The Computer Journal Advance Access published April 30, 2009

doi:10.1093/comjnl/bxp032

The Random Neural Network: A Survey

Stelios Timotheou*

Intelligent Systems and Networks Group, Department of Electrical and Electronic Engineering, Imperial College, London SW7 2BT, UK

**Corresponding author: stelios.timotheou05@imperial.ac.uk*

The random neural network (RNN) is a recurrent neural network model inspired by the spiking behaviour of biological neuronal networks. Contrary to most artificial neural network models, neurons in the RNN interact by probabilistically exchanging excitatory and inhibitory spiking signals. The model is described by analytical equations, has a low complexity supervised learning algorithm and is a universal approximator for bounded continuous functions. The RNN has been applied in a variety of areas including pattern recognition, classification, image processing, combinatorial optimization and communication systems. It has also inspired research activity in modelling interacting entities in various systems such as queueing and gene regulatory networks. This paper presents a review of the theory, extension models, learning algorithms and applications of the RNN.

Keywords: random neural network (RNN); survey; RNN extension models; learning algorithms; applications

Received 01 November 2008; revised 17 February 2009
Handling editor: Taskin Kocak

1. INTRODUCTION

The random neural network (RNN) is a neural network model inspired by the spiking behaviour of biophysical neurons [1]. When a biophysical neuron is excited, it transmits a train of signals, called action potentials or spikes, along its axon to either excite or inhibit the receiving neurons. The combined effect of excitatory and inhibitory inputs changes the potential level of the receiving neuron and determine whether it will become excited. In RNN, these signals are represented as excitatory and inhibitory spikes of amplitude $+1$ and -1, respectively, that are transmitted either from other neurons or from the outside world. Each neuron can fire only when its potential is strictly positive. The potential is equal to the number of positive spikes received that have not yet been fired or cancelled by inhibitory spikes.

The RNN has attracted a lot of attention in the scientific community. Various aspects of it have been explored, while several extension models and learning algorithms have been developed. In addition, the RNN has found widespread application in diverse areas of engineering and physical sciences. The success of the model can be attributed to its unique features which include the following [2].

- Although it is a recurrent neural network, its steady-state probability distribution is described by an analytical equation that can be easily and efficiently computed without the use of Monte Carlo methods.
- Its standard learning algorithm has low complexity and strong generalization capacity even for a relatively small training data set.
- It represents in a closer manner the signals transmitted in a biological neuronal network than other artificial neural networks (ANNs).
- It can be easily implemented in both software and hardware since its neurons can be represented by simple counters.
- There is a direct analogy between the RNN and the connectionist ANN.
- The neuron potential is represented as an integer rather than a binary variable resulting in a more detailed system-state description.
- It is a universal approximator for bounded continuous functions.
- The stochastic excitatory and inhibitory interactions in the network make it an excellent modelling tool for various interacting entities.

This paper attempts to briefly and comprehensively present the large amount of research published on the RNN since its introduction two decades ago. Our intention is to inform the interested reader about the theory and the different tools associated with the RNN that can be utilized for the solution of practical problems.

The paper is organized as follows: The RNN model and its steady-state properties are described in Section 2, while extension models are discussed in Section 3. Following is a presentation of the RNN learning algorithms as well as algorithms proposed for its extensions. The applications of RNNs are summarized in Section 5 and the paper is concluded in Section 6.

2. THE RNN MODEL

In this section, a mathematical description and the main results of the standard RNN model are given. We also discuss the stability of the network as well as its analogy to the connectionist ANN.

2.1. Mathematical model

The RNN is a recurrent network of N fully connected neurons which exchange positive and negative signals in the form of unit amplitude spikes. At any time t, the state of neuron i is described by its signal potential $k_i(t)$, which is a non-negative integer associated with the accumulation of positive signals at the neuron. We say that neuron i is excited when $k_i(t) > 0$, else if $k_i(t) = 0$, then it is idle or quiescent. A closely related parameter is $q_i(t) = \Pr[k_i(t) > 0] \leq 1$, which is the excitation probability of the neuron.

When neuron i is excited, it can randomly fire according to the exponential distribution with rate r_i resulting in the reduction of its potential by 1. The fired spike either reaches neuron j as a positive signal with probability $p^+(i, j)$ or as a negative signal with probability $p^-(i, j)$, or it departs from the network with probability $d(i)$. These probabilities must sum up to one yielding

$$\sum_{j=1}^{N} \left[p^+(i, j) + p^-(i, j)\right] + d(i) = 1 \quad \forall i \tag{1}$$

Hence, when neuron i is excited, it fires positive and negative signals to neuron j with rates:

$$w^+(i, j) = r_i p^+(i, j) \geq 0, \tag{2}$$

$$w^-(i, j) = r_i p^-(i, j) \geq 0. \tag{3}$$

Combining Equations (1)–(3) an expression which associates r_i with $w^+(i, j)$ and $w^-(i, j)$ is derived:

$$r_i = (1 - d(i))^{-1} \sum_{j=1}^{N} [w^+(i, j) + w^-(i, j)]. \tag{4}$$

Positive and negative signals can also arrive from the outside world according to Poisson processes of rates Λ_i and λ_i, respectively. Positive signals have an excitatory effect in the sense that they increase the signal potential of neuron j by 1. In contrast, negative signals have an inhibitory effect and cancel a positive spike if $k_j(t) > 0$, while if $k_j(t) = 0$, the negative signal has no effect.

The main symbols of the model are presented in Table 1, to facilitate the reader with the understanding of the text.

TABLE 1. List of RNN symbols

Notation	Definition
$k_i(t)$	Potential of neuron i at time t
q_i	Probability neuron i is excited at time t
Λ_i [λ_i]	External arrival rate of positive (negative) signals to neuron i
$\lambda^+(i)$ [$\lambda^-(i)$]	Average arrival rate of positive (negative) signals to neuron i
$p^+(i, j)$ [$p^-(i, j)$]	Probability the neuron j receives a positive (negative) signal from firing neuron i
$w^+(i, j)$ [$w^-(i, j)$]	Rate of positive (negative) signals to neuron j from firing neuron i
$d(i)$	Probability that a signal from firing neuron i departs from the network
r_i	Firing rate of neuron i
N	Number of neurons in the network
K	Number of input–output training pairs
x_{ik}	ith input value of the kth training pair
y_{ik}	ith output value of the kth training pair

2.2. Network behaviour in steady state

The state of the network is described by the vector of signal potentials at time t, $\mathbf{k}(t) = [k_1(t), \ldots, k_N(t)]$. Due to the stochastic nature of the network, we are interested in determining the stationary probability distribution $\pi(\mathbf{k}) = \lim_{t\to\infty} \pi(\mathbf{k}, t) = \lim_{t\to\infty} \Pr[\mathbf{k}(t) = \mathbf{k}]$, which can be described by the steady-state Chapman–Kolmogorov equations for continuous time Markov chain systems [1]:

$$\begin{aligned}
&\pi(\mathbf{k}) \sum_{i=1}^{N} \left[\Lambda_i + (\lambda_i + r_i)\mathbf{1}_{\{k_i>0\}}\right] \\
&\quad= \sum_{i=1}^{N} \Big\{\pi\left(\mathbf{k}_i^+\right) r_i d(i) + \pi\left(\mathbf{k}_i^-\right) \Lambda_i \mathbf{1}_{\{k_i>0\}} + \pi\left(\mathbf{k}_i^+\right) \lambda_i \\
&\qquad+ \sum_{j=1}^{N} \Big[\pi\left(\mathbf{k}_{ij}^{+-}\right) r_i p^+(i, j) \mathbf{1}_{\{k_j>0\}} + \pi\left(\mathbf{k}_{ij}^{++}\right) r_i p^-(i, j) \\
&\qquad+ \pi\left(\mathbf{k}_i^+\right) r_i p^-(i, j) \mathbf{1}_{\{k_j=0\}}\Big]\Big\}.
\end{aligned} \tag{5}$$

The values of the stationary parameters of the network, the stationary excitation probabilities $q_i = \lim_{t\to\infty} q_i(t)$, $i = 1, \ldots, N$, and the stationary probability distribution $\pi(\mathbf{k})$ are derived from Theorem 1.

THEOREM 1 [1]. *Let the total arrival rates of positive and negative signals $\lambda^+(i)$ and $\lambda^-(i)$, $i = 1, \ldots N$ be given by the following system of equations*

$$\lambda^+(i) = \Lambda_i + \sum_{j=1}^{N} r_j q_j p^+(j, i), \tag{6}$$

$$\lambda^-(i) = \lambda_i + \sum_{j=1}^{N} r_j q_j p^-(j,i), \tag{7}$$

where

$$q_i = \min\left\{1, \frac{\lambda^+(i)}{r_i + \lambda^-(i)}\right\}. \tag{8}$$

If a unique non-negative solution $\{\lambda^-(i), \lambda^+(i)\}$ exists for the non-linear system of Equations (6–8) such that $q_i < 1, \forall i$, then:

$$\pi(\mathbf{k}) = \prod_{i=1}^{N} \pi_i(k_i) = \prod_{i=1}^{N} (1-q_i)\, q_i^{k_i}. \tag{9}$$

The theorem states that whenever a solution to the signal flow Equations (6–8) can be found such that $q_i < 1, \forall i$, then the stationary joint probability distribution of the network has the simple product form (9) associated with the marginal probabilities of each neuron, $\pi_i(k_i)$. The condition $q_i < 1$ can be viewed as a 'stability condition' that guarantees that the excitation level of each neuron remains finite with probability one. Product form implies the independence of the neurons despite the fact that the neurons are coupled through the exchanged signals. A result of their independence is that we can easily compute parameters that are associated with a single neuron such as the average steady-state excitation level of neuron i, which is equal to $A_i = q_i/(1-q_i)$.

In [1], the case where a number of neurons are saturated is also discussed. Neuron i is *saturated* if $\lambda^+(i) \geq r_i + \lambda^-(i)$ so that it continuously fires in steady state and its excitation probability is equal to one. It is shown that the product form solution given by Equation (9) is still valid for the set of non-saturated neurons.

2.3. Network stability

The network is stable if the signal potential of each neuron does not tend to increase without bounds. Due to the product form stationary probability distribution of the system, stability is guaranteed if a unique solution exists to the non-linear system of Equations (6–8) and $q_i < 1, \forall i$. In addition, it can be easily shown that if a solution to Equations (6–8) exists with $q_i < 1, \forall i$ then it is unique [3]. The result stems from the fact that $\pi(\mathbf{k})$ is unique when $0 < q_i < 1, \forall i$ because the process $\{\mathbf{k}(t), t \geq 0\}$ is an irreducible continuous time Markov chain and $\pi(\mathbf{k})$ is positive with unit norm, which follows from Theorem 1. Furthermore, for any i it is impossible to have two different values q_i and q_i' satisfying the unique $\pi(\mathbf{k})$ when $k_i = 0$; hence existence of the solution implies its uniqueness.

