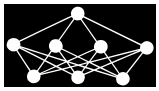


RANDOM NEURAL NETWORK MODEL FOR SUPERVISED LEARNING PROBLEMS

S. Basterrech[*], *G. Rubino*[†]

tutorial

Abstract: Random Neural Networks (RNNs) are a class of Neural Networks (NNs) that can also be seen as a specific type of queuing network. They have been successfully used in several domains during the last 25 years, as queuing networks to analyze the performance of resource sharing in many engineering areas, as learning tools and in combinatorial optimization, where they are seen as neural systems, and also as models of neurological aspects of living beings. In this article we focus on their learning capabilities, and more specifically, we present a practical guide for using the RNN to solve supervised learning problems. We give a general description of these models using almost indistinctly the terminology of Queuing Theory and the neural one. We present the standard learning procedures used by RNNs, adapted from similar well-established improvements in the standard NN field. We describe in particular a set of learning algorithms covering techniques based on the use of first order and, then, of second order derivatives. We also discuss some issues related to these objects and present new perspectives about their use in supervised learning problems. The tutorial describes their most relevant applications, and also provides a large bibliography.

Key words: *neural networks, random neural networks, supervised learning, pattern recognition, G-networks*

Received: January 21, 2015 **DOI:** 10.14311/NNW.2015.25.024
Revised and accepted: October 13, 2015

1. Introduction

Supervised Learning is an area of the Machine Learning field that refers to a set of problems wherein the information is presented according to an outcome measurement associated with a set of input features. The information is presented as a dataset of labeled samples. The aim is *"to learn"* the relationship between input and output features. This learning process is done based on a set of examples in order to generate a learning model with the power of *"generalising"*, this is to make

[*]Sebastián Basterrech – Corresponding author, National Supercomputing Center, VŠB-Technical University of Ostrava, IT 453, Studentská 1, Ostrava-Poruba, Czech Republic, E-mail: Sebastian.Basterrech.Tiscordio@vsb.cz

[†]Gerardo Rubino, INRIA – Rennes, Beaulieu university campus, 35042 Rennes Cedex, France, E-mail: Gerardo.Rubino@inria.fr

"good" predictions for new unseen inputs. The research on *Neural Networks (NNs)* is considered to have started with the work of Warren McCulloch and Walter Pitts in 1943 [84], and it has produced a rich literature with a strong concentration of papers in the 80s and 90s. In the 80s Rumelhart *et al.* explored the relationship between *Parallel Distributed Processing (PDP)* systems and various aspects of human cognition. The authors defined a general framework of a PDP system reactivating the research on connectionist models [98]. The most popular PDP systems are NNs. In the last decades several books and journals have been dedicated to the research on NNs. The interest in the NN area arises from both its theoretic aspects and its computational power for solving real problems. NNs have been successfully applied in many different fields such as engineering, biology, pattern recognition, theoretical physics, applied mathematics, statistics, etc.

There are many types of NNs, and the related literature is huge. This article focuses on a particular class of NNs called *Random Neural Networks*. The RNN model was introduced by E. Gelenbe in 1989 [37, 38]. RNNs are mathematical objects that combine features of both NNs and queueing models. They been successfully employed in many types of applications: in learning problems, in optimization, in image processing, in associative memories, etc. Here, we are specifically interested in the situations where the model is applied for solving supervised learning tasks. A RNN is a PDP composed of a pool of interconnected nodes, which process and transmit information (signals) between them. Each node is a simple processor and it is characterized by its state, a whole number. The nodes receives two kinds of signals (negative and positive) from their neighbors or from outside. When a negative signal arrives to a node, it produces an effect that can be related to neural inhibition, its state its decreased by one. The arrivals of positive signals provoke the opposite effect, the state is increased by one. The fire of signals by the nodes is modeled by Poisson processes, and the pattern of connectivity among the neurons follows stochastic rules.

The design of the model was inspired from the biological behavior of neuron circuits in the neo-cortex. The model considers the following biological aspects: the action potentials in the form of spikes, the exchange of excitatory and inhibitory signals among the neurons, the synapses (weighted connections between two neurons), random delays between spikes, reduction of neuronal potential after firing, arbitrarily topology [37]. The model has been also proven very powerful, from the computational viewpoint. In [58] the authors shown that under certain algebraic hypothesis the RNN is an universal approximator. Besides, it can be easily implemented in both software and hardware. In order to apply the model for solving learning tasks, several learning algorithms have been adapted from the classic NN to RNNs, such as the Gradient Descent [35] and Quasi-Newton methods [16, 74]. The number of applications of the model in the learning area is very large, but the model has been also applied to solve combinatorial optimization problems, such that the Traveling Salesman Problem or the Minimum Vertex Covering Problem [42, 47].

Main contributions

The first overview about RNN was presented in 2000 [9]. A survey about RNN focused on networking application and self-aware networks was introduced in [99]. Another general and helpful survey about RNN was presented in [104], where the authors describe the main applications of RNNs, covering several topics including biology, reinforcement learning, and optimization problems. In [60] the authors focused on RNN for solving learning problems, they identified some drawbacks of the RNN learning applications. In addition, an extensive literature about RNN was presented in [29]. In the 25th anniversary of the RNN model, we present this tutorial that contains the following contributions with respect to the previous published material.

- We introduce the model as a simple computational processor in a PDP framework, instead of using concepts coming from queueing systems. Besides, we present a parallelism between this particular PDP and the model as belonging to the queueing area.

- We provide a structured overview about the numerical optimization algorithms used for training RNNs. We introduce algorithms that use the first derivative information of a quadratic cost function, such that the gradient descent type algorithms. We then present Quasi-Newton methods that use the information of the second derivative of the cost function. In this practical guide, all the algorithms used for training are shown in detail following a homogeneous format.

- We present a critical review and new perspectives on RNN in supervised learning. We discuss technical issues concerning stability problems in the model itself, as well as problems related to the parameters' optimization in the learning process. We discuss some points related to the computational advantages of the model, as well as about its weaknesses and limitations. The overview concludes with remarks concerning some new trends and future research lines.

In addition, this article presents an overview of some selected applications of the RNN in the supervised learning area. In particular, we comment on two applications where the experimental results show a better performance of the model with respect to other techniques of the literature.

Organization of the article

This article is structured as follows. Section 2 formally describes the RNN model as a learning tool and in the framework of queueing theory. Section 3 presents algorithms for training the RNN model. It starts with a formal specification of the computational problems in supervised learning. In Section 3.2 we give a general description of RNN in the learning context. We present the Gradient Descent algorithm in Section 3.3, and we introduce second order optimization methods in 3.4. We describe the following algorithms: the Broyden-Fletcher-Goldfarb-Shanno in Section 3.4.1, the Davidon-Fletcher-Powell in Section 3.4.2, the Levenberg-Marquardt

in Section 3.4.3 and one variation of it in Section 3.4.4. We present a critical review about the RNN model for solving learning problems in Section 4. Section 5 presents an overview of applications. We conclude and present new research trends in Sections 6.

2. The Random Neural Network model

This Section formally introduces the RNN model. It has four parts. First, we describe a single neuron (Random Neuron) as an elementary processor. Second, we present the RNN as a system composed by interconnected neurons. Third, we review the model in the framework of queuing networks. The section ends introducing the different topologies and structural concepts of the RNN.

2.1 Random Neuron (RN)

A *Random Neuron (RN)* is a real parametric function of two real variables, with a real parameter called the neuron's *rate*. The input variables are assumed to be non-negative. The rate is positive. If $x \geq 0$ is the first input variable, $y \geq 0$ is the second one, and if $r > 0$ is the rate of the neuron, then the output is the real z given by the expression

$$z = \frac{x}{r+y} r. \tag{1}$$

See that a RN is characterized by its rate r. We can see the neuron as an input-output system with two "input ports", one for x and the other one for y, and one output port for z. The ports associated with the output and with the first input value are called *positive*; the input port corresponding to the second input variable y is called *negative* (the reason for this is explained later), but all the variables involved are non-negative real numbers. Fig. 1 shows a neuron as an input-output device. When $x \geq r + y$ we say that the neuron is *saturated*.

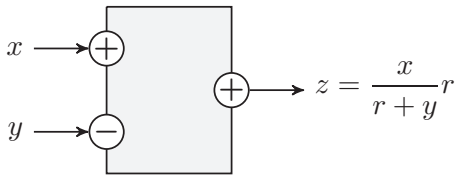


Fig. 1 *A zoom on a random neuron (RN) seen as a "black-box" system; the inputs are the reals $x, y \geq 0$; the parameter is the rate $r > 0$, and the output is the real z; we say that the first input variable x is connected to the positive input port of the RN (depicted '+') and the second input variable y to the negative input port (depicted '−'); the output port is also said to be positive (and it is depicted '+' in the figure)*

The output value z is seen as a measure of the activity of the neuron (as in most input-output systems). As such, see that z is increasing in x and decreasing in y. In real neurons, which also are input-output systems, the input signals belong to

two types, excitatory signals, which are those contributing to the neuron's activity measured by its output (the higher the excitatory signal, the higher the neuron's activity) and inhibiting inputs playing the opposite role. This is why we call positive the signals arriving at the '+' input port, and negative those arriving at the '−' one.

We will say that a RN is *controlled* if its output z is modified according to the rule

$$z = \min\left\{\frac{x}{r+y}, 1\right\} r. \tag{2}$$

So, in this case the RN's output is always less than or equal to its rate, and it is equal to its rate when the neuron is saturated. In the case of the initial definition (1), the neuron is said to be *uncontrolled*.

2.2 Random Neural Network (RNN)

A *Random Neural Network* (RNN) is a network composed of N interconnected RNs, that implements a function from $\mathbb{R}^{2N}$ into $\mathbb{R}^O$, for some $1 \leq O \leq N$, in the following way. We are given N RNs denoted $1, 2, \ldots, N$ (that is, we are given N strictly positive reals $r_1, r_2, \ldots, r_N$), and two $N \times N$ matrices denoted by $\mathbf{P}^+ = (p_{ij}^+)$ and $\mathbf{P}^- = (p_{ij}^-)$, whose components are probabilities. Both matrices *and their sum* are substochastic, that is, for any of their rows, say the ith one, we have

$$\sum_{j=1}^{N} \left(p_{ij}^+ + p_{ij}^-\right) \leq 1.$$

Also, for at least one of the neurons, we have $\sum_{j=1}^{N}(p_{ij}^+ + p_{ij}^-) < 1$. The neurons i for which $\sum_{j=1}^{N}(p_{ij}^+ + p_{ij}^-) < 1$ are called *output* neurons. We denote by O their number (so, $1 \leq O \leq N$). The network outside is often referred to as the neuron's *environment* with which the system operates [98].

Let us denote the $2N$ input variables of the network as $x_1, \ldots, x_N, y_1, \ldots, y_N$. Then, the output of the network is the set of outputs of each of its output neurons. We need only to specify how are determined the inputs to the N RNs (the outputs are given by the previously described rules, in the uncontrolled or controlled cases). Let us call u_i (respectively v_i) the positive (respectively negative) input to neuron i. Then, the following equations must be satisfied:

$$u_i = x_i + \sum_{j=1}^{N} z_j p_{ji}^+, \qquad v_i = y_i + \sum_{j=1}^{N} z_j p_{ji}^-.$$

In words, the fraction p_{ji}^+ of the output z_j of neuron j adds to the positive input u_i to neuron i, and the fraction p_{ji}^- of the output z_j of neuron j adds to the negative input v_i to i. Of course, this means that the reals $z_1, \ldots, z_N$ must satisfy the

non-linear system of equations

$$z_i = \frac{x_i + \sum_{j=1}^{N} z_j p^+_{ji}}{r_i + y_i + \sum_{j=1}^{N} z_j p^-_{ji}} r_i, \qquad i = 1, 2, \ldots, N.$$

This needs some technical discussions about the existence and unicity of solutions to this system, as we will see below.

Observe that if we define

$$d_i = 1 - \sum_{j=1}^{N} \left(p^+_{ij} + p^-_{ij} \right), \tag{3}$$

we have $0 \leq d_i \leq 1$, and that neuron i is an output neuron when $d_i > 0$. We can also say that the network of neurons sends the part $d_i z_i$ of z_i through the output port of i.

Observation: in general in the learning applications, we use a RNN with N neurons as a function from $\mathbb{R}^I$ to $\mathbb{R}^O$ where $I < 2N$ or even $I < N$, by setting $2N - I$ of the standard $2N$ input variables to a fixed value (typically to 0). We will see soon this frequent situation. An important particular case covering all the applications done so far for these objects as learning tools is as follows. The network with N neurons implements a function with $I \leq N$ input variables and $O \leq N$ output variables. The input variables are denoted by $x_1, \ldots, x_I$, which are all connected to the positive port of I neurons called *input* neurons. In other words, no input variable is connected to a negative port. The function output is the set of outputs generated by the O output neurons. A group of neurons can have no interactions with the environment (when $I + O < N$). We call those units *hidden* neurons. Note that a neuron can be both an input and an output one.

2.3 A queueing view of the Random Neural Networks

The RNN method has been used with two different interpretations both referring to exactly the same mathematical model. One is the already described type of interconnected RNs. Another one is a type of queueing systems called G-queues and G-networks. The first interpretation is often employed in the Machine Learning contexts and the second one is applied in Performance Evaluation, for example.

We begin by describing a single queue where customers arrive according to a Poisson process, say with rate $\lambda > 0$, and *service times* are exponentially distributed with parameter $r > 0$. It is assumed that service times are mutually independent and that they are also independent of the inter-arrival times. This server queue is named *M/M/1* queueing model [72]. At any time t the state of the system $S(t)$ is the number of customers present in the queue. The queue storage capacity is infinite. The stochastic process $\{S(t),\ t \geq 0\}$ is a continuous time homogeneous Markov process on the non-negative integers. We define the *utilization factor* of the queue as the ratio $\varrho = \lambda / r$. When the process is ergodic ($\varrho < 1$), the

steady-state is given by

$$p(k) = \lim_{t\to\infty} \mathbb{P}(S(t) = k) = \varrho^k(1-\varrho). \tag{4}$$

A Jackson queueing network consists of N interconnected queues with the following characteristics. For each queue i the service time is exponentially distributed with rate r_i. When a customer completes the service at queue i, it will either move to queue j with routing probability p_{ij} or leave the network with probability d_i ($d_i = 1 - \sum_{j=1}^{N} p_{ij}$). Customers arrive from the environment to queue i according to a Poisson process with rate λ_i^+. At any time t, the system state is the vector $\mathbf{S}(t) = (S_1(t), \ldots, S_N(t))$, where $S_i(t)$ denotes the number of customers in queue i at time t. The assumptions about the independence among the processes can be summarized as follows:

- arrival processes, service processes and switching (routing) processes are independent of each other;
- at each server, the service times are independent of each other;
- at each switching point, the successive switching results are independent of each other.

We define T_i as the mean throughput at queue i. In order to avoid a trivial case, we assume that at least one of the λ_i^+'s is non-zero (strictly positive). In addition, assuming that the system is irreducible (for any two nodes i and j in the Markovian graph there exists a path from i to j), and in equilibrium, T_i for all i can be determined by solving the *flow balance equations*:

$$T_i = \lambda_i^+ + \sum_{j=1}^{N} T_j p_{ji}. \tag{5}$$

The strongly connected property of the Markovian graph implies that exists an unique (and strictly positive) solution. The utilization factor of queue i is given by $\varrho_i = T_i/r_i$.

