*LETTER* Communicated by Pik-Yin Lai

# Random Neural Networks with Synchronized Interactions

**Erol Gelenbe**
*e.gelenbe@imperial.ac.uk*
**Stelios Timotheou**
*stelios.timotheou@imperial.ac.uk*
*Intelligent Systems and Networks Group, Department of Electrical and Electronic Engineering, Imperial College, London SW7 2BT, U.K.*

**Large-scale distributed systems, such as natural neuronal and artificial systems, have many local interconnections, but they often also have the ability to propagate information very fast over relatively large distances. Mechanisms that enable such behavior include very long physical signaling paths and possibly saccades of synchronous behavior that may propagate across a network. This letter studies the modeling of such behaviors in neuronal networks and develops a related learning algorithm. This is done in the context of the random neural network (RNN), a probabilistic model with a well-developed mathematical theory, which was inspired by the apparently stochastic spiking behavior of certain natural neuronal systems. Thus, we develop an extension of the RNN to the case when synchronous interactions can occur, leading to synchronous firing by large ensembles of cells. We also present an $O(N^3)$ gradient descent learning algorithm for an $N$-cell recurrent network having both conventional excitatory-inhibitory interactions and synchronous interactions. Finally, the model and its learning algorithm are applied to a resource allocation problem that is NP-hard and requires fast approximate decisions.**

## 1 Introduction

Synchronized firing (SF) has been observed among cultured cortical neurons (Muramoto, Kobayashi, Nakanishi, Matsuda, & Kuroda, 1988; Robinson et al., 1993), and it is believed that it serves a prominent role in information processing functions of both sensory and motor systems (Konig, & Engel, 1995). Temporal firing synchrony is a result of functional coupling, which dynamically varies according to the internal state of the neural system and the stimuli, and it appears in both homogeneous and clustered neuronal networks (Segev, Shapira, Benveniste, & Ben-Jacob, 2001); it has been observed that under special population density conditions, a neuronal culture can self-organize into linked clusters (Segev, Benveniste, Shapira, & Ben-Jacob, 2003) to generate synchronous firing activity similar to the one observed in homogeneous networks (Mann-Metzer, & Yarom, 1999;

 

Segev et al., 2003). This behavior may also be related to the correlation in connectivity, which is usually measured in neuron cultures because it is difficult to identify the synaptic strength among neurons and, hence, determine the characteristic node connectivity (Jia, Sano, Lai,& Chan, 2004). The experimental observations of synchronized firing in cultured or sliced neuron cell ensembles may be bursts of firing resulting from the nonlinear dynamics of the neuronal interactions. Thus, in this letter, we develop a mathematical model of a spiked random network that can exhibit synchronized firing between cells, where one cell may trigger firing in another one, and cascades of such triggered firings can occur. Our model describes the triggering of firing between two cells and triggered firing by cascades of cells, including feedback loops in the cascades, so that lengthy bursts of firing can also be modeled. Thus, the model could be used to mimic the spike bursts that have been experimentally observed.

In this letter, we discuss an extension of the random neural network model (RNN), which incorporates the usual excitatory and inhibitory interactions between cells but also offers the possibility that cells synchronously act together on other cells, triggering successive firing instants in other cells.

The spiked RNN model was introduced in Gelenbe (1989a, 1989b, 1990a, 1990b) and shown to have a "product form" solution, which implies that the joint stationary probability distribution of the excitation state of all the cells in the network is the product of the stationary marginal probabilities of each cell. The RNN has been used to study oscillations in corticothalamic circuits in Gelenbe and Cramer (1998). Its gradient descent learning algorithm was derived in Gelenbe (1993b), and in Gelenbe, Mao, and Li (1999, 2004) it is shown that the RNN can be used to approximate continuous and bounded functions. A generalization of the RNN to represent multiple classes of signals was introduced in Gelenbe and Fourneau (1999), and applied in Atalay, Gelenbe, and Yalabik (1992) and Gelenbe and Hussain (2002) to the reproduction and learning of image textures. The RNN has been applied to several other image processing problems, such as image fusion (Bakircioglu, Gelenbe, & Koçak, 1998), image and video compression (Cramer and Gelenbe, 2000; Cramer, Gelenbe, & Bakircioglu, 1996; Gelenbe, Cramer, Sungur, & Gelenbe, 1996), and the detection of tumors in brain images (Gelenbe, Feng, & Krishnan, 1996) and also to the detection of unusual signals inside periodic sequences (Gelenbe, Harmanci, & Krolik, 1998), as well as in a variety of pattern recognition problems (Abdelbaki, Gelenbe, & Koçak, 2005; Gelenbe, & Koçak, 2000, 2004). It has also been used in several papers to obtain approximate solutions of $NP$-hard problems (Aguilar & Gelenbe, 1997; Gelenbe, 1992; Gelenbe, Ghanwani, & Srinivasan, 1997; Gelenbe, Koubi, & Pekergin, 1994). A model similar to the RNN has also been suggested for genetic algorithms (Gelenbe, 1997). The RNN has also been applied to control the routing of packets in computer networks (Gelenbe, Lent, & Nunez, 2004; Gelenbe, Lent, & Xu, 2001). Other related models can be found in Gelenbe (2007a, 2007b).

In addition to introducing the learning algorithm for the RNN with synchronized interactions and showing that it is of computational complexity $O(N^3)$ for an $N$-cell network, in this letter, we also present how the network can be used as a decision-making device for an NP-hard resource allocation problem. The approach here differs from previous work in this area in that we use the learning algorithm to train the network on a large number of similar instances of the optimization problem at hand, and then we use it on problem instances that it has not encountered before. The evaluation that we present in this letter shows that this approach can be very effective: the trained network decides about the resource allocation very quickly and solves it quite accurately, as we discuss in detail in section 5.