As a result, the key to proving stability is to show the existence of the solution, a result that is non-trivial due to the non-linearity of the signal-flow equations. Early studies have examined the solution existence in special RNN architectures. In [1], it has been proved that a solution always exists in the feed-forward RNN architecture since the computation of q_i in one layer, depends only upon the values of neurons in the preceding layer which have already been computed. In [3], the existence of a solution is presented for balanced networks that have identical $q_i, \forall i$ and damped networks that are governed by the hyper-stability condition:

$$r_i + \lambda_i > \Lambda_i + \sum_{j=1}^{N} r_j p^+(j,i). \tag{10}$$

Although the hyper-stability condition appears to be strong, it can be used to appropriately select parameters of the network to guarantee stability [4].

The existence of a solution to the general case has been established in [5]. The approach followed is general and has also been used to examine the existence of a solution in extensions of RNN. Next, the proof to the existence of a solution $\{\lambda^+(i), \lambda^-(i)\}$ to Equations (6) and (7) is outlined.

Initially, the q_i terms are eliminated from Equations (6–7) and the latter are combined to obtain:

$$\boldsymbol{\lambda}^- - \boldsymbol{\lambda} = \boldsymbol{\lambda}^+\mathbf{F}\mathbf{P}^- = \boldsymbol{\Lambda}(\mathbf{I} - \mathbf{F}\mathbf{P}^+)^{-1}\mathbf{F}\mathbf{P}^-,$$
$$\boldsymbol{\lambda}^-, \boldsymbol{\lambda}^+, \boldsymbol{\lambda}, \boldsymbol{\Lambda} \in \mathbb{R}^{1\times N} \quad \text{and} \quad \mathbf{I}, \mathbf{F}, \mathbf{P}^{\pm} \in [0,1]^{N\times N}, \tag{11}$$

where $\boldsymbol{\lambda}^{\pm}$, $\boldsymbol{\Lambda}$ and $\boldsymbol{\lambda}$ are vectors representing the total and exogenous arrival rates of excitatory–inhibitory signals, $\mathbf{P}^+$ and $\mathbf{P}^-$ are square matrices, the elements of which are the probabilities $p^{\pm}(i,j)$, $\mathbf{I}$ is the identity matrix and $\mathbf{F}$ is a diagonal matrix with elements $f_{ii} = r_i/(r_i + \lambda^-(i)) \leq 1$.

Because $\mathbf{P}^+$ is sub-stochastic and all elements of $\mathbf{F}$ are smaller than 1, the series $\sum_{m=0}^{\infty}(\mathbf{F}\mathbf{P}^+)^m$ is geometrically convergent and hence we can write:

$$(\mathbf{I} - \mathbf{F}\mathbf{P}^+)^{-1} = \sum_{m=0}^{\infty}(\mathbf{F}\mathbf{P}^+)^m. \tag{12}$$

Defining $\mathbf{y} = \boldsymbol{\lambda}^- - \boldsymbol{\lambda}$ the system can be written in the fixed point form:

$$\mathbf{y} = \mathbf{g}(\mathbf{y}) = \sum_{m=0}^{\infty}(\mathbf{F}\mathbf{P}^+)^m\mathbf{F}\mathbf{P}^-, \tag{13}$$

where the dependence of $\mathbf{g}$ on $\mathbf{y}$ comes from $\mathbf{F}$, while $\mathbf{g}(\mathbf{y})$ is continuous and always non-negative. According to Brouwer's fixed point theorem, Equation (13) has at least one fixed point solution. In this case, exactly one fixed point $\mathbf{y}^*$ must exist since solution uniqueness has already been established. As a result, a solution to Equations (6–8) always exists and it is unique.

2.4. Analogy with the formal neural networks

In [1], the analogy between formal neurons and RNN neurons is discussed. In formal neural networks, the input to neuron i, v_i, is a combination of the weighted sum of other neuron outputs, y_j, and a threshold value θ_i such that $v_i = \sum_j w_{ji}^A y_j - \theta_i$. Whether neuron i will be excited or not is determined by an activation function according to $y_i = f(v_i)$. The analogy of RNN with this model is established for the unit-step activation function.

Because the RNN weights are non-negative, each weight $w^A_{ij} \in \mathbb{R}$ is represented by a pair of weights such that:

$$w^+(i,j) = \max\{0, w^A_{ij}\},\ w^-(i,j) = \max\{0, -w^A_{ij}\}$$

Moreover, non-output RNN neurons do not dissipate, $d(i) = 0$, and their firing rate r_i is given by Equation (4), while for output neurons, $d(i) = 1$. Parameters θ_i and y_i are associated with λ_i and q_i, respectively. When y_i is binary, a threshold value, α, can separate 0 and 1 according to:

$$[y_i = 0] \iff q_i < 1 - \alpha \quad \text{and} \quad [y_i = 1] \iff q_i \geq 1 - \alpha \quad \forall i.$$

Note that all RNN parameters are mapped to formal neurons' parameters except from the firing rates of the output neurons, the rate of external positive signal Λ_i and α. These parameters are set to appropriate values so that output neurons have the desired behaviour.

3. RNN EXTENSION MODELS

Apart from the original RNN, models of RNN with additional capabilities have been developed. Similar to the original RNN, all models maintain a product-form solution which may differ according to the model considered. In this section we describe the bipolar RNN (BRNN), a model of RNN with state-dependent firing (RNNSDF), the RNN with synchronized interactions (RNNSI) and the multiple class RNN (MCRNN).

3.1. Bipolar RNN

The bipolar RNN has been introduced in [6] to represent bipolar patterns and facilitate associative memory capabilities. Contrary to the original RNN there are two different types of neurons: (a) the positive neurons, which have behaviour same as the neurons of the original RNN and (b) the negative neurons, which have behaviour opposite to that of the positive ones. In other words, negative neurons accumulate negative signals so that the reception of positive signals has a suppressive role. Hence, signals emitted from a negative neuron i arrive to neuron j as positive (resp. negative) signals with probability $p^-(i,j)$ (resp. $p^+(i,j)$). The model is governed by similar signal-flow equations to the original RNN, taking into consideration the effect of both positive and negative neurons, while it retains a geometric product-form stationary probability distribution for the neuron potentials.

The BRNN has been applied in associative memory to obtain better separation between bipolar patterns. Moreover, the feed-forward BRNN has been utilized to prove the universal approximation properties of RNN discussed in Section 5.4.

3.2. RNN with state-dependent firing

Although in the original RNN the firing rate of neurons is constant, it is more biologically plausible to assume that the firing rate of neurons depends on the signal potential. In [7], a model with state-dependent firing has been proposed and its properties have been investigated. The RNNSDF defers from the original RNN in two aspects:

(i) The firing rate is exponentially distributed but it is potential-dependent instead of constant. Dependence is added as a multiplication factor so that the firing rate is $r_i f_i(k_i)$, where $f_i(k_i) > 0$ for $k_i > 0$ and bounded above by B_i.

(ii) When a negative signals arrives at an excited neuron j, it reduces the potential of the neuron by 1 with a state-dependent probability $f_j(k_j)/B_j$, otherwise it has no effect.

The authors proved that under certain conditions the model has a simple product-form solution, which is dependent on $f_i(k_i), \forall i$; this implies that the RNNSDF can exhibit a variety of stationary probability distribution structures by altering $f_i(k_i)$, in contrast to the RNN whose distribution is geometric and decreasing with respect to k_i.

3.3. RNN with synchronized interactions

The RNN with synchronized interactions has been introduced in [8,9] to capture the synchronous firing observed in nature by a large ensemble of neurons, as well as second or higher-order interactions of entities in different systems. In addition to the ordinary excitatory and inhibitory interactions, a firing neuron i may create a *synchronous interaction* at time t together with neuron j to affect some third neuron m, with probability $Q(i,j,m)$. The result of the interaction is to reduce the potential of neurons i and j and increase the potential of neuron m by 1. However if $k_j(t) = 0$, then the only thing that will occur is that $k_i(t^+) = k_i(t) - 1$, and the firing of i will have no other effect.

Thus, synchronous interactions take the form of a joint second-order excitation by cells i, j on m only if both neurons i and j are excited. This effect can generalize to an arbitrary number of neurons that fire synchronously. Indeed, if we have a sequence of neurons $j_1, \ldots, j_{n+1}, j_{n+2}$ such that $Q(j_l, j_{l+1}, j_{l+2}) = 1$ for $l = 1, \ldots, n$, then if neurons j_1 and j_2 are excited, eventually all the neurons $j_1, \ldots, j_{n+1}, j_{n+2}$ will fire.

As in the original RNN, the RNNSI is also governed by the same product form stationary probability distribution given by Equation (9) and the same stability condition, while the signal-flow equations are modified to include synchronous firing.

The RNNSI can be applied to problems where higher-order interactions between entities are present. For example, in gene regulatory networks apart from the interactions between individual genes, more than one genes may act together to excite another gene [10].

3.4. Multiple class RNN

In the MCRNN [11], there are C different classes of positive signals and a class of negative signals. As a result, the potential of neuron i is described by a vector of signal potentials, each associated with a different class of signals, $\mathbf{k}_i = [k_{i1}, \ldots, k_{iC}]$ so that $k_i = \sum_c k_{ic}$. Positive exogenous signals arrive to neuron i according to a Poisson distribution of rate Λ_{ic} and increase the potential k_{ic} by 1. Negative exogenous signals also arrive according to a Poisson distribution with rate λ_i. If at time t a negative signal is received by neuron i, then if it is excited, $k_i(t) > 0$, the potential of class c signals will become $k_{ic}(t^+) = k_{ic}(t) - 1$ with probability k_{ic}/k_i. When a neuron is excited it fires a class c signal with probability k_{ic}/k_i at rate r_{ic} and the potential k_{ic} is reduced by 1. If such an event occurs, the following can happen: (a) with probability $p^+(i, c; j, \xi)$ it goes to neuron j as a positive class ξ signal (b) with probability $p^-(i, c; j)$ it goes to neuron j as a negative signal or (c) it leaves the network with probability $d(i, c)$. As the other models, the MCRNN also obeys to a product-form solution for each neuron and each class of signals.

The MCRNN can be used in applications associated with the concurrent processing of different streams of information such as colours in image processing or attributes in a data network.

4. LEARNING ALGORITHMS

One of the most important features of a neural network model is its ability to learn from examples. In this section we describe the standard gradient descent supervised learning algorithm for the RNN [5] and other supervised learning algorithms proposed for the model and its extensions. Initialization algorithms that can be exploited by the supervised learning algorithms are also discussed. The section is completed with a discussion on RNN reinforcement learning (RL) algorithms.

4.1. Gradient descent supervised learning algorithm for the RNN model

A gradient descent supervised learning algorithm for the recurrent RNN has been developed by Gelenbe in [5]. In RNNs, the kth input training pattern $\mathbf{x}_k$ is represented by the vectors $\boldsymbol{\Lambda_k} = [\Lambda_{1k}, \ldots, \Lambda_{Nk}]$ and $\boldsymbol{\lambda_k} = [\lambda_{1k}, \ldots, \lambda_{Nk}]$. Usually the approach taken is to assign the input training values, x_{ik} to the exogenous arrival rates such that:

- If $x_{ik} > 0$, then $\Lambda_{ik} > 0$ and $\lambda_{ik} = 0$;
- If $x_{ik} \leq 0$, then $\Lambda_{ik} = 0$ and $\lambda_{ik} > 0$.