A G-network (or equivalently, an RNN) is an extension of a Jackson's network where there is a new entity in the system: *negative customers*. As in the previous network, in a G-network there are Poisson arrivals, probabilistic routing among the queues, exponential service rates and usual independence among the corresponding stochastic processes. There are two types of customers in the system, positive ones that operate as we defined for the Jackson network, and the negative ones that operate as follows. When a negative customer arrives at a non-empty queue, it destroys a positive customer in this queue, if any, and disappears. If there are no customers in the queue, a negative customer does not operate, it just disappears from the system. In several works negative customers are referenced as *signals*, thus there are two entities, customers (positive customers) and signals (negative customers).

In [37, 36] Gelenbe shows that, in an equilibrium situation, the ϱ_is satisfy the following flow balance equations:

$$\text{for each node } i, \qquad \varrho_i = \frac{T_i^+}{r_i + T_i^-}, \tag{6}$$

$$\text{for each node } i, \qquad T_i^+ = \lambda_i^+ + \sum_{j=1}^{N} \varrho_j r_j p_{ji}^+, \tag{7}$$

and

$$\text{for each node } i, \qquad T_i^- = \lambda_i^- + \sum_{j=1}^{N} \varrho_j r_j p_{ji}^-, \tag{8}$$

with the supplementary condition that, for all neuron i, we have $\varrho_i < 1$. An important result associated with open Jackson networks and with G-networks is called the *product form theorem*. Gelenbe proved that under Markovian assumptions G-networks have a product form equilibrium distribution. This means that the joint equilibrium distribution of the queue states is the product of the marginal distributions. For more details see [37].

Observation: Let us unify the notation that will be used through this article. So far we introduced the RNN as a function, next we presented the concept using a queueing point of view. In the rest of the article, we follow the most often used notation presented in [37]. Let N be the number of interconnected neurons. For each neuron i its service rate is denoted by r_i, the value at its positive port is denoted by T_i^+ and to the negative port is T_i^-. The positive input value λ_i^+ (the Poisson rate of the customers coming from outside), the negative input value λ_i^- (the Poisson rate of the negative customers coming from outside), and the probability to send information to the environment denoted by d_i characterize the interaction of i with outside. The output of neuron i is its activation rate ϱ_i. The

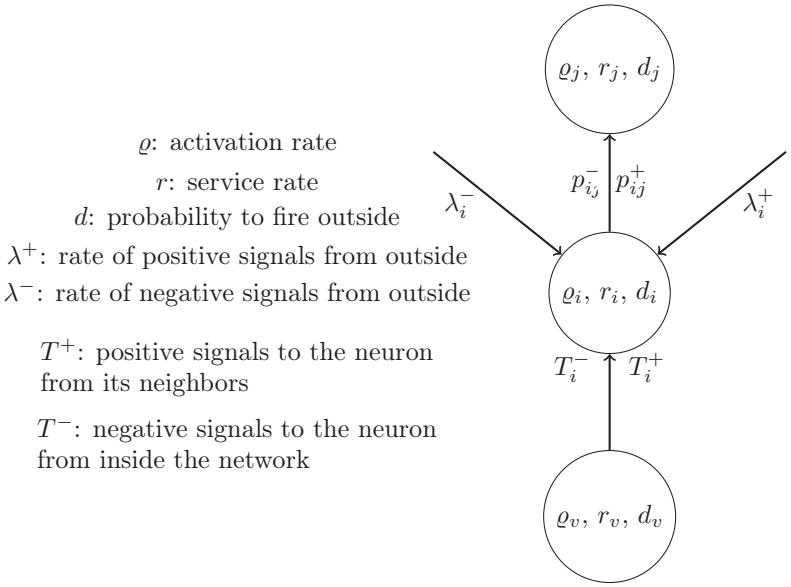


Fig. 2 *A representation of a RN. The figure shows the main parameters involved in a RN embedded in a network.*

connections between two neurons i and j are given by the probabilities $p^+_{i,j}$ and $p^-_{i,j}$. Fig. 2 shows the main parameters involved in a RNN. We will introduce in our notation the concept of weights. For any two neurons i and j, they are defined as: $w^+_{i,j} = r_i p^+_{i,j}$ and $w^-_{i,j} = r_i p^-_{i,j}$. The first one is called *positive weight* and the second one is called *negative weight*. Note that the weights are, by definition, positive reals. In the context of NNs, the traditional notation used for the weight connection (direct edge) between the nodes i to j is often denoted as (j, i). In the RNN context, the reverse order is traditionally used. This originates in the first paper about supervised learning with RNNs [35].

2.4 The network topology

So far, we defined the RNN as a parallel distributed system composed of simple processors (RNs). Therefore, the network is a graph where the RNs are their nodes; the existence of an arc between two nodes is given by certain probability. The two most common topologies of networks are *multi-layer feedforward* and *recurrent* networks.

2.5 Feedforward topology

We start describing the feedforward case. The identifying property is that there are no cyclic connections among the neurons, no circuits in the (directed) graph. The architecture of the graphs consists of multiple layers of neurons in a directed graph. There are three types of layers popularly known as *input*, *hidden* and *output* layers. The neurons can have only connections in a forward direction, from the input neurons to the output neurons, traveling through the hidden ones. Only neurons belonging to the input and to the output layers can exchange information with the environment. The activity rate for each output neuron is computed using a forward propagation procedure. A representation of a feedforward network with one hidden layer is illustrated in Fig. 3.

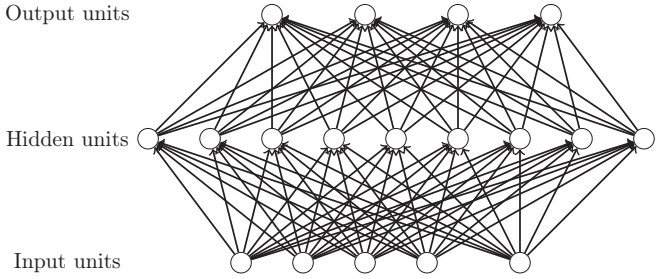


Fig. 3 *A representation of a Feedforward Neural Network. The figure shows a network with a single hidden layer. The flow of information is from the the input neurons through the output ones. In this example there are 5 input neurons full connected to 9 hidden neurons, and the hidden neurons are full connected with 4 output neurons. A network with this topology is used for mapping a relationship from a 5-dimensional space into a 4-dimensional space.*

The feedforward case has been widely used in supervised learning due to the fact that training process is much faster than in the recurrent case. Besides, the feedforward networks are easier to analyze than networks with recurrent topologies. One advantage is that the non-linear system of equations (6), (7) and (8) can be formally solved. Then, we can express the activity rate of the output units as functions of the inputs variables of the system. Let I be the number of input neurons, H is the number of hidden neurons and let O be the number of output neurons. We arbitrary index the input neurons from 1 to I, the hidden neurons from $I+1$ to $I+H$ and the output neurons from $I+H+1$ to $I+H+O=N$. We can compute the activity rate of the neurons using a forward procedure as follows. At the first step, we compute the activity rate of the input neurons, next the activities of the hidden neurons and finally those of the output neurons. Input neurons are the only ones that receive signals from the environment; so we set $\lambda_i^+ = \lambda_i^- = 0$ for all $i \in [I+1, N]$. The activity rates are given by the following explicit expressions:

$$\varrho_i = \frac{\lambda_i^+}{r_i + \lambda_i^-}, \quad \forall i \in [1, I], \qquad \varrho_h = \frac{\displaystyle\sum_{i=1}^{I} \varrho_i w_{i,h}^+}{r_h + \displaystyle\sum_{i=1}^{I} \varrho_i w_{i,h}^-}, \quad \forall h \in [I+H+1, N],$$

and

$$\varrho_o = \frac{\displaystyle\sum_{i=I+1}^{I+H} \varrho_h w_{h,o}^+}{r_o + \displaystyle\sum_{i=I+1}^{I+H} \varrho_h w_{h,o}^-}, \quad \forall o \in [I+H+1, N].$$

More general feedforward networks consist of successive layers where the signals can circulate only in one direction.

2.6 Recurrent topology

In the case of recurrent networks circuits are allowed. The existence of directed cycles has an important impact in the model: we can not compute the rate activities of the output neurons as functions of the network inputs (except, of course, when $N \leq 4$). A RNN with circuits connects to the concept of dynamical systems, rather than to functions, there is an idea of time implicit in the model. For simplicity we assume discrete time and we avoid to use temporal notation in $\boldsymbol{\varrho}$. At each time instant, the network is characterized by an internal state $\boldsymbol{\varrho}$ formed by the activity rates $\boldsymbol{\varrho} = (\varrho_1, \ldots, \varrho_N)$. When an input pattern is presented to the network, the network updates its internal state. For computing the network state we must solve the system of equations (6), (7) and (8), where the unknown parameters are ϱ_i, T_i^+ and T_i^-, for all i. For solving this system is necessary to perform a fixed point procedure (a summary about this computation is given in [104]). The output of the network is given by the state of the output neurons. Unlike the feedforward case, a recurrent network can use its internal states to process sequences of inputs. As a consequence, the recurrent case is often used for solving problems where the dataset presents temporal dependencies.

3. Random Neural Networks in supervised learning problems

In this Section we present the algorithms used for *learning*. The Section starts with a formal definition of the supervised learning problem. Next, we present the algorithms of Gradient Descent type for training the RNN. Then, we introduce the algorithms that use the Hessian or an approximation of the Hessian matrix for training the RNN. We close the Section with a general discussion that covers topics such as: limitations of the algorithms in the numerical optimisation, analysis of the algorithmic time complexity, applications of the RNN concepts in the Reservoir Computing area, a discussion about the computational power of the RNN for approximating any regular function, and an analogy of the model with other NNs.

3.1 Specification of a supervised learning problem

We begin by specifying a supervised learning problem. Given a dataset $\mathcal{L} = \{(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}), k = 1, \ldots, K\}$, where $\mathbf{a}^{(k)} \in \mathcal{A}$ and $\mathbf{b}^{(k)} \in \mathcal{B}$, with $\mathcal{A}$ and $\mathcal{B}$ some given finite dimensional spaces (typically, sets of real vectors, or of vectors of elements in some alphabet, or a mix of both types of objects). The learning procedure consists in inferring a mapping $\nu(\mathbf{a}, \mathcal{L})$ in order to predict the $\mathbf{b}$ values, such that some distance $d(\nu(\mathbf{a}^{(k)}, \mathcal{L}), \mathbf{b}^{(k)})$ is minimized for all $k \in \{1, 2, \ldots, K\}$. We denote by I the dimension of the input vector $\mathbf{a}$ and O the dimension of the output vector $\mathbf{b}$. For each instance $\mathbf{a}^{(k)}$, let us denote $\boldsymbol{\varrho}^{(k)}$ the output produced by the network, that is $\boldsymbol{\varrho}^{(k)} = \nu(\mathbf{a}^{(k)}, \mathcal{L})$. The distance above referred is a function $L(\cdot)$ named *loss function* or *cost function* that measures the deviations of the model predictions $\boldsymbol{\varrho}$s and the targets $\mathbf{b}$s. Several types of loss functions have been used, the main examples are the criteria of *Sum-of-Squared Errors* (L_{RSS}) and the *Kullback-Leibler distance* (L_{KL}), also called *cross-entropy* [66, 100]. The RSS is defined as

$$L_{\mathrm{RSS}} = \sum_{i=1}^{N} \sum_{k=1}^{K} c_i \big(b_i^{(k)} - \varrho_i^{(k)}\big)^2, \tag{9}$$

where $c_i = 1$ when i is an output neuron, otherwise $c_i = 0$. There are several slight modifications of the previous distances, one of those is the *Mean Square Error (MSE)* given by:

$$\mathrm{MSE} = \frac{1}{K} L_{\mathrm{RSS}}. \tag{10}$$

In supervised learning when the targets are categorical or discrete variables the problem is called *classification problem*; when the target is a real vector, the problem is called *regression problem*.

3.2 Random Neural Network as a learning tool

A first approach for applying the RNN model in supervised learning tasks was introduced at the beginning of the 90s by Erol Gelenbe [35]. This procedure is based on the classical backpropagation algorithm [97]. As in practice, the input and output variables in learning problems are bounded with known bounds, the

algorithm described in [35] assumes that $\mathbf{a}^{(k)} \in [0..1]^I$ and $\mathbf{b}^{(k)} \in [0..1]^O$, for all sample k. The RNN model as a predictor is a parametric mapping $\nu(\mathbf{a}, \mathbf{w}^+, \mathbf{w}^-, \mathcal{L})$, where the parameters $\mathbf{w}^+$ and $\mathbf{w}^-$ are adjusted minimizing the loss function. In [35] was considered the quadratic error presented in the expression (10). The network architecture is defined with I input nodes and O output nodes. There are not additional constraints regarding the network topology, that means the network can be feedforward with one or several layers, or it can be recurrent network. We set the port of the input neurons each time that an input pattern $\mathbf{a}^{(k)}$ is offered to the network. The inputs to the positive ports are set with the input pattern: $\lambda_i^+ = a_i^{(k)}$; the negative ports of input neurons are conventionally set to zero ($\lambda_i^- = 0$). The output of the model is a vector of the activity rates produced by the output neurons. The adjustable parameters of the mapping are the weights connections among the neurons. We follow this Section describing the optimization algorithms that have been introduced over the last decades.

3.3 The gradient descent optimization algorithm

We can now describe the gradient-based algorithm that was used so far for training the RNN model [35]. We define two set of neurons $\mathcal{I}$ and $\mathcal{O}$ that correspond to the set of input neurons and the output neurons, respectively. The weights are initialized at some arbitrary values $w_{u,v}^{+\,(0)}$ and $w_{u,v}^{-\,(0)}$, for all u and v. At the τth-iteration, we select a data pattern $\big(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}\big)$, $k = 1, \ldots, K$, where $k = \tau - 1 \bmod K + 1$. The weight correction is computed following the *delta learning rule* [97], meaning that the weight correction is proportional to the partial derivative of the loss function with respect to each weight. From (3), the service rate of neuron i verifies

$$r_i = \frac{1}{1 - d_i} \sum_{j=1}^{N} \Big(w_{i,j}^+ + w_{i,j}^- \Big), \tag{11}$$

for all $i \in \mathcal{I} \cup \mathcal{H}$. Also note that r_i is a free-parameter when i is an output neuron.