## 2 The Model

The RNN model with synchronized interactions is a network of $N$ cells that have independent and exponentially distributed firing times with firing rates $r_1, \ldots, r_N \geq 0$. The internal state of a neuron $i$ is the nonnegative variable $k_i(t)$. We say that the neuron is excited if $k_i(t) \geq 1$, and that it is quiescent or idle if $k_i(t) = 0$. A neuron may fire if it is excited, and when $i$ fires at time $t$, right after $t$, we have $k_i(t^+) = k_i(t) - 1$. The following events can occur at time $t$:

- A cell $i$ receives an excitatory spike from the outside world; this increases the internal state of the receiving neuron by 1. Excitatory spikes arrive at cell $i$ from the outside world according to a Poisson process of rate $\Lambda(i)$.
- A cell $i$ receives an inhibitory spike from the outside world at time $t$; if $k_i(t) > 0$, this decreases the value of the internal state of the receiving neuron by 1 and has no effect if $k_i(t) = 0$. Inhibitory spikes arrive at cell $i$ from the outside world according to a Poisson process of rate $\lambda(i)$.
- Cell $i$ fires at time $t$ if $k_i(t) > 0$, and the probability of this event is $r_i \Delta t + o(\Delta t)$ so that the cell's firing rate $r_i$ is exponentially distributed. If this happens, $k_i(t^+) = k_i(t) - 1$. The resulting spike will go to cell $j$ as an excitatory spike with probability $p^+(i, j)$ or as an inhibitory spike with probability $p^-(i, j)$, or the spike departs the network going to the outside world with probability $d(i)$, or it creates a synchronous interaction together with cell $j$ to affect some third cell $m$, with probability $Q(i, j, m)$.
- An excitatory or inhibitory spike arriving at a neuron $j$ from another neuron $i$ is treated by $j$ exactly the same way as if such spikes arrive from the outside world.
- If at time $t$ a spike from $i$ reaches $m$ to provoke a second-order effect on $j$, then the following happens: of course, $k_i(t^+) = k_i(t) - 1$, but

also $k_m(t^+) = k_m(t) - 1$ and $k_j(t^+) = k_j(t) + 1$ if $k_m(t) > 0$. However, if $k_m(t) = 0$, then $k_i(t^+) = k_i(t) - 1$.

Thus, synchronous interactions take the form of a joint excitation by cells $i, m$ on $j$ and can occur only if both cell $i$ and $m$ are excited. Note also that

$$d(i) = 1 - \sum_{j=1}^{N} \left[ p^+(i, j) + p^-(i, j) + \sum_{m=1}^{N} Q(i, j, m) \right], \tag{2.1}$$

where $Q(i, j, m)$ is the probability that when $i$ fires, then if $j$ is excited, it will also fire immediately, with an excitatory spike being sent to cell $m$. This synchronous behavior can be extended to an arbitrary number of cells when we have a sequence of cells $j_1, \ldots, j_{n+1}, j_{n+2}$ such that $Q(j_i, j_{i+1}, j_{i+2}) = 1$ for $1 \leq i \leq n$. In this case, if cells $j_1$ and $j_2$ are excited, then eventually all the cells $j_1, \ldots, j_{n+1}, j_{n+2}$ will fire. Thus, the generalized RNN model we have described can be used to model some quite general forms of synchronized firing.

## 3 Steady-State Solution

Let the state of the network be $\underline{k}(t) = [k_1(t), k_2(t), \ldots, k_N(t)]$. With the previous assumptions, the system state is a continuous time Markov chain, and the probability distribution of the system state $\{\underline{k}(t) : t \geq 0\}$ satisfies a set of Chapman-Kolmogorov equations. Let us use the following vectors to denote specific values of the network state, where all of these vectors' values must be nonnegative:

$$\begin{aligned}
\underline{k} &= [k_1, \ldots, k_N] \\
\underline{k}_i^+ &= [k_1, \ldots, k_i + 1, \ldots, k_N] \\
\underline{k}_i^- &= [k_1, \ldots, k_i - 1, \ldots, k_N] \\
\underline{k}_{ij}^{+-} &= [k_1, \ldots, k_i + 1, \ldots, k_j - 1, \ldots, k_N] \\
\underline{k}_{ij}^{++} &= [k_1, \ldots, k_i + 1, \ldots, k_j + 1, \ldots, k_N] \\
\underline{k}_{ijm}^{++-} &= [k_1, \ldots, k_i + 1, \ldots, k_j + 1, \ldots, k_m - 1, \ldots, k_N].
\end{aligned}$$

If the steady-state distribution $\pi(\underline{k}) = \lim_{t \to \infty} P[\underline{k}(t) = \underline{k}]$ exists, it satisfies the Chapman-Kolmogorov equations given in the steady-state:

$$\pi(\underline{k}) \sum_{i=1}^{N} \left[ \Lambda(i) + (\lambda(i) + r_i) \mathbf{1}_{\{k_i > 0\}} \right] =$$

$$\sum_{i=1}^{N} \left\{ \pi\left(\underline{k}_i^+\right) r_i d(i) + \pi\left(\underline{k}_i^-\right) \Lambda(i) \mathbf{1}_{\{k_i > 0\}} + \pi\left(\underline{k}_i^+\right) \lambda(i) \right.$$

$$+\sum_{j=1}^{N}\Bigg[\pi(\underline{k}_{ij}^{+-})r_i\,p^+(i,\,j)\mathbf{1}_{\{k_j>0\}}+\sum_{m=1}^{N}\pi(\underline{k}_i^{+})r_i\,Q(i,\,j,\,m)\mathbf{1}_{\{k_j=0\}}$$
$$+\pi(\underline{k}_{ij}^{++})r_i\,p^-(i,\,j)+\pi(\underline{k}_i^{+})r_i\,p^-(i,\,j)\mathbf{1}_{\{k_j=0\}}$$
$$+\sum_{m=1}^{N}\pi(\underline{k}_{ijm}^{++-})r_i\,Q(i,\,j,\,m)\mathbf{1}_{\{k_m>0\}}\Bigg]\Bigg\}, \tag{3.1}$$

where $\mathbf{1}_{\{Y\}}$ is equal to 1 if $Y$ is true and 0 otherwise.

