of the squared differences of actual and desired outputs as the one depicted in Eq. (17) for the RNN network. We also assume that the reader is aware that the computational complexity of an epoch of MLP training with the back-propagation learning procedure applied to a feed-forward MLP neural network with I, H, O input, hidden, and output nodes, respectively, is $O(PT \cdot [I \cdot H + H \cdot O])$, where PT is the number of patterns in the training set. In Fig. 5, we compare the RNN performance, trained with the RPROP algorithm, with the MLP performance, trained with the RPROP algorithm. For all the experiments, MLP used the same number of hidden units as the RNN. In particular, in Fig. 5 we show the average *Epochs*, and *PCC* attained by RNN RPROP and MLP RPROP on the test data for each one of the aforementioned datasets. The averages are over 20 experiments started with different initial weight values.

Some observations are in order from Fig. 5.

- RNN RPROP takes, in most cases, more epochs to converge to a solution compared to the MLP RPROP. The computational complexity of an RNN RPROP epoch is $O(PT \cdot I \cdot H \cdot O)$, while the computational complexity an MLP RPROP epoch is $O(PT \cdot [I \cdot H + H \cdot O])$, which implies that the complexity of an RNN RPROP epoch is slightly higher than the computational complexity of an MLP RPROP epoch (note that for classification problems the number of output nodes O is small). Hence, for the datasets that we experimented with RNN MLP is faster than RNN RPROP.
- With a fixed computational budget (i.e., maximum number of epochs allowed) the trained MLP generalizes better (and sometimes statistically significantly better) than the trained RNN; this statement is valid for the circle in the square datasets that we have experimented with.

Critical Review Comment 1: *Our examination of the relevant RNN literature showed that there is not an exhaustive investigation of the different derivative-based approaches to solve the associated sum of the squared error problem. Some papers related to this topic have appeared in the literature (e.g.,* [29,30]*) but both of them compare their learning algorithms with the simple gradient descent approach on a limited set of problems. On the other hand, the MLP neural network and its associated back-propagation learning algorithm (gradient descent applied on the sum of the squared errors function) has received significant attention in the literature by a variety of researchers who have suggested improvements to its speed of convergence (e.g.,* [26–28,76–79] *and many others). There is a need for a more thorough investigation of which of the many derivative-based methods is most efficient*

with the RNN feed-forward neural network model and this investigation should be applied to a good number of benchmark problems.

ART, SVMs, Decision Trees: Adaptive Resonance Theory (ART) was developed by Grossberg [80]. Some of the ART architectures that have appeared in the literature include Fuzzy ARTMAP (FAM) [81], Ellipsoidal ARTMAP (EAM) [82,83], and Gaussian ARTMAP (GAM) [84]. All of these ART architectures possess a number of desirable properties, such as they can solve arbitrarily complex classification problems, they converge quickly to a solution (within a few presentations of the list of input/output patterns belonging to the training set), they have the ability to recognize novelty in the input patterns presented to them, they can operate in an on-line fashion (new input patterns can be learned by the ART system without retraining with the old input/output patterns), and they produce answers that can be explained with relative ease. We assume here that that the reader is familiar with the class of ART classifier models, a good representative of which is Fuzzy ARTMAP [81].

In [85] we proposed a GA approach [86] for the evolution of ART architectures. Genetic ART starts with a population of trained ART networks, with the number of hidden layer nodes and the interconnection weights to these nodes fully determined (at the beginning of the evolution) by ARTs training rules. To this initial population of ART networks, GA operators are applied to modify these trained ART architectures (i.e., number of nodes in the hidden layer, and values of the interconnection weights) in a way that encourages better generalization and smaller size architectures. The GA optimization problem of ART has two objectives: maximize classification accuracy on a validation set, and minimize network complexity (size of the network), measured in terms of the number of hidden nodes (categories). The proposed GA relies on adaptive parameter control. This has the advantage of avoiding the reliance of the performance on genetic parameters such as the probability of mutation and probability of deletion of an ART category (i.e., hidden node). Therefore, the proposed approach avoids the need for experimentation with GA parameter values to achieve the best possible performing ART network. Furthermore, it alleviates the need to experiment with the ART network parameter values. The ART network parameters are simply sampled to define the initial population of ART neural networks, while the GA algorithm parameters are automatically adapted during evolution to fit the classification problem at hand, resulting in a classifier that does not require any user intervention. Furthermore, the proposed improved GA approach relies on multi-objective optimization techniques (thus overcoming the issues associated with framing a multi-objective optimization problem as a single objective optimization problem [87]), while it chooses the best GA parameters in an adaptive, problem-dependent, fashion. We named this improved GA approach MO-GART and more selectively MO-GFAM, MO-GEAM, and MO-GGAM. In [85] we assessed the performance of MO-GART on a number of classification problems, some of which were the problems that we tested RNN with in this paper. For instance for all the Gaussian problems (G05, G15, G25, and G40) MO-GART not only attained the optimal classification performance (Bayes classification performance) but it achieved this performance by designing an ART architecture with only two hidden nodes (see Figs. 2 and 3 for the corresponding performance of an RNN with five hidden nodes). Furthermore in [85], MO-GART's classification performance on the C00, IRIS, and Pima datasets was found to be 99.80%, 95.24%, and 82.67%, utilizing only 2, 2, and 2 hidden nodes respectively (contrast this performance with the RNN performance of Figs. 2 and 3).

In the same paper [85] we compared MO-GART's performance with the performance of two other popular classifiers: SVM [88] and CART [89]. Once more, we assume that the reader is familiar with the basic functionality of the SVM and CART classifier models.

For SVM training, we used LIBSVM (v. 2.8, see [90]), which is an implementation of the Sequential Minimal Optimization (SMO) algorithm first developed by Platt [91]. As far as we are aware, this is the best-known implementation of SMO available. The SVM classifier has several adjustable parameters including, C, a regularization parameter, and parameters associated with the kernel. For this test, we employed the RBF kernel (or Gaussian kernel, $\exp(-\gamma \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2)$). The linear kernel may be more suitable in some cases where the data is known to be linearly separable and can produce fewer support vectors. However, the RBF kernel is more applicable to a wider variety of instances including linearly separable data. In general, it is not known, a priori, which are the best parameter settings for a particular dataset for providing the best generalization performance. For this reason, it is not uncommon to train the classifier for a wide range of settings to find the optimal generalization performance. For this test, we trained the SVM classifier across a grid of C and γ parameters. The range of values used is as follows: $\log_2(C)$ was varied between -5 and 15, with step size of 2 and $\log_2(\gamma)$ was varied between 3 and -15 with step size of -2, resulting in a total number of 110 C and γ parameter values. The best results that we obtained with the SVM classifier resulted in an SVM classifier that achieved the optimal classification performance (Bayes classification performance) for the G05, G15, G25, and G40 problems utilizing only two support vectors in each one of these cases. SVM's classification performance on the C00, IRIS, and Pima datasets was found to be 99.67%, 95.06%, and 73.71%, utilizing 88, 64, and 2 support vectors, respectively (contrast this performance with the RNN performance of Figs. 2 and 3).

Another classifier we compared MO-GART with in [85] was CART. Classification and Regression Tree (CART) (see [89]) is a classical methodology for training decision trees. It provides systematic and complete solutions to the key problems in training, including the selection of splits, the separation of training into growing and pruning phases to prevent overtraining, the handling of missing values, and the cross-validation of trees. It also provides theoretical support for speed-up algorithms and the performance evaluation of trees. During training, CART offers options and parameters to tune its performance, but it also provides default settings, such as the usage of the Gini Index as the impurity measure in growing the tree and the 1-SE rule (see [89] for more details) in selecting the best pruned tree; these choices usually produce good results. We applied these default settings in our experiments. The results that we obtained with the CART classifier resulted in a

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

CART classifier that achieved the optimal classification performance (Bayes classification performance) for the G05, G15, G25, and G40 problems utilizing only two intermediate nodes in each one of these cases for the decision trees constructed. CART's classification performance on the CS00, IRIS, and Pima datasets was 97.56%, 94.02%, and 73.70%, utilizing 28, 2, and 4 intermediate decision tree nodes, respectively (contrast this performance with the RNN performance of Figs. 2 and 3).

It is worth noting that all the results reported in this section for MO-GART, SVM and CART, as well as MLP, use the same training set for the construction of the classifier model and the same test set for the assessment of the model's performance.