At each step τ, the current weight value descends in the direction of the negative gradient of $L(\cdot)$; the update rule for positive and negative weights (denoted with superscript $*$) of any connection (u, v) is:

$$w_{u,v}^{*\,(\tau)} = w_{u,v}^{*\,(\tau-1)} + \delta_{u,v}^{*(\tau)}, \tag{12}$$

where

$$\begin{aligned} \delta_{u,v}^{*(\tau)} &= -\eta \sum_{i=1}^{N} c_i \big(\varrho_i^{(k)} - b_i^{(k)}\big) \left. \frac{\partial \varrho_i^{(k)}}{\partial w_{u,v}^*} \right|_{\mathbf{w}=\mathbf{w}^{(\tau-1)}} \\ &= -\eta \left(\frac{\partial}{\partial w_{u,v}^*} \frac{1}{2} \sum_{i=1}^{N} c_i \big(\varrho_i^{(k)} - b_i^{(k)}\big)^2 \right)_{\mathbf{w}=\mathbf{w}^{(\tau-1)}} \\ &= -\eta \sum_{i=1}^{N} c_i \big(\varrho_i^{(k)} - b_i^{(k)}\big) \left. \frac{\partial \varrho_i^{(k)}}{\partial w_{u,v}^*} \right|_{\mathbf{w}=\mathbf{w}^{(\tau-1)}}. \end{aligned} \tag{13}$$

The parameter $\eta \in [0, 1]$ is called *learning factor*. It is used for tuning the convergence speed of the algorithm. Here, we set $c_i = 1$ for all output neuron i,

otherwise $c_i = 0$. Equation (13) leads to the following simplified expressions. For each connection (u, v), define the vectors $\boldsymbol{\gamma}^+_{u,v}$ and $\boldsymbol{\gamma}^-_{u,v}$ by

$$\gamma^+_{uv;i} = \begin{cases} -\dfrac{1}{r_i + T^-_i}, & \text{if } u = i, \quad v \neq i, \\ \dfrac{1}{r_i + T^-_i}, & \text{if } u \neq i, \quad v = i, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\gamma^-_{uv;i} = \begin{cases} -\dfrac{1 + \varrho_i}{r_i + T^-_i}, & \text{if } u = i, \quad v = i, \\ -\dfrac{1}{r_i + T^-_i}, & \text{if } u = i, \quad v \neq i, \\ -\dfrac{\varrho_i}{r_i + T^-_i}, & \text{if } u \neq i, \quad v = i, \\ 0, & \text{otherwise.} \end{cases}$$

Then, denoting by $\boldsymbol{\varrho}$ the vector of activity rates $\boldsymbol{\varrho} = (\varrho_1, \ldots, \varrho_N)$:

$$\frac{\partial \boldsymbol{\varrho}}{\partial w^+_{u,v}} = \boldsymbol{\gamma}^+_{u,v} \varrho_u \left[\mathbf{I} - \boldsymbol{\Omega}\right]^{-1} \qquad \text{and} \qquad \frac{\partial \boldsymbol{\varrho}}{\partial \mathrm{w}^-_{\mathrm{u,v}}} = \boldsymbol{\gamma}^-_{\mathrm{u,v}} \varrho_{\mathrm{u}} \left[\mathbf{I} - \Omega\right]^{-1}, \tag{14}$$

where $\mathbf{I}$ and $\boldsymbol{\Omega}$ are N-dimensional matrices, $\mathbf{I}$ is the identity, and the element (i, j) of $\boldsymbol{\Omega}$ is given by

$$\Omega_{i,j} = \frac{w^+_{i,j} - w^-_{i,j} \varrho_j}{r_j + T^-_j}. \tag{15}$$

The partial derivatives were explicitly computed for a feedforward RNN with a single layer in [60].

An *online version* of the *Gradient Descent (GD)* algorithm is an iterative method that processes the input patterns one-by-one realizing the following two main operations: to compute the direction of the gradient of the loss function and to update the weights using the expression (12). The method can either be stopped using an arbitrary number of iterations or when the performance measure is smaller than some threshold value. The online version of the GD algorithm is specified in Algorithm 1. In contrast, an *offline* training scheme (also called *batch* algorithm) uses the whole pattern data before modifying the model parameters. An input is offered to the network, the direction of the gradient is computed. When all data have been presented, the gradient directions are averaged. Finally, each weight is updated using the average of the gradient directions. In the Machine Learning literature coexists two opposite views concerning these two training schemes. As far as we know there has been no consensus on which scheme (on-line or offline) is more efficient for training a learning model [88, 106].

3.3.1 Slight modification of the gradient descent algorithm

A slight variation of the GD algorithm for RNN was proposed in [12]. The authors increase the amount of adjustable parameters during the training of the gradient descent algorithm without modifying the network topology and the time complexity

Algorithm 1: Specification of the GD learning algorithm for the RNN model (online version).

Inputs : $\{(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}) : k = 1, \ldots, K\}$ (training dataset), η (learning rate), maxIters (max. number of iterations), the topology of the RNN (that is, the routing probabilities)
Outputs: $\mathbf{w} = \{w^{+}_{i,j}, w^{-}_{i,j} : i, j = 1, \ldots, N\}$ (network's weights)

```
τ = 0;
Initialize all weights (for instance, randomly); // we get w*(0)_{u,v} for all u, v with u either input or hidden neuron
Choose the value of r_i for all i ∈ 𝒪;
while ((τ < maxIters) or (until convergence)) do
    τ = τ + 1; // iteration step τ
    k = τ − 1 mod K + 1;
    λ⁺ = a^(k); // read input
    For all i ∉ 𝒪 compute r_i using (11); // weights are those at τ − 1
    For all i, compute ϱ_i using (6), (7) and (8);
    Compute [I − Ω]^{−1} (see (15));
    For all connections (u, v), update w⁺_{u,v} and w⁻_{u,v} using (12); // see also 3.3.2 for many relevant technicalities
    Evaluate convergence.
```

of the algorithm. They consider as adjustable parameters in the training objective the following ones: the connection weights $\{w^{+}_{i,j}, w^{-}_{i,j} : i, j \in [1, N]\}$, the positive and negative input signals from the environment λ^{+}_{i}, λ^{-}_{i} for all hidden and output neuron i, and the service rate r_i for all output neuron i.

Considering the training error given by the expression (10), the update learning rule is given as follows. Let $\mathbf{\Delta}$, $\mathbf{P}$, $\mathbf{\Lambda}^{+}$ and $\mathbf{\Lambda}^{-}$ be matrices of dimensions $N \times N$, where the matrix $\mathbf{\Delta}$ has elements

$$\Delta_{i,j} = 0 \text{ if } i \neq j \qquad \text{and} \qquad \Delta_{\mathrm{i,i}} = \mathrm{r_i} + \mathrm{T^{-}_i},$$

the matrix $\mathbf{P}$ is defined as

$$\mathrm{P}_{i,j} = 0 \text{ if } i \neq j, \qquad \text{and} \qquad \mathrm{P_{i,i}} = \varrho_{\mathrm{i}},$$

and the matrices $\mathbf{\Lambda}^{+}$ and $\mathbf{\Lambda}^{-}$ have at the position (i, u) the value $\dfrac{\partial \varrho_i}{\partial \lambda^{+}_{u}}$ and $\dfrac{\partial \varrho_i}{\partial \lambda^{-}_{u}}$, respectively. By computing the elements of $\mathbf{\Lambda}^{+}$ and $\mathbf{\Lambda}^{-}$ we have:

$$\mathbf{\Lambda}^{+} = \mathbf{\Delta}^{-1}((\mathbf{I} - \mathbf{\Omega})^{-1})^{\mathrm{T}} \qquad \text{and} \qquad \mathbf{\Lambda}^{-} = \mathbf{\Delta}^{-1}\mathbf{P}((\mathbf{I} - \mathbf{\Omega})^{-1})^{\mathrm{T}},$$

where $\mathbf{\Omega}$ was defined in the expression (15). Then, for each input pattern $(\mathbf{a}^{(k)}, \mathbf{b}^{(k)})$ at the τth iteration, we have the following update rule:

$$w^{*(\tau)}_{u,v} = w^{*(\tau-1)}_{u,v} - \eta \varrho^{(\tau)}_{u} \boldsymbol{\gamma}^{*(\tau)}_{u,v} \left[\mathbf{I} - \mathbf{\Omega}\right]^{-1} (\boldsymbol{\varrho}^{(\tau)} - \mathbf{b}^{(\tau)}), \quad \forall u, v, \tag{16}$$

$$\lambda_u^{*(\tau)} = \lambda_u^{*(\tau-1)} - \eta_1 (\boldsymbol{\varrho}^{(\tau)} - \mathbf{b}^{(\tau)})^{\mathrm{T}} \Lambda_u^*, \quad \forall u \in \mathcal{H} \cup \mathcal{O}, \tag{17}$$

$$r_u^{(\tau)} = r_u^{(\tau-1)} - \eta_2 \big(\varrho_u^{(\tau)} - b_u^{(\tau)}\big) \frac{-T_u^+}{(r_u + T_u^-)^2}, \quad \forall u \in \mathcal{O}, \tag{18}$$

where $[\mathbf{I} - \mathbf{\Omega}]^{-1}$, $\mathbf{\Lambda}^*$ and T^* are computed using the current input $(\mathbf{a}, \mathbf{b})$ and Λ_u^* denotes the column u of the matrix $\mathbf{\Lambda}^*$.

3.3.2 Technical issues

We discuss here some technical issues related to the learning process, well illustrated by the GD procedure. Recall that the model can be seen as a network of queues (it is actually born in this way). This has some consequences, that have an impact on the design algorithmic decisions. A first point concerns the use of (12) for updating the weights. Indeed, it may happen that (12) leads to a new value for some weight that is negative or null. This does not fit the analogy with a network of queues, or even a network of spiking neurons where the weights model mean throughputs of spikes: weights should be positive numbers. We can accept a null value for some $w_{u,v}^*$ interpreted as the fact that there is actually no such connection between u and v, but a negative one has no interpretation. The usage is to respect this analogy, modifying the updating rule such that the weights are never negative. Three possible approaches are proposed in [35]:

- To use the following updating rule

$$w_{u,v}^{*\,(\tau)} = \max\left\{ w_{u,v}^{*\,(\tau-1)} + \delta_{u,v}^{*(\tau)}, 0 \right\}, \tag{19}$$

  and in the case that some weight is assigned value zero, then to apply one of the following rules:

  - fix a null value to this weight, and do not change it anymore in future iterations;
  - assign a zero value to this weight, but allow positive updates in subsequent iterations, keeping using (19).

- Another option is to decrease the value of η and update again the weight using (12). If the new weight is still negative, repeat until obtaining a positive number or stop the loop using some control parameter. Formally, this means that the learning factor becomes a variable parameter in the method. In a nutshell, the global idea in descent methods is to decrease little by little the learning factor, as we get closer and closer to a local minimum. Global accuracy can also be improved (but also cost) if $\eta^{(\tau)}$, say, is built by a supplementary optimization process (this is called *line searching* in the area) [92]. We do not enter these details here.

- An alternative option was presented in [74]. The authors propose a change of variable: instead of using $w^*_{u,v}$ they use new variables $\beta^+_{u,v}$ and $\beta^-_{u,v}$, such that

$$w^+_{u,v} = \left(\beta^+_{u,v}\right)^2 \qquad \text{and} \qquad w^-_{u,v} = \left(\beta^-_{u,v}\right)^2.$$

  Then, instead of using the expression (14), we proceed as follows

$$\frac{\partial \varrho}{\beta^+_{u,v}} = 2\beta^+_{u,v}\frac{\partial \varrho}{\partial w^+_{u,v}}, \qquad \frac{\partial \varrho}{\beta^-_{u,v}} = 2\beta^-_{u,v}\frac{\partial \varrho}{\partial w^-_{u,v}}. \tag{20}$$

3.3.3 Computational cost of the gradient descent algorithm

When one data pattern is presented to update each weight in the network the main computational effort consists of computing $[\mathbf{I} - \mathbf{\Omega}]^{-1}$ using (14) [35]. This effort has $O(N^3)$ time complexity. A remark made in [35] consists in that when a m-step relaxation method is applied the time complexity decreases to $O(mN)$.

Additionally, the general scheme of the algorithm can be adapted when we use a feedforward RNN. In this case the matrix $\mathbf{I} - \mathbf{\Omega}$ becomes triangular, so the computational cost of computing its inverse decreases to $O(N^2)$. Also, the computational effort to compute each activity rate in feedforward networks is reduced, due to the the activity rate of any neuron depends only on the neurons in the preceding layers.

3.4 Second order optimization methods

In this Section, we present the optimisation methods for RNN that use the information given by the second derivative of the loss function. We start introducing the *Gauss-Newton (GN)* methods, next we explore the *Quasi-Newton (QN)* techniques. We present four particular algorithms developed for training RNNs: the *Broyden-Fletcher-Goldfarb-Shanno (BFGS)*, the *Davidon, Fletcher and Powell (DFP)*, the *Levenberg-Marquardt (LM)* and the *LM with Adaptative Momentum (LM-AM)*.

The *Gauss-Newton (GN)* algorithm is a technique for solving non-linear least squares problems that incorporates the second derivatives of the loss function or an approximation of those. Unlike the algorithms of first derivatives that can solve a large non-sparse optimization problems, a GN method can only be used when the loss function is given by a quadratic objective function, for instance the expression (10). The methods of the GN type are generally considered more powerful in terms of accuracy and time than the algorithms that only use the first derivative information.

The GN method is based on an expansion of the loss function in the Taylor series. Let M be the number of adjustable parameters (the number of weights $w^+_{i,j}$ and $w^-_{i,j}$). We define the M-dimensional vector $\mathbf{w}$ that collects in some arbitrary order the weights $w^+_{i,j}$ and $w^-_{i,j}$. Let $\mathbf{a}$ be an input vector on the network. The GN algorithm employs a linear approximation with the first three terms of the Taylor series

$$L(\mathbf{a}, \mathbf{w} + \boldsymbol{\delta}) \approx L(\mathbf{a}, \mathbf{w}) + \sum_{m=1}^{M} \frac{\partial L(\mathbf{a}, \mathbf{w})}{\partial w_m}\delta_m + \sum_{i,j}^{M} \frac{\partial L(\mathbf{a}, \mathbf{w})}{\partial w_i \partial w_j}\delta_i\delta_j, \tag{21}$$

where $\boldsymbol{\delta}$ is a M-dimensional vector that represents a small correction of the weights. The solution is found by solving the $M \times M$ set of equations (called *normal equations*)

$$\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol{\delta} = -\mathbf{G}, \tag{22}$$

where $\mathbf{G}$ and $\mathbf{J}$ are the gradient vector and the Jacobian matrix, respectively. For computing $\mathbf{G}$ and $\mathbf{J}$ we proceed as follows. Let $\mathbf{e}^{(k)}$ be the *residual* row vector of dimension $\mathcal{O}$ for the kth input-output training pair,

$$\mathbf{e}^{(k)} = \mathbf{b}^{(k)} - \boldsymbol{\varrho}^{(k)}. \tag{23}$$

Collecting those residuals, we have a vector $\mathbf{E}$ of $S \times 1$ dimensions, with $S = K\mathcal{O}$. Then, the gradient vector of $L(\cdot)$ has $M \times 1$ dimensions and its mth element is

$$G_m = \sum_{s=1}^{S} \frac{\partial e_s}{\partial w_m} e_s. \tag{24}$$

The Jacobian matrix has dimensions $S \times M$ and its (s, m) element is

$$J_{s,m} = \partial E_s / \partial w_m. \tag{25}$$

For computing the partial derivatives of (24) and (25) we use the expressions presented in (14).