The following is an application of on earlier result, shown in Gelenbe (1993a).

**Theorem 1.** *Let $\lambda^-(i)$ and $\lambda^+(i)$, $i = 1, \ldots, N$ be given by the following system of equations*

$$\lambda^-(i)=\lambda(i)+\sum_{j=1}^{N}r_jq_j\left[p^-(j,i)+\sum_{m=1}^{N}Q(j,i,m)\right] \tag{3.2}$$

$$\lambda^+(i)=\sum_{j=1}^{N}r_jq_j\,p^+(j,i)+\sum_{j=1}^{N}\sum_{m=1}^{N}q_jq_mr_j\,Q(j,m,i)+\Lambda(i), \tag{3.3}$$

*where*

$$q_i=\lambda^+(i)/\left(r_i+\lambda^-(i)\right). \tag{3.4}$$

*If a unique nonnegative solution $\{\lambda^-(i), \lambda^+(i)\}$ exists for the nonlinear system of equations, 3.2 to 3.4, such that $q_i < 1\ \forall i$, then*

$$\pi\left(\underline{k}\right)=\prod_{i=1}^{N}(1-q_i)\,q_i^{k_i}. \tag{3.5}$$

Thus, whenever a solution can be found to equations 3.2 to 3.4, such that all the $q_i < 1$, then the network's steady state has the simple product form, equation 3.5. The condition $q_i < 1$ can be viewed as a stability condition that guarantees that the excitation level of each neuron remains finite with probability one. Note also that the average excitation level of neuron $i$ in steady state is $q_i/(1-q_i)$.

We now introduce notation similar to that used in Gelenbe (1993b), where we replace the firing rates $r_i$ and the probabilities $p^+(i,\,j)$, $p^-(i,\,j)$, and $Q(i,\,j,\,l)$ by weights that in this model represent the rates at which the cells

or neurons interact. Let:

$$w^+(i,\, j) = r_i\, p^+(i,\, j), \tag{3.6}$$

$$w^-(i,\, j) = r_i\, p^-(i,\, j), \tag{3.7}$$

and

$$w(i,\, j, l) = r_i\, Q(i,\, j, l). \tag{3.8}$$

As a result we can write

$$r_i = \frac{\sum_{j=1}^{N} \left[ w^+(i,\, j) + w^-(i,\, j) + \sum_{m=1}^{N} w(i,\, j, m) \right]}{1 - d(i)}. \tag{3.9}$$

The denominator of $q_i$ can be written as

$$D(i) = r_i + \sum_{j=1}^{N} q_j \left[ w^-(j, i) + \sum_{m=1}^{N} w(j, i, m) \right] + \lambda(i), \tag{3.10}$$

while its numerator becomes

$$N(i) = \sum_{j=1}^{N} q_j\, w^+(j, i) + \sum_{j=1}^{N} \sum_{m=1}^{N} q_j q_m w(j, m, i) + \Lambda(i), \tag{3.11}$$

so that $q_i = N(i)/D(i)$. The results summarized in this section will now be used to design an efficient learning algorithm for this network with second-order effects.

## 4 Gradient Descent Learning of Computational Complexity $O(N^3)$

Gradient-descent-based algorithms in neural networks are used to select the weights of the network so that if the network is presented with a given input $\mathbb{X}$, the network's output is a very good match for a desired output vector $\underline{y}$. In our case, the input vector $\mathbb{X}$ will be the vector of the external excitatory and inhibitory arrival rates $(\Lambda(1), \ldots, \Lambda(n))$ and $(\lambda(1), \ldots, \lambda(N))$. The output vector will correspond to the steady-state values of the network, for instance, a vector of the values taken by the $N$ probabilities $(q_1, \ldots, q_N)$, which result from applying the input vector to the network. The gradient descent algorithm is usually provided with a set of input and output vectors that are used to adjust the network weights so that the difference between the given and obtained outputs is minimized over the set of input vectors.

Now consider a set of $K$ input-output pairs $(\mathbb{X}, \mathbb{Y})$, with inputs given by $\mathbb{X} = [\mathbb{X}_1, \mathbb{X}_2, \ldots, \mathbb{X}_K]^T$ and $\mathbb{X}_k = [\underline{\Lambda}_k, \underline{\lambda}_k]^T$.

The vectors $\underline{\Lambda}_k = [\Lambda_k(1), \ldots, \Lambda_k(N)]$ and $\underline{\lambda}_k = [\lambda_k(1), \ldots, \lambda_k(N)]$ are obviously the external arrival rates of customers and signals entering the $N$ neurons, respectively. The $K$ desired outputs are $\mathbb{Y} = [\underline{y}_1, \ldots, \underline{y}_K]^T$ with $\underline{y}_k = [y_{k1}, \ldots, y_{kN}]$, where $y_{ki} \in [0, 1]$ are the desired outputs of the $i$th neuron for the $k$th training set.

A learning algorithm computes values of the network weights so as to find a local minimum of a cost or error function such as

$$E_k = \frac{1}{2} \sum_{i=1}^{N} c_i \left( f_i(q_i) - y_i \right)^2, \quad c_i \geq 0, \tag{4.1}$$

where $f_i(q_i)$ is a differentiable function of $q_i$ associated with neuron $i$. If for some reason we want a particular neuron $i$ not to be considered as an output, then we can set $c_i = 0$. Note that more general forms of $E_k$ can be selected without significant changes in the algorithm we will consider as long as $E_k$ is nonnegative, differentiable in all of the $q_i$, and has at least one minimum for the set of all parameter values $w^+(i, j), w^-(i, j) \geq 0$, and $w(i, j, l) \geq 0$.