Critical Review Comment 2: *Our examination of the relevant RNN literature showed that there has not been a thorough examination of the feed-forward RNN's performance on classification problems, an important class of problems for a variety of application domains. On the other hand, for the other classifier models that we have mentioned there is a good number of published papers (e.g.,* [85,92–94]*) reporting their performance on a variety of benchmark classification problems. Therefore, in order for RNN to get its fair treatment in the literature, a thorough investigation of its classification performance, size, and complexity needs to be conducted and contrasted against other popular classifier models, such as MLP, ART, SVMs, decision trees, and others.*

5.2. Other error functions

The error function typically used for the training of the RNN (see [4,20,29,30]) is the sum of the squared errors between actual outputs and desired outputs for all the output nodes of the RNN and all input/output pairs of the classification problem under consideration. This was defined earlier and repeated below.

$$E = \sum_{p=1}^{PT} E(p) = \frac{1}{2} \sum_{p=1}^{PT} \sum_{\tilde{o}=1}^{O} \left[q_{\tilde{o}}^{(2)}(p) - d_{\tilde{o}}^{(2)}(p) \right]^2. \tag{35}$$

In the experiments that we have conducted in the previous section we only dealt with two-class classification problems, where RNN has one output node. We would like in that case the output of the RNN ($q_1^{(2)}$) to represent the posterior probability, $P(1|\Lambda, \lambda)$ for class 1 data. The posterior probability of class 2 data will then be given by $P(2|\Lambda, \lambda) = 1 - q_1^{(2)}$. This can be achieved if we consider a target coding scheme for which $d_1^{(2)} = 1$ for an input vector that belongs to class 1 and $d_1^{(2)} = 0$ for an input vector that belongs to class 2. We can combine these into a single expression, so that the probability of observing either target value is

$$P\left(d_1^{(2)}|\Lambda, \lambda\right) = [q_1^{(2)}]^{d_1^{(2)}} [1 - q_1^{(2)}]^{1-d_1^{(2)}}. \tag{36}$$

It is not that difficult to then see that the likelihood of observing the training data set, assuming that the data-points are drawn independently from their corresponding distribution, is then given by

$$\prod_{p=1}^{PT} [q_1^{(2)}(p)]^{d_1^{(2)}(p)} [1 - q_1^{(2)}(p)]^{1-d_1^{(2)}(p)}. \tag{37}$$

So, it makes sense to maximize the likelihood of observing the training data, or minimize the negative of the logarithm of this likelihood, resulting in a modified error function for the RNN, given below.

$$E = -\sum_{p=1}^{PT} (d_1^{(2)}(p) \ln\left(q_1^{(2)}(p)\right) + (1 - d_1^{(2)}(p)) \ln(1 - q_1^{(2)}(p))). \tag{38}$$

The minimization of this error function results in similar weight change equations as the ones that we have discovered for the minimization of the sum of the squared errors error function. The only difference is an additional factor term that appears when the error function of Eq. (38) is employed. In particular,

$$\frac{\partial E}{\partial w_{ho}^{+(2)}} = \sum_{p=1}^{PT} \left[q_1^{(2)}(p) - d_1^{(2)}(p) \right] \frac{\partial q_1^{(2)}(p)}{\partial w_{ho}^{+(2)}} \frac{1}{q_1^{(2)}(p)\,(1 - q_1^{(2)}(p))}. \tag{39}$$

In the above equation, the third factor in the sum is the additional factor that is introduced, now that the error function has been changed from the sum of the squared errors to the cross-entropy error function (see Eq. (38)).

Fig. 6 contrasts the results (*Epochs* and *PCC*) needed for the training of the RNN using the PSO learning procedure and the sum of the squared error function versus the cross-entropy error function. From the results in Fig. 6 it is obvious that the error function used in the training of the RNN has a significant effect on the classification performance attained on the test set. A similar type of conclusions was extracted by comparing the BGD and DE performance results for the sum of the squared error function and the cross-entropy function.

The topic of minimizing alternative error functions (other than the sum of the squared errors function) has received significant attention in the neural network community, examples of which are represented below. In particular, Karayiannis, and Venetsanopoulos introduced in [95] an alternative error function after the initial adaptation cycles with the squared error function and showed faster convergence to a solution for an MLP neural network on a number of benchmark problems (XOR, *N-M-N* encoder problem). In [96] Juang and Katagiri present a new formulation of the pattern recognition problem, aiming at achieving a minimum error rate classification. This new formulation results in a smooth error function that

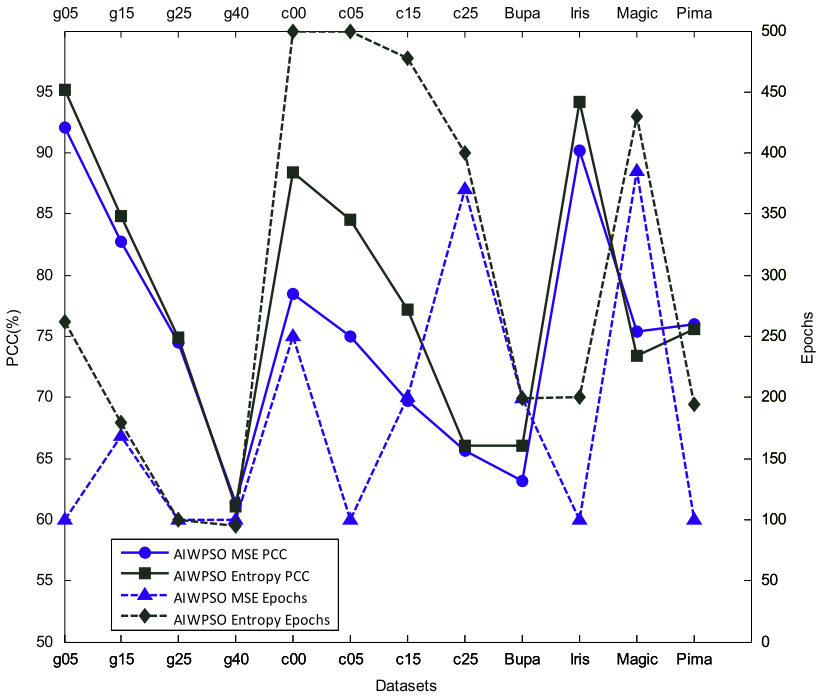

Fig. 6. Average percentage of correct classification (*PCC*), and average number of epochs (*Epochs*) until the end of training for the PSO learning procedure when the mean squared error and the cross-entropy error functions are used, and for each of the Gaussian, Circle in the Square, IRIS, Bupa, Pima and Magic datasets; the average is considered over 20 runs. The experiments were run for a maximum number of 100, 200, 300, 400, and 500 epochs of training.

approximates the empirical error for the design sample set arbitrarily close. In their paper they suggest how the error back-propagation algorithm can work with this new error criterion to achieve the minimum error result. In [97] Van Ooyeen and Nienhuis used the cross-entropy function and showed that for a number of benchmark problems it improves the convergence speed of the MLP neural network. In [98] Nedeljkovic introduced a new MLP training algorithm that minimizes the classification error and demonstrated its potential benefits on a limited number of small scale classification problems. In [99] Cid-Sueiro, et al., established necessary and sufficient conditions for Strict Sense Bayesian (*SSB*) functions (i.e., those that provide estimates of the a posteriori probabilities of the classes) in networks whose outputs are consistent with probability laws; in the same paper the authors provide a general formula for SSB cost functions, satisfying two important properties, the properties of symmetry and separability, both of which are satisfied by the mean squared error function and the cross-entropy error function.