The GN method is a batch type algorithm. We call an *epoch* of the GN algorithm when all the patterns in the training set are used [101]. At each epoch τ, the weight correction δ is computed, next the weights are updated as follows:

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \alpha^{(\tau)}\boldsymbol{\delta}^{(\tau)}, \tag{26}$$

where $\alpha \in (0, 1]$ is computed using a line search technique [91]. In the canonical GN method this parameter is set to 1. A better strategy is tuning α with less values until some suitable point. For details about how to tune α see Chapter 9 of [91].

The GN method for solving the problem of minimization using NNs presents several drawbacks. The method requires a good initial solution, that is often not available [30]. Another drawback is that the GN method requires computing the Hessian matrix $\mathbf{H}$ ($\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{J}$) and its inverse, both computations can be expensive. Therefore, the method is expensive in time and in storage.

A *Quasi-Newton (QN)* method type is a variant of the GN algorithms that uses an approximation of the Hessian matrix ($\widetilde{\mathbf{H}}$) for solving the normal equations. The general approach behind a QN method is an iterative procedure that consists of starting with a positive and symmetric matrix and updating it in successive steps in such a way that the matrix remains positive definite and symmetric. The update rule always moves in a downhill direction for solving the normal equations and guarantees that $\widetilde{\mathbf{H}}$ approximates $\mathbf{H}$. As we already commented so far, the implementation of the second order methods is offline, thus at each epoch the network outputs are computed for the whole of input patterns. We present in Schema 2 a procedure that shows how to compute those model outputs. In the following of this Section we will use this schema as a black box being a part of the GN and Quasi-Newton algorithms. In the remainder of this Section, we present four algorithms based on approximations of the Hessian matrix.

Algorithm 2: Auxiliary schema. Given a RNN the procedure shows how to compute the network outputs for the whole input dataset. The procedure returns a $K \times N$ matrix, that has the vector $\boldsymbol{\varrho}^{(k)}$ computed with the input pattern $\mathbf{a}^{(k)}$ in its k-row.

Inputs : $\{(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}) : k = 1 \ldots, K\}$ (training dataset), the topology of the RNN

Outputs: The neuron activity rate produced by the whole of input patterns: C a $K \times N$ matrix

```
Choose the value of r_i for all output neuron i;
For all i ∉ 𝒪 compute r_i using (11);
for (k ← 1 to K) do
    λ⁺ = a^(k); // read input
    For all i, compute ϱ_i^(k) using (6), (7) and (8);
    // see also 3.3.2 for many relevant technicalities
    Set the row k of C with the vector ϱ^(k);
```

3.4.1 The Broyden-Fletcher-Goldfarb-Shanno algorithm

The *Broyden-Fletcher-Goldfarb-Shanno (BFGS)* method for the RNN model was introduced in [74]. The BFGS is an offline algorithm, which at each epoch τ an approximation of the Hessian matrix $\widetilde{\mathbf{H}}^{(\tau)}$ is computed. The method starts using the identity matrix as the initial Hessian approximation $\widetilde{\mathbf{H}}^{(0)} = \mathbf{I}$. The Choleski factorization is used for decomposing a symmetric and positive definite matrix into two triangular matrices. Choleski factorization is more efficient than alternative methods for solving linear equations, it is about two times faster than the alternative ones. For details about the implementation of this factorization see [91]. The matrix $\widetilde{\mathbf{H}}^{(\tau)}$ is decomposed using Choleski factorization as

$$\widetilde{\mathbf{H}}^{(\tau)} = \mathbf{L}^{(\tau)}\mathbf{L}^{\mathrm{T}(\tau)}. \tag{27}$$

Let c be an auxiliary scalar defined at each epoch as

$$c^{(\tau)^2} = \frac{(\mathbf{w}^{(\tau)} - \mathbf{w}^{(\tau-1)})^{\mathrm{T}}(\mathbf{G}^{(\tau)} - \mathbf{G}^{(\tau-1)})}{(\mathbf{w}^{(\tau)} - \mathbf{w}^{(\tau-1)})^{\mathrm{T}}\widetilde{\mathbf{H}}^{(\tau)}(\mathbf{w}^{(\tau)} - \mathbf{w}^{(\tau-1)})}. \tag{28}$$

We define an auxiliary vector v as

$$\mathbf{v}^{(\tau)} = c^{(\tau)}\mathbf{L}^{(\tau)}(\mathbf{w}^{(\tau)} - \mathbf{w}^{(\tau-1)}). \tag{29}$$

Next, we compute

$$\mathbf{A}^{(\tau)} = \mathbf{L}^{(\tau)} + \frac{((\mathbf{G}^{(\tau)} - \mathbf{G}^{(\tau-1)}) - \mathbf{L}^{(\tau)}\mathbf{v}^{(\tau)})\mathbf{v}^{\mathrm{T}(\tau)}}{\mathbf{v}^{\mathrm{T}(\tau)}\mathbf{v}^{(\tau)}}. \tag{30}$$

The update of the Hessian matrix approximation is given by

$$\widetilde{\mathbf{H}}^{(\tau+1)} = \mathbf{A}^{(\tau)}\mathbf{A}^{\mathrm{T}(\tau)}. \tag{31}$$

Finally, the weight update is given by $\boldsymbol{\delta}$ solving

$$\widetilde{\mathbf{H}}^{(\tau+1)}\boldsymbol{\delta}^{(\tau+1)} = -\mathbf{G}^{(\tau)}. \tag{32}$$

In summary, the BFGS method for RNN presented in [74] is defined in Algorithm 3.

Algorithm 3: Specification of the BFGS algorithm for the RNN model.

Inputs : $\{(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}) : k = 1 \ldots, K\}$ (training dataset), maxIters (max. number of iterations), the topology of the RNN

Outputs: The weights: $\mathbf{w} = \{w^+_{i,j}, w^-_{i,j} : i, j = 1, \ldots, N\}$ (network's weights)

$\tau = 0$;
Initialize all weights (for instance, randomly); `// we get` ${\mathbf{w}^*_{u,v}}^{(0)}$ `for all` u, v `with` u `either input or hidden neuron`
Choose the value of r_i for all output neuron i;
$\widetilde{\mathbf{H}} = \mathbf{I}$;
Compute $\mathbf{G}$ using (24) and (14);
Compute $\boldsymbol{\delta}$ solving (32);
For all connections (u, v), update $w^+_{u,v}$ and $w^-_{u,v}$ using (26);
while $\big((\tau <$ maxIters$)$ *or (until convergence)*$\big)$ **do**
    $\tau = \tau + 1$;
    Apply Schema 2;
    Compute $\mathbf{G}$ using (24) and (14);
    Compute $\mathbf{L}$ using Choleski factorization see (27);
    Compute c, $\mathbf{v}$ and $\mathbf{A}$ using (28), (29) and (30), respectively;
    Update $\widetilde{\mathbf{H}}$ using (31);
    Compute $\boldsymbol{\delta}$ solving (32);
    For all connections (u, v), update $w^+_{u,v}$ and $w^-_{u,v}$ using (26); `// see also 3.3.2 for many relevant technicalities`
    Evaluate convergence;

3.4.2 The Davidon-Fletcher-Powell algorithm

The *Davidon-Fletcher-Powell (DFP)* algorithm is another widely used QN method sometimes referred as Fletcher-Powell [91]. The algorithm is a slight variation of BFGS algorithm, the difference between them is given in the following terms. The scalar c is defined as

$$c^{(\tau)^2} = \frac{(\mathbf{w}^{(\tau)} - \mathbf{w}^{(\tau-1)})^{\mathrm{T}}(\mathbf{G}^{(\tau)} - \mathbf{G}^{(\tau-1)})}{(\mathbf{G}^{(\tau)} - \mathbf{G}^{(\tau-1)})^{\mathrm{T}}\widetilde{\mathbf{H}}^{(\tau)}(\mathbf{G}^{(\tau)} - \mathbf{G}^{(\tau-1)})}, \tag{33}$$

and the vector $\mathbf{v}$ is such that solves the linear system,

$$\mathbf{L}^{(\tau)}\mathbf{v}^{(\tau)} = c^{(\tau)}(\mathbf{G}^{(\tau)} - \mathbf{G}^{(\tau-1)}). \tag{34}$$

The matrix $\mathbf{A}$ is determined by computing

$$\mathbf{A}^{(\tau)} = \mathbf{L}^{(\tau)} - \frac{(\mathbf{G}^{(\tau)} - \mathbf{G}^{(\tau-1)})((\mathbf{w}^{(\tau)} - \mathbf{w}^{(\tau-1)})^{\mathrm{T}}\mathbf{L}^{(\tau)} - \mathbf{v}^{\mathrm{T}(\tau)})}{(\mathbf{w}^{(\tau)} - \mathbf{w}^{(\tau-1)})^{\mathrm{T}}(\mathbf{G}^{(\tau)} - \mathbf{G}^{(\tau-1)})}. \tag{35}$$

Finally, yielding the Hessian approximation

$$\widetilde{\mathbf{H}}^{(\tau)} = \mathbf{A}^{(\tau)}\mathbf{A}^{\mathrm{T}(\tau)}, \tag{36}$$

and we compute the search direction $\boldsymbol{\delta}$ for update the weights solving the expression (32).

According empirical results the BFGS performs better than the DFP method [91]. Although, for some specific benchmark problems the DFP reached better accuracy than DFGS [74]. The algorithm is summarized in 4.

Algorithm 4: Specification of the DFS algorithm for the RNN model. The DFS and the BFGS algorithms differ only in details. As a consequence, we introduce the DFS referencing the schema already presented in Algorithm 3.

Inputs : $\{(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}) : k = 1 \ldots, K\}$ (training dataset), maxIters (max. number of iterations), the topology of the RNN
Outputs: The weights: $\mathbf{w} = \{w_{i,j}^{+}, w_{i,j}^{-} : i, j = 1, \ldots, N\}$ (network's weights)

```
// Perform the lines 1 until 7 of Algorithm 3.
```
while $\big((\tau <$ maxIters$)$ *or (until convergence)*$\big)$ **do**
- $\tau = \tau + 1$;
- Apply Schema 2;
- Compute $\mathbf{G}$ using (24) and (14);
- Compute c using (33);
- Compute $\mathbf{L}$ solving (34);
- Compute $\mathbf{A}$ using (35);
- `// Perform the lines 14 until 17 of Algorithm 3.`

3.4.3 The Levenberg-Marquardt algorithm

The *Levenberg-Marquardt (LM)* algorithm is one of the most standard optimization methods used in the NN area [91, 4, 63]. The LM is a sort of compromise between an offline version of the GD algorithm and a GN method [80, 91]. The algorithm was introduced for training RNN in [16].

At each epoch τ, the approximation of the Hessian matrix is given by,

$$\widetilde{\mathbf{H}}^{(\tau)} = \mathbf{J}^{\mathrm{T}(\tau)}\mathbf{J}^{(\tau)} + \mu^{(\tau)}\mathbf{I}, \tag{37}$$

where $\mu^{(\tau)} > 0$ is called *dumping* term, $\mathbf{I}$ is the identity matrix of dimension $M \times M$, and $\mathbf{J}$ is the Jacobian matrix that is computed using (25). The dumping term μ is modified at each epoch. In the case that the prediction error decreases, then the dumping term is reduced by some constant value β

$$\mu \leftarrow \mu/\beta. \tag{38}$$

Otherwise, the dumping value is increased by a factor of β,

$$\mu \leftarrow \mu\beta. \tag{39}$$

So far, the factor for modifying the dumping term was set as $\beta = 10$ [91, 16].

The LM algorithm computes the weight correction $\boldsymbol{\delta}$ solving the system (32). Then, the update rule for the weights is given by the expression (26). In [16], this weight update considers only the search direction $\boldsymbol{\delta}$. In other words, the authors set $\alpha = 1$ in the expression (26). The algorithm can evolve through either of extreme possible situations are [63, 91]:

- If the dumping term approaches to zero, the LM basically performs as the Gauss-Newton method.
- Otherwise, when the dumping term is very large, the matrix $\widetilde{\mathbf{H}}$ becomes diagonal dominant, so the update rule is similar to the updating expression of gradient descent method using a learning factor of $1/\mu$.

Concerning the stopping conditions, the method can fail if the Jacobian matrix becomes singular or nearly to singular. Even if this situation is rare in practice, a control of the condition number of $\mathbf{J}$ can be useful [91]. Besides, it is necessary to control that the dumping factor satisfies some boundary conditions. It is not recommended to stop after an epoch wherein the training objective error increases. For more technical discussion about the stopping criteria of the LM see [91]. We present the LM procedure in Algorithm 5.

3.4.4 Levenberg-Marquardt with adaptive momentum training

A variation of the LM method applied to NNs was developed in [4, 5]. This approach was adapted for the case of RNN on learning problems in [16]. The idea consists in inserting a *momentum term* that controls the directions followed in the searching space. The algorithm was introduced under the name of *Levenberg-Marquardt with Adaptative Momentum (LM-AM)* [4, 16]. The approach consists in maintaining the *conjugacy* of successive searching vectors [10]. This means that at an epoch τ the new direction $\boldsymbol{\delta}^{(\tau)}$ depends on the selected direction at the previous epoch $\boldsymbol{\delta}^{(\tau-1)}$. It is desirable that the motion along a direction at the current step positively interferes with the minimization along the previous step. Formally, this property occurs when both vectors $\boldsymbol{\delta}^{(\tau-1)}$ and $\boldsymbol{\delta}^{(\tau)}$ are *mutually conjugate*, the same principle is used in the Conjugate Gradient algorithm [91].

In the LM-AM the update rule for the weights at the epoch τ is given by:

$$\boldsymbol{\delta}^{(\tau)} = -\frac{\lambda_1}{2\lambda_2}\left[\widetilde{\mathbf{H}}^{(\tau)}\right]^{-1}\mathbf{G}^{(\tau)} + \frac{1}{2\lambda_2}\boldsymbol{\delta}^{(\tau-1)}, \tag{40}$$

where

$$\lambda_1 = \frac{-2\lambda_2\Delta Q^{(\tau)} + c_2}{c_1} \tag{41}$$

and

$$\lambda_2 = \frac{1}{2}\sqrt{\frac{c_1c_3 - {c_2}^2}{c_1(\Delta P)^2 - (\Delta Q^{(\tau)})^2}}, \tag{42}$$

Algorithm 5: Specification of the LM algorithm for RNN.