**4.1 Restricting the Optimization to $w(i, j, l) = w^-(i, j)a(j, l)$.** In general, we can select the $w(i, j, l)$ in an arbitrary manner as long as $w(i, j, l) \geq 0$, and $w^+(i, j), w^-(i, j) \geq 0$. However, we see that $w(i, j, l)$ in fact acts as an inhibitory term from $i$ to $j$, followed by an excitatory term from $j$ to $l$. Thus, we propose to simplify the computation involved in seeking a minimum of the cost function (4.1) by writing

$$w(i, j, l) = w^-(i, j)a(j, l), \text{ for all } 1 \leq i, j, l \leq N, \tag{4.2}$$

where $a(j, l) \geq 0$.

We will therefore design a gradient descent algorithm to obtain the unknown parameters of the network—the matrices $\mathbb{W}^+ = \{w^+(i, j)\}$, $\mathbb{W}^- = \{w^-(i, j)\}$, and $\mathbb{A} = \{a(i, j)\}$ for $i, j = 1, \ldots, N$—in order to minimize the cost function.

In the sequel we will use the generic term $w(u, v)$ to represent either $w(u, v) = w^+(u, v)$ or $w(u, v) = w^-(u, v)$ or $w(u, v) = a(u, v)$. The weights are updated using the gradient descent rule so that for the $k$th input-output pair, the $n$th computational step is

$$w_{k,n+1}(u, v) = w_{k,n}(u, v) - \eta \left. \frac{\partial E_k}{\partial w(u, v)} \right|_{w(u,v)=w_n(u,v),\ \mathbb{X}=\mathbb{X}_k,\ \underline{y}=\underline{y}_k}, \tag{4.3}$$

where $n$ denotes the update step, $\eta > 0$ is known as the learning rate, and the partial derivative of the cost function is computed with the $n$th updated values of the weights. Clearly,

$$\frac{\partial E_k}{\partial w(u,v)} = \sum_{i=1}^{N} c_i \left(f_i\left(q_{ik}\right) - y_{ik}\right) \frac{\partial f_i(q_i)}{\partial q_i} \frac{\partial q_i}{\partial w(u,v)} \Bigg|_{w(u,v)=w_{k-1}(u,v),\ q_i=q_{ik-1}}. \tag{4.4}$$

By taking the derivative of $q_i = N(i)/D(i)$ with respect to the generic variable $w(u, v)$, one obtains after some calculations each of the terms of interest. Writing the vector $\underline{q} = [q_1, q_2, \ldots, q_N]$ and using the $N \times N$ matrix,

$$\mathbb{W} = \frac{1}{D(j)} \cdot \left\{ w^+(i,\, j) - q_j w^-(i,\, j) + \sum_{m=1}^{N} q_m w^-(i,m) a(m,\, j) \right.$$
$$\left. + a(i,\, j) \sum_{m=1}^{N} q_m w^-(m,i) - q_j w^-(i,\, j) \sum_{m=1}^{N} a(j,m) \right\} \; i,\, j = 1, 2, \ldots, N, \tag{4.5}$$

we obtain

$$\frac{\partial \underline{q}}{\partial w^+(u,v)} = \frac{\partial \underline{q}}{\partial w^+(u,v)} \mathbb{W} + \underline{\gamma}^+(u,v) \tag{4.6}$$

$$\frac{\partial \underline{q}}{\partial w^-(u,v)} = \frac{\partial \underline{q}}{\partial w^-(u,v)} \mathbb{W} + \underline{\gamma}^-(u,v) \tag{4.7}$$

$$\frac{\partial \underline{q}}{\partial a(u,v)} = \frac{\partial \underline{q}}{\partial a(u,v)} \mathbb{W} + \underline{\gamma}^*(u,v), \tag{4.8}$$

where we have used

$$\underline{\gamma}^+(u,v) = [\gamma_1^+(u,v), \gamma_2^+(u,v), \ldots, \gamma_N^+(u,v)], \tag{4.9}$$

$$\underline{\gamma}^-(u,v) = [\gamma_1^-(u,v), \gamma_2^-(u,v), \ldots, \gamma_N^-(u,v)], \tag{4.10}$$

and

$$\underline{\gamma}^*(u,v) = [\gamma_1^*(u,v), \gamma_2^*(u,v), ..., \gamma_N^*(u,v)], \tag{4.11}$$

which are given by:

$$\gamma_i^+(u,v) = \frac{1}{D(i)} \cdot \begin{cases} q_u - q_u/(1-d(i)) & u=i, v=i \\ -q_u/(1-d(i)) & u=i, v\neq i \\ q_u & u\neq i, v=i \\ 0 & u\neq i, v\neq i \end{cases} \tag{4.12}$$

$$\gamma_i^-(u,v) = \frac{1}{D(i)} \cdot$$

$$\begin{cases} q_u q_v\big[a(v,i) - 1 - \sum_{m=1}^N a(v,m)\big] & v=i, u=i \\ \quad -q_u\big[1+\sum_{m=1}^N a(v,m)\big](1-d(i))^{-1} & \\ q_u q_v\big[a(v,i) - 1 - \sum_{m=1}^N a(v,m)\big] & v=i, u\neq i \\ q_u q_v a(v,i) - q_u\big[1+\sum_{m=1}^N a(v,m)\big](1-d(i))^{-1} & v\neq i, u=i \\ q_u q_v a(v,i) & v\neq i, u\neq i \end{cases} \tag{4.13}$$

$$\gamma_i^*(u,v) = \frac{1}{D(i)} \cdot$$

$$\begin{cases} -q_i w^-(i,u)(1-d(i))^{-1} & v=i, u=i \\ -q_i w^-(i,u)(1-d(i))^{-1} + q_u \sum_{j=1}^N q_j w^-(j,u) & v=i, u\neq i \\ -q_i w^-(i,u)(1-d(i))^{-1} - q_u \sum_{j=1}^N q_j w^-(j,u) & v\neq i,\ u=i \\ -q_i w^-(i,u)(1-d(i))^{-1} & v\neq i,\ u\neq i \end{cases}. \tag{4.14}$$