Critical Review Comment 3: *Our examination of the relevant RNN literature showed that all of the RNN papers examined (e.g.,* [4,20,29,30]*) consider the minimization of the frequently used sum of the squared errors function to find the RNN interconnection weights. Although this function is appropriate for regression problems, it might not be the most appropriate one for classification problems. As the literature is suggesting (e.g.,* [95–98]*, and many others papers), as well as our limited experimentation with the RNN and the cross-entropy error function* [100] *there might be advantages in considering alternative error functions to find the weights in a neural network. Therefore, the RNN learning algorithm procedures (derivative based or not) should be examined with other error functions in addition to the frequently used squared error function, especially if the type of problems that RNN addresses are classification problems.*

5.3. Evolutionary approaches and RNN

A very thorough review of the relevant literature pertaining to the evolution of artificial neural networks can be found in a survey paper by Yao [101]. Since Yao's paper, published in 1999, a lot more papers have been published in the literature on the topic of evolutionary neural networks (e.g., [102–104,45]), but none of them has been applied towards the optimization of the weights and/or the topology of an RNN model. In this paper we considered two evolutionary approaches, the PSO approach [38] and the Differential Evolution approach [34], to solve the optimization problem pertaining to learning in the feed-forward RNN (i.e., minimization of the sum of the squared errors (Eq. (17)) or minimization of the entropic error function (Eq. (38)). These approaches were not chosen lightly. The PSO approach has been investigated and referenced extensively in the literature, since its introduction by Eberhart and Kennedy [33]; we also mentioned in the introduction of this paper that PSO has been successfully used for finding interconnection weights in an artificial neural network (e.g., [43–49]). On the other hand, the paper on Differential Evolution [34] has been extensively cited in the literature and it illustrated convincingly that Differential Evolution has a number of advantages in solving optimization problems

since it outperforms a good number of various state-of-the-art optimization methods, such as Annelaed Nelder and Mead strategy [105], the Adaptive Simulated Annealing [106], Breeder Genetic Algorithm [107], Evolutionary Algorithm with Soft Genetic Operators [108], and the Stochastic Differential Equations approach [109]. These observations lead us in a natural way into the following critical review comment.

Critical Review Comment 4: *Our examination of the relevant RNN literature showed that in finding the interconnection weights of the RNN, primarily first and second order derivative methods have been considered. Our limited experimentation has illustrated that there might be an advantage using existing evolutionary approaches, such as PSO and DE, as well as other evolutionary methods, to optimize the weights of an RNN in an effort to avoid getting trapped in local minima, which has been one of the problems of first and second order derivative based learning methods* [110–112].

5.4. Multi-objective evolutionary approaches and RNN

Many real world problems involve the simultaneous optimization of conflicting objectives. This is the basic challenge of multi-objective optimization research. Evolutionary algorithms have been used extensively to solve multi-objective optimization problems, resulting in a body of knowledge known as multi-objective evolutionary algorithms (MOEA). A number of authors has published surveys of MOEA, such as [113], and the reader can find more details about MOEA's there. With conflicting multiple objectives, there is no single optimal solution, but rather, there is a set of good solutions. It is often desirable to find these good solutions as they provide alternative solutions to the problem at hand. Evolutionary algorithms (EAs) are suitable for solving multi-objective optimization problems because EAs are population-based search algorithms, and, as such, they can find, in a single run, multiple good solutions on the surface defined by the multiple objectives that are to be optimized.

Using a multi-objective approach to optimize a classifier is a topic that has attracted considerable attention in the literature. In 1997, Ishibuchi, et al., [114] compared a number of genetic-based single objective approaches and a multi-objective approach to design a fuzzy classifier that maximizes classification accuracy and minimizes the number of produced linguistic rules. The multi-objective method selects half of the population from one generation to the next using a single objective function with random weights (not constant weights); this way it drives the population in many different directions in the objective space. In one of the variations of the multi-objective approach the confidence of a fuzzy rule is taken into consideration in making classification decisions. It turned out that the multi-objective approach that incorporates the confidence of the fuzzy rule produced the best results. The paper by Ishibuchi, et al. [114] is a single example of a very rich literature on multi-objective optimization of fuzzy classifiers, pioneered by Professor Ishibuchi and his colleagues, but a literature that has many more contributors as the *Evolutionary Multi-Objective Optimization of Fuzzy Rule-Based Systems Bibliography* page by M. Cococcioni demonstrates.

The literature is also rich of contributions related to multi-objective optimization of neural networks.

For instance, Everson and Fieldsend [115] have provided a multi-class ROC (receiver operating characteristic) analysis for a 3-class problem using a k-nearest neighbor and a multi-layer classifier. Jin, and his colleagues [116] optimize classification accuracy and connectivity within the framework of spiking neural networks. Graning and his colleagues [117] show on a few benchmark problems that multi-objective optimization of the structure and the weights of multilayer neural networks, solving binary classification problems, results in networks that generalize better compared to the case where the objectives are optimized through a single-objective optimization approach. Jin and Sendhoff provide a very comprehensive review of Pareto-Based Multi-Objective Machine Learning approaches in [118]. There, they illustrate that three benchmark problems (generating interpretable models, model selection for generalization, and ensemble generation) can benefit from a Pareto-based multi-objective approach. Abass [51] used a multi-objective optimization approach and a variation of the Differential Evolution algorithm to successfully evolve the weights and the structure of multi-layer perceptron neural network architecture. An example of a multi-objective approach to evolve ART neural network architectures has already been mentioned earlier [85] and it has demonstrated the potential to produce classifiers of very good generalization, and small size, with a reasonable computational cost and void of any required user intervention to tweak the network's parameters for a number of benchmark classification problems.

Critical Review Comment 5: *Our examination of the relevant RNN literature showed that there has been no attempt to co-jointly optimize the weights and the structure of the RNN network using a multi-objective optimization approach. Considering the extensive literature on the subject of the multi-objective optimization of classifiers and the good results obtained therein (e.g.,* [51,85,114], *many others), it is worth considering a multi-objective optimization of the structure and the interconnection weights of an RNN model.*

5.5. Analysis of the RNN

One of the major criticisms that neural networks have sustained throughout the years is their lack of ability to explain the answers that they produce. This criticism has been directed more towards global neural network models, such as the MLP and the RNN models, where the responsibility of correctly producing a desired output for a specific input is distributed amongst all the network's interconnection weights. This criticism is less directed towards local neural network models, such as RBF-NNs [119], and ART neural networks [120]. In particular, for ART neural networks a number of results has appeared in the literature that attempts to explain how learning works in an ART neural network, and shed light on how this

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

neural network produces its answers (see [121–126]). Furthermore, some attempts have been published in the literature to explain the answers that an MLP neural network produces, when it is confronted with classification problems (e.g., [127]), or understanding some of the limitations of gradient descent learning and the MLP (e.g., [128,129]). It will be worth examining if such a careful investigation of the feed-forward RNN model's functionality will lead into its better understanding.

Critical Review Comment 6: *A review of the RNN literature revealed that a careful analysis of the functionality of the feed-forward RNN model has not been conducted. This research avenue will facilitate the future work, suggested by all critical comments* 1–5, *which are primarily of experimental nature, but whose successful implementation relies on a good understanding of the feed-forward RNN model, its capabilities and its limitations.*

6. Summary/conclusions

In this paper we have focused on the feed-forward RNN model and its associated learning algorithms. Furthermore, we emphasized classification problems. This focus was intentional, so that we can narrow the breadth of the RNN knowledge to a manageable level.

Within the aforementioned context, to motivate some of the omissions that we found in the available RNN literature, a number of experiments was performed with the feed-forward RNN model and its performance was assessed on a small number of classification problems, using a variety of learning strategies. These results and other results that have appeared in the literature for other competitive classifiers (neural-network-based or not) led us into the identification of a number of critical comments or future research directions that would, in our opinion, advance the understanding of the RNN model, as well as its usefulness in a variety of applications. These are: *Critical Comment* 1: Exhaustive experimentation to assess the RNN feed-forward model's performance on a number of benchmark classification problems using a multitude of derivative-based learning methods, *Critical Comment* 2: Investigation of alternative error functions (other than the mean square error function) and assessment of the RNN feed-forward model's performance on a number of benchmark classification problems, *Critical Comment* 3: Thorough comparison of the feed-forward RNN model's performance (accuracy, speed, size) and other popular classifier models on a number of benchmark classification problems, *Critical Comment* 4: Investigation of a number of evolutionary approaches to find the interconnection weights of a feed-forward RNN model on a number of benchmark classification problems, and comparison of the effectiveness (accuracy) and efficiency (speed) of these methods with the most frequently used derivative-based methods, *Critical Comment* 5: Examination of a variety of multi-objective evolutionary approaches to co-jointly optimize the weights and the structure of a feed-forward RNN model and comparison of these approaches with the frequently used fixed RNN structure's optimization of the weights approach. *Critical Comment* 6: Analysis of the functionality of the RNN feed-forward model, to gain a better understanding of its capabilities and limitations.

Acknowledgements

Michael Georgiopoulos and Cong Li recognize the partial support of NSF grant numbers: 0341601, 0647018, 0717674, 0717680, 0647120, 0525429, 0806931, 0837332, and 0963146.