Inputs : $\{(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}) : k = 1 \ldots, K\}$ (training dataset), maxIters (max. number of iterations), the topology of the RNN, μ (dumping term), β (constant to modify μ)

Outputs: The weights: $\mathbf{w} = \{w_{i,j}^{+}, w_{i,j}^{-} : i, j = 1, \ldots, N\}$ (network's weights)

$\tau = 0$;
Initialize all weights (for instance, randomly); `// we get` ${\mathbf{w}_{u,v}^{*}}^{(0)}$ `for all` u, v `with` u `either input or hidden neuron`
Choose the value of r_i for all output neuron i;
while $\big((\tau <$ maxIters$)$ *or (until stopping conditions)*$\big)$ **do**
  $\tau = \tau + 1$;
  Apply Schema 2;
  Compute $L(\mathbf{w})$ using (10);
  Compute $\mathbf{G}$ using (24) and (14);
  Compute $\mathbf{J}$ using (25);
  Compute $\widetilde{\mathbf{H}}$ using (37);
  Compute $\boldsymbol{\delta}$ solving (32);
  Compute temporal weights $\mathbf{w}_{\text{tmp}}^{*} = \mathbf{w}^{*} + \boldsymbol{\delta}$;
  `// weights` $\mathbf{w}^{*}$ `are those at` $\tau - 1$`, see also 3.3.2 for technicalities`
  Apply Schema 2 using the weights $\mathbf{w}_{\text{tmp}}^{*}$;
  Compute $L(\mathbf{w}_{\text{tmp}}^{*})$ using (10);
  if $\big(L(\mathbf{w}_{\text{tmp}}) < L(\mathbf{w})\big)$ **then**
    Update μ using (38);
    Set the weights $\mathbf{w}^{*}$ with $\mathbf{w}_{\text{tmp}}^{*}$;
    Set $L(\mathbf{w}^{*})$ with $L(\mathbf{w}_{\text{tmp}}^{*})$;
  else
    Update μ using (39);
  Evaluate stopping conditions;

with

$$\Delta Q^{(\tau)} = \frac{1}{2\lambda_2}(c_2 - \lambda_1 c_1),$$

and the constants c_1, c_2 and c_3 are three real numbers defined as follows:

$$c_1 = \mathbf{G}^{\mathrm{T}(\tau)} \left[\widetilde{\mathbf{H}}^{(\tau)}\right]^{-1} \mathbf{G}^{(\tau)}, \tag{43}$$

$$c_2 = \mathbf{G}^{\mathrm{T}(\tau)} \boldsymbol{\delta}^{(\tau-1)} \tag{44}$$

and

$$c_3 = \boldsymbol{\delta}^{\mathrm{T}(\tau-1)} \widetilde{\mathbf{H}}^{(\tau)} \boldsymbol{\delta}^{(\tau-1)}. \tag{45}$$

In practice, it is suggested to set ΔQ as

$$\Delta Q^{(\tau)} = -\zeta \Delta P \sqrt{c_1}, \tag{46}$$

where ζ is some constant between 0 and 1 [4, 16]. Then, the parameters for the LM-AM procedure are ζ and ΔP. When the LM-AM is applied for optimizing classic NNs is suggested to experiment with $0.85 \leq \zeta \leq 0.95$ [4]. In [16], the authors applied LM-AM for optimizing the RNN model achieving the best results when $0.1 \leq \Delta P \leq 0.6$. The Algorithm 6 presents the pseudo-code of the LM-AM.

Algorithm 6: Specification of the LM-AM algorithm for RNN. This algorithm is a variation of Algorithm 5, as a consequence we introduce it referencing the schema already presented there.

Inputs : $\{(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}) : k = 1 \ldots, K\}$ (training dataset), maxIters (max. number of iterations), the topology of the RNN, μ (dumping term), β (constant to modify μ), ζ and ΔP (specific parameters of the LM-AM)

Outputs: The weights: $\mathbf{w} = \{w^{+}_{i,j}, w^{-}_{i,j} : i, j = 1, \ldots, N\}$ (network's weights)

```
// Performs the lines 1 until 3 of Algorithm 5
```

1 **while** $\big((\tau <$ maxIters$)$ *or (until stopping conditions)*$\big)$ **do**

```
    // Performs the lines 5 until 10 of Algorithm 5
```

2 Compute c_1, c_2 and c_3 using (43), (44) and (45);

3 Compute ΔQ using (46);

4 Compute λ_1 and λ_2 using (41), (42);

5 Compute $\boldsymbol{\delta}$ using (40);

```
    // Performs the lines 12 until 21 of Algorithm 5
```

4. Critical review

In this section we discuss some stability issues of the model when applied to supervised learning tasks. Next, we analyze the time complexity and memory stockage of Gauss-Newton methods. We discuss the difficulties of training recurrent topologies and we present an alternative for using recurrent networks without the drawback of learning the network parameters. Next, we present some properties of the RNN model and its analogy with a specific type of NN. The section ends with a presentation of RNN variations.

4.1 Stability issues

Originally, the model was introduced as a network of queues. Some of the consequences of that were already discussed in Sections 3.3.2. Actually, the model has been applied respecting the analogy with queueing networks. As a consequence, the weights are controlled in order to keep them in positive intervals. If we see neuron i as a queue, the interpretation of ϱ_i makes basically sense in the stable case only, when the queue is in equilibrium (that is, when the underlying stochastic process is ergodic). The same holds for the whole network. This is a tricky point. Seeing i as a queue, we have stability only when $T_i^{+} < r_i + T_i^{-}$. If we want to keep this true, we need supplementary constraints in the optimization processes, since,

at each step, a new neuron rate r_i is computed, as a function of the previously computed weights, and intuitively, it must be high enough such that the neuron remains stable. The point is also relevant regarding the use of the model in supervised learning, because roughly speaking, stability implies that the non-linear system of equations (6), (7) and (8) has an unique solution. The usage is, however, to ignore this point and only check stability for the output neurons. In this case, if for some neuron i we obtain $\varrho_i > 1$, we replace this number by 1.

These remarks mean that there is an open research line here, where other learning schemes could be designed weakening the connexion with the queuing world.

4.2 Time complexity

The LM algorithm has been proved to be efficient in terms of computational time and accuracy rate. In spite of that, the method presents some drawbacks. One of the weak points is that it may require a large amount of memory, due to the need of storing large matrices [10]. The procedure is offline, at each iteration the algorithm uses the whole data pattern for computing the Jacobian matrix and the inverse of the pseudo-Hessian matrix. These two operations can be expensive when the dimensions of both matrices are large. An exact and efficient method for computing the Hessian for the feedforward case was introduced in [18]. However, this approach is not practical in the case of large networks [83]. Another operation that has a high computational cost is the computation of the inverse of $\widetilde{\mathbf{H}}$. For a $M \times M$ matrix, we have the following well known techniques for inverting a matrix and their corresponding computational costs: Gauss-Jordan elimination method (time complexity $O(M^3)$) [91], Strassen method (time complexity $O(M^{\log_2 7})$) [92], Coppersmith-Winograd method (time complexity $O(M^{2.376})$) [23]. A slight variation of the LM algorithm was proposed in [105], wherein the authors reduce the memory space used for storing the Jacobian and pseudo-Hessian matrix. However, the proposal is much slower in terms of convergence speed.

4.3 Difficulties for optimizing recurrent topologies

In the machine learning community there have been numerous efforts to develop algorithms for training a NN with a recurrent topology. In spite of that, in practice is hard to train recurrent networks. An algorithm based on the gradient information has often stability problems, due to the volatile relationship between the weights and the states of the neurons [81]. These phenomena were studied by several researchers. In the literature, they are identified as *vanishing* and the *exploding* gradient problems [17]. The first one occurs when the gradient norm tends fast towards zero. The exploding gradient phenomenon refers to the opposite situation, that is, when the gradient norm tends to get very large [89]. As far as we know, the vanishing and exploding gradient phenomena have not been yet studied for RNNs. The research effort continues to address these issues. For instance, a new attempt to train NN with recurrences was recently introduced under the name of *Hessian-Free Optimization* [81]. This could also be explored for training recurrent RNNs.

4.4 Reservoir computing and Random Neural Networks

During the last fifteen years, *Echo State Network (ESN)* has received much attention in the NN community, due to its good performance for solving time-series learning problems. The ESN introduces a new approach to design and train NN with recurrences. In these models, learning only occurs in the weights that are not involved in recurrences. Those involved in circuits are deemed fixed during the training process.

In the canonical ESN the neurons have associated a sigmoid transfer function. A variation of the ESN model named *Echo State Queueing Network (ESQN)* that uses the dynamics of the RNN (based on Eqns. (6), (7) and (8)) was introduced in [13, 14]. The ESQN has been successfully applied for solving temporal learning tasks, for instance, in predicting future Internet traffic based on past observations [13].

4.5 Universal approximator

George Cybenko investigated in 1989 the conditions under which feedforward NNs are dense functions in the space of continuous functions defined, say, in the hypercube $[0,1]^{|\mathcal{O}|}$) [28]. Cybenko proved that any continuous function can be uniformly approximated by a continuous classic NN having a finite number of neurons and only one hidden layer. The considered activation function of the neurons was a sigmoid function. In [48], the authors studied this property for the RNN model. They proved that RNNs constitute a family of functions that can approximate any real continuous function $f : [0,1]^{|\mathcal{I}|} \to [0,1]^{|\mathcal{O}|}$ with an arbitrary precision [48, 58]. In order to prove that RNNs satisfy this *approximator universal property*, the authors consider a specific topology and two extensions of the model called *Bipolar Random Neural Network (BRNN)* and *Clamped Random Neural Network (CRNN)*. For details about the proof see [49, 40, 48, 58].

4.6 Analogy between RNNs and other NNs

In [37] was studied an analogy between a specific class of Artificial NN and the RNN model. Consider a classical NN and let w_{uv} be the weight from neuron u to neuron v (note that the notation is in reverse order to that used in the standard NN literature [19, 98]). Given a sequence of inputs $y_1, y_2, \ldots, y_N$ to neuron u, its output is

$$y_u = f\left(\sum_{v=1}^{N} w_{vu} y_v - \theta_u\right), \tag{47}$$

where $f(\cdot)$ is a sigmoid function [37]. Assume that the input-output pattern in the training data is binary. The input space is $\{0,1\}^{|\mathcal{I}|}$ and the output space is $\{0,1\}^{|\mathcal{O}|}$, where $|\mathcal{I}|$ and $|\mathcal{O}|$ are positive integers. The network topology is of the feeedforward type with multiple layers. The adjustable parameters of the model are the weight connections (w_{uv}) and the bias parameter (θ_u) for $u, v = 1, \ldots, N$. Let us consider the indexes u and v to denote the neurons in the ANN and the indexes i and j to denote the neurons in the RNN model. The analogy between both networks is built using the same number of input, hidden and output neurons in the two networks, as follows. The threshold of neuron u is associated with the flow of negative signals

to neuron i: $T_i^- = \theta_u$. The relation among the weight connections is given by the following rules: if $w_{uv} > 0$, then $r_i p_{i,j}^+ = w_{uv}$; if $w_{uv} < 0$, then $r_i p_{i,j}^- = |w_{uv}|$. If i is an output neuron, then $d_i = 1$, and r_i is a free parameter. The authors propose to set $d_i = 0$ and $r_i = \sum_{j=1}^{N} \left(p_{i,j}^+ + p_{i,j}^-\right) = \sum_{v=1}^{N} |w_{uv}|$, when i and u are an input and a hidden neuron respectively. Then, the input signals of an input node of the RNN are set as follows:

- If the element i of the input pattern is equal to 1, then λ_i^+ is a non null constant in $(0, 1]$ while $\lambda_i^- = 0$. The authors propose an ad-hoc setting for the input rate λ_i^+, chosen according to the training data in order to obtain the desired outputs.
- If the input pattern is equal to 0, then $\lambda_i^- \neq 0$ and $\lambda_i^+ = 0$.

The vector of output activation values $(y_1, \ldots, y_N)$ corresponding to each neuron in the ANN is associated with the vector $(\varrho_1, \ldots, \varrho_N)$ in the RNN. In the paper, the authors proposed to use some "cut–points" $(\alpha_1, \ldots, \alpha_N)$ such that

$$y_i = 0 \Longleftrightarrow \varrho_i < 1 - \alpha_i \qquad \text{and} \qquad y_i = 1 \Longleftrightarrow \varrho_i > 1 - \alpha_i.$$

The values of the flow of input signals λ_i^+ and the cut-points α_i must be chosen in order to obtain the expected effect at the output values. It can be observed that the task of tuning these parameters can be a non trivial problem.

4.7 Model variations

There are several extensions of the canonical RNN model. We present a brief summary of a few of them.

- In [34], negative arriving customers (signals) make that the potential of the neurons goes to zero.
- In [44, 46, 53, 32] multiple classes of neurons are allowed, leading to an increased flexibility in the design of the model.
- In [54, 59, 22] negative signal arrivals can trigger customers' movements in other parts of the network.
- In [67, 69, 33, 68] the rates of the neurons can be state-dependent, and batch movements are also allowed.

5. Applications in the supervised learning area

Since its apparition in 1989, the RNN model has been applied in a large variety of supervised learning problems. This article is not a survey about the RNN applications. We focus on the numerical optimization algorithms applied for solving the supervised tasks, as well as in the usage of these algorithms. Both points were already discussed in the previous sections. However, in the following we briefly describe several applications of the model. The main references used in this overview are [104, 60, 9, 99, 29].

5.1 Application for multimedia quality assessment

Neural Networks have been applied for developing mechanisms of controlling the quality of multimedia applications. Two main classes of methods can be considered for assessing the perceived quality of multimedia streams:

- Objective methods: they are usually based on a comparison between the original and distorted sequences. The main example is the Peak Signal to Noise Ratio (PSNR).

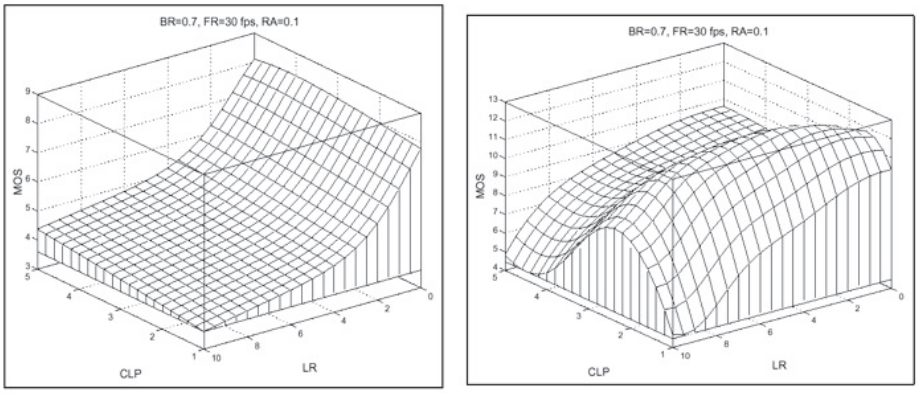
- Subjective methods: the most commonly used measure is the Mean Opinion Score (MOS), where a group of people evaluate media samples according to a predefined quality scale, under carefully controlled experimental conditions.

Thus, the quality is measured as a distance function between the original and the distorted sequences in the case of objective methods, and it is some statistical function based on human evaluations in the case of subjective methods. These approaches have the following drawbacks. Objective methods require the original streams which often are not available in practical applications. In particular, this restriction makes that these techniques can not be used for controlling applications, which need real-time reactions. Subjective methods are expensive and, by definition, they can not be used in real-time problems. Another remark is that these methods do not take into account the effects of the packet network's parameters in the perceived quality the streams.

A new approach was presented in [95], based on subjective techniques and QoS metrics, in order to estimate in real time the quality of the media. We can describe the main idea as follows. A set of original signals of video or audio (depending of our problem) is considered as the data set. The training data is generated using a codec and network simulation distortion considering loss rate, delay and other network parameters which affect quality. Several models to characterize the loss processes in the Internet are used to simulate distorted data, such as independent losses [20] or fixed–size loss bursts [64], the Gilbert Model [62] and k^{th} order Markov chains [107]. Next, a group of subjects evaluates the training set using a MOS protocol, thus a collection of images or sound distorted is subjectively evaluated.