The values of the non-zero elements produced from the above expression can be taken equal to $|x_{ik}|$, or some constant value Λ and λ, respectively, to ensure network stability.

The desired values of the kth pattern, $\mathbf{y}_k$, are represented by the steady-state excitation probabilities of the neurons $\mathbf{q}_k = [q_{1k}, \ldots, q_{Nk}]$ emanating from applying input training pattern k to the network. The RNN weights updated during the learning process are $w^+(i, j)$ and $w^-(i, j)$.

Without loss of generality we assume that the error function to be minimized is a general quadratic function of the form:

$$E = \sum_{k=1}^{K} E_k = \frac{1}{2} \sum_{k=1}^{K} \sum_{i=1}^{N} c_i (f_i(q_{ik}) - y_{ik})^2, \qquad (14)$$

where E_k is the error function of the kth input–output pair, $c_i \in \{0, 1\}$ shows whether neuron i is an output neuron and $f_i(q_{ik})$ is a differentiable function of neuron i.

In the proposed approach by Gelenbe, the training examples are sequentially processed and the weights of the network are updated according to the gradient descent rule until a minimum of the error function is reached. If we denote by the generic term $w(u, v)$ either $w^+(u, v)$ or $w^-(u, v)$, the rule for updating the weights using the kth input–output pair at step $(\tau + 1)$ is

$$w_{\tau+1}(u, v) = w_\tau(u, v) - \eta \left[\frac{\partial E_k}{\partial w(u, v)} \right]_\tau . \qquad (15)$$

The partial derivative of the error function with respect to $w(u, v)$ can be calculated based on Equation (14) and yields:

$$\left[\frac{\partial E_k}{\partial w(u, v)} \right]_\tau = \sum_{i=1}^{N} c_i \left(f_i(q_{ik}) - y_{ik} \right) \times \left[\frac{\partial f_i(q_i)}{\partial q_i} \frac{\partial q_i}{\partial w(u, v)} \right]_\tau , \qquad (16)$$

where the operator $[\cdot]_\tau$ denotes that all calculations are performed using the weight values of step τ and the q_{ik} values derived from solving Equations (6–8) when the current weights $(w_\tau(u, v))$ are used. The challenging step in the evaluation of Equations (15) and (16) is the derivation of a closed expression for the term $[\partial q_i / \partial w(u, v)]_\tau$, which depends on the non-linear system of Equations (6–8). Gelenbe proved that the particular term can be brought to the form:

$$\frac{\partial \mathbf{q}}{\partial w(u, v)} = \boldsymbol{\gamma}(u, v) \, (\mathbf{I} - \mathbf{W})^{-1} , \qquad (17)$$

where $\mathbf{I}$ is the identity matrix and $\boldsymbol{\gamma}(u, v)$ denotes either $\boldsymbol{\gamma}^+(u, v)$ or $\boldsymbol{\gamma}^-(u, v)$, which are associated with $\partial \mathbf{q} / \partial w^+(u, v)$ and $\partial \mathbf{q} / \partial w^-(u, v)$, respectively. The parameters $\boldsymbol{\gamma}^+(u, v)$, $\boldsymbol{\gamma}^-(u, v)$ and $\mathbf{W}$ depend on the current values of $w(u, v)$ and $\mathbf{q}_k$ [5].

The steps of the gradient descent RNN learning algorithm are the following:

(1) Initialize the weights $w^+(u, v)$ and $w^-(u, v)$ $\forall u, v$ and appropriately choose the learning rate η.
(2) For each successive input–output pattern k do:
   (a) Initialize Λ_{ik} and λ_{ik} according to x_{ik};
   (b) Solve the system of Equations (6–8) using the current weight values;
   (c) Based on the values attained, calculate $\mathbf{W}$, $\boldsymbol{\gamma}^+(u, v)$ and $\boldsymbol{\gamma}^-(u, v)$, $\forall u, v$;

(d) Calculate $[\partial\mathbf{q}/\partial w^+(u,v)]_\tau$ and $[\partial\mathbf{q}/\partial w^-(u,v)]_\tau$ according to Equation (17);
(e) Update the weights from Equations (15) and (16). To satisfy the weight non-negativity constraint either the negative values can be set to zero or the iteration can be repeated with a smaller value of η.

(3) Repeat the procedure of step (2) until convergence.

The complexity of the algorithm for updating one weight $w(u,v)$ is $O(N^3)$ because of the inversion operation in Equation (17), which is the most demanding step of the algorithm [5]. Note that according to an iterative algorithm presented in [12], the complexity of solving the system of Equations (6–8) is $O(N^2)$ per iteration and the system converges at a rate better than a geometric sequence; hence it is less costly than the inversion operation.

The complexity of the algorithm can be further reduced if $(\mathbf{I}-\mathbf{W})^{-1}$ is approximated by the linear term $(\mathbf{I}+\mathbf{W})$; the approximation holds when $\|\mathbf{W}\| < 1$ [13].

4.1.1. Weight initialization

Supervised learning can be considered as a non-linear and non-convex optimization problem where our goal is to minimize the error function subject to the satisfaction of the signal-flow equations and the non-negativity constraints for our decision vector, the weights $w^+(i,j)$ and $w^-(i,j), \forall i,j$; hence, convergence to a global optimum cannot be guaranteed. For this reason, developing efficient weight initialization algorithms can help us obtain good solutions.

In the context of RNNs, the weight initialization methods proposed include random initialization, the Hebbian rule and a quadratic optimisation approach.

In random initialization small random values are used for the weights which are drawn from the uniform distribution in the range $[0, w_{\max}]$. In practice, a good choice for $w_{\max}$ is 0.2.

A Hebbian rule has been introduced in connectionist ANNs. According to this rule, a weight w_{ij}^H is assigned a positive value if the inputs to neurons i and j are either both high or both low. Likewise, w_{ij}^H is assigned a negative value if one neuron has high input and the other low. In RNNs, the weights are initialized according to the input values of the training patterns x_{ik}. When $x_{ik} \in [0,1]$, the rule described above can be quantified according to Equation (18) [14]:

$$w_{ij}^H = \sum_{k=1}^{K} (2x_{ik}-1)(2x_{jk}-1), \quad x_{ik} \in [0,1]. \tag{18}$$

If $w_{ij}^H > 0$, then we set $w^+(i,j) = w_{ij}^H$ and $w^-(i,j) = 0$, while if $w_{ij}^H < 0$, then the negative weight is assigned the non-zero value $|w_{ij}^H|$.

The quadratic optimization approach takes advantage of the fact that when all neurons in the network have desired values, the system of Equations (6–8) becomes linear, if $q_i < 1, \ \forall i$. Hence, the optimization problem is a linear non-negative least squares (NNLS) problem which is a convex optimization problem that can be solved to optimality. This approach was utilized in [15], where a solution method was developed for a special RNN architecture. This architecture is composed only of output neurons and it is not fully recurrent since each neuron can interact only with its local neighbours.

To confront the general case that the network is fully recurrent, and comprised of not only output neurons there are two main difficulties: (a) the formulated problem is of very large dimensionality (NK equations and $2N^2$ unknowns) and (b) we must deal efficiently with the non-output neurons. In [16], a projected gradient algorithm for the NNLS problem was developed which is suitable for large-scale problems. Furthermore, an efficient weight initialization algorithm was proposed that manages the case where the network is not comprised only of output neurons [17]. Performance evaluation in a combinatorial optimization problem indicated that this approach is better than random initialization.

4.2. Alternative RNN supervised learning algorithms

Apart from the standard gradient descent algorithm described in Section 4.1, other authors have also examined supervised learning in the context of the RNN.

The author of [18] has modified the Resilient Propagation (RPROP) algorithm and utilized it for RNN supervised learning. In RPROP, the weights are updated based only on the temporal behaviour of the sign of the error function derivative. It is considered to be a resilient and transparent method because it is not influenced by any unexpected behaviour of the value of the error function derivative. Nevertheless, the RPROP–RNN method has the same complexity with the standard learning algorithm, while the two methods produce comparable results in terms of recognition of noisy patterns.

In [19], the use of genetic algorithms (GA) in conjunction with the gradient descent RNN learning algorithm is proposed to address the problem of converging to a local solution. M RNNs are trained in parallel according to an iterative two stage process until convergence. The first stage involves training of the networks with the RNN supervised learning algorithm. In the second stage, new weights are generated for each RNN by applying genetic operations, such as mutation and crossover, to the network topologies. The genetic representation of each network is achieved through an extended direct coding scheme where both the presence of links and the values of the weights are included accomplishing both parametric and structural modifications. Apart from mutation and crossover operations, local search and optimization are also performed. Although this algorithm performs better than the gradient descent RNN algorithm it is significantly slower.

In [20], a quasi-Newton algorithm is developed for RNN supervised learning. Quasi-Newton algorithms are a well established class of iterative non-linear optimization techniques which rely on second-order gradient information; however, instead of computing the Hessian matrix, which holds the second-order partial derivatives of the error function, an approximation matrix $\mathbf{B}_\tau$ with desirable properties is updated in each iteration of the algorithm. The authors employ the well-known Broyden-Fletcher-Goldfarb-Shanno (BFGS) and Davidon-Fletcher-Powell (DFP) rules for updating $\mathbf{B}_\tau$. Contrary to the standard RNN learning algorithm, their approach is a batch learning algorithm where the weights are updated after processing all the training examples. The developed quasi-Newton algorithm is evaluated on the parity problem and comparison with the gradient descent algorithm demonstrates better performance and convergence to less than half steps compared with the standard RNN learning algorithm. Nonetheless, the algorithm cannot be used for online learning because it operates in a batch mode.

4.3. Supervised learning algorithms for RNN extension models

As discussed in Section 3, RNN extension models have more capabilities than the original RNN and are preferable in certain applications. As a result, it is important to develop efficient supervised learning algorithms for these models as well. In fact, gradient descent algorithms have been developed for the MCRNN and the RNNSI models based on the original RNN learning algorithm.

In the MCRNN learning algorithm [21], the gradient descent rule is employed to update the weights $w^+(i, c; j, \xi) = r_{ic}p^+(i, c; j, \xi)$ and $w^-(i, c; j) = r_{ic}p^-(i, c; j)$. This is achieved by obtaining the partial derivatives of $q_{ic}, \forall i, c$ with respect to these weights; the equations obtained have the same form with Equation (17). However, in the multiple class case, the computational complexity of updating a weight is $\mathrm{O}((NC)^3)$ because although we have N neurons, they must accumulate C different classes of positive signals.