Appendix A. Notation

RNN notation	Explanation
O:	The maximum index of an output RNN node. So, output nodes in the RNN are indexed from 1 to O with a generic index designated as o
I:	The maximum index of an input RNN node. So, input nodes in the RNN are indexed from 1 to I, with a generic index designated as i
H:	The maximum index of an RNN node that is not an input node, nor an output node; we call these nodes *hidden*. So, hidden nodes in the RNN are indexed from 1 to H, with a generic index designated as h.
n:	The number of neurons in the RNN network. For a feed-forward RNN network with I, H, O, input, hidden and output neurons, respectively, $n = I \cdot H + H \cdot O$
PT:	The maximum index of an input/output pair presented to the RNN. The generic index of an input/output pair is designated as p
$E(p)$:	Error term associated with the input/output pair p
Layer Index	The input layer is layer 0, the hidden layer is layer 1, and the output layer is layer 2
$q_i^{(0)}$:	The output of input layer neuron i, located in layer 0
$q_h^{(1)}$:	The output of hidden-layer neuron h, located in layer 1
$q_o^{(2)}$:	The output of output-layer neuron o, located in layer 2

(*continued on next page*)

$d_o^{(2)}(p)$:	The desired output of output node o in layer 2
$N_i^{(0)}$:	The numerator of the ratio defining the output of neuron i in layer 0; this corresponds to the rate of positive signals (exogenous and endogenous) arriving at input neuron i of layer 0
$D_i^{(0)}$:	The denominator of the ratio defining the output of neuron i in layer 0; this corresponds to the rate of negative signals (exogenous and endogenous) arriving at input neuron i of layer 0 and the rate $r_i^{(0)}$, defined below
$N_h^{(1)}$:	The numerator of the ratio defining the output of neuron h in layer 1; this corresponds to the rate of positive signals (exogenous and endogenous) arriving at input neuron h of layer 1
$D_h^{(1)}$:	The denominator of the ratio defining the output of neuron h in layer 1; this corresponds to the rate of negative signals (exogenous and endogenous) arriving at input neuron h of layer 1 and the rate $r_h^{(1)}$, defined below
$N_o^{(2)}$:	The numerator of the ratio defining the output of neuron o in layer 2; this corresponds to the rate of positive signals (exogenous and endogenous) arriving at input neuron o of layer 2
$D_o^{(2)}$:	The denominator of the ratio defining the output of neuron o in layer 2; this corresponds to the rate of negative signals (exogenous and endogenous) arriving at input neuron o of layer 2, and the rate $r_o^{(2)}$, defined below
$\Lambda_i^{(0)}$:	The arrival rate of the Poisson process corresponding to the exogenous positive signal arriving at input node i of layer 0
$\lambda_i^{(0)}$:	The arrival rate of the Poisson process corresponding to the exogenous negative signal arriving at input node i of layer 0
$\lambda_i^{+(0)}$:	The arrival rate of the Poisson process corresponding to the endogenous and exogenous positive signal arriving at input node i of layer 0; is equal to $N_i^{(0)}$
$\lambda_i^{-(0)}$:	The arrival rate of the Poisson process corresponding to the endogenous and exogenous negative signal arriving at input node i of layer 0; this term is such that $D_i^{(0)} = r_i^{(0)} + \lambda_i^{-(0)}$, where $r_i^{(0)}$ is defined below
$\Lambda_h^{(1)}$:	The arrival rate of the Poisson process corresponding to the exogenous positive signal arriving at hidden node h of layer 1. The usual assumption made is that the exogenous signals are non-zero only for the input nodes (the nodes in layer 0). For all other nodes in the feed-forward RNN (i.e., hidden and outputs) these exogenous signals are assumed to be equal to zero
$\lambda_h^{(1)}$:	The arrival rate of the Poisson process corresponding to the exogenous negative signal arriving at hidden node h of layer 1. The usual assumption made is that the exogenous signals are non-zero only for the input nodes (the nodes in layer 0). For all other nodes in the feed-forward RNN (i.e., hidden and outputs) these exogenous signals are assumed to be equal to zero
$\lambda_h^{+(1)}$:	The arrival rate of the Poisson process corresponding to the endogenous and exogenous positive signal arriving at hidden node h of layer 1; is equal to $N_h^{(1)}$
$\lambda_h^{-(1)}$:	The arrival rate of the Poisson process corresponding to the endogenous and exogenous negative signal arriving at hidden node h of layer 1; this term is such that $D_h^{(1)} = r_h^{(1)} + \lambda_h^{-(1)}$, where $r_h^{(1)}$ is defined below
$\Lambda_o^{(2)}$:	The arrival rate of the Poisson process corresponding to the exogenous positive signal arriving at output node o of layer 2. The usual assumption made is that the exogenous signals are non-zero only for the input nodes (the nodes in layer 0). For all other nodes in the feed-forward RNN (i.e., hidden and outputs) these exogenous signals are assumed to be equal to zero
$\lambda_o^{(2)}$:	The arrival rate of the Poisson process corresponding to the exogenous negative signal arriving at output node o of layer 2. The usual assumption made is that the exogenous signals are non-zero only for the input nodes (the nodes in layer 0). For all other nodes in the feed-forward RNN (i.e., hidden and outputs) these exogenous signals are assumed to be equal to zero
$\lambda_o^{+(2)}$:	The arrival rate of the Poisson process corresponding to the endogenous positive signal arriving at output node o of layer 2; this term is equal to $N_o^{(2)}$

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

$\lambda_o^{-(2)}$:	The arrival rate of the Poisson process corresponding to the endogenous negative signal arriving at output node o of layer 2; this term is such that $D_o^{(2)} = r_o^{(2)} + \lambda_o^{-(2)}$, where $r_o^{(2)}$ is defined below
$w_{ih}^{+(1)}$:	The positive interconnection weight emanating from node i in layer 0 and converging to node h in layer 1
$w_{ih}^{-(1)}$:	The negative interconnection weight emanating from node i in layer 0 and converging to node h in layer 1
$w_{ho}^{+(2)}$:	The positive interconnection weight emanating from node h in layer 1 and converging to node o in layer 2
$w_{ho}^{-(2)}$:	The negative interconnection weight emanating from node h in layer 1 and converging to node o in layer 2
$s_i^{(0)}$:	The rate with which a signal is emitted from input neuron i to the outside environment; in our paper, $s_i^{(0)}$ is equal to 0
$s_h^{(1)}$:	The rate with which a signal is emitted from hidden neuron h to the outside environment; in our paper, $s_h^{(1)}$ is equal to 0
$s_o^{(2)}$:	The rate with which a signal is emitted from output neuron o to the outside environment; in our paper, $s_o^{(2)}$ is equal to 0
$r_i^{(0)}$:	When the potential of a neuron i, in the input layer, is positive the neuron might fire. The average time between two successive signal departures from the ith neuron is denoted by $1/r_i^{(0)}$, which means that the rate with which neuron i fires is denoted by $r_i^{(0)}$. The firings of the ith active input neuron follow a Poisson process with average inter-arrival between two successive signal firings equal to $1/r_i^{(0)}$; note that with the assumption that $s_i^{(0)} = 0, r_i^{(0)} = \sum_{h=1}^{H}[w_{ih}^{+(1)} + w_{ih}^{-(1)}]$
$r_h^{(1)}$:	When the potential of a neuron h, in the hidden layer, is positive the neuron might fire. The average time between two successive signal departures from the hth neuron is denoted by $1/r_h^{(1)}$, which means that the rate with which neuron h fires is denoted by $r_h^{(1)}$. The firings of the hth active hidden neuron follow a Poisson process with average inter-arrival between two successive signal firings equal to $1/r_h^{(1)}$; note that with the assumption that $s_h^{(1)} = 0, r_h^{(1)} = \sum_{o=1}^{O}[w_{ho}^{+(2)} + w_{ho}^{-(2)}]$
$r_o^{(2)}$:	When the potential of a neuron o, in the output layer, is positive the neuron might fire. The average time between two successive signal departures from the oth neuron is denoted by $1/r_o^{(2)}$, which means that the rate with which neuron o fires is denoted by $r_o^{(2)}$. The firings of the oth active output neuron follow a Poisson process with average inter-arrival between two successive signal firings equal to $1/r_o^{(2)}$. Note than an output neuron in the feed-forward RNN is not connected to any other neuron, and as such $r_o^{(2)} = 0$; furthermore, in this case $s_o^{(2)} = 1$
p:	The generic index used to define the index of an input/output pair. All of the RNN quantities defined above can be redefined, using this index, to designated the fact that they correspond to RNN values produced during the presentation of a specific input/output pair; for instance $q_o^{(2)}(p)$ corresponds to the output of output neuron o in the RNN layer 2 due to the presentation of the pth input output pair
$(\Lambda(p), \lambda(p); d(p))$:	The pth input output pair from the training collection that is presented to the RNN neural network. For all practical purposes we can assume that $\Lambda(p)$, $\lambda(p)$ are vectors of dimensionality I, and their ith component is designated as $\Lambda_i^{(0)}(p)$, $\lambda_i^{(0)}(p)$, respectively. For all practical purposes we can assume that $d(p)$ is an O-dimensional vector, and its corresponding oth component is designated by $d_o^{(2)}(p)$
W:	The matrix involved in the calculation of the $\partial q/\partial w$ values for a recurrent RNN network. This matrix is nxn; the lth column, mth row entry of this matrix is equal to $(w_{lm}^{+} - w_{lm}^{-})q_l/D_l$