The RNN has been used for learning the relationship between the network parameters (loss rate, delay, jitter,...) and quality as evaluated by panels of human users. This approach of estimating the multimedia quality the has been studied in the following articles [94, 96, 86, 87], among many others. In [96] the model was compared with Naive Bayesian Classifiers and with classic Neural Networks, for this specific supervised learning application the RNN model shows a better performance.

In [95], other learning techniques were explored, including those included in commercial packages. Fig. 4 shows one of the situations where RNN behaved better than a competitor, on the same data and keeping the sizes of the vectors of weights identical. We can see the classical *overtraining* phenomenon appearing in the other technique.



(a) Correctly trainined (b) Example of an over-trained ANN

Fig. 4 *Comparison between the RNN model in (a) and a classical commercial ANN tool in (b); the picture shows the learned function plotted when varying two of its input variables. Physical considerations say that the function was corrected learned in (a), and that (b) is unrealistic.*

5.2 Solving temporal supervised learning

Recently a computational model that uses the dynamics of the RNN has been analyzed for studying sequential dataset [13, 14]. The model name is *Echo State Queueing Network (ESQN)* because it is an hybrid produced from two different types of Neural Networks: RNNs and Echo State Networks (ESN) [78]. The ESN are recurrent NNs with sigmoid activation functions. They have been successful used for solving time-series problems. The RNN variation named ESQN uses the circuits of the network for memorizing the sequential data. Besides, the recurrent part of the network acts as a projection method (as in ESNs and in Kernel Methods) in order to enhance the linear separability of the input data. The network has fixed the weights that are involved in the circuits and only the weights that generate the output of the model are updated in the training phase. The state of the nodes in this variation of the RNN model is vector $\boldsymbol{\varrho}$. When an input pattern $\mathbf{a}^{(k)}$ is presented to the network, the vector of states evolves according to the dynamics given by the following equations, where $\mathcal{I}$ denotes the set of input neurons and $\mathcal{H}$ denotes the set of hidden ones (the "reservoir"):

$$\varrho_i^{(k)} = \frac{a_i^{(k)}}{r_i}, \quad i \in \mathcal{I}, \tag{48}$$

and for $i \in \mathcal{H}$,

$$\varrho_i^{(k)} = \frac{\displaystyle\sum_{j\in\mathcal{I}} w_{ij}^{+} \frac{a_j^{(k)}}{r_j} + \sum_{j\in\mathcal{I}\cup\mathcal{H}} w_{ij}^{+} \varrho_j^{(k-1)}}{r_i + \displaystyle\sum_{j\in\mathcal{I}} w_{ij}^{-} \frac{a_j^{(k)}}{r_j} + \sum_{j\in\mathcal{I}\cup\mathcal{H}} w_{ij}^{-} \varrho_j^{(k-1)}}, \tag{49}$$

Traffic time series	Model	NMSE	CI
ISP	NN variation (ESN)	0.0062	$\pm 9.8885 \times 10^{-7}$
	RNN variation (ESQN)	0.0100	$\pm 1.2436 \times 10^{-4}$
UKERNA	NN variation (ESN)	0.3781	± 0.0066
	RNN variation (ESQN)	0.2030	± 0.0335

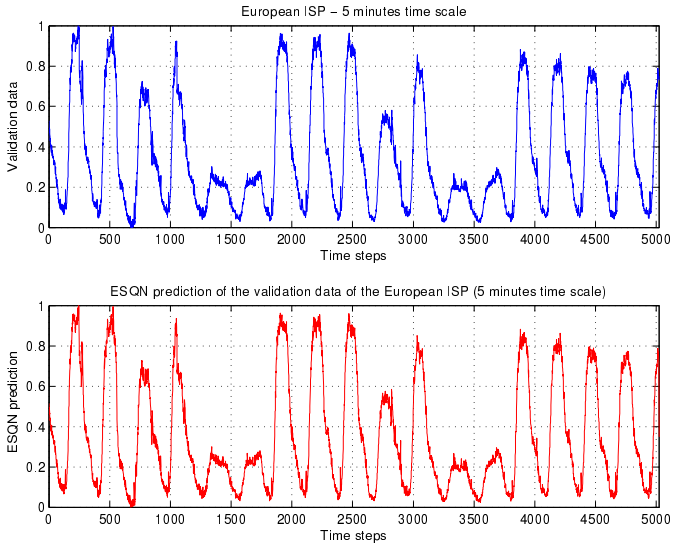
Tab. I *Comparison between the ESN (RNN with sigmoid activation function and fixed recurrent structure) and the variation of RNN for time series problems (named ESQN). The table presents the Normalized MSE reached on* 20 *independent trials, and the corresponding Confidence Interval (CI).*

for all $i \in \mathcal{H}$. The expression (49) is a dynamical system that has at the left the state values at time k, and on the right hand side the state values at time $k-1$. In the same way as in the ESN model, the parameters are computed generated using a linear ridge regression from $\boldsymbol{\varrho}$ to $\mathbf{b}$.

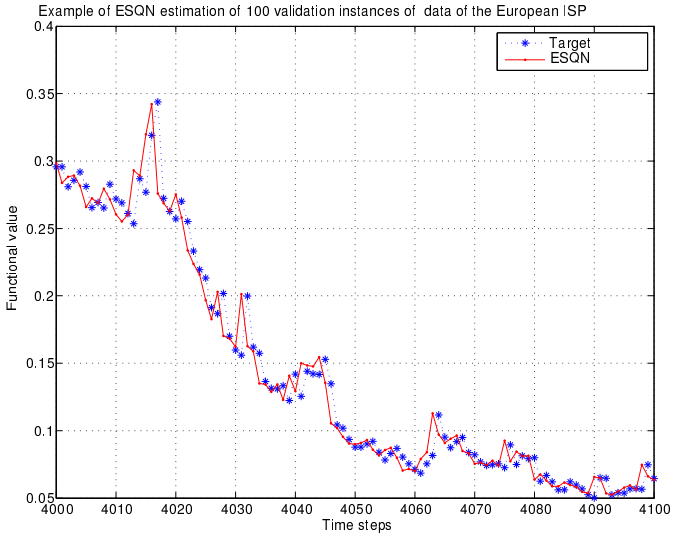
According to the experimental results presented in [13] the model presents a competitive accuracy if one refers to the classic Neural Networks with sigmoid activation functions (as, for instance, the ESN model). In particular, the model has been applied for predicting the Internet traffic on real dataset from an Internet Service Provider (ISP) working in 11 European cities [24, 71]. The data was collected from a private ISP from 7 June to 29 July, 2005. It covers traffic information of 11 European cities. In addition, the model has been applied for traffic prediction using another benchmark data from the *Traffic data from United Kingdom Education and Research Networking Association (UKERNA)* [24, 71]. The data was collected from 19 November and 27 January, 2005. It was studied in [24]. In order to analyze the sequential behavior, the problem was studied collecting the data and using three-time scales: day, hour and intervals of five minutes. Different scales of time capture the strength of trend and seasonality. Fig. 5 shows an example of the Internet traffic prediction using the ESQN model on the European ISP dataset. Table I illustrates the accuracies of the ESQN and ESN models for predicting the Internet traffic. For more details about the implementation of ESQNs to solve time-series problems see [13, 14, 15].

5.3 Approximation of nonlinear functions

A supervised task consists in generating a parametric mapping between inputs and outputs. One of the most referenced and studied nonlinear benchmark has been the xor problem. In 1969, Marvin Minsky and Seymour Papert proved the inability of perceptrons to solve it [85]. Since then, xor is considered as a classical reference to study the ability of a model to solve nonlinear classification problems. A RNN that tries to match the xor function was studied in [37, 39]. The same problem solved using learning algorithms was studied in [16]. In [16] a generalization of the xor problem named the parity problem was analyzed. The RNNs have been used also for approximating real functions. For instance, in [16] it was studied for approximating a sinusoidal function. In [82] an extension of the canonical RNN has been used for solving real function approximation problems.



(a) Internet traffic prediction using the RNN model called ESQN on the European ISP traffic validation data (5 minutes scale)



(b) Internet traffic prediction using the RNN variation called ESQN on the European ISP traffic validation data (5 minutes scale). Example of ESQN prediction for 1000 time steps

Fig. 5 *Example of Internet traffic prediction using the RNN variation called ESQN on the European ISP traffic validation data (5 minutes scale).*

5.4 Image processing

The RNN model has been widely used in image processing problems. In this section we give a brief description about this type of applications.

Image compression

Feedforward NNs have been successful used for compressing images [31]. The network has at least one hidden layer, and the same number of input and output neurons. The number of hidden neurons is smaller that the number of units in the other layers. The compression ratio is the rate between the number of input and of hidden neurons. This specific topology is often referred to as a *bottleneck* network. We illustrate it in Fig. 6 [79]. The input data is a collection that represents the original image. For this purpose, some relevant features of the image is used [9]. In the learning procedure the feedforward network is used as an *auto-associator tool*, which means that the model is trained in order to recreate the input data [31].

The same approach was used with the RNNs on image data compression tasks [25, 51, 26]. In [9], RNNs were compared to other more traditional compression tools from the performance point of view. The measure of quality considered was the *Peak Signal to Noise Ratio (PSNR)*. The authors remarked that traditional methods, such as JPG and Wavelet Compression, reached higher performance level (with respect to PSNR) than compression with RNNs. However, RNN presents the advantage to be faster that the other techniques. Besides, RNNs for image compression can be adapted for an implementation using parallel computing. For instance, the image can be fragmented in several parts, and for each part an RNN can be used for compressed/decompressed the image fragments. This procedure can be implemented in parallel. Thus, the computational time of this compression/decompression technique considerably decreases with respect to the other more classical techniques.

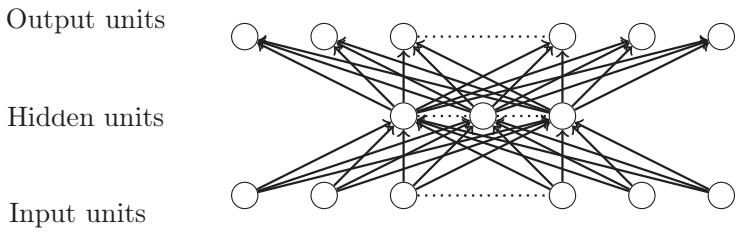


Fig. 6 *Bottleneck network used for compressing images. The network has the same number of input and of output neurons. The compression ratio is given by ratio between the number of input and of hidden neurons.*

Image enlargement

The *Image Enlargement* is a technique for resizing a digital image in order to increase its resolution. A procedure to solve this problem using RNNs was developed in [8, 9]. The model was applied for enlarging two well know images called *Lena*

and *Peppers*. The procedure requires a training data composed by pairs of images. Each pair is composed of the original (the smaller one) and the target (the larger one). The authors propose to use a feedforward RNN and the gradient descent training algorithm. The training function is defined using the *zero order interpolation*, a technique for signal reconstruction. For details about the technicalities of this procedure see [31, 9].

Image fusion

In the fusion of images the goal is to obtain a new image with high-resolution from a set of images with low-resolution. The problem using RNNs was examined in [9]. The training inputs are composed of several images, for instance produced by sensors. As usual in supervised learning, each input pattern has associated with a target. In this task, the target is an image with better resolution that the one, present at the input. The network is used for learning the mapping between the set of sensor images and the target image. The authors in [9] use Gradient Descent with RNN for solving this problem.

Medical image computing

RNNs have been applied in image segmentation for *Magnetic Resonance Imaging (MRI)* of the brain [50, 43]. The application on MRI is based on the following procedure. Given a reference image that contains a finite set of regions, the goal is to use RNNs for classifying those regions. The approach consists of training several RNNs, each of them using a specific region of the reference image. Then, after this training phase, an image is decomposed into many blocks of small sizes, and each block is classified using the corresponding RNN. Thus, we can assign a label to each small block (eventually individual pixels can be considered) of the image. The model has been applied to the segmentation of ultrasound images [77].

Textural features for image classification

Textural features classification is an area of image processing and computer vision where the goal is to categorize or to classify pictorial information. This is done using a set of meaningful features of the image. Texture is one of the main characteristics for identifying or classifying individual pixels or pixel blocks in an image. Several works analyze what kind of textural features are useful for classifying images [65, 93]. Once the features are defined, the blocks of images can be categorized using some pattern recognition methods. In [103] the authors analyze the performance of using RNNs to classify pixels into texture classes. According to this article, the classification performance of RNNs is comparable with that of the other methods presented in the pattern recognition literature.

An extension of RNNs called *Multiple Classes Random Networks* [44] was applied to the Color Pattern Recognition problem in [2]. The model was also applied to another classification problem called *Laser Intensity Vehicle Classification System (LIVCS)* [70]. Basically, laser intensity images are collected in an information system. This kind of images produce information that is less sensitive to environmental conditions that information produced by sensors and video systems. Once

the information is collected, specific features are used for training the RNN model. Another area of application was image reconstruction, see for instance [56].

Other areas of image processing

In a similar way as for image compression, the RNN model has also been applied to video compression [51]. The authors develop the technique on video sequences over Broadband ISDN networks. According to [25, 26, 27, 51], RNNs exhibit a good performance in these tasks.

Image synthesis consists of creating digital images from some texture and form of image description. The approach using RNNs considers an architecture where each neuron is associated with each pixel of the image. To train the networks the method uses a training set composed of images where the aim is to learn certain textural features such as granularity, inclination, contrast, homogeneity and others. Once the network is trained, it can be used to generate images with similar textural characteristics as the images used in the learning process. Models to create image synthesis in gray level using RNNs have been examined in [6, 45, 103]. There is also some research for color images [7, 46].

5.5 Other applications of the RNN model

In addition to supervised learning, the model has been used in the field of combinatorial optimization problems. Some of the main applications are the following ones (the list is not exhaustive):

- The *Traveling Salesman Problem (TSP)* is a classical case within hard combinatorial optimization problems. The goal is to design a set of minimum-cost vehicle routes delivering goods to a set of customers where the vehicles start and finish at a central point. This problem was studied using RNNs in [47].
- Another combinatorial optimization problem is the *Minimum Vertex Covering* (MVC) problem which was studied in [1, 42].
- The *satisfiability* (SAT) problem consists of determining if the variables of a given Boolean formula can be assigned in such a way that the formula evaluates to TRUE. The SAT problem has a great importance in computer science and it was the first known NP–complete problem. It was studied using RNNs in [41].
- The well known *Minimum Steiner Tree* (MST) problem in graphs is a sort of benchmark in combinatorial optimization. A RNN-based approach has been developed to find a solution in [55, 61].
- RNNs have also been used to solve two problems close to MSTs: the *dynamic multicast* problem [55, 3] and the *access network design* problem [21].
- The *Independent Set problem* was studied with a variation of the RNN model in [90].
- The *Optimal Resource Allocation* on a distributed system was analyzed using RNNs in [108].

- Associative memory: works about associative networks for storage and retrieval symbolic information that use the RNN model can be found in [102, 52, 49, 73]. In addition, an analogy between the Hopfield Network and the RNN was studied in [75].