The challenge in the RNNSI learning algorithm is to develop an algorithm that takes advantage of the synchronized interactions in an efficient manner. The authors of [8], in addition to the weights of the original RNN, introduce the new weight $w(i, j, l) = r_i Q(i, j, l)$, which is associated with the rate at which synchronized interactions take place between neurons i, j and l. To simplify the computation they assume that $w(i, j, l) = w^-(i, j)a(j, l)$, $a(j, l) \geq 0$, which holds the non-negativity property and also has similar effect on the neurons. By performing gradient descent learning with respect to the weights $w^+(u, v)$, $w^-(u, v)$ and $a(u, v)$, the equations describing the derivatives of q_i with respect to these weights are obtained. Although, RNNSI deals also with second-order interactions, the model maintains the equation form of the original RNN and hence the complexity of updating a weight remains $\mathrm{O}(N^3)$. Note that the RNNSI gradient descent algorithm has the capability to learn better than the original RNN since the latter corresponds to the special case of RNNSI that $a(u, v) = 0, \forall u, v$.

4.4. Reinforcement learning in RNN

RL methodologies have also been developed in the context of RNNs. In RL a system takes a sequence of cascaded decisions related to the perceived state of the environment and accordingly receives external reinforcement either positive (reward) or negative (punishment). The goal is to find an optimal policy to obtain maximum reward for each perceived state.

Halici proposed a RL scheme for a tree RNN-architecture for single and cascaded decisions [22–24]. In the general case, the system is composed of an input, a number of the intermediate and an output layer of neurons. The input neurons perceive the state of the environment, while the neurons of intermediate layer m represent the possible states that can be reached after the mth decision step. Each connection (i_{m-1}, j_m) represents the transition from state i_{m-1} to state j_m when decision a_m^τ of the τth trial is taken.

In each trial a signal is propagated from the input neurons to the output neurons, which dissipate and excite the environment that returns the external reinforcement. Decisions in the network are taken probabilistically. When neuron i is excited at trial τ, which means that it is an activated state, it fires a signal that reaches neuron j with probability $p_\tau^+(i, j)$ and activates it. The sequence of activated neurons from the input to the output layer is the decision cascade of trial τ, $\mathbf{a}_\tau$.

RL in RNNs works by updating the weights of the connections at the end of each trial. The update is related to the attained environmental reinforcement so that ‘good’ decisions are rewarded. The reinforcement $R_\tau(i, \mathbf{a}_\tau)$ that neuron i receives in this cascaded decision environment is a function of several parameters such as the trial, the external reinforcement associated with the output neuron and the cost of the activated decision path.

Halici has proposed three different weight update rules for single and cascaded decision rules: the reward rule (R-rule) [22], the reward and punishment rule (L-rule) [23] and the update rule with internal expectation of the reward (E-rule) [23,24]. Experimental analysis showed that the E-rule is superior to the other rules both in terms of learning and adaptivity to environmental changes.

The success of the E-rule relies on an adaptive internal expectation of the reward. In this rule, the weights are updated according to the difference between the actual reinforcement and the internal expectation. For example, if the difference is positive, the weights of all selected neurons in the decision path are reinforced proportionally to the difference, while their neighbour neurons are punished. In this way, the algorithm is adaptive to changes in the environment and results in obtaining time-varying reinforcement.

In the following Section we describe a variation of the E-rule that is extensively used for routing packets in the cognitive packet network (CPN) discussed in Section 5.6.

4.4.1. RNN–RL in CPN

The RNN reinforcement learning (RNN–RL) algorithm used in the CPN is a simple cascaded decision algorithm that employs the idea of the internal expectation of the reward as the E-rule. However, contrary to the tree-RNN architecture a recurrent architecture is employed, while decisions are based on the excitation probabilities of the neurons rather than the connection probabilities.

In this algorithm the reinforcement value is a quality of service (QoS) metric of the communication system such as path delay or packet loss. The reinforcement $R^+_\tau(i, \mathbf{a}_\tau)$ is also a function of the network node. For instance if the reinforcement is associated with the delay experienced in a source–destination pair in the communication network, then the reward is the inverse value of the delay from the particular node to the destination.

For each destination and QoS class, a node maintains a recurrent RNN, with non-dissipating neurons. Each neuron in an RNN corresponds to a neighbour of the particular node. The weights and the internal expectation of the reward, $R^+_{\tau,\beta}$, are updated whenever a new reward $R^+_{\tau+1}(i, \mathbf{a}_{\tau+1})$ reaches node i according to the following rule for all $i \neq j$:

- If $R^+_{\tau,\beta} \leq R^+_{\tau+1}(i, \mathbf{a}_{\tau+1})$

$$w^+_{\tau+1}(i, j) = w^+_\tau(i, j) + R^+_{\tau+1}(i, \mathbf{a}_{\tau+1}),$$

$$w^-_{\tau+1}(i, k) = w^-_\tau(i, k) + \frac{R^+_{\tau+1}(i, \mathbf{a}_{\tau+1})}{N-2}, \quad \forall k \neq j.$$

- Else

$$w^+_{\tau+1}(i, k) = w^+_\tau(i, k) + \frac{R^+_{\tau+1}(i, \mathbf{a}_{\tau+1})}{N-2}, \quad \forall k \neq j,$$

$$w^-_{\tau+1}(i, j) = w^-_\tau(i, j) + R^+_{\tau+1}(i, \mathbf{a}_{\tau+1}).$$

In the above equation, index j corresponds to the selected neuron of the network that resulted in the particular reward value. Numerical instability is avoided by re-normalizing the weights after each update by performing two operations. First, for each neuron i its new r^*_i is computed:

$$r^*_i = \sum_{m=1}^{n} \left[w^+_{\tau+1}(i, m) + w^-_{\tau+1}(i, m)\right].$$

Second, the weights are re-normalized such that:

$$w^+_{\tau+1}(i, j) \leftarrow w^+_{\tau+1}(i, j) * \frac{r_i}{r^*_i}, \tag{19}$$

$$w^-_{\tau+1}(i, j) \leftarrow w^-_{\tau+1}(i, j) * \frac{r_i}{r^*_i}. \tag{20}$$

Whenever an action is to be taken, the system of the signal flow Equations (6–8) is solved for the most recent weights to obtain values for the excitation probabilities of the neurons. The most excited neuron corresponds to the action to be taken and in the context of CPN represents the next node in the route of a packet.

5. RNN APPLICATIONS

In this section we report research on applications of RNN such as modelling queueing and biological networks, image processing, pattern recognition, classification, combinatorial optimization and communication systems. The different problems addressed in each application area are summarized in Table 2.

TABLE 2. Summary of RNN applications.

Application Area	Investigated problems and related references
Modelling	G-networks [25–33], genetic chromosome population [34], gene regulatory networks [10], corticothalamic oscillatory behaviour [35]
Optimization	Minimum vertex covering [4], assignment in emergency response [9], task assignment in distributed systems [39], satisfiability problem [40], MST [41], dynamic multicast [42], access network design [43], independent set [45], travelling salesman [46], optimal resource allocation [47], graph partitioning [49]
Hardware	Digital logic neuron [50], analog circuit neuron [51], simplified RNN–RL algorithm [52,53], probabilistic CMOS [55]
Image processing	Texture generation of gray [58,59] and colour [21,60] images, texture classification and retrieval [61], biomedical image segmentation [62,63], image fusion [64], image enlargement [64,65], image and video compression [66–69]
Communication systems	Cognitive packet network [70–73,75–77], DoS attack detection [78], automatic quantification of the PSQA metric for multimedia applications [79–83], call admission control in ATM networks [84], multimedia server modelling [85]
Simulation pattern recognition and classification	LAs [86], injection of autonomous agents in augmented reality [87] associative memory [6,14,18,88–90] target recognition [91], laser intensity vehicle classification system [92], wafer surface reconstruction [93], mine detection [94,95]

5.1. Modelling

The RNN is a prominent modelling tool that can capture the behaviour of interacting entities in different frameworks such as biological and queueing networks.

To begin with, due to the direct analogy between the RNN and queueing networks [25], the notion of inhibitory signals in RNNs inspired the use of negative customers in queueing networks for work removal [26]. This model attracted a lot of attention, resulting in a generalized class of networks called G-networks [27].

In G-networks positive and negative customers (excitatory and inhibitory signals) circulate in the network. When a positive customer arrives at a server (neuron), it increases the size of the queue (neuron potential) by 1, while the arrival of a negative customer cancels a positive one, if at least one is present in the queue. Service completion (neuron firing) decreases the queue size by one and causes the movement of a customer that will reach another node as a positive or negative customer or depart from the network in a probabilistic manner.

Negative customers may be replaced by signals that have a more general role. They can be used to trigger the instantaneous movement of a customer from one queue to another [28], or reset the queue to a new value when it is empty [29]. It is also possible to have multiple classes of positive and negative customers [30], as well as multiple classes of signals with a triggering effect [31]. As in RNNs, the stationary probability distribution of the queue lengths in G-networks have product form under certain conditions. Furthermore, it has been derived that two G-networks of the same size that have the same stationary rates of positive customers and signals for all queues (flow equivalence) are also governed by the same joint queue length probability distribution (flow equivalence) [32]. An extended survey of G-networks can be found in [33].

In [34], an analytical solution for a class of stochastic genetic algorithms is derived according to the behaviour of a population of different chromosome types. Different genetic operations are modelled using first-and second-order probabilistic interactions including the birth and death of a type i chromosome, the mutation of a type i chromosome to a type j chromosome, as well as the crossover operation where a type i chromosome combines with a type j chromosome to produce a type k chromosome.

In [10], the modelling of a gene regulatory network with interacting genes or other biochemical substances is studied. As in the genetic algorithms model, interactions between agents include not only first-order interactions but also second-order interactions, where more than two agents have a combined effect on a third one. Logical dependencies among agents are also modelled and an analytic steady-state solution of the overall system is derived.

Moreover, the RNN has been used to model interactions between different areas of the human brain to predict the oscillatory behaviour of those areas with response to somatosensory inputs [35]. In particular, a recurrent RNN is constructed with three neurons, each representing a neuronal layer associated with the thalamus, the cortex and the reticular layer. The strength of excitatory and inhibitory interactions betweens these areas, the measured firing rates and the delays in the signal propagation from one area to the other are considered in the model. The parameters of the constructed model are estimated using experimental data and exploited to predict the oscillatory behaviour of the system relating to different interaction strengths and delays.

5.2. Optimization

Combinatorial optimization problems routinely arise in many applications. However, they are usually NP-hard and cannot be optimally solved in a timely manner. Hopfield and Tank [36] proposed the use of ANNs, such as the discrete Hopfield network, for the solution of such problems attracting a lot of attention. Since their seminal work, different types of ANNs have been employed for the solution of various combinatorial optimization problems [37] and several neural techniques have been developed [38]. In this section we investigate the use of RNNs for the solution of discrete optimization problems. Emphasis is given to explain different solution approaches which include:

(i) The parameter association approach;
(ii) The dynamical RNN (DRNN) approach;
(iii) The energy function approach;
(iv) Supervised learning approach.