RPROP notation	Explanation
$\triangle_{uv}$:	The amount of change applied to an RNN interconnection weight, as dictated by the RPROP algorithm

(*continued on next page*)

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

$\triangle_{\min}$:	The minimum amount of change applied to an RNN interconnection weight, as dictated by the RPROP algorithm
$\triangle_{\max}$:	The maximum amount of change applied to an RNN interconnection weight, as dictated by the RPROP algorithm
η^+:	The factor by which the change of the interconnection weight, as dictated by the RPROP algorithm, increases.
η^-:	The factor by which the change of the interconnection weight, as dictated by the RPROP algorithm, decreases
PSO/DE notation	Explanation
S:	Number of particles (solutions) in a PSO or DE algorithm
X_m^t:	The vector of interconnection weights in an RNN, represented as a particle of index m, at epoch (generation) t. This vector represents a solution for the problem at hand (i.e., minimizing the error function for RNN)
V_m^t:	The velocity of a particle of index m, at epoch (generation) t. This vector expresses the amount of change that needs to be applied to a particle's position as it moves from its position at epoch t to its position at epoch $t+1$
w_m^t:	This is the exploration constant that determines how much a particle is allowed to explore new solutions in its input space
ISA:	This parameter controls the value of the exploration constant, is particle dependent, and dimension dependent, and is affected by the particle's best prior position and the best of all particles' position
c_1:	The conscience constant in the PSO. This provides the amount of influence of the best prior position of a particle on the particle's next position
c_2:	The conscience constant in the PSO. This provides the amount of influence of the best of all particles position on the particle's next position
Y_m^t:	The mutated particle, corresponding to particle of index m at epoch t
Z_m^t:	The mutated and cross-over particle, corresponding to particle of index m at epoch t
SVM notation	Explanation
C:	The regularization parameter in a Support Vector Machine problem formulation that determines the penalty in the objective function applied for every input/output pair that violates the problem constraints; high C value means that high penalty is applied for a constraint violation
γ:	The kernel parameter used in a Support Vector Machine problem that determines the width of the Gaussian kernels used

Appendix B. Step-by-step AIW-PSO procedure

Step 1 (*Initialization Step*): Start with a population of S particles where each of their components represent the weights in the RNN, that is $w_{ih}^{+(1)}$, $w_{ih}^{-(1)}$, $w_{ho}^{+(2)}$, $w_{ho}^{-(2)}$, for $1 \le i \le I$, $1 \le h \le H$, $1 \le o \le O$; all these weights are random values uniformly distributed over a positive interval of the real line (say interval $(0, w_{\max}]$). The c_1, c_2 constant values are taken to be equal to 2.0. The constant α is chosen to be equal to 0.5.

Step 2 (*Apply the PSO Equations*): You apply the PSO equations to move each particle from its position in generation t to generation $t+1$ according to the following equations.

$$V_{md}^{t+1} = w_{md}^t V_{md}^t + c_1\ rand()\left(P_{md}^t - X_{md}^t\right) + c_2\ rand()\left(P_{gd}^t - X_{md}^t\right)$$

$$X_{md}^{t+1} = X_{md}^t + V_{md}^{t+1}$$

$$w_{md}^t = 1 - \frac{1}{1+\exp(-\alpha \cdot ISA_{md}^t)}$$

$$ISA_{md}^t = \frac{\left|X_{md}^t - P_{md}^t\right|}{\left|P_{md}^t - P_{gd}^t\right| + \varepsilon}.$$

In applying the above equations we use the error (fitness) function below to find the personal best of every particle, up to generation t, and the global best of all particles up to generation t.

$$E = \frac{1}{2}\sum_{p=1}^{PT}\sum_{o=1}^{O}[q_o^{(2)}(p) - d_o^{(2)}(p)]^2.$$

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

To calculate the fitness function of every particle at generation t it suffices to compute the RNN output $q_o^{(2)}$ for every output node of the RNN and for every input pattern applied to the RNN input layer. This can be accomplished by applying the Eqs. (13)–(15), located in the main text of the paper.

Step 3 *(Check for convergence):* Check the value of the total average squared error, given by the following equation for the best particle found by generation t:

$$\frac{1}{PT \cdot O} E = \frac{1}{PT \cdot O} \frac{1}{2} \sum_{p=1}^{PT} \sum_{o=1}^{O} [q_o^{(2)}(p) - d_o^{(2)}(p)]^2$$

If this average squared error is smaller than a user designated threshold or if the error has not changed significantly over a number of epochs, then the learning is complete; otherwise go back to Step 2 and apply the PSO equations for each particle for one more generation (epoch).

Appendix C. Step-by-step DE procedure

Step 1 (*Initialization Step*): Start with a population of S particles where each of their components represents the weights in the RNN, that is $w_{ih}^{+(1)}$, $w_{ih}^{-(1)}$, $w_{ho}^{+(2)}$, $w_{ho}^{-(2)}$, for $1 \le i \le I$, $1 \le h \le H$, $1 \le o \le O$; all these weights are random values uniformly distributed over a positive interval of the real line (say interval $(0, w_{\max}]$). The values of F and CR are chosen to be equal to 0.5 and 0.1, respectively.

Step 2 (*Mutation of a Particles*): Apply the DE equations to mutate every particle of the population ($1 \le m \le S$). These are:

$$Y_{md}^{t+1} = X_{r_1 d}^{t} + F(X_{r_2 d}^{t} - X_{r_3 d}^{t})$$

where in the above equation the mutated mth particle at dimension d is equal to the value of another particle at dimension d, plus the difference of two other particle values at the same dimension. As it can be seen F is a parameter, typically chosen in the interval [0, 2] that controls the amount of the contribution of $(X_{r_2 d}^{t} - X_{r_3 d}^{t})$.

Step 3 (*Cross-Over of Particles*): Apply the DE equations to cross-over the mutated particle and the original particle. The cross-over obeys the following equation:

$$Z_{md}^{t+1} = Y_{md}^{t+1} \text{ if } rand() \le CR, \text{ while } Z_{md}^{t+1} = X_{md}^{t} \text{ if } rand() > CR \text{ and } Z^{t+1} \ne X^{t}.$$

If Z^{t+1} has better fitness than X^t then X^{t+1} becomes equal to Z^{t+1}; otherwise X^{t+1}becomes equal to X^t. The error (fitness) function shown below is used, and the lower the fitness value of a particle is the better the fitness value of the particle is.

$$E = \frac{1}{2} \sum_{p=1}^{PT} \sum_{o=1}^{O} [q_o^{(2)}(p) - d_o^{(2)}(p)]^2.$$

To calculate the fitness function of particles at generation t it suffices to compute the RNN output $q_o^{(2)}(p)$ for every output node of the RNN and for every input pattern applied to the RNN input layer. This can be accomplished by using Eqs. (13)–(15), located in the main text of the paper.

Step 4 (*Check for Convergence*): Check the value of the total average squared error, given by the following equation for the best particle found by generation t:

$$\frac{1}{PT \cdot O} E = \frac{1}{PT \cdot O} \frac{1}{2} \sum_{p=1}^{PT} \sum_{o=1}^{O} [q_o^{(2)}(p) - d_o^{(2)}(p)]^2.$$

If this average squared error is smaller than a user designated threshold or if the error has not changed significantly over a number of epochs, then the learning is complete; otherwise go back to Step 2 and apply the mutation and cross-over operation to all the particles for one more generation (epoch).