- Applications in networking using a routing protocol called the Cognitive Packet Networks are found in [76, 57, 99].

6. Conclusions and perspectives

Since the early 1990s, the *Random Neural Network (RNN)* model has gained importance in the Neural Networks (NNs) area as well as in the field of the Queueing Theory. The model can be seen as an extension of open Jackson's networks (it can actually be seen in several different ways). It has characteristics that are present in biological NNs, such that the action potential, firing spikes among neurons, inhibitory and exhibitory spikes, random delays between spikes, and so on.

This article intends to be a practical guide for applying RNNs in supervised learning tasks. Several learning algorithms that have been used for training RNNs are presented in some detail. In 1993, a learning algorithm of the *Gradient Descent (GD)* type was adapted for the RNN case. At the beginning of the 2000s, *Quasi-Newton (QN)* methods were applied for training these models. More recently, other second-order algorithms, namely the *Levenberg-Marquardt (LM)* method and some variations were also developed for training RNNs. The LM procedure is a more robust technique than Gauss-Newton algorithms and it is very popular in NNs. In general, the available experimental results have shown that QN methods are much faster than GD algorithms. However, QN methods (for instance LM) can present robustness problems. First order optimization models are usually more robust but slower. In the case of a recurrent topology, a first order method can present also problems of stability. Those drawbacks are identified in the NN literature as the "exploding" and "vanishing" phenomena. To the best of our knowledge, those phenomena have not been analyzed yet in the RNN case.

RNNs have been successfully used as learning tools in many applications of different types. In this overview, we only scratched the surface of some of those applications. We described some solutions in the area of pattern recognition, image processing, compression, as well as in some combinatorial optimization problems.

Due to the good results previously mentioned, we encourage researchers to continue the ongoing effort around, in particular, the following lines:

- All the learning algorithms presented in the RNN literature use the same type of cost function: a sum of squared "individual" errors. It can be interesting to develop algorithms that use other measures of errors, as for instance the Kullback-Leiber one. It is known that this metric is specially useful for evaluate the learning performance on classification tools [66].

- A point that deserves specific attention is the study of the difficulties of training a RNN with a recurrent topology, in particular with GD-type algorithms. See for instance [17, 89] in the case of classical NNs. As far as we know, these studies have not been extended yet to the case of RNNs.

- There are several optimization algorithms developed in the NN area that are not still studied for the RNN case. For instance, we can mention the Hessian-free optimization and back-propagation curvature [83].
- In [11] RNNs have been implemented in parallel multiple GPUs. In this implementation, the authors used only the gradient descent method. Therefore, similar works using second-order methods in parallel implementations should be explored as-well.

Acknowledgments

This article has been elaborated in the framework of the project New creative teams in priorities of scientific research, reg. no. CZ.1.07/2.3.00/30.0055, supported by Operational Programme Education for Competitiveness and co-financed by the European Social Fund and the state budget of the Czech Republic and supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, and by the Project SP2015/105 *DPDM - Database of Performance and Dependability Models of the Student Grand System*, VŠB-Technical University of Ostrava and by Project SP2015/146 *Parallel processing of Big data 2*, of the Student Grand System, VŠB-Technical University of Ostrava.

Curriculum Vitae

Sebastián Basterrech received a M.Sc. degree in Applied Mathematics in 2008 from the Aix-Marseille University, France. From 2009 to 2012 he received a doctoral fellowship from the National Institute for Research in Computer Science and Control (INRIA), France. He obtained the Ph.D. degree in Computer Sciences in November, 2012 from INRIA-Rennes and University of Rennes 1, France. In July, 2013 he received the M.Sc. degree in Computer Arts at the University of Rennes 2 in France. Since June, 2013 he is a postdoctoral researcher at the National Super Computer Center and VŠB-Technical University of Ostrava, Czech Republic. He is TC member of the IEEE SMC Soft-Computing Society. The main research interests of S. Basterrech include Spatio-temporal Data Mining, Recurrent Neural Networks and Bio-inspired Algorithms for Machine Learning Applications.

Gerardo Rubino is a senior researcher at INRIA (the French National Institute for Research in Computer Science and Control) where he is the leader of the DIONYSOS group, working on the analysis and design of networking technologies. He is Board Member of the Media and Networks Cluster, Brittany, France, and INRIA's representative at the SISCom Brittany Research Cluster. Among past responsibilities, he has been Scientific Delegate for the Rennes unit of INRIA for 5 years, responsible of

research in networking at the Telecom Bretagne engineering school for 5 years, Associate Editor of the Operations Research international journal "Naval Research Logistics" for 9 years, former member of the Steering Board of the European Network of Excellence EuroFGI and responsible of the relationships between the network and European industry. He has also been the head of the International Partnership Office at INRIA Rennes for 5 years. He is a member of the IFIP WG 7.3. He belongs to the Steering Committee of QEST (www.qest.org). He is interested in quantitative analysis of complex systems using probabilistic models. He presently works on performance and dependability analysis, and on perceptual quality assessment of audio and video applications and services. In particular, he is the author of the PSQA (Pseudo-Subjective Quality Assessment) technology for automatic perceptual quality real-time evaluation. He also works on rare event analysis, he is a member of the Steering Committee of RESIM, the only workshop dedicated to the topic, and co-author of the book entitled "Rare Event Simulation Using Monte Carlo Methods" (Wiley, 2009). He co-authored the book "Markov Chains and Dependability Theory" published in 2014 by Cambridge University Press. He is the author of more than 200 scientific works in applied mathematics and computer science.

References

[1] AGUILAR J. Definition of an energy function for the random neural to solve optimization problems. *Neural Networks*. 1983, 11, pp. 731–737, doi: 10.1016/s0893-6080(98)00020-3.

[2] AGUILAR J. Learning Algorithm and Retrieval Process for the Multiple Class Random Neural Network Model. *Neural Processing Letters*. 2001, pp. 81–91, doi: 10.1023/A:1009611918681.

[3] AIELLO G., GAGLIO S., RE G., STORNIOLO P., URSO A. The Random Neural Network Model for the On-Line Multicast Problem. *Biological and Artificial Intelligence Environments*. 2005, pp. 157–164, doi: 10.1007/1-4020-3432-6_19.

[4] AMPAZIS N., PERANTONIS S. J. Levenberg-Marquardt algorithm with adaptive momentum for the efficient training of feedforward networks. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*. 2000, pp. 126–131, doi: 10.1109/ijcnn.2000.857825.

[5] AMPAZIS N., PERANTONIS S. J., TAYLOR J. G. Dynamics of multiplayer network in the vicinity of temporary minima. *Neural Networks*. 1999, 12(1), pp. 45–58, doi: 10.1016/S0893-6080(98)00103-8.

[6] ATALAY V., GELENBE E., YALABIK N. The random neural network model for texture generation. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*. 1992, 6, pp. 131–141, doi: 10.1142/S0218001492000072.

[7] ATALAY V., GELENBE E. Parallel algorithm for colour texture generation using the random neural network model. In: *Parallel image processing*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1992, pp. 227–236. Available also from: http://dl.acm.org/citation.cfm?id=183390.183574.

[8] BAKIRCIOĞLU H., GELENBE E., KOÇAK T. Image processing with the Random Neural Network model. *ELEKTRIK*. 1998, 5(1), pp. 65–77.

REFERENCES

[9] BAKIRCIOĞLU H., KOÇAK T. Survey of Random Neural Network applications. *European Journal of Operational Research*. 2000, 126(2), pp. 319–330, doi: 10.1016/S0377-2217(99)00481-6.

[10] BARNARD E. Optimization for Training Neural Nets. *Neural Networks, IEEE Transactions on*. 1992, 3(2), pp. 232–240, doi: 10.1109/72.125864.

[11] BASTERRECH S., JANOUŠEK J., SNÁŠEL V. A Study of Random Neural Network Performance for Supervised Learning Tasks in CUDA. In: J.-S. PAN, V. SNÁŠEL, E. S. CORCHADO, A. ABRAHAM, S.-L. WANG, eds. *Intelligent Data analysis and its Applications, Volume II*. Springer International Publishing, 2014, pp. 459–468. Advances in Intelligent Systems and Computing series.

[12] BASTERRECH S., RUBINO G. A More Powerful Random Neural Network Model in Supervised Learning Applications. In: *Soft Computing and Pattern Recognition (SoCPaR), 2013 International Conference of*, 2013, pp. 201–206, doi: 10.1109/SOCPAR.2013.7054127.

[13] BASTERRECH S., RUBINO G. Echo State Queueing Network: A new Reservoir Computing Learning Tool. In: *10th IEEE Consumer Communications and Networking Conference, CCNC 2013, Las Vegas, NV, USA, January 11-14, 2013*, 2013, pp. 118–123, doi: 10.1109/CCNC.2013.6488435.

[14] BASTERRECH S., SNÁŠEL V. Initializing Reservoirs With Exhibitory And Inhibitory Signals Using Unsupervised Learning Techniques. In: *International Symposium on Information and Communication Technology (SoICT)*, Danang, Viet Nam: ACM Digital Library, 2013, doi: 10.1145/2542050.2542087.

[15] BASTERRECH S., SNÁŠEL V., RUBINO G. An Experimental Analysis of Reservoir Parameters of the Echo State Queueing Network Model. In: A. ABRAHAM, P. KRÖMER, V. SNÁŠEL, eds. *Innovations in Bio-inspired Computing and Applications*. Springer International Publishing, 2014, pp. 13–22. Advances in Intelligent Systems and Computing series.

[16] BASTERRECH S., MOHAMED S., RUBINO G., SOLIMAN M. Levenberg-Marquardt Training Algorithms for Random Neural Networks. *Computer Journal*. 2011, 54(1), pp. 125–135, doi: 10.1093/comjnl/bxp101.

[17] BENGIO Y., SIMARD P., FRASCONI P. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*. 1994, 5(2), pp. 157–166, doi: 10.1109/72.279181.

[18] BISHOP C. Exact Calculation of the Hessian Matrix for the Multilayer Perceptron. *Neural Computation*. 1992, 4(4), pp. 494–501, doi: 10.1162/neco.1992.4.4.494.

[19] BISHOP C. M. *Neural Networks for Pattern Recognition*. New York: Oxford University Press Inc., 1995.

[20] BOLOT J.-C. Characterizing End-to-End Packet Delay and Loss in the Internet. *Journal of High-Speed Networks*. 1993, 2(3), pp. 305–323.

[21] CANCELA H., ROBLEDO F., RUBINO G. A GRASP algorithm with RNN based local search for designing a WAN access network. *Electronic Notes in Discrete Mathematics*. 2004, 18, pp. 59–65, doi: 10.1016/j.endm.2004.06.010.

[22] CHAO X. A note on queueing networks with signals and random triggering times. *Probablility in the Engineering and Informational Sciences*. 1994, 8, pp. 213–219, doi: 10.1017/s0269964800003351.

[23] COPPERSMITH D., WINOGRAD S. Matrix Multiplication via Arithmetic Progressions. *Journal of Symbolic Computation*. 1990, 9, pp. 251–280, doi: 10.1016/S0747-7171(08)80013-2.

[24] CORTEZ P., RIO M., ROCHA M., SOUSA P. Multiscale Internet traffic forecasting using Neural Networks and time series methods. *Expert Systems*. 2012, doi: 10.1111/j.1468-0394.2010.00568.x.

[25] CRAMER C., GELENBE E., BAKIRCIOGLU H. Low Bit Rate Video Compression with Neural Networks and Temporal Sub-Sampling. *Proceedings of the IEEE*. 1996, 84(10), pp. 1529–1543, doi: 10.1109/5.537116.

[26] CRAMER C., GELENBE E., GELENBE P. Image and video compression. *Potentials, IEEE*. 1998, 17(1), pp. 29 –33, doi: 10.1109/45.652854.

[27] CRAMER C., GELENBE E. Video Quality and Traffic QoS in Learning-based Subsampled and Receiver-interpolated Video Sequences. *Selected Areas in Communications, IEEE Journal on*. 2000, 18(2), pp. 150 –167, doi: 10.1109/49.824788.

[28] CYBENKO G. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*. 1989, 2(4), pp. 303–314, doi: 10.1007/BF02551274.

[29] DO T. V. An initiative for a classified bibliography on G-networks. *Performance Evaluation*. 2011, 68, pp. 385–394, doi: 10.1016/j.peva.2010.10.001.

[30] DRUCKER H., LE CUN Y. Improving Generalization Performance Using Double Backpropagation. *IEEE Transaction on Neural Networks*. 1992, 3(6), pp. 991–997, doi: 10.1109/72.165600.

[31] EGMONT-PETERSEN M., de RIDDER D., HANDELS H. Image processing with neural networks - a review. *Pattern Recognition*. 2002, 35, pp. 2279–2301, doi: 10.1016/S0031-3203(01)00178-9.

[32] FOURNEAU J. M., KLOUL L., VERCHÉRE D. Multiple class G-networks with list-oriented deletions. *European Journal of Operational Research*. 2000, 126(2), pp. 250 –272, doi: 10.1016/S0377-2217(99)00477-4.

[33] FOURNEAU J.-M., VERCHÈRE D. G-Networks with Triggered Batch State-Dependent Movement. In: *MASCOTS '95: Proceedings of the 3rd International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Washington, DC, USA: IEEE Computer Society, 1995, pp. 33–37, doi: 10.1109/mascot.1995.378713.

[34] GELENBE E. G-networks with signals and batch removal. *Probability in the Engineering and Informational Sciences*. 1993, 7, pp. 335–342, doi: 10.1017/S0269964800002953.

[35] GELENBE E. Learning in the Recurrent Random Neural Network. *Neural Computation*. 1993, 5(1), pp. 154–164, doi: 10.1162/neco.1993.5.1.154.

[36] GELENBE E. Product-Form Queueing Networks with Negative and Positive Customers. *Journal of Applied Probability*. 1991, 28(3), pp. 656–663, doi: 10.2307/3214499.

[37] GELENBE E. Random Neural Networks with Negative and Positive Signals and Product Form Solution. *Neural Computation*. 1989, 1(4), pp. 502–510, doi: 10.1162/neco.1989.1.4.502.

[38] GELENBE E. Réseaux stochastiques ouverts avec clients négatifs et positifs, et réseaux neuronaux. *Comptes Rendus de l'Académie des Sciences 309, Série II*. 1989, 309, pp. 979–982.

[39] GELENBE E. Stability of the Random Neural Network Model. In: *Proc. of Neural Computation Workshop*, Berlin, West Germany, 1990, pp. 56–68, doi: 10.1007/3-540-52255-7_27.

[40] GELENBE E. The Spiked Random Neural Network: Nonlinearity, Learning and Approximation. In: *Proc. Fifth IEEE International Workshop on Cellular Neural Networks and Their Applications*, London, England, 1998, pp. 14–19, doi: 10.1109/CNNA.1998.685674.

[41] GELENBE E. Une généralisation probabiliste du probleme SAT. *Comptes Rendus de l'Académie des Sciences 309, Série II*. 1992, 313, pp. 339–342.

[42] GELENBE E., BATTY F. Minimum cost graph covering with the random neural network. In: *Computer Science and Operations Research*. New York: Pergamon, 1992, pp. 139–147.