5.2.1. Parameter association approach

In the parameter association approach, different parameters of the RNN are associated with parameters of the optimization problem under consideration. Binary decision variables of the problem are represented by the q_i parameters of the RNN, while input parameters are usually associated with positive external arrivals Λ_i. The positive and negative probabilities of the RNN, $p^+(i, j)$ and $p^-(i, j)$ are employed to capture the interactions between decision variables, the neurons, that stem from the constraints of the problem. For example, if in a row of neurons only one must be selected, then each of these neurons attempts to inhibit the others from being selected. Excitation interactions are also possible when two neurons benefit from the mutual selection. Some parameters such as the firing rate r_i can be selected to guarantee hyper-stability of the network so that there is a balance between excited and not excited neurons.

The procedure for the solution of problems using this approach is very simple. After the RNN parameters have been given appropriate values, the q_i values are computed and the most excited neuron is selected. Then the problem is reduced and the procedure is repeated until the final solution is reached.

The approach has been applied for the solution of several NP-hard optimization problems including the minimum vertex covering (MVC) problem [4], a task assignment problem with communication and imbalance costs emerging in distributed systems [39] and the satisfiability (SAT) problem [40].

In the above problems, the RNN approach has been used to construct a solution from scratch. An alternative approach is to consider a good heuristic for the examined problem to find an initial solution and then apply the parameter association method to indicate variables that could be included in the solution to improve its quality. This method has been proposed in [41] for the solution of the minimum steiner tree (MST) problem. The authors use a known heuristic for the MST problem to obtain a solution and then employ an RNN to select a vertex-neuron that can be a good 'candidate' to be included in the solution. A re-optimization algorithm is then adopted to find a new solution based on the current MST and the selected vertex. The new solution is definitely not worse than the current one. By repeating this procedure until all vertices have been considered, the final solution is at least as good as the initial. Experimentation illustrated that the particular method can improve the solution quality by 10–20% without increasing the computational complexity.

The RNN approach for the solution of the MST problem has also been exploited for the solution of the dynamic multicast problem [42] and the access network design problem [43], which are closely related to the MST problem.

In general, the parameter association approach provides a trade-off between solution quality and computational complexity. It has consistently outperformed greedy heuristics for the different problems, but exhibited worse performance than metaheuristic techniques such as simulated annealing and genetic algorithms, which are very computationally demanding. The performance of the parameter association approach has also been compared with a Hopfield neural network algorithm in the MVC problem [44]. It was shown that the RNN is substantially better in solving the particular problem especially for large instances. Despite its success in many optimization problems, one of the limitations of this approach is that it cannot be easily generalized. Hence, it cannot be directly adopted for new problems but different parameter mappings must be tested to discover a good association.

5.2.2. *Dynamical RNN approach*

The idea behind the use of the Hopfield recurrent neural network for the solution of optimization problems relies on mapping the parameters of the problem to the energy function of this dynamic network. The energy function of the Hopfield network is guaranteed to converge to an attractor of the energy landscape, which is hopefully a good solution to the problem. Although the RNN is a recurrent neural network as well, it cannot perform dynamic behaviour in terms of its output parameters in the form $\mathrm{d}\mathbf{q}/\mathrm{d}t = f(\mathbf{q})$, because of its unique equilibrium point. Therefore, the DRNN has been developed to exhibit dynamic evolution with regard to its input [45,46].

The DRNN uses only positive inputs and negative weights with no signals leaving the network ($d(i) = 0$), whereas its dynamical equation is similar to the Cohen–Grossberg equation with time delayed feedbacks:

$$\frac{\mathrm{d}\Lambda_i}{\mathrm{d}t} = a(q_i)\left(B(q_i) - \frac{\partial F(\mathbf{q})}{\partial q_i}\right) \quad \forall i, \tag{21}$$

where function $a(q_i)$ regulates the convergence rate while $B(q_i)$ is appropriately selected to place the attractors of the dynamical system in the best possible positions. $F(\mathbf{q})$ is the penalty function associated with the optimization problem we are dealing with. Any constraints associated with the problem can be incorporated into $F(\mathbf{q})$ using the Lagrange multipliers method. The negative weights are assigned appropriate values so that a neuron inhibits the neurons that cannot be simultaneously excited with it.

After assigning appropriate values to the parameters of the DRNN model, the procedure progresses by iteratively computing the q_i values based on Equations (6–8) and updating the Λ_i values according to Equation (21) until the Λ_i values have stabilized.

The DRNN has been exploited for the solution of several optimization problems including the independent set [45] and travelling salesman problems [46], as well as a problem of optimal resource allocation with minimum and maximum activation levels for each resource and fixed costs [47]. Interestingly, the latter problem includes both binary and continuous variables. For modelling each of the continuous variables, the authors used n neurons. Neuron i_r, $1 \le i_r \le n$ is associated with variable a_r and represents a value 2^{-i_r} so that:

$$a_r = \sum\nolimits_{i_r=1}^{n} q_{i_r} 2^{-i_r}.$$

Performance comparison in the various problems has demonstrated the superiority of the DRNN over heuristic and Hopfield neural network approaches. An advantage of the DRNN compared with the parameter association approach is that it incorporates into the formulation the penalty function $F(\mathbf{q})$; nevertheless, some parameters such as the $B(q_i)$ and $w^-(i, j)$ are assigned values in an ad-hoc manner.

5.2.3. *Energy function approach*

In the DRNN the dynamic evolution of the input parameters Λ_i is used for the solution of optimization problems. Another approach is to evolve Λ_i according to the gradient of the cost function $F(\mathbf{q})$ [48]:

$$\Lambda_i^{\tau+1} = \Lambda_i^{\tau} - \eta\left[\frac{\partial F(\mathbf{q})}{\partial \Lambda_i}\right]_{\tau} = \Lambda_i^{\tau} - \eta\left[\frac{\partial F(\mathbf{q})}{\partial q_i}\frac{\partial q_i}{\partial \Lambda_i}\right]_{\tau}, \quad \forall i \tag{22}$$

In the above equation, the term $\partial q_i/\partial \Lambda_i$ can be computed based on Equation (8) yielding a linear system of equations as analysed in [49]. Furthermore, to simplify the computation of the term $\partial F(\mathbf{q})/\partial q_i$, the general quadratic energy function $E(\mathbf{q})$ has been proposed:

$$E(\mathbf{q}) = \sum_{i=1}^{N}\sum_{j<i} a_{ij}q_i q_j + \sum_{i=1}^{N} a_{ii}q_i^2 + \sum_{i=1}^{N} b_i q_i + c. \quad (23)$$

A result of the above formulation is that if the optimization problem at hand is of similar form, then by mapping the parameters of the problem to the energy function, the term $\partial E(\mathbf{q})/\partial q_i$ can be easily computed without any additional effort. The other parameters of the RNN are initialized so that the constraints associated with the problem are strengthened through the neuron interactions, as discussed in previous approaches.

The solution procedure is similar to that of the DRNN approach, with the difference that the Λ_i values are updated not based on Equation (21), but based on Equation (22). Additionally, both the RNN and the energy function parameters must be assigned appropriate values.

The approach has been evaluated with respect to two optimization problems: the graph partitioning problem and the MVC problem [49]. The method exhibits better or comparable performance compared with greedy heuristics and two metaheuristic search techniques (simulated annealing and genetic algorithms).

5.2.4. *Supervised learning approach*

In [9] supervised learning has been considered for the solution of combinatorial optimization problems and particularly for an NP-complete assignment problem arising in an emergency response. The idea is to train an RNN architecture with instances of the optimization problem under consideration. The inputs to the network are the input parameters of one problem instance, while the desired output is the optimal solution corresponding to the particular instance.

Then, the trained network is used to obtain fast and close to optimal solutions for other instances of the optimization problem. The problem studied in [9] involves the optimal allocation of emergency units to locations of incidents having a different number of injured civilians with the objective of collecting as many civilians as possible in minimum time. The experimental results obtained for varying number of emergency units and incidents demonstrate the efficiency of the approach. In all cases examined, the obtained results diverged at most 10% from the optimal solution, while the number of civilians collected was above 95% in almost all cases.

5.3. Hardware implementations

The processing capabilities of brain neuronal networks rely on their massively parallel architecture. ANNs are parallel as well, but software implementations result in sequential execution. Therefore, it is important to implement neural networks in hardware. In [50], an implementation of the RNN using discrete logic integrated circuits is presented. The realization of the network is achieved using four modules. An input module is needed for the input signals, while a second module is responsible for the aggregation of signals for each neuron. A random number generator is required for the realization of the exponential firing of neurons. The last component is the routing module for the propagation of signals between neurons.

An analog implementation of the RNN that captures the addition and multiplication operations performed in RNNs has been proposed in [51]. Nonetheless, in both implementations, once the circuit is realized the routing probabilities cannot be altered restricting the functionality of the circuit.

Research has also been undertaken in developing hardware implementations of the RNN–RL algorithm, which is used in CPN. The authors of [52], introduced a simplified RNN–RL algorithm with $2N$ instead of $2N^2$ weights for the implementation of a CPN smart packet processor. This simplification is based on the fact that all weights of the form $w(j,i)$, $\forall j$ are always updated by equal increments so that if their initial values are equal, they remain equal after any update. As a result, for given i, the N weights $w(j,i)$, $\forall j$ can be represented by the weight $w(i)$. The weight number reduction significantly enhances the performance in terms of execution time, memory storage and realization simplicity. The complexity of the RNN–RL CPN algorithm can be further reduced if the q_i values are directly updated from the reinforcement [53]. This algorithm was implemented on an FPGA and performance evaluation showed almost an order of magnitude speed-up without deterioration of the results.

The stochastic nature of the RNN allows its efficient realization on probabilistic CMOS (PCMOS). PCMOS harness the probabilistic behaviour of the circuits exhibited in the nanoscale regime, because of process variations and noise, yielding significant improvements in energy and performance [54]. The authors of [55], realized the RNN on a PCMOS co-processor for the solution of the MVC problem. They implemented the core probabilistic module of RNN associated with the random firing of neurons on PCMOS, instead of a pseudo-random number generator, and the rest of the network on a conventional microprocessor. Experimental evaluation showed that the PCMOS RNN co-processor exhibited less energy consumption and execution speed-up by orders of magnitude compared with an implementation on a conventional microprocessor.

5.4. Function approximation

One important feature of a neural network model is its ability to approximate functions with an arbitrary degree of accuracy. The authors of [56] have proved that the feed-forward BRNN and the clamped RNN (CRNN) have the universal approximation ability for any continuous function on a bounded set [0,1], i.e. functions of the form $f : [0,1]^s \to \mathbb{R}^w$. Such functions can be separated into one-dimensional functions of the form

$f_w : [0, 1]^s \to \mathbb{R}$; hence the important step is to prove universal approximation of this case.

To prove the universal approximation property, the authors first established the result for a 1-input 1-output function and then generalized it to the s-input 1-output case. Their method is based on constructing an RNN that reproduces a polynomial that is an estimator of the bounded continuous function under consideration. The polynomial consists of m terms of the form $c_j(1/(1+x))^j$ or $c_j(x/(1+x)^j)$, where $c_j \in \mathbb{R}$.