References

[1] E. Gelenbe, Random neural networks with negative and positive signals and product form solution, Neural Computation 1 (1989) 502–510.
[2] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, J.L. McClelland, PDP Research Group (Eds.), Parallel Distributed Processing; Explorations in the Microstructure of Cognition, MIT Press, Cambridge, MA, 1986.
[3] E. Gelenbe, Stability of the random neural network model, Neural Computation 2 (1990) 239–247.
[4] E. Gelenbe, Learning in the recurrent neural network, Neural Computation 5 (1993) 154–164.
[5] U. Halici, E. Karaoz, A linear approximation for the Gelenbe's learning algorithm for recurrent random neural networks, in: TAINN VI Proceedings of the 6th Turkish Symposium on Artificial Intelligence and Neural Networks, Istanbul, Turkey, 1996, pp. 86–94.
[6] V. Atalay, Learning by optimization in random neural networks, in: Proceedings of the Thirteenth International Symposium on Computer and Information Sciences, Antalya, Turkey, 1998, pp. 143–148.
[7] H. Bakircioglu, T. Kocak, Survey of random neural network applications, European Journal of Operational Research 126 (2000) 319–330.
[8] S. Timotheou, The random neural network: a survey, Computer Journal 53 (3) (2010) 251–267.
[9] E. Gelenbe, R. Lent, Z. Xu, Design and performance of cognitive packet networks, Performance Evaluation 46 (2001) 155–176.

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

[10] T. Kocak, J. Seeber, H. Terzioglu, Design and implementation of a random neural network routing engine, IEEE Transactions on Neural Networks 14 (2003) 1128–1143.
[11] V. Atalay, E. Gelenbe, N. Yalabik, The random neural network model for texture generation, International Journal of Pattern Recognition and Artificial Intelligence 6 (1992) 131–141.
[12] E. Gelenbe, T. Feng, K.R.R. Krishnan, Neural network methods for volumetric magnetic resonance imaging of the human brain, Proceedings of IEEE 84 (1996) 1488–1496.
[13] H. Bakircioglu, E. Gelenbe, T. Kocak, Image enhancement and fusion with the random neural network, Turkish Journal of Electrical Engineering and Computer Science 5 (1997) 65–77.
[14] C. Cramer, E. Gelenbe, P. Gelenbe, Image and video compression, IEEE Potentials 17 (1998) 29–33.
[15] H. Bakircioglu, E. Gelenbe, Random neural network recognition of shaped objects in strong clutter, in: Proceedings of 3rd Conference on Applications of Artificial Neural Networks in Image Processing, San Jose, CA, USA, 1998, pp. 22–28.
[16] H. Abdelbaki, E. Gelenbe, T. Kocak, Matched neural filters for EMI based mine detection, in: Proceedings of Internatinal Joint Conference on Neural Networks, Washington, DC, 1999, pp. 3236–3240.
[17] K. Hussain, G.S. Moussa, Laser intensity vehicle classification system based on random neural network, in: Proceedings of 43rd Annual Southeast Regional Conference, Kennesaw, Georgia, USA, 2005, pp. 31–35.
[18] E. Gelenbe, T. Kocak, Wafer surface reconstruction from top-down scanning electron microscope images, Microelectronic Engineering 75 (2004) 216–233.
[19] E. Gelenbe, J.M. Fourneau, Random neural networks with multiple classes of signals, Neural Computation 11 (1999) 953–963.
[20] E. Gelenbe, K. Hussain, Learning in the multiple class random neural network, IEEE Transactions on Neural Networks 13 (2002) 1257–1267.
[21] E. Gelenbe, Z.-H. Mao, Y.-D. Li, Function approximation with spiked random networks, IEEE Transactions on Neural Networks 10 (1) (1999) 3–9.
[22] E. Gelenbe, Z.-H. Mao, Y.-D. Li, Function approximation by random neural networks with bounded number of layers, Journal of Differential Equations and Dynamical Systems 12 (1&2) (2004) 143–170.
[23] E. Gelenbe, S. Timotheou, Synchronized intercations in spiked neuronal models, The Computer Journal 51 (6) (2008) 723–730.
[24] E. Gelenbe, S. Timotheou, Random neural networks with synchronized interactions, Neural Computation 20 (2008) 2308–2324.
[25] S. Timotheou, Nonegative least squares learning for the random neural network, in: 18th International Conference on Artificial Neural Networks, Prague, Czech Republic, 2008, pp. 195–204.
[26] R.A. Jacobs, Increased rates of convergence through learning rate adaptation, Neural Networks 1 (1988) 151–160.
[27] M. Riedmiller, H. Braun, A direct adaptive method for faster back-propagation learning: the RPROP algorithm, in: Proceedings of the International Conference on Neural Networks, 1993, pp. 586–591.
[28] M.T. Hagan, M. Menhaj, Training feed-forward neural networks with the marquardt algorithm, IEEE Transactions on Neural Networks 5 (6) (1994) 989–993.
[29] A. Likas, A. Stafylopatis, Training the random neural network using quasi-newton methods, European Journal of Operational Research 126 (2000) 331–339.
[30] S. Basterrech, S. Mohammed, G. Rubino, M. Soliman, Levenberg–Marquardt training algorithms for random neural networks, The Computer Journal (2009).
[31] C. Hubert, Pattern completion with the random neural network using the RPROP learning algorithm, in: Proceedings of International Conference on Man and Cybernetics, LeTouquet, France, vol. 2, 1993, pp. 613–617.
[32] J. Aguilar, A. Colmenares, Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm, Pattern Analysis and Applications 1 (1998) 52–61.
[33] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, Perth Australia, vol. 4, 1995, pp. 1942–1947.
[34] R. Stork, K. Price, Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
[35] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: Proceedings of the IEEE Conference on Evolutionary Computation, Singapore, 1998, pp. 69–73.
[36] Y. Shi, R.C. Eberhart, Fuzzy adaptive particle swarm optimization, in: Proceedings of the 2001 Congress on Evolutionary Computation, Seoul, Korea, 2001, pp. 101–106.
[37] L. Zhang, H. Yu, S. Hu, A new approach to improve particle swarm optimization, Genetic and Evolutionary Computation (2003) 134–139.
[38] Q. Zheng, Y. Fan, Z. Shi, Y. Wang, Adaptive inertia weight particle swarm optimization, in: Lecture Notes in Computer Science, vol. 4029, Springer, Berlin, Heidelberg, 2006, pp. 450–459.
[39] G. Zee-Lee, Particle swarm optimization to solve the economic dispatch considering the generator constraints, IEEE Transactions on Power Systems 18 (2003) 1187–1195.
[40] P. Jinho, C. Kiyong, D. Allstot, Parasitic-aware RF circuit design and optimization, IEEE Transactions on Circuits and Systems 51 (2004) 1953–1965.
[41] M.A. Abido, Optimal design of power-system stabilizers using particle swarm optimization, IEEE Transactions on Energy Conversion 17 (2002) 406–413.
[42] J. Salerno, Using the particle swarm optimization technique to train a recurrent neural model, in: Proceedings of the 9th International Conference on Tools with Artificial Intelligence, IEEE Press, pp. 45–49.
[43] C.F. Juang, A hybrid genetic algorithm and particle swarm optimization for recurrent neural network design, IEEE Transactions on Systems Man and Cybernetics 32 (2004) 997–1006.
[44] Y. Da, X.R. Ge, An improved PSO-based ANN with simulated annealing technique, Neurocomputing Letters 63 (2005) 527–533.
[45] M. Karvalho, T.B. Ludermir, Particle swarm optimization of neural network architectures and weights, in: Proceedings of the 7th International Conference on Hybrid Intelligent Systems, Kaiserslautern, Germany, 2007, pp. 336–339.
[46] M. Meissner, M. Schmuker, G. Schneider, Optimized particle swarm optimization (OPSO) and its application to artificial neural network training, BMC Bioinformatics 7 (125) (2006).
[47] C. Zhang, H. Shao, An ANN's evolved by a new evolutionary system and its application, in: Proceedings of the 39th IEEE Conference on Decision and Control, Sydney, Australia, vol. 4, 2000, pp. 3562–3563.
[48] M. Settles, B. Rodebaugh, T. Soule, Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, Chicago, IL, in: Lecture Notes in Computer Science, vol. 2723, 2003, pp. 151–152.
[49] Y. Yu, L. Xi, S. Wang, An improved particle swarm optimization for evolving feed-forward artificial neural networks, Neural Processing Letters 26 (2007) 217–231.
[50] S. Kiranyaz, I. Turker, A. Yildirim, M. Gabbou, Evolutionary artificial neural networks by multi-dimensional particle swarm optimization, Neural Networks 22 (2009) 1448–1462.
[51] H.A. Abass, Speeding up back-propagation using multi-objective evolutionary algorithms, Neural Computation 15 (2003) 2705–2726.
[52] C. Cramer, E. Gelenbe, H. Bakircioglu, Low bit rate video compression with neural networks and temporal sub-sampling, Proceedings of the IEEE 84 (10) (1996) 1529–1543.
[53] E. Gelenbe, C. Cramer, M. Sungur, P. Gelenbe, Traffic and video quality in adaptive neural compression, Multimedia Systems 4 (6) (1996) 357–369.
[54] E. Gelenbe, M. Sungur, Random network learning and image compression, in: IEEE International Conference on IEEE World Congress on Computational Intelligence, 1994, pp. 3996–3999.