[43] GELENBE E., FENG Y., KRISHNAN K. Neural Network Methods for Volumetric Magnetic Resonance Imaging of the Human Brain. *Proceedings of the IEEE*. 1996, 84(10), pp. 1488 –1496, doi: 10.1109/5.537113.

[44] GELENBE E., FOURNEAU J. Random neural networks with multiple classes of signals. *Neural Computation*. 1999, 11(4), pp. 721–731, doi: 10.1162/089976699300016520.

[45] GELENBE E., HUSSAIN K., ABDELBAKI H. M. Random Neural Network texture model. In: *Conference on Applications of Artificial Neural Networks in Image Processing V*, San Jose, CA, USA, 2000, pp. 104–111, doi: 10.1117/12.382903.

[46] GELENBE E., HUSSAIN K. Learning in the multiple class random neural network. *Neural Networks, IEEE Transactions on*. 2002, 13(6), pp. 1257–1267, doi: 10.1109/TNN.2002.804228.

[47] GELENBE E., KOUBI V., PEKERGIN F. Dynamical random neural network approach to the traveling salesman problem. In: *Proc. IEEE Symp. Syst., Man, Cybern.* 1993, pp. 630–635, doi: 10.1109/icsmc.1993.384945.

[48] GELENBE E., MAO Z., LI Y. Function Approximation with Spiked Random Networks. *IEEE Trans. on Neural Networks*. 1999, 10, pp. 3–9, doi: 10.1109/72.737488.

[49] GELENBE E., STAFYLOPATIS A., LIKAS A. Associative Memory Operation of the Random Network Model. In: *Proc. Int. Conf. Artificial Neural Networks, ICANN 91*. Espoo, Finland, 1991, pp. 307–312.

[50] GELENBE E., FENG Y., RANGA K., KRISHNAN R. Neural Networks for volumetric magnetic resonance imaging of the brain. In: *International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing (NICROSP'96)*, Venice, Italy: IEEE Computer Society, 1996, pp. 194–202, doi: 10.1109/NICRSP.1996.542760.

[51] GELENBE E., CRAMER C., SUNGUR M., GELENBE P. Traffic and Video Quality in Adaptive Neural Compression. *Multimedia Systems*. 1996, 4(6), pp. 357–369, doi: 10.1007/s005300050037.

[52] GELENBE E. Distributed associative memory and the computation of membership functions. *Information Sciences*. 1991, 57-58(0), pp. 171–180, doi: 10.1016/0020-0255(91)90076-7.

[53] GELENBE E. G-Networks: Multiple Classes of Positive Customers, Signals, and Product form Results. In: *Lecture Notes in Computer Science. Performance Evaluation of Complex Systems: Techniques and Tools*, Springer, 2002, pp. 129–141, doi: 10.1007/3-540-45798-4_1.

[54] GELENBE E. G-Networks with Triggered Customer Movement. *Journal of Applied Probability*. 1993, 30(3), pp. 742–748, doi: 10.2307/3214781.

[55] GELENBE E., GHANWANI A., SRINIVASAN V. Improved Neural Heuristics for Multicast Routing. *IEEE Journal on Selected Areas in Communications*. 1997, 15(2), pp. 147–155, doi: 10.1109/49.552065.

[56] GELENBE E., KOÇAK T., WANG R. Wafer surface reconstruction from top-down scanning electron microscope images. *Microelectronic Engineering*. 2004, 75(2), pp. 216 –233, doi: 10.1016/j.mee.2004.05.006.

[57] GELENBE E., LOUKAS G. A self-aware approach to denial of service defence. *Computer Networks*. 2007, 51(5), pp. 1299–1314, doi: 10.1016/j.comnet.2006.09.009.

[58] GELENBE E., MAO Z.-H., LI Y.-D. Function Approximation by Random Neural Networks with a Bounded Number of Layers. *Journal of Differential Equations and Dynamical Systems*. 2004, 12(1-2), pp. 143–170, doi: 10.1142/9781860948923_0005.

[59] GELENBE E., SHACHNAI H. On G-networks and resource allocation in multimedia systems. *European Journal of Operational Research*. 2000, 126(2), pp. 308–318, doi: 10.1016/S0377-2217(99)00480-4.

[60] GEORGIOPOULOS M., LI C., KOÇAK T. Learning in the feed-forward random neural network: A critical review. *Performance Evaluation*. 2011, 68(4), pp. 361–384, doi: 10.1016/j.peva.2010.07.006.

[61] GHANWANI A. Neural and delay based heuristics for the Steiner problem in networks. *European Journal of Operational Research*. 1998, 108, pp. 231–232, doi: 10.1016/s0377-2217(97)00369-x.

[62] GILBERT E. Capacity of a Burst-loss Channel. *Bell Systems Technical Journal*. 1960, 5(39), doi: 10.1002/j.1538-7305.1960.tb03959.x.

[63] HAGAN M. T., MENHAJ M. B. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*. 1994, 5(6), pp. 989–993, doi: 10.1109/72.329697.

[64] HANDS D., WILKINS M. A Study of the Impact of the Network Loss and Burst Size on Video Streaming Wuality and Acceptability. *Interactive Distributed Multimedia Systems and Telecommunication Services Workshop*. 1999.

[65] HARALICK R. M., SHANMUGAM K., DINSTEIN H. Textural Features for Image Classification. *Systems, Man and Cybernetics, IEEE Transactions on*. 1973, 3(6), pp. 610–621, doi: 10.1109/TSMC.1973.4309314.

[66] HASTIE T., TIBSHIRANI R., FRIEDMAN J. *The elements of Statistical Learning*. New York, USA: Springer-Verlag, 2001. Spring series in statistics series.

[67] HENDERSON W. Queueing networks with negative customers and negative queue lengths. *Journal of Applied Probability*. 1993, 30, pp. 931–942, doi: 10.2307/3214523.

[68] HENDERSON W., NORTHCOTE B., TAYLOR P. G. Geometric equilibrium distributions for queues with interactive batch departures. *Annals of Operations Research*. 1994, 48, pp. 493–511, doi: 10.1007/BF02033316.

[69] HENDERSON W., NORTHCOTE B., TAYLOR P. G. State-dependent signalling in queueing networks. *Advances in Applied Probability*. 1994, 26, pp. 436–455, doi: 10.2307/1427445.

[70] HUSSAIN K. F., MOUSSA G. S. Laser Intensity Vehicle Classification System Based on Random Neural Network. In: *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 1*, Kennesaw, Georgia: ACM, 2005, pp. 31–35. ACM-SE 43 series, doi: 10.1145/1167350.1167372.

[71] HYNDMAN R. *Time Series Data Library* [online]. Qlik Ltd., [viewed 2015-08-28]. Available from: https://datamarket.com/data/list/?q=cat:ecd%20provider:tsdl.

[72] KENDALL D. G. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*. 1953, 24(3), pp. 338–354, doi: 10.1214/aoms/1177728975.

[73] LIKAS A., STAFYLOPATIS A. High Capacity Associative Memory Based on the Random Neural Network Model. *International Journal of Pattern Recognition and Artificial Intelligence*. 1997, 10(8), pp. 919–937, doi: 10.1142/S0218001496000529.

[74] LIKAS A., STAFYLOPATIS A. Training the Random Neural Network using Quasi-Newton Methods. *Eur.J.Oper.Res.* 2000, 126, pp. 331–339, doi: 10.1016/S0377-2217(99)00482-8.

[75] LIKAS A., STAFYLOPATIS A. An Investigation of the Analogy between the Random Network and the Hopfield Network. In: *Proc. 6th International Symposium on Computer and Information Sciences*. Turkey: Antalya, 1991 [viewed 2015-08-28], pp. 849–857. Available from: http://www.cs.uoi.gr/~arly/papers/antalya.ps.gz.

[76] LIU P., GELENBE E. Recursive routing in the cognitive packet network. In: *Proceedings of 3rd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom'07)*, 2007, pp. 1–6, doi: 10.1109/tridentcom.2007.4444727.

[77] LU R., SHEN Y. Image Segmentation Based on Random Neural Network Model and Gabor Filters. In: *27th Annual International Conference of the Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005.* 2005, pp. 6464–6467, doi: 10.1109/IEMBS.2005.1615979.

[78] LUKOŠEVIČIUS M., JAEGER H. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*. 2009, pp. 127–149, doi: 10.1016/j.cosrev2009.03.005.

[79] MANEVITZ L., YOUSEF M. One-class Document Classification via Neural Networks. *Neurocomputing*. 2007, 70(7-9), pp. 1466–1481, doi: 10.1016/j.neucom.2006.05.013.

[80] MARQUARDT D. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Math.* 1963, pp. 431–441, doi: 10.1137/0111030.

[81] MARTENS J., SUTSKEVER I. Learning Recurrent Neural Networks with Hessian-Free Optimization. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. Bellevue, Washington, USA: ACM, 2011 [viewed 2015-08-28], pp. 1033–1040. Available from: http://www.icml-2011.org/papers/532_icmlpaper.pdf.

[82] MARTINELLI G., MASCIOLI F. F., PANELLA M., RIZZI A. Extended Random Neural Networks. In: *Neural Nets*, Berlin, Germany: Springer Berlin / Heidelberg, 2002, pp. 75–82. Lectures Notes in Computer Science series, doi: 10.1007/3-540-45808-5_7.

[83] MARTINS J., SUTSKEVER I., SWERSKY K. Estimating the Hessian by Back-propagation Curvature. In: *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*. Edinburgh, Scotland, GB: Omnipress, 2012 [viewed 2015-08-28], pp. 1783–1790. Available from: http://icml.cc/2012/papers/866.pdf.

[84] MCCULLOCH W., PITTS W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*. 1943, 5(4), pp. 115–133, doi: 10.1007/BF02478259.

[85] MINSKY M. L., PAPERT S. A. *Perceptrons*. Cambridge, MA: MIT Press, 1969.

[86] MOHAMED S., RUBINO G., VARELA M. A method for quantitative evaluation of audio quality over packet networks and its comparison with existing techniques. In: *Measurement of Speech and Audio Quality in Networks (MESAQIN)*. 2004 [viewed 2015-08-28]. Available from: http://goo.gl/SC2c0C.

[87] MOHAMED S., RUBINO G., VARELA M. Performance evaluation of real-time speech through a packet network: a random neural networks-based approach. *Performance Evaluation*. 2004, 57(2), pp. 141–161, doi: 10.1016/j.peva.2003.10.007.

[88] NAKAMA T. Theoretical Analysis of Batch and On-Line Training for Gradient Descent Learning in Neural Networks. *Neurocomputing*. 2009, 73, pp. 151–159, doi: 10.1016/j.neucom.2009.05.017.

[89] PASCANU R., MIKOLOV T., BENGIO Y. On the difficulty of training recurrent neural networks. In: Atlanta, Georgia, USA, 2013 [viewed 2015-08-28], pp. 37–48. Available from: http://jmlr.org/proceedings/papers/v28/pascanu13.pdf.

[90] PEKERGIN F. Combinatorial optimization by random neural network model - application to the independent set problem. In: *Proc. International Conference on Artificial Neural Networks (ICANN'92)*. Brighton, United Kingdom: North-Holland, Amsterdam, 1992 [viewed 2015-08-28], pp. 437–440. Available from: http://www.gbv.de/dms/tib-ub-hannover/128443588.pdf.

[91] PRESS W., TEUKOLSKY S., VETTERLING W., FLANNERY B. *Numerical Recipes in C*. 2nd ed. Cambridge, UK: Cambridge University Press, 1992.

[92] PRESS W. H., TEUKOLSKY S. A., VETTERLING W. T., FLANNERY B. P. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2002.

[93] ROSENFELD A., TROY E. *Visual Texture Analysis*. USA: Maryland Univ., College Park (USA). Computer Science Center, 1970. Technical report TR-70-116.

[94] RUBINO G. Quantifying the Quality of Audio and Video Transmissions over the Internet: the PSQA Approach. In: *Design and Operations of Communication Networks: A Review of Wired and Wireless Modelling and Management Challenges*. Imperial College Press, 2005. Edited by J. Barria series.

[95] RUBINO G., MOHAMED S. A Study of Real-time Packet Video Quality Using Random Neural Networks. *IEEE Transactions On Circuits and Systems for Video Technology*. 2002, 12(12), pp. 1071–1083, doi: 10.1109/TCSVT.2002.806808.

[96] RUBINO G., TIRILLY P., VARELA M. Evaluating users' satisfaction in packet networks using Random Neural Networks. In: *16th International Conference on Artificial Neural Networks (ICANN'06)*, Athens, Greece, 2006, doi: 10.1007/11840817_32.

[97] RUMELHART D., HINTON G., WILLIAMS R. J. Learning Internal Representations by Error Propagation. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986, pp. 318–362. Computational Models of Cognition and Perception series.

[98] RUMELHART D. E., HINTON G. E., MCCLELLAND J. L. A general framework for parallel distributed processing. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986, pp. 45–76. Computational Models of Cognition and Perception series.

[99] SAKELLARI G. The Cognitive Packet Network. *Computer Journal*. 2010, 53(3), pp. 268–279, doi: 10.1093/comjnl/bxp053.

[100] SCHUMACHER M., ROSSNER R., VACH W. Neural Networks and Logistic Regression: Part I. *Computational Statistics & Data Analysis*. 1996, 21, pp. 661–682, doi: 10.1016/0167-9473(95)00032-1.

[101] SCHWENK H., BENGIO Y. Boosting Neural Networks. *Neural Comput.* 2000, 12(8), pp. 1869–1887, doi: 10.1162/089976600300015178.

[102] STAFYLOPATIS A., LIKAS A. Pictorial Information Retrieval Using the Random Neural Network. *IEEE Transactions on Software Engineering*. 1992, 18(7), pp. 590–600, doi: 10.1109/32.148477.

[103] TEKE A., ATALAY V. Texture Classification and Retrieval Using the Random Neural Network Model. *Computational Management Science*. 2006, 3(3), pp. 193–205, doi: 10.1007/s10287-006-0012-1.

[104] TIMOTHEOU S. The Random Neural Network: A Survey. *The Computer Journal*. 2010, 53(3), pp. 251–267, doi: 10.1093/comjnl/bxp032.

[105] UKIL A. *Intelligent Systems & Signal Processing in Power Engineering*. Berlin Heidelberg: Springer, 2007. Power systems series.

[106] WILSON D. R., MARTINEZ T. R. The General Inefficiency of Batch Training for Gradient Descent Learning. *Neural Networks*. 2003, 16(10), pp. 1429–1451, doi: 10.1016/S0893-6080(03)00138-2.

[107] YAJNIK M., MOON S., KUROSE J. F., TOWSLEY D. F. Measurement and Modeling of the Temporal Dependence in Packet Loss. *Proccedings of IEEE INFOCOM'99*. 1993, pp. 345–352, doi: 10.1109/infcom.1999.749301.

[108] ZHONG Y. J., SUN D. C., WU J. J. Dynamical Random Neural Network Approach to a Problem of Optimal Resource Allocation. In: J. CABESTANY, A. PRIETO, F. SANDOVAL, eds. *Computational Intelligence and Bioinspired Systems*. Springer Berlin / Heidelberg, 2005, pp. 387–415. Lecture Notes in Computer Science series.