To prove the result for the BRNN model, sub-RNNs are built to represent each of the terms $(1/(1+x))^j$. Then, the sub-RNNs corresponding to positive (resp. negative) c_j coefficients are connected with appropriate weight values to a positive (resp. negative) output neuron. The sum of the average potentials of the two output neurons reproduces the polynomial that estimates the desired function. The proof for the CRNN, which is the original RNN with the addition of a constant c, is again by construction but the approximating polynomial is the one with terms $c_j(x/(1+x)^j)$.

Note that in the above approach no restrictions are imposed to the structure of the network besides being feed-forward. In [57], the approximation capabilities of RNN were further explored to limit the number of total layers required. The authors proved that for both the BRNN and CRNN models, functions of the form $f : [0, 1] \to \mathbb{R}$ can be arbitrarily approximated with an architecture of one input, one hidden and one output layers. Extending the result for the approximation of functions $f_w : [0, 1]^s \to \mathbb{R}$, it was derived that for both the BRNN and CRNN models arbitrary approximation can be accomplished by considering an architecture with s-hidden layers.

5.5. Image processing

The recurrent nature of the RNN and its low complexity supervised learning algorithm make it highly applicable in image processing. Numerous image processing problems have been explored with the RNN including the generation, classification and retrieval of textures, the compression of still and moving images as well as image enlargement and fusion.

In texture modelling and generation (synthesis) [58,59], each pixel is associated with a neuron in a recurrent RNN architecture with connections between immediate spatial neighbour neurons. The gray level of each pixel is normalized and associated with the excitation probability of the corresponding neuron. Texture examples with desired characteristics such as granularity, inclination, contrast and homogeneity are used to train the RNN. The trained network can be regarded as the texture model that is used to synthesize textures with similar characteristics. Experimental evaluation confirmed the ability of the RNN to generate textures with the same statistical properties as the training ones, at low computational cost. Texture synthesis has also been examined for colour images [21,60]. In this case, the MCRNN model and its learning algorithm are utilized so that each colour is represented by a different RNN signal class.

Texture classification and retrieval using the RNN has been investigated in [61]. Given an input image, the purpose in texture classification is to categorize each pixel into a texture class. For this purpose an RNN is constructed and trained with sample images for each texture class. The pixels of a test image are labelled one by one. Specifically, each pixel along with its neighbour pixels are presented to the trained networks to produce an output image window. The pixel under examination is associated with the class whose RNN produced the smallest Euclidean distance between the input and the output. Similarly, texture retrieval is achieved by training one neural network with respect to the desired texture class and then selecting the neurons-pixels whose distance is less than a pre-specified threshold.

Another image processing area which involves classification is image segmentation, an area particularly useful in biomedical image analysis. In image segmentation, the objective is to separate the image into segments with specific characteristics. The RNN has been applied to the segmentation of both magnetic resonance [62] and ultrasound images [63]. The general approach followed is very similar to the approach followed in [61] for the classification of textures. A number of recurrent RNNs, equal to the regions of interest, are trained each for a different region. As a result, by presenting different blocks of an image to the RNNs, each block is assigned to the segment that resembles it most.

To apply the RNN in image enlargement, the authors of [64,65] construct a feed-forward RNN composed of an input layer with a number of neurons equal to the number of pixels of the small image, a hidden layer and an output layer equal to the number of pixels in the enlarged image. The input is the image to be enlarged and the output is the difference image between the zero-order interpolation and the desired output images. The output image is then added to the interpolated image to obtain the enlarged one. Training is performed using a set of large images that are representative of the images of interest. In order to facilitate learning and decrease the computational time, the input images are divided into blocks that are learned separately.

A similar approach has been followed in image fusion where the goal is to produce a good quality image from a number of low quality sensor images [64]. The main difference is that the inputs to the training network are the fused images and the outputs are the respective real images.

The application of RNN in the compression of still and moving images (video) has been investigated in [66–69]. Compression of still images can be achieved by training a feed-forward RNN. The training procedure is similar to the image enlargement approach, but in this case the number of neurons in the hidden layer is chosen to achieve a particular compression ratio. In addition, the objective function is related to the minimization of the error between the original image (input) and the compressed one (output).

Video compression depends heavily on the compression of still images but also involves efficient management of blocks of different frames in the image sequence. The latter has been accomplished using three different techniques. First, a motion detector is used to conside only blocks of frames where motion has been detected. This is achieved by comparing the mean gray value of a block with its corresponding block in the previous frame [67,68]. The second technique involves an adaptive compression scheme for selecting a well-suited compression ratio for each block according to a quality threshold [67,68]. Finally, subsampling is applied so that only a portion of the frames is considered; the rest are generated by interpolation. In this case the function approximation capability of the RNN can be utilized [69]. Using the aforementioned techniques, significant image and video compression ratios can be achieved, which are comparable to other compression and interpolation techniques, while decompression is done at a faster rate.

5.6. Communication systems

Both the learning and modelling capabilities of RNNs have been extensively exploited in communication systems.

The most important exploitation of RNNs in communication networks corresponds to the utilization of the RNN–RL algorithm to control the routing of packets in the *Cognitive Packet Network* developed in [70–73]. A CPN is a connectionless, packet switching network with intelligent capabilities added to the packets rather than the nodes, for routing and flow control. This is achieved by sending into the network special cognitive packets called *smart packets* which discover new routes from a source to a destination according to a QoS metric such as throughput, delay, jitter and packet loss [74]. The smart packet routing decisions are taken either randomly or based on the RNN–RL algorithm discussed in Section 4.4.1. Random decisions contribute to the exploration of the network while RL-based decisions contribute to the attainment of the best available route for the desired QoS metric. The information gathered by smart packets is sent back to the source via acknowledgement packets so that the state of the RNNs is updated accordingly. Another class of packets, the *dump packets*, carry the payload through the best route discovered so far.

A prominent characteristic of the CPN, is its ability to adapt according to the user goals so that diverse data streams follow different routes based on their QoS requirements. The flexibility of the CPN for goal-oriented routing has also been extended to *ad-hoc* networks where power utilisation and path availability is of primary concern [75], as well as sensor networks where the preservation of energy is essential [76]. A CPN can also play an important role in future intelligent networks where users can both dynamically request services according to their QoS needs and provide services, especially in the search for services [77].

Apart from the RNN–RL, RNN supervised learning has also been exploited in the CPN to detect denial of service (DoS) attacks [78]. The objective is to collect information from traffic parameters affected during a DoS attack and use it to train the RNN to detect an attack. Performance of the network for six such variables showed that DoS attacks can be correctly identified in approximately 95% of the cases.

Moreover, supervised RNN learning has been applied for automatic quantification of the quality of traffic associated with real-time multimedia applications such as video, speech and interactive voice [79–82]. In order to quantify traffic quality the authors introduced a pseudo-subjective quality assessment (PSQA) metric and identified quality-affecting parameters such as delay, jitter and packet loss. Automatic quantification was accomplished by training the RNN with appropriate input–output pairs. The inputs corresponded to distorted samples of the affecting parameters collected from the network, while the outputs were the mean opinion score (MOS) for each of the samples, subjectively quantified from a human expert. RNN training was performed with a subset of these data and validated with its complement. Experiments conducted with ANNs and Bayesian Classifiers confirmed the superiority of the RNN in terms of performance, generalization and real-time computational complexity [83].

The RNN's modelling capacity has also been helpful in communication networks. The author of [84] used RNN modelling and learning for call admission control in ATM networks. Specifically, each server in the network is modelled using three neurons. The first neuron corresponds to the server's buffer occupancy, the second to the utilization of the server and the third is representative of the traffic variation. It is assumed that each server exhibits the behaviour of an M/G/1 queue and hence the M/G/1 formulas are employed to map the parameters of the neurons to the parameters of the traffic. The overall network is modelled by connecting the sub-RNNs together according to the connections of the actual communication system. Weight learning at each of the sub-RNNs is performed to adapt to the dynamically changing conditions of the traffic. The RNN utilization allowed the prediction of the traffic characteristics so that call admission control could be successfully performed.

Modelling the behaviour of a communication system using an RNN has also been considered in [85]. The authors exploited a variation of the G-network with batch removals to model the behaviour of a multimedia server that is responsible for satisfying the customer requests to view a video document. The G-network usage resulted in an efficient service scheme that maximizes the utilization of the server under limited resources.

5.7. Simulation

In traditional simulation approaches, the designer has full control over the behaviour of the synthetic entities represented. For each type of entity, rules must be specified for all possible conditions so that the entities can take appropriate decisions, a process that is both complex and time consuming. Furthermore, human decisions are not only based on current conditions but are also influenced by attributes such as observation, learning and emotions.

These problems can be overcome with the introduction of learning agents (LAs) as proposed in [86]. LAs acquire information from the environment and other agents to act in favour of their pursued goals. The cognitive map of each agent, associated with the gained experience of the agent, is stored in RNNs and updated using RL. Different RNNs are associated with different actions and goals. Similar to RNN–RL in the CPN, an RNN is responsible for both storing the internal representation of the agent and taking decisions. The LA approach was tested in the context of a simulation where agents represent vehicles with the goal of traversing a dangerous grid area with minimum delay and damage. In each intersection, agents can decide to change direction based on the local conditions, as well as the delay and damage experienced by other agents who passed from the particular intersection and had the same final destination. Experimentation showed that RNN–RL had comparable performance to an algorithm that incorporated a *priori* information about the condition of the environment. Supervised learning has also been tested but experimental evaluation indicated that it is inferior to RL in the particular context.

LAs have also been employed to generate autonomous realistic behaviour for artificial entities that are injected into realistic settings [87]. Augmenting reality with synthetic conditions is important in generating realistic training exercises offering the authenticity of a real exercise at much lower cost and hazards. As in [86], the authors employed the RNN–RL algorithm for agent navigation, but also introduced imitation and grouping capabilities. Consequently, the agents demonstrated realistic behaviour in a scenario that involved a group of agents going from location A to location B in a hostile terrain without being hit.

5.8. Pattern recognition and classification

It is well known that neural networks are highly successful in pattern recognition and classification problems. This stems from their ability for learning and generalization. The RNN has been particularly successful in these problems.

Early applications of the RNN concentrated on its use as an associative memory, which can recognize noisy patterns. Recurrent RNN architectures are usually used with each neuron representing one bit of information. The authors of [88,89] investigated the use of the BRNN model for associative memory operations. In [88], they proposed a technique to retrieve stored symbolic images from incomplete or erroneous information. Their technique was based on adapting the external arrival rates and the sign of each neuron (positive or negative) so that the neurons can closely represent the stored patterns. More techniques concerning the storage and reconstruction of patterns with the BRNN model were examined in [89]. It was shown that the BRNN has the capacity to store up to $N/2$ patterns outperforming other well-known associative neural network models. Associative memory for the recall of noisy patterns has also been studied in the context of the original RNN [14,18]. Contrary to [88,89], RNN learning was performed by adapting the weights of the network and fixing the external arrival rates. Results attested the ability of the RNN to recognize noisy patterns achieving recognition rates greater than 98% with-distortion rates less than 25%. Finally, the MCRNN and its learning algorithm have been exploited to learn how to retrieve noisy colour patterns, by representing each colour with a different signal class of the model [90].