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

[55] C.E. Cramer, E. Gelenbe, Video quality and traffic QOS in learning-based subsampled and receiver interpolated video sequences, IEEE Journal on Selected Areas in Communications 18 (2) (2000) 150–166.
[56] H. Bakircioglu, E. Gelenbe, L. Carin, Random neural network recognition of shaped objects in strong clutter, in: Proceedings of the 7th International Conference on Neural Networks, in: Lecture Notes in Computer Science, vol. 1327, 1997, pp. 961–966.
[57] H. Bakircioglu, E. Gelenbe, Feature-based RNN target recognition, in: SPIE Proceedings, Orlando, FL, in: Aerosense 98, vol. 3370, (Algorithms for Synthetic Aperture Radar Imagery, V), 1998, pp. 508–518.
[58] S. Mohamed, G. Rubino, A study of real-time packet video quality using random neural networks, IEEE Transactions on Circuits and Systems for Video Technology 12 (12) (2002) 1071–1073.
[59] S. Mohamed, G. Rubino, H. Afifi, F. Cervantes, Real-time video quality assessment in packet networks: a neural network model, in: Proceedings of International Conference of Parallel and Distibuted Processing Techniques and Applications, PDPTA'01, Las Vegas, NV, 2001.
[60] S. Mohamed, G. Rubino, M. Varela, Performance evaluation of real time speech through a packet network, Performance Evaluation 57 (2004) 141–161.
[61] G. Rubino, M. Varela, J. Bonnin, Controlling multimedia QoS in the future home network using the PSQA metric, Computer Journal 49 (2006) 137–155.
[62] G. Rubino, P. Tirilly, M. Varela, Evaluating users satisfaction in packet networks using random neural networks, in: Proceedings 16th International Conference on Artificial Neural Networks, ICANN 2006, Athens, Greece, pp. 303–12.
[63] H. Abdelbaki, E. Gelenbe, T. Kocak, E. El-Khamy, Random neural network filter for land mine detection, in: Proceedings of the 16th National Radio Science Conference, NRSC, Am Shams University, Cairo Egypt, 1999.
[64] G. Loukas, G. Oke, A biologically inspired denial of service detector using the random neural network, in: Proceeding of the 4th IEEE International Conference on Mobile Adhoc and Sensor Systems, MASS'07, Pisa, Italy, IEEE Computer Society, Washington, DC, USA, 2007, pp. 1–7.
[65] G. Oke, G. Loukas, E. Gelenbe, Detecting denial of service attacks with bayesian classifiers and the random neural network, in: Proceedings of Fuzz-IEEE, London, UK, 2007 pp. 1964–1969.
[66] G. Loukas, G. Oke, Likelihood ratios and recurrent random neural networks in detection of denial of service attacks, in: Proceedings of International Symposium of Computer and Telecommunication Systems, SPECTS'07, San Diego, CA, USA, 2007.
[67] Gülay Öke, Georgios Loukas, A denial of service detector based on maximum likelihood detection and the random neural network, Computer Journal 50 (6) (2007) 717–727.
[68] E. Gelenbe, E. Seref, Z. Xu, Simulation with learning agents, Proceedings of the IEEE 89 (2001) 148–157.
[69] U. Halici, Reinforcement learning with internal expectation for the random neural network, European Journal of Operations Research 126 (2000) 288–307.
[70] E. Gelenbe, M. Gellman, R. Lent, P. Liu, P. Su, Autonomous smart routing for network QoS, in: Proceedings of the First International Conference on Autonomic Computing, New York, NY, 2004, pp. 232–239.
[71] E. Gelenbe, Steps toward self-aware networks, Communications of the ACM 52 (7) (2009) 66–75.
[72] G. Sakellari, The cognitive packet network: a survey, The Computer Journal 53 (2010) 268–279.
[73] E. Gelenbe, Product form queuing networks with negative and positive customers, Journal of Applied Probability 28 (1991) 656–663.
[74] J.R. Artalejo, G-networks: a versatile approach for work removal in queuing networks, European Journal of Operations Research 126 (2000) 233–249.
[75] UCI repository of machine learning databases. URL: http://www.ics.uci.edu/mlearn/MLRepository.html.
[76] S.E Fahlman, Faster learning variations on back-propagation: an empirical study, in: D.E. Touretzky, G. Hinton, T. Senjnowski (Eds.), Proceedings of the Connectionist Models Summer School San Mateo, Morgan Kaufmann, 1988, pp. 38–51.
[77] Y. LeCun, Efficient BackProp, in: Genevieve B. Orr, Klauss-Robert Muller (Eds.), Neural Networks: Tricks of the Trade, in: Lecture Notes in Computer Science, Springer, 1998, pp. 9–50.
[78] L. Beheva, S. Kumar, A. Patnaik, On adaptive learning rate that guarantees convergence in feedforward networks, IEEE Transactions on Neural Networks 17 (5) (2006) 1116–1125.
[79] C-T. Kim, J.-J. Lee, Training two-layer feedforward networks with variable projection method, IEEE Transactions on Neural Networks 19 (2) (2008) 371–375.
[80] S. Grossberg, Adaptive pattern classification and universal recoding, II: feedback, expectation, olfaction, and illusions, Biological Cybernetic (1976) 187–202.
[81] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, D.B. Rosen, Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps, IEEE Transactions on Neural Networks (1992) 698–713.
[82] G. Anagnostopoulos, Novel approaches in adaptive resonance theory for machine learning, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, University of Central Florida, May 2001.
[83] G.C. Anagnostopoulos, M. Bharadwaj, M. Georgiopoulos, S.J. Verzi, G.L. Heileman, Exemplar-based pattern recognition via semi-supervised learning, in: Proceedings of the 2003 International Joint Conference on Neural Networks, Portland, Oregon, USA, vol. 4, 2003, pp. 2782–2787.
[84] J.R. Williamson, Gaussian ARTMAP: a neural network for fast incremental learning of noisy multidimensional maps, Neural Networks 9 (5) (1996) 881–897.
[85] A. Kaylani, M. Georgiopoulos, M. Mollaghasemi, G.C. Anagnostopoulos, C. Sentelle, M. Zhong, An adaptive multi-objective approach to evolving ART architectures, IEEE Transactions on Neural Networks 21 (4) (2010) 529–550.
[86] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.
[87] I. Das, J. Dennis, A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems, Structural Optimization 14 (1997) 6–69.
[88] C.J.C. Burges, A tutorial on support vector machines for pattern recognition, Data Mining and Knowledge Discovery 2 (2) (1998) 121–167.
[89] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth, 1984.
[90] C.-C. Chan, C.-J. Lin, LIBSVM: a library for support vector machines. URL: http://www.csie.ntu.edu.tw/cjlin/libsvm.
[91] J.C. Platt, Fast training of support vector machines using sequential minimal optimization, Advances in Kernel Methods: Support Vector Learning (1999) 185–208.
[92] T. Lim, W. Loh, Y. Shih, A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, Machine Learning 40 (2000) 203–229.
[93] J. Demsar, Statistical comparisons of classifiers over multiple datasets, Journal of Machine Learning Research 7 (2006) 1–30.
[94] S. Garcia, F. Herrera, An extension on "statistical comparisons of classifiers over multiple data-sets" for all pairwise comparions, Journal of Machine Learning Research 9 (2008) 2677–2694.
[95] N.B. Karayiannis, A.N. Venetsanopoulos, Fast learning algorithms for neural networks, IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing 39 (7) (1992).
[96] B-H. Juang, S. Katagiri, Discriminative learning for minimum error classification, IEEE Transactions on Signal Processing 40 (12) (1992) 3043–3054.
[97] A. van Ooyen, B. Nienhuis, Improving the convergence of the back-propagation algorithm, Neural Networks 5 (1992) 465–471.
[98] V. Nedeljkovic, A novel multi-layer neural networks training algorithm that minimizes the probability of classification error, IEEE Transactions on Neural Networks 4 (4) (1993).
[99] J. Cid-Sueiro, J. Ignacio Arribas, S. Urban-Munoz, A.R. Figueiras-Vidal, Cost functions to estimate a posteriori probabilities in multi-class problems, IEEE Transactions on Neural Networks 10 (3) (1999) 645–656.
[100] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 2008.
[101] X. Yao, Evolving artificial neural networks, Proceedings of the IEEE 87 (9) (1999) 1423–1447.
[102] P.A. Castillo, J. Carpio, J.J. Merelo, A. Prieto, V. Rivas, Evolving multi-layer perceptrons, Neural Processing Letters 12 (2000) 115–127.
[103] J.M. Yang, C.Y. Kao, A robust evolutionary algorithm for training neural networks, Neural Computing and Applications 10 (2001) 214–230.