Notice that equations 4.6 to 4.8 can also be written as

$$\frac{\partial \underline{q}}{\partial w^+(u,v)} = \underline{\gamma}^+(u,v)\,(\mathbb{I}-\mathbb{W})^{-1} \tag{4.15}$$

$$\frac{\partial \underline{q}}{\partial w^-(u,v)} = \underline{\gamma}^-(u,v)\,(\mathbb{I}-\mathbb{W})^{-1} \tag{4.16}$$

$$\frac{\partial \underline{q}}{\partial a(u,v)} = \underline{\gamma}^*(u,v)\,(\mathbb{I}-\mathbb{W})^{-1}, \tag{4.17}$$

where $\mathbb{I}$ is the $N \times N$ identity matrix. We now summarize the steps of the learning algorithm:

1. Initialize the matrices $\mathbb{W}^+$, $\mathbb{W}^-$, and $\mathbb{A}$, and choose a value for $\eta$. The larger the value of $\eta$, the greater the change in the weight update in one step.
2. Set the input values for $\mathbb{X}_k = [\underline{\Lambda}_k, \underline{\lambda}_k]^T$ and $\underline{y}_k$ for a particular $k$.

3. Solve the system of the $N$ nonlinear equations 3.2 to 3.4 based on the above values.
4. Solve the three systems of the $N$ linear equations 4.15 to 4.17 using the values of $q_{ik}$ obtained. The complexity of solving these systems is $O(N^3)$, because of the matrix inversion operation or $O(mN^2)$ if an $m$-step relaxation method is adopted.
5. Using the results from steps 3 and 4, update the matrices $\mathbb{W}^+ = \{w^+(i, j)\}, \mathbb{W}^- = \{w^-(i, j)\}$ and $\mathbb{A} = \{a(i, j)\}$ using equations 4.3 and 4.4. If during some update a particular parameter does not satisfy the constraint that they must be nonnegative, then either the particular update can be repeated with a smaller value of $\eta$ or the particular parameter can be set to the closest values within the constraints.

## 5 Fast Optimization in a Resource Allocation Problem

When a limited set of resources needs to be allocated simultaneously to a set of needs, one either formulates an optimization problem, which is then solved using specific and often time-consuming algorithms, or one resorts to some heuristic technique. When the problem needs to be solved rapidly, with limited computational resources and preferably in real time, then heuristic solutions are the approach of choice. Thus, the application presented in this section uses the RNN with synchronized interactions to compute a fast approximate decision for a resource allocation that arises in emergency management for a problem whose exact solution would require an impractically large computation time.

The approach taken is to train the RNN with the learning algorithm derived in this letter using numerous instances of the optimization problem, with exact solutions that are obtained offline. The trained RNN is then evaluated by testing it with randomly generated instances of the optimization problem. This evaluation shows (1) that the RNN with triggered interactions generally provides better (lower-cost) solutions than one where the triggered interactions are not allowed and (2) this approach yields results that are quite close to the optimum values.

In the emergency problem considered, $N_L$ incidents occur simultaneously at different locations with $I_j$ people injured at incident $j$. $N_U$ emergency units or ambulances (say) are spatially distributed before the time of the incident, with unit $i$ being able to collect $c_i > 0$ injured and having response time to incident $j$ given by $T_{ij} > 0$. We assume that only one unit can be allocated to one incident and that the initial locations of the units and the locations of incidents are fixed. If the capacity of the ambulances is sufficient to collect all the injured, then our goal is not only to collect the injured but also to minimize the average response time of the ambulances.

Let us introduce the binary decision variables $x_{ij}$:

$$x_{ij} = \begin{cases} 1 \text{ if unit } i \text{ is allocated to incident } j \\ 0 \text{ if unit } i \text{ is not allocated to incident } j \end{cases}. \tag{5.1}$$

The optimization problem is then

$$\min \ f(\underline{x}) = \sum_{i=1}^{N_U} \sum_{j=1}^{N_L} T_{ij} x_{ij}, \tag{5.2}$$

subject to

$$\sum_{j=1}^{N_L} x_{ij} = 1 \quad \forall i \tag{5.3}$$

$$\sum_{i=1}^{N_U} c_i x_{ij} \geq I_j \quad \forall j \tag{5.4}$$

$$x_{ij} \epsilon \{0, 1\} \qquad \forall i, \ j \tag{5.5}$$

The constraint 5.3 indicates that an emergency unit must be allocated to exactly one incident, while equation 5.4 expresses the fact that the total capacity of the units allocated to an incident must be at least equal to the number of people injured there. The above problem is known to be NP-hard.

To map the problem to a supervised learning context, we have to select parameters $I_j$ that represent the inputs to the decision problem, while the outputs are the $x_{ij}$. Because $I_j \geq 0 \ \forall j$, in the RNN they will be represented by the parameters $\Lambda(j)$ of the input neurons. The output variables are represented by the use of two neurons: a "positive" neuron and a negative neuron. If $x_{ij} = 1$, then the excitation level of the corresponding positive output neuron is high (close to 1), while the excitation level of the negative output neuron is low (close to 0). If $x_{ij} = 0$, then the excitation levels of the corresponding neurons are antisymmetric to the first case.