Apart from the recognition of noisy patterns, the RNN has been utilized in the automatic target recognition of objects based on synthetic aperture radar images [91]. The approach followed is to train feed-forward RNNs to learn different segments of the target. In each RNN an output neuron shows whether the particular segment of the target is present or not. Each RNN is trained to detect its target segment. To detect a target in an image, neighbouring blocks of the image are sequentially examined by the RNN and their decisions are fused to make a common decision for the location of the target. The detection probability of the network for noise levels up to 20% was almost equal to 1.

In [92], a vehicle classification system was presented. The system extracts features from intensity images produced from laser sensors such as geometrical characteristics of the vehicle, speed and edge points and uses them to train a feed-forward RNN. The RNN achieves a misclassification percentage less than 10% when classifying vehicles into five different categories.

The authors of [93] addressed a problem in semiconductor fabrication associated with the prediction of the profile of cross-section wafer surface images from the intensity profile of top-down images. This reconstruction process is important as it allows failure analysis engineers to recognize whether a wafer surface is defective. Three approaches were proposed using the RNN supervised learning algorithm. The first relied on using training data from real images of both normal and defective surfaces to directly map intensity profiles to cross-section profiles. The other two approaches relied on mathematical and physical modelling in obtaining expressions that relate the intensity profile function with the cross-section profile and applying RNN supervised learning to acquire good values for the unknown expression parameters. Experimental evaluation indicated that the direct mapping method is more successful for abnormal surfaces, whereas the physical modelling method works better for normal surfaces.

In the search for mines, detection is sometimes not sufficient because false alarms can increase the cost and time spent. In [94,95], two robust non-parametric techniques based on the RNN were developed for the identification of mines according to data from a calibrated landmine field. The first approach is based on training a feed-forward RNN to identify mines using input values from two features extracted from the minefield

[96]. The second is based on determining the presence of mines with respect to the electromagnetic induction (EMI) energy measured at different locations. Apart from the detection of mines, the problem of autonomously and effectively exploring a minefield has been considered in [97]. To address the fact that different minefield regions have different probabilities for having mines, the authors proposed stochastic modelling of the minefield. Furthermore, to accomplish the goal of maximizing the number of mines found per unit time, they developed a gradient descent algorithm in association with their modelling, which is similar to the RNN supervised learning algorithm.

6. CONCLUSIONS

In this paper we have surveyed the research work undertaken on the RNN. The RNN is a biologically inspired, open, recurrent neural network with closed form expression for the probability steady-state and analytically solvable signal-flow equations. The properties of the model, extensions and learning algorithms have been described. Furthermore, numerous RNN applications have been reviewed with the emphasis placed on their underlying methodology. The plethora and diversity of applications reflect the prominence of the RNN both as a modelling and as a learning tool.

FUNDING

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Networks) project and is jointly funded by a BAE (British Aerospace) Systems and EPSRC (Engineering and Physical Sciences Research Council) strategic partnership (EP/C548051/1).

ACKNOWLEDGEMENT

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Networks) project and is jointly funded by a BAE Systems and EPSRC strategic partnership (EP/C548051/1).

REFERENCES

[1] Gelenbe, E. (1989) Random neural networks with negative and positive signals and product form solution. *Neural Comput.*, **1**, 502–510.

[2] Gelenbe, E. (1991) Theory of the Random Neural Network. In Gelenbe, E. (ed.), *Neural Networks: Advances and Applications*, pp. 1–20. Elsevier Science Publishers B.V.

[3] Gelenbe, E. (1990) Stability of the random neural network. *Neural Comput.*, **2**, 239–247.

[4] Gelenbe, E. and Batty, F. (1992) Minimum Graph Covering with the Random Neural Network Model. In Gelenbe, E. (ed.), *Neural Networks: Advances and Applications*, Vol. 2, pp. 215–222. Elsevier Science Publishers B.V.

[5] Gelenbe, E. (1993) Learning in the recurrent random network. *Neural Comput.*, **5**, 154–164.

[6] Gelenbe, E., Stafylopatis, A. and Likas, A. (1991) Associative Memory Operations of the Random Neural Network. *Proc. Int. Conf. Artificial Neural Networks*, Espoo, Finland, June 24–28 pp. 307–312. North-Holland, Amsterdam.

[7] Sungho, J., Yin, J. and Zhi-Hong, M. (2005) Random neural networks with state-dependent firing neurons. *IEEE Trans. Neural Netw.*, **16**, 980–983.

[8] Gelenbe, E. and Timotheou, S. (2008) Synchronised interactions in spiked neuronal networks. *Comp. J.*, **51**, 723–730.

[9] Gelenbe, E. and Timotheou, S. (2008) Random neural networks with synchronised interactions. *Neural Comput.*, **20**, 2308–2324.

[10] Gelenbe, E. (2007) Steady-state solution of probabilistic gene regulatory networks. *Phys. Rev. E*, **76**, 031903.

[11] Gelenbe, E. and Fourneau, J.M. (1999) Random neural networks with multiple classes of signals. *Neural Comput.*, **11**, 953–963.

[12] Fourneau, J. Computing the Steady State Distribution of Networks with Positive and Negative Customers. *Proc. 13th IMACS World Cong. Computational and Applied Mathematics*, Dublin, Ireland, July. North-Holland, Amsterdam.

[13] Halici, U. and Karaoz, E. (1998) A Linear Approximation for Training Recurrent Random Neural Networks. *Proc. 13th Int. Symp. Computer and Information Sciences*, Antalya, Turkey, October 26–28, pp. 149–156. IOS Press, Amsterdam.

[14] Hubert, C. (1993) Supervised learning and recall of simple images with the random neural network. *C. R. Acad. Sci. I*, **316**, 93–96.

[15] Atalay, V. (1998) Learning by Optimization in Random Neural Networks. *Proc. 13th Int. Symp. Computer and Information Sciences*, Antalya, Turkey, October 26–28, pp. 143–148. IOS Press, Amsterdam.

[16] Timotheou, S. (2008) Nonnegative Least Squares Learning for the Random Neural Network. *Proc. 18th Int. Conf. Artificial Neural Networks* (ICANN 2008), Prague, Czech Republic, September 3–6, pp. 195–204. Springer, Berlin, Heidelberg.

[17] Timotheou, S. (2009). A novel weight initialization method for the random neural network. *Neurocomputing*, to appear.

[18] Hubert, C. (1993) Learning internal representations with the N–M–N random neural network. *C. R. Acad. Sci. II*, **317**, 451–456.

[19] Aguilar, J. and Colmenares, A. (1998) Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm. *Pattern Anal. Appl.*, **1**, 52–61.

[20] Likas, A. and Stafylopatis, A. (2000) Training the random neural network using quasi-Newton methods. *Eur. J. Oper. Res.*, **126**, 331–339.

[21] Gelenbe, E. and Hussain, K. (2002) Learning in the multiple class random neural network. *IEEE Trans. Neural Netw.*, **13**, 1257–1267.

[22] Halici, U. (1997) Reinforcement learning in random neural networks for cascaded decisions. *Biosystems*, **40**, 83–91.

[23] Halici, U. (2000) Reinforcement learning with internal expectation for the random neural network. *Eur. J. Oper. Res.*, **126**, 288–307.

[24] Halici, U. (2001) Reinforcement learning with internal expectation in the random neural networks for cascaded decisions. *Biosystems*, **63**, 21–34.

[25] Gelenbe, E. (1994) G-networks: a unifying model for neural and queueing networks. *Ann. Oper. Res.*, **48**, 433–461.

[26] Gelenbe, E. (1991) Product-form queueing networks with negative and positive customers. *J. Appl. Probab.*, **28**, 656–663.

[27] Gelenbe, E. (2000) The first decade of G-networks. *Eur. J. Oper. Res.*, **126**, 231–232.

[28] Gelenbe, E. (1993) G-networks with triggered customer movement. *J. App. Probab.*, **30**, 742–748.

[29] Gelenbe, E. and Fourneau, J.-M. (2002) G-networks with resets. *Perform. Eval.*, **49**, 179–191.

[30] Fourneau, J.M., Gelenbe, E. and Suros, R. (1996) G-networks with multiple classes of positive and negative customers. *Theoret. Comput. Sci.*, **155**, 141–156.

[31] Gelenbe, E. and Labed, A. (1998) G-networks with multiple classes of signals and positive customers. *Eur. J. Oper. Res.*, **108**, 293–305.

[32] Fourneau, J.-M. and Gelenbe, E. (2004) Flow equivalence and stochastic equivalence in G-networks. *Comput. Manage. Sci.*, **1**, 179 – 192.

[33] Artalejo, J.R. (2000) G-networks: a versatile approach for work removal in queueing networks. *Eur. J. Ope. Res.*, **126**, 233–249.

[34] Gelenbe, E. (1997) A class of genetic algorithms with analytical solution. *Robot. Auton. Syst.*, **22**, 59–64.

[35] Gelenbe, E. and Cramer, C. (1998) Oscillatory corticothalamic response to somatosensory input. *Biosystems*, **48**, 67–75.

[36] Hopfield, J. and Tank, D. (1985) "Neural" computation of decisions in optimization problems. *Biol. Cybern.*, **52**, 141–152.

[37] Smith, K. (1999) Neural networks for combinatorial optimization: a review of more than a decade of research. *INFORMS J. Comput*, **11**, 15–34.

[38] Smith, K., Palaniswami, M. and Krishnamoorthy, M. (1998) Neural techniques for combinatorial optimization with applications. *IEEE Trans. Neural Netw.*, **9**, 1301–1318.

[39] Aguilar, J. and Gelenbe, E. (1997) Task assignment and transaction clustering heuristics for distributed systems. *Inform. Sci. – Inform. Comput. Sci*, **97**, 199–221.

[40] Gelenbe, E. (1992) A probabilistic generalization of the SAT problem. *C. R. Acad. Sci. I*, **315**, 339–342.

[41] Gelenbe, E., Ghanwani, A. and Srinivasan, V. (1997) Improved neural heuristics for multicast routing. *IEEE J. Sel. Areas Commun.*, **15**, 147–155.

[42] Aiello, G., Gaglio, S., Re, G.L. and Urso, A. (2004) A random neural network for the dynamic multicast problem. *WSEAS Trans. Comput.*, **3**, 1545–1550.

[43] Cancela, H., Robledo, F. and Rubino, G. (2004) A GRASP algorithm with RNN based local search for designing a WAN access network. *Electron. Notes Discrete Math.*, **18**, 59–65.

[44] Ghanwani, A. (1994) A qualitative comparison of neural network models applied to the vertex covering problem. *Turk. J. Electr. Eng. Comput. Sci.*, **2**, 11–18.

[45] Pekergin, F. (1992) Combinatorial optimization by random neural network model – Application to the independent set problem. *Int. Conf. Artificial Neural Networks* (*ICANN'92*), Brighton, England, September 4–7, pp. 437–440. North-Holland, Amsterdam.