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006

[104] E. Cantu-Paz, C. Kamath, An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems, IEEE Transactions on Systems, Man and Cybernetics Part B 35 (2005) 915–927.
[105] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, Cambridge University Press, 1992.
[106] L. Ingber, Simulated annealing: practice versus theory, Journal of Mathematical and Computer Modeling 18 (11) (1993) 29–57.
[107] H. Muehlenbein, Schlierkamp-Vosen, predictive models for the breeder genetic algorithm, I. Continuous parameter optimizations, Evolutionary Computation 1 (1) (1993) 25–49.
[108] H.-M. Voigt, Soft Genetic Operators in Evolutionary Computation, Evolution and Biocomputation, in: Lecture Notes in Computer Science, vol. 899, Springer, Berlin, 1995, pp. 123–141.
[109] F. Aluffi-Pentini, V. Parisi, F. Zirilli, Global optimization and stochastic differential equations, Journal of Optimization Theory and Applications 47 (1) (1985) 1–16.
[110] B.D. Ripley, Neural Networks and related methods for classification, Journal of Royal Statistical Society Series B 56 (1994) 409–456.
[111] L.F.A. Wessels, E. Barnard, Avoiding false local minima by proper initialization of connections, IEEE Transactions on Neural Networks 3 (1992) 899–905.
[112] I. Jordanov, A. Georgieva, Neural network learning with global heuristic search, IEEE Transactions on Neural Networks 18 (3) (2007) 937–942.
[113] C.C. Coello, Evolutionary multi-objective optimization: a historical view of the field, IEEE Computational Intelligence Magazin (1) (2006) 2–36.
[114] H. Ishibuchi, T. Murata, I.B. Turksen, Single-objective and two-objective genetic algorithms for selecting limguistic rules for pattern classification problems, Fuzzy Sets and Systems 89 (2) (1997) 13–150.
[115] R.M. Everson, J.E. Fieldsend, Multi-class ROC analysis from a multi-objective optimization perspective, Pattern Recognition Letters 27 (8) (2006) 91–927.
[116] Y. Jin, R. Wen, B. Sendhoff, Evolutionary multi-objective optimizationg of Spiking neural networks, in: Proceedings of the International Conference of Artificial Neural Networks, in: LNCS, vol. 4668, Springer Verlag, Berlin, Heidelberg, 2007, pp. 37–379.
[117] L. Graning, Y. Jin, B. Sendhoff, Generalization improvement in multi-objective learning, in: Proceedings of the 2006 International Joint Conference on Neural Networks, 2006, pp. 9983–9900.
[118] Y. Jin, B. Sendhoff, Pareto-based multi-objective Machine Learning: an overview and case studies, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 38 (3) (2008) 397–415.
[119] J. Park, J.W. Sandber, Universal approximation using radial basis function neural networks, Neural Computation 3 (2) (1991) 246–257.
[120] G.A. Carpenter, S. Grossberg, The ART of adaptive pattern recognition by a self-organizing neural network, Computer 21 (3) (1988) 77–88.
[121] M. Georgiopoulos, G.L. Heileman, J. Huang, Convergence properties of learning in ART1, Neural Computation 2 (4) (1990) 502–510.
[122] M. Georgiopoulos, G.L. Heileman, J. Huang, Properties of learning related to pattern diversity in ART1, Neural Networks 4 (6) (1991) 751–758.
[123] M. Georgiopoulos, J. Huang, G.L. Heileman, Properties of learning in ARTMAP, Neural Networks 7 (3) (1994) 495–506.
[124] J. Huang, M. Georgiopoulos, G.L. Heileman, Fuzzy ART properties, Neural Networks 8 (2) (1995) 203–213.
[125] M. Georgiopoulos, H. Fernlund, G. Bebis, G.L. Heileman, Order of search in Fuzzy ART and Fuzzy ARTMAP: effect of the choice parameter, Neural Networks 9 (9) (1996) 1541–1559.
[126] G.C. Anagnostopoulos, M. Georgiopoulos, Category regions as new geometrical concepts in fuzzy ART and fuzzy ARTMAP, Neural Networks 15 (10) (2002) 1205–1221.
[127] G.J. Gibson, C.F.N. Cowan, On the decision regions of multi-layer perceptrons, Proceedings of the IEEE 78 (10) (1990).
[128] D. Hush, J.M. Salas, B. Horne, Error surfaces for multi-layer perceptrons, IEEE Transactions for Systems, Man and Cybernetics 22 (1992) 1152–1161.
[129] M. Brady, R. Raghavan, J. Slawny, Gradient descent fails to separate, in: Proceedings IEEE International Conference on Neural Networks, 1988, vol. 1, pp. 649–656.

Michael Georgiopoulos received the Diploma in EE from the National Technical University in Athens, his MS degree and Ph.D. degree in EE from the University of Connecticut, Storrs, CT, in 1983 and 1986, respectively. He is currently a Professor in the School of EECS at the University of Central Florida in Orlando, FL. His research interests lie in the areas of Machine Learning and applications with special emphasis on neural network algorithms, neuroevolutionary algorithms, and applications. He has published more than 60 journal papers and more than 180 conference papers in a variety of conference and journal venues. He has been an Associate Editor of the IEEE Transactions on Neural Networks from 2002–2006, and he is currently serving as an Associate Editor of the Neural Networks journal. He served as the General Chair of the S+SSPR 2008, the satellite workshop affiliated with the ICPR 2008 conference. He is currently serving as the Technical Co-Chair of IJCNN 2011.

Cong Li received the Bachelor's degree in Electronic & Information Engineering and a dual degree in Applied Mathematics from Tianjin University, China, in 2009. He is now working toward his Ph.D. degree in EE at the University of Central Florida. His research interests lie in the area of Machine Learning with special emphasis in kernel methods and outlier detection algorithms.

Taskin Kocak received B.S. degrees in Electrical and Electronic Engineering, and in Physics (as a double major) from Bogazici University, Istanbul, Turkey in 1996 as well as M.S. and Ph.D. degrees in Electrical and Computer Engineering from Duke University, Durham, NC, USA in 1998 and 2001, respectively. He is currently an associate professor and chairman of the Computer Engineering Department at Bahcesehir University, Istanbul, Turkey. Previously, he was an associate professor in the Electrical and Electronic Engineering Department at the University of Bristol, England, UK (2007–2009) and, he was an assistant professor of Computer Engineering at the University of Central Florida, Orlando, FL, USA (2001–2007). His research interests are in computer networks, hardware design and applied machine learning. His research activities have produced nearly 100 peer-reviewed publications, including 33 journal papers. He founded and organized the Advanced Networking and Communications Hardware Workshop series (2004–2006) which were supported by both IEEE and ACM. He is the founding editor-in-chief of the ICST Transactions on Network Computing. He also served as an associate editor for the Computer Journal (2007–2009) and as a guest editor for special issues of the ACM Journal on Emerging Technologies in Computing Systems and the Computer Journal.

Please cite this article in press as: M. Georgiopoulos, et al., Learning in the feed-forward random neural network: A critical review, Performance Evaluation (2010), doi:10.1016/j.peva.2010.07.006