We will evaluate two distinct neural network architectures, both of which contain an input and an output layer, with the output layer having $N_{out} = 2 * N_U * N_L$ neurons. The input layer in the first case contains $N_L$ neurons, while in the second case, a positive and a negative neuron for each input are used. The key difference between the two architectures is that the first one is fully connected in terms of the $\mathbb{W}^+$ and $\mathbb{W}^-$ weight matrices, while in the second, each neuron excites output neurons of the same type and inhibits output neurons of the opposite type. Figure 1 presents the second

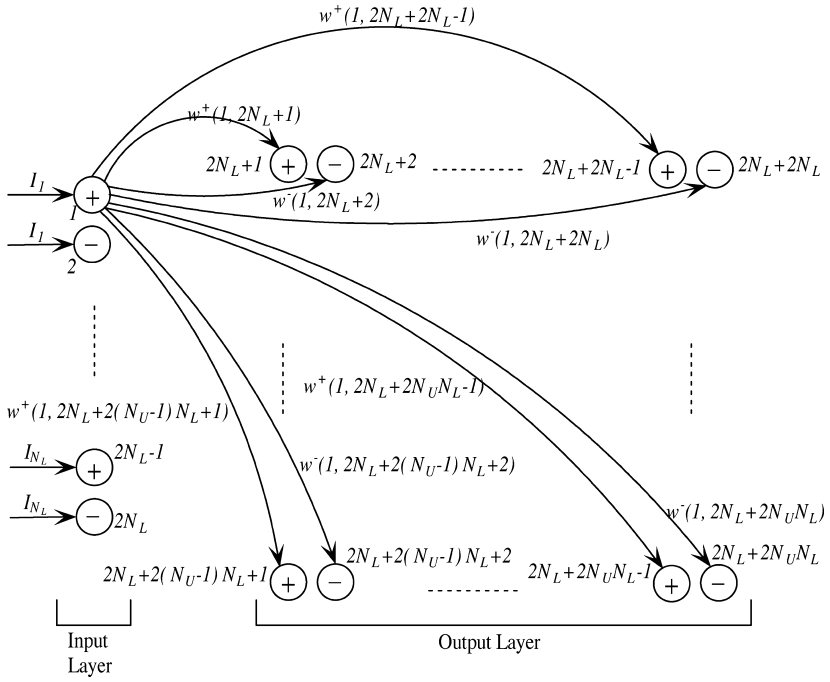


Figure 1: Second neural network architecture.

architecture showing the excitation and inhibition weights of a positive neuron.

The synchronized firing weights $\mathbb{A}$ are selected in four different ways:

**A1:** All the weights of the $\mathbb{A}$ matrix are used in learning.

**A2:** To deal with constraint 5.3, the synchronized weights corresponding to a decision variable $x_{ij}$, $a(ij, im)$ $i = 1, \ldots, N_U$, $j, m = 1, \ldots, N_L$ are high when the neurons are of different types and low if they are of the same type. The other synchronized weights are kept to zeros: $a(ij, lm) = 0$ $i, l = 1, \ldots, N_U$ $l \neq i$, $j, m = 1, \ldots, N_L$.

**A3:** The same as approach A2 but $a(ij, lm) \neq 0$ $i, l = 1, \ldots, N_U$ $l \neq i$, $j, m = 1, \ldots, N_L$, so these weights must be learned as well.

**A4:** The synchronized weights are all zero—$\mathbb{A} = 0$.

In the learning procedure, the weights are updated after processing each input pattern (i.e., an instance of the decision problem), where a pattern consists of input values and the corresponding output decision variables obtained from an "exact" solution of this instance of the decision problem.

The patterns are presented to the network at random, and after all patterns are processed, the whole procedure is repeated until convergence.

**5.1 Evaluating the RNN as a Decision Tool.** We first generated at random 200 training instances for different numbers of emergency units and for the locations of incidents. The remaining parameters have been chosen at random, with $T_{ij} \epsilon \mathbb{R}$ and $c_i \epsilon \mathbb{Z}$ being uniformly distributed in the intervals 0–1 and 1–4, respectively. For each of the training patterns, the $I_j \epsilon \mathbb{Z}$ are also uniformly distributed in the interval $0.5 * c_t/N_L, c_t/N_L$, where $c_t = \sum_i (c_i)$ is the total capacity of all the emergency units.

We have performed experiments with the following numbers of emergency units and incidents: $N_U = 5, 10, 15, 20$ and $N_L = 3, 5$. Among the test cases considered, we chose only those whose required capacity was within the total available capacity of the ambulances. The optimum solution in each case was then obtained accurately by solving the combinatorial optimization problem in Matlab using the function *bintprog*.

Testing after training was performed using a distinct but similarly generated set of 200 test cases so that the training and testing were disjoint, but with the same probability distributions for all parameters. Eight network architectures were used:

**Case 1:** Network architecture 1 with approach A1
**Case 2:** Network architecture 1 with approach A2
**Case 3:** Network architecture 1 with approach A3
**Case 4:** Network architecture 1 with approach A4
**Case 5:** Network architecture 2 with approach A1
**Case 6:** Network architecture 2 with approach A2
**Case 7:** Network architecture 2 with approach A3
**Case 8:** Network architecture 2 with approach A4

The results were evaluated on the basis of the following metrics:

- The percentage of instances that were solved so that all of the injured were evacuated. The results are shown in Figure 2.
- The percentage of people collected averaged over all the instances, summarized in Figure 3.
- The metric that evaluates the closeness to the optimality of the solution, given by the average of the ratio $f_{NN}(\underline{x})/f_{opt}(\underline{x})$, taken over all of the solutions where the ambulances assigned cover the casualties, which is summarized in Figure 4.