[46] Gelenbe, E., Koubi, V. and Pekergin, F. (1994) Dynamical random neural network approach to the traveling salesman problem. *Turk. J. Electr. Eng. Comput. Sci.*, **2**, 1–10.

[47] Zhong, Y., Sun, D. and Wu, J. (2005) Dynamical Random Neural Network Approach to a Problem of Optimal Resource Allocation. *Proc. 8th Int. Work-Conf. Artificial Neural Networks*, Barcelona, Spain, June 8–10, pp. 1157–1163. Springer, Berlin.

[48] Koubi, V. (1994) Reseaux de neurones et optimization combinatoire. PhD Thesis, Universit de Paris 05, Paris, France.

[49] Aguilar, J. (1998) Definition of an energy function for the random neural to solve optimization problems. *Neural Netw.*, **11**, 731–737.

[50] Cerkez, C., Aybay, I. and Halici, U. (1997) A Digital Neuron Realization for the Random Neural Network Model. *Proc. Int. Conf. Neural Networks*, Houston, TX, USA, June 09–12, pp. 1000–1004. IEEE, Piscataway, NJ.

[51] Abdelbaki, H., Gelenbe, E. and El-Khamy, S. (2000) Analog Hardware Implementation of the Random Neural Network Model. *Proc. IEEE/INNS/ENNS Int. Joint Conf. Neural Networks* (*IJCNN 2000*), Como, Italy, July 24–27, pp. 197–201. IEEE Computer Society, Los Alamitos, CA.

[52] Kocak, T., Seeber, J. and Terzioglu, H. (2003) Design and implementation of a random neural network routing engine. *IEEE Trans. Neural Netw.*, **14**, 1128–1143.

[53] Hey, L.A. (2008) Reduced complexity algorithms for cognitive packet network routers. *Comput. Commun.*, **31**, 3822–3830.

[54] Akgul, B.E.S., Chakrapani, L.N., Korkmaz, P. and Palem, K.V. (2006) Probabilistic CMOS Technology: A Survey and Future Directions. *Proc. 14th IFIP Int. Conf. Very Large Scale Integration of System-on-Chip*, Nice, France, October 16–18, pp. 1–6. IEEE, Piscataway, NJ.

[55] Chakrapani, L.N., Korkmaz, P., Akgul, B.E.S. and Palem, K.V. (2007) Probabilistic system-on-a-chip architectures. *ACM Trans. Des. Autom. Electron. Sys.*, **12**, 1–28.

[56] Gelenbe, E., Mao, Z.-H. and Li, Y.-D. (1999) Function approximation with spiked random networks. *IEEE Trans. Neural Netw.*, **10**, 3–9.

[57] Gelenbe, E., Mao, Z.-H. and Li, Y.-D. (2004) Function approximation by random neural networks with a bounded number of layers. *J. Differ. Equ. Dyn. Syst.*, **12**, 143–170.

[58] Atalay, V., Gelenbe, E. and Yalabik, N. (1992) The random neural network model for texture generation. *Int. J. Pattern Recognit. Artif. Intell.*, **6**, 131–141.

[59] Gelenbe, E., Hussain, K. and Abdelbaki, H. (2000) Random Neural Network Texture Model. *Proc. 5th Conf. Applications of Artificial Neural Networks in Image Processing*, San Jose, CA, USA, January 27–28, pp. 104–111. SPIE, Bellingham, WA.

[60] Atalay, V. and Gelenbe, E. (1992) Parallel algorithm for colour texture generation using the random neural network model. *Int. J. Pattern Recognit. Artif. Intell.*, **6**, 437–446.

[61] Teke, A. and Atalay, V. (2006) Texture classification and retrieval using random neural network model. *Comput. Manage. Sci.*, **3**, 193–205.

[62] Gelenbe, E., Feng, T. and Krishnan, K.R.R. (1996) Neural network methods for volumetric magnetic resonance imaging of the human brain. *Proc. IEEE*, **84**, 1488–1496.

[63] Lu, R. and Shen, Y. (2005) Image Segmentation Based on Random Neural Network Model and Gabor Filters. *Proc. 27th Annual Int. Conf. Engineering in Medicine and Biology Society*, Shanghai, China, September 1–4, pp. 6464–6467. IEEE, Piscataway, NJ.

[64] Bakircioglu, H., Gelenbe, E. and Koçak, T. (1997) Image Enhancement and Fusion with the Random Neural Network. *Turk. J. Electr. Eng. Comput. Sci.*, **5**, 65–77.

[65] Bakircioglu, H. and Kocak, T. (2000) Survey of random neural network applications. *Eur. J. Oper. Res.*, **126**, 319–330.

[66] Cramer, C.,Gelenbe, E. and Gelenbe, P. (1998) Image and video compression. *IEEE Potentials*, **17**, 29–33.

[67] Gelenbe, E., Cramer, C., Sungur, M. and Gelenbe, P. (1996) Traffic and video quality in adaptive neural compression. *Multimedia Syst.*, **4**, 357–369.

[68] Cramer, C., Gelenbe, E. and Bakircioglu, H. (1996) Low bit rate video compression with neural networks and temporal subsampling. *Proc. IEEE*, **84**, 1529–1543.

[69] Cramer, C. and Gelenbe, E. (2000) Video quality and traffic QoS in learning-based subsampled and receiver-interpolated video sequences. *IEEE J. Sel. Areas Commun.*, **18**, 150–167.

[70] Gelenbe, E., Lent, R. and Xu, Z. (2001) Measurement and performance of a cognitive packet network. *Comput. Netw.*, **37**, 691–791.

[71] Gelenbe, E., Lent, R. and Xu, Z. (2001) Design and performance of cognitive packet networks. *Perform. Eval.*, **46**, 155–176.

[72] Gelenbe, E., Lent, R. and Nunez, A. (2004) Self-aware networks and QoS. *Proc. IEEE*, **92**, 1478–1489.

[73] Gelenbe, E., Sakellari, G. and Arienzo, M.D. (2008) Admission of QoS aware users in a smart network. *ACM Trans. Auton. Adapt. Syst.*, **3**, 4:1–4:28.

[74] Gelenbe, E. (2003) Sensible decisions based on QoS. *Comput. Manage. Sci.*, **1**, 1–14.

[75] Gelenbe, E. and Lent, R. (2004) Power-aware ad hoc cognitive packet networks. *Ad Hoc Netw. J.*, **2**, 205–216.

[76] Hey, L. (2008) Power Aware Smart Routing in Wireless Sensor Networks. *Proc. Conf. Next Generation Internet* Networks, Krakow, Poland, April 28–30, pp. 195–202. IEEE, Piscataway, NJ.

[77] Gelenbe, E. (2006) Users and services in intelligent networks. *Proc. IEE* (*ITS*), **153**, 213–220.

[78] Oke, G. and Loukas, G. (2007) A denial of service detector based on maximum likelihood detection and the random neural network. *Comp. J.*, **50**, 717–727.

[79] Mohamed, S. and Rubino, G. (2002) A study of real-time packet video quality using random neural networks. *IEEE Trans. Circuits Syst. Video Technol.*, **12**, 1071–1083.

[80] Mohamed, S., Rubino, G. and Varela, M. (2004) Performance evaluation of real-time speech through a packet network: a random neural networks-based approach. *Perform. Eval.*, **57**, 141–161.

[81] Rubino, G., Varela, M. and Bonnin, J. (2006) Controlling multimedia QoS in the future home network using the PSQA metric. *Comp. J.*, **49**, 137–155.

[82] da Silva, A.P.C., Varela, M., de Souza e Silva, E., Leao, R. M.M. and Rubino, G. (2008) Quality assessment of interactive voice applications. *Comput. Netw.*, **52**, 1179–1192.

[83] Rubino, G., Tirilly, P. and Varela, M. (2006) Evaluating Users Satisfaction in Packet Networks Using Random Neural Networks. *Proc. 16th Int. Conf. Artificial Neural Networks* (*ICANN 2006*), Athens, Greece, September 10–14, pp. 303–312. Springer, Berlin.

[84] Aussem, A. (1994) Call Admission Control in ATM Networks with the Random Neural Network. *Proc. IEEE Int. Conf. Neural Networks*, Orlando, FL, USA, June 28 – July 2, pp. 2482–2487. IEEE, Piscataway, NJ.

[85] Gelenbe, E. and Shachnai, H. (2000) On G-networks and resource allocation in multimedia systems. *Eur. J. Oper. Res.*, **126**, 308–318.

[86] Gelenbe, E., Seref, E. and Xu, Z. (2001) Simulation with learning agents. *Proc. IEEE*, **89**, 148–157.

[87] Gelenbe, E., Hussain, K. and Kaptan, V. (2005) Simulating autonomous agents in augmented reality. *J. Syst. Softw.*, **74**, 255–268.

[88] Stafylopatis, A. and Likas, A. (1992) Pictorial information-retrieval using the random neural network. *IEEE Trans. Softw. Eng.*, **18**, 590–600.

[89] Likas, A. and Stafylopatis, A. (1996) High capacity associative memory based on the random Neural Network model. *Int. J. Pattern Recognit. Artif. Intell.*, **10**, 919–937.

[90] Aguilar, J. (2001) Learning algorithm and retrieval process for the multiple classes random neural network model. *Neural Process. Lett.*, **13**, 81–91.

[91] Bakircioglu, H. and Gelenbe, E. (1998) Random Neural Network Recognition of Shaped Objects in Strong Clutter. *Proc. 3rd Conf. Applications of Artificial Neural Networks in Image Processing*, San Jose, CA, USA, January 26–27, pp. 22–28. SPIE, Bellingham, WA.

[92] Hussain, K. and Moussa, G.S. (2005) Laser Intensity Vehicle Classification System Based on Random Neural Network. *Proc. 43rd Annual Southeast Regional Conf.*, Kennesaw, Georgia, Alabama, USA, March 18–19, pp. 31–35. ACM, New York.

[93] Gelenbe, E. and Kocak, T. (2004) Wafer surface reconstruction from top-down scanning electron microscope images. *Microelctron Eng.*, **75**, 216–233.

[94] Abdelbaki, H., Gelenbe, E. and Kocak, T. (1999) Matched Neural Filters for EMI Based Mine Detection. *Proc. Int. Joint Conf. Neural Networks* (*IJCNN 99*), Washington, DC, July 10–16, pp. 3236–3240. IEEE, Piscataway, NJ.

[95] Abdelbaki, H., Gelenbe, E. and Kocak, T. (2005) Neural algorithms and energy measures for EMI based mine detection. *J. Differ. Equ. Dyn. Sys.*, **13**, 63–86.

[96] Gelenbe, E. and Kocak, T. (2000) Area-based results for mine detection. *IEEE Trans. Geosci. Remote Sens.*, **38**, 1–14.

[97] Gelenbe, E. and Cao, Y. (1998) Autonomous search for mines. *Eur. J. Oper. Res.*, **108**, 319–333.