Based on these results, we can say that:

- Cases 1, 2, 3, and 4 are equally effective, and clearly better than the other cases, in obtaining the highest percentage of solutions where

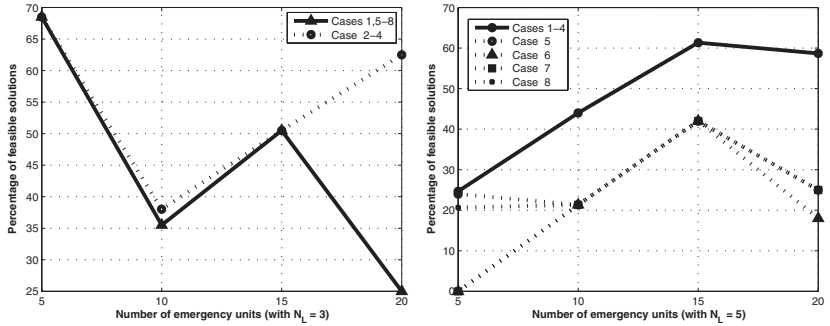


Figure 2: Percentage of solutions in which all casualties are evacuated. These solutions are called "feasible" in the graphs, for want of a better term.

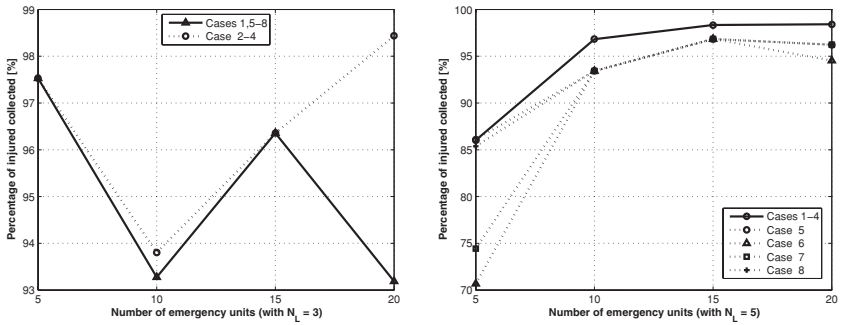


Figure 3: Percentage of casualties that are collected.

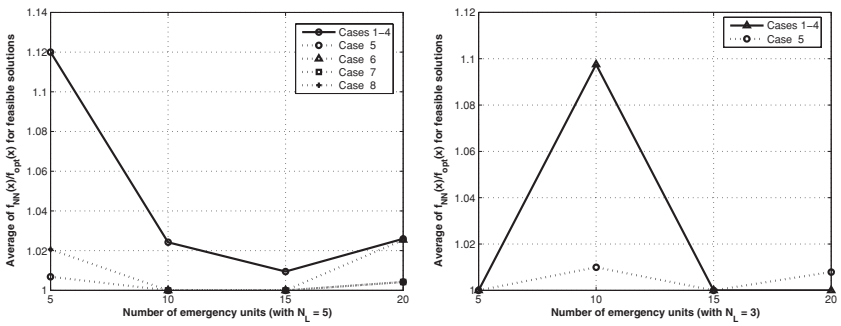


Figure 4: Average of $f_{NN}(x)/f_{opt}(x)$ for the solutions where the units are able to remove all the casualties (i.e., the "feasible" ones).

all casualties are evacuated and collecting the highest percentage of injured, except for the set of instances with $N_U = 20$ and $N_L = 3$.
- With respect to the ratio $f_{NN}(\underline{x})/f_{opt}(\underline{x})$, cases 5, 6, 7, and 8 (using the second architecture) are within a few percent of the optimum, while all results obtained are basically within 10% of the optimum.

Thus, the first architecture is better at finding a greater proportion of solutions where all casualties are evacuated, while the second is better at getting close to the optimum. In addition, among the cases studied the most successful are those when all the synchronized interaction weights are used (cases 1 and 5).

## 6 Conclusions

In this letter, we have developed a model of spiked neural networks with synchronized firing and then derived a gradient descent learning algorithm for the recurrent network. We showed that the learning algorithm is of complexity $O(N^3)$ for an $N$-node network and then applied it to solve a class of optimization problems that arise in emergency management.

The practical application that we consider is that of dispatching a set of vehicles or "emergency units" to a given set of locations that need these units, with knowledge of the number of "victims" at each location, the capacity of each emergency unit to treat victims, and knowledge of the time it takes the units to reach the locations where they are needed. The purpose is to evacuate as many victims as possible in the shortest possible time. We first formulate this problem in mathematical terms and train the RNN based on instances of the problem for which the optimal solution is known. The performance of the trained RNN is then tested on a set of randomly chosen instances. Our results show that the RNN will in this manner provide quasi-optimal solutions.

## Acknowledgments

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project and is jointly funded by a BAE Systems and EPSRC (UK Engineering and Physical Research Council) strategic partnership under grant no. EP/C548051/1.

## References

Abdelbaki, H., Gelenbe, E., & Koçak, T. (2005). Neural algorithms and energy measures for EMI based mine detection. *J. Diff. Equations and Dynamical Systems*, *13*(1–2), 63–86.

Aguilar, J., & Gelenbe, E. (1997). Task assignment and transaction clustering heuristics for distributed systems. *Information Sciences: Informatics and Computer Science*, *97*(1 & 2), 199–221.

Atalay, V., Gelenbe, E., & Yalabik, N. (1992). The random neural network model for texture generation. *International Journal of Pattern Recognition and Artificial Intelligence*, *6*(1), 131–141.

Bakircioglu, H., Gelenbe, E., & Koçak, T. (1998). Image processing with the random neural network model. *ELEKTRIK*, *5*(1), 65–77.

Cramer, C., & Gelenbe, E. (2000). Video quality and traffic QoS in learning-based subsampled and receiver-interpolated video sequences. *IEEE Journal on Selected Areas in Communications*, *18*(2), 150–167.

Cramer, C., Gelenbe, E., & Bakircioglu, H. (1996, October). Low bit rate video compression with neural networks and temporal subsampling. *Proceedings of the IEEE*, *84*(10), 1529–1543.

Gelenbe, E.(1989a). Random neural networks with positive and negative signals and product form solution. *Neural Computation*, *1*(4), 502–510.

Gelenbe, E. (1989b). Réseaux stochastiques ouverts avec clients négatifs et positifs, et réseaux neuronaux. *Comptes-Rendus Acad. Sci. Paris II*, *309*, 979–982.

Gelenbe, E. (1990a). Réseaux neuronaux aléatoires stables. *Comptes-Rendus Acad. Sci. Paris II*, *310*, 177–180.

Gelenbe, E. (1990b). Stability of the random neural network. *Neural Computation*, *2*(2), 239–247.

Gelenbe, E. (1992). Une généralisation probabiliste du problème sat. *Comptes-Rendus Acad. Sci. Paris II*, *313*, 339–342.

Gelenbe, E. (1993a). G-networks with triggered customer movement. *Journal of Applied Probability*, *30*(3), 742–748.

Gelenbe, E. (1993b). Learning in the recurrent random network. *Neural Computation*, *5*, 154–164.

Gelenbe, E. (1997). A class of genetic algorithms with analytical solution. *Robotics and Autonomous Systems*, *22*, 59–64.

Gelenbe, E. (2007a). Steady-state solution of probabilistic gene regulatory networks. *Physical Review E*, *76*(1), 031903.

Gelenbe, E. (2007b). Dealing with software viruses: A biological paradigm. *Information Security Technical Reports*, *12*, 242–250.

Gelenbe, E., & Cramer, C. (1998). Oscillatory corticothalamic response to somatosensory input. *Biosystems*, *48*(1–3), 67–75.

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

Gelenbe, E., Feng, T., & Krishnan, K. R. R. (1996). Neural network methods for volumetric magnetic resonance imaging of the human brain. *Proceedings of the IEEE*, *84*(1), 1488–1496.

Gelenbe, E., & Fourneau, J. M. (1999). Random neural networks with multiple classes of signals. *Neural Computation*, *11*(4), 953–963.

Gelenbe, E., Ghanwani, A., & Srinivasan, V. (1997). Improved neural heuristics for multicast routing. *IEEE Journal of Selected Areas of Communications*, *15*(2), 147–155.

Gelenbe, E., Harmanci, K., & Krolik, J. (1998). Learning neural networks for detection and classification of synchronous recurrent transient signals. *Signal Processing*, *64*(3), 233–247.

Gelenbe, E., & Hussain, K. (2002). Learning in the multiple class random neural network. *IEEE Trans. on Neural Networks*, *13*(6), 1257–1267.

Gelenbe, E., & Koçak, T. (2000). Area-based results for mine detection. *IEEE Trans. on Geoscience and Remote Sensing*, *38*(1), 1–14.

Gelenbe, E., & Koçak, T. (2004). Wafer surface reconstruction from top-down scanning electron microscope images. *Microlectronic Engineering*, *75*, 216–233.

Gelenbe, E., Koubi, V., & Pekergin, F. (1994). Dynamical random neural approach to the traveling salesman problem. *ELEKTRIK*, *2*(2), 1–10.

Gelenbe, E., Lent, R., & Nunez, A. (2004). Self-aware networks and QoS. *Proceedings of the IEEE*, *92*(9), 1478–1489.

Gelenbe, E., Lent, R., & Xu, Z. (2001). Measurement and performance of a cognitive packet network. *Computer Networks*, *37*, 691–791.

Gelenbe, E., Mao, Z.-H., & Li, Y.-D. (1999). Function approximation with spiked random networks. *IEEE Trans. on Neural Networks*, *10*(1), 3–9.

Gelenbe, E., Mao, Z.-H., & Li, Y.-D. (2004). Function approximation by random neural networks with a bounded number of layers. *J. Differential Equations and Dynamical Systems*, *12*(1 & 2), 143–170.

Jia, L., Sano, M., Lai, P.-Y., & Chan, C. (2004). Connectivities and synchronous firing in cortical neuronal networks. *Physical Review Letters*, *93*(8), 088101-1–088101-4.

Konig, P., & Engel, A. K. (1995). Correlated firing in sensory-motor systems. *Current Opinion in Neurobiology*, *5*(4), 511–519.

Mann-Metzer, P., & Yarom, Y. (1999). Electrotonic coupling interacts with intrinsic properties to generate synchronized activity in cerebellar networks of inhibitory interneurons. *J. Neuroscience*, *19*(9), 3298–3306.

Muramoto, K., Kobayashi, K., Nakanishi, S., Matsuda, Y., & Kuroda, Y. (1988). Functional Synapse formation between cultured neurons of rat cerebral cortex. *Proc. Japan Acad. Ser. B*, *64*, 319–322.

Robinson, H. P., Kawahara, M., Jimbo, Y., Torimitsu, K., Kuroda, Y., & Kawana, A. (1993). Periodic synchronized bursting and intracellular calcium transients elicited by low magnesium in cultured cortical neurons. *J. Neurophysiology*, *70*(4), 1606–1616.

Segev, R., Benveniste, M., Shapira, Y., & Ben-Jacob, E. (2003). Formation of electrically active clusterized neural networks. *Physical Review Letters*, *90*(16), 168101.

Segev, R., Shapira, Y., Benveniste, M., & Ben-Jacob, E. (2001). Observations and modeling of synchronized burst in two-dimensional neural networks. *Physical Review E*, *64*(1), 011920.

Received April 16, 2007; accepted December 17, 2007.