Fast Nonnegative Least Squares Learning in the Random Neural Network

Stelios Timotheou
KIOS Research Center for Intelligent Systems and Networks
University of Cyprus, Cyprus
timotheou.stelios@ucy.ac.cy

Abstract

The random neural network is a biologically inspired neural model where neurons interact by probabilistically exchanging positive and negative unit-amplitude signals that has superior learning capabilities compared to other artificial neural networks. This paper considers nonnegative least squares supervised learning in this context, and develops an approach that achieves fast execution and excellent learning capacity. This speedup is a result of significant enhancements in the solution of the nonnegative least squares problem which regard (a) the development of analytical expressions for the evaluation of the gradient and objective functions and (b) a novel limited-memory quasi-Newton solution algorithm. Simulation results in the context of optimizing the performance of a disaster management problem using supervised learning verify the efficiency of the approach, achieving two orders of magnitude execution speedup and improved solution quality compared to state-of-the-art algorithms.

Keywords: random neural network, supervised learning, nonnegative least-squares, limited-memory, quasi-Network, optimization.

I. Introduction

The Random Neural Network (RNN) is a neural network model inspired by the spiking behaviour of biophysical neurons [1], [2]. When a neuron is excited, it transmits unit amplitude signals called *spikes* to either excite or inhibit the receiving neurons. The combined effect of excitatory and inhibitory inputs changes the potential level of the receiving neuron and determines whether it will become excited. RNN has attracted a lot of attention in the scientific community due to its analytical solvability, excellent learning capacity, implementation ease, as well as its representational, modelling and universal approximation capabilities [3]. RNN has also been applied for the solution for different types of problems including optimization (e.g. minimum Steiner tree [4], assignment of assets to tasks under uncertainty [5], task assignment in distributed systems [6], rescuer assignment of emergency evacuation [7], cognitive packet networks [8]) and modelling (e.g. G-networks [9]–[12], gene regulatory networks [13], and protein interaction networks [14]) problems. Nonetheless, the most important application area of RNN regards the solution of supervised learning problems such as laser intensity vehicle classification [15], wafer surface reconstruction [16], mine detection [17] and denial-of-service attack detection [18].

In the context of supervised RNN learning, several algorithms have been designed over the years aiming to optimize excitatory and inhibitory weight values to minimize some non-convex cost function associated with the learning task. In [19] the standard gradient descent supervised learning algorithm for recurrent RNN was developed while gradient descent learning algorithms have also been developed for extension models such as the Multiple-Class RNN [20], the RNN with synchronized interactions [21] and the feedforward RNN [22]. In the same context more advanced optimization algorithms have also been developed based on contrastive learning [23], quasi-Newton [24] and Levenberg-Marquardt [25] methods. A survey of RNN models, learning algorithms and applications can be found in [26].

Departing from the traditional approach of solving the supervised RNN learning problem as a nonlinear, non-convex optimization problem, in [27], [28] the problem was reformulated into a convex program and solved to optimality. This was accomplished by approximating the equations governing the RNN into a linear system of equations with nonnegativity constraints, when the excitation levels of all neurons are fixed for each input pattern (either to desired or random values). This approximation yielded a linear least squares problem with nonnegativity constraints (NNLS) which is a quadratic programming problem that can be optimally solved using convex optimization algorithms. Because the large-scale nature of the underlying problem prohibited the accurate solution of the NNLS problem using quadratic programming, a first-order recursive algorithm was developed for its solution. NNLS learning has been shown to provide comparable performance to gradient descent RNN learning with lower execution times.

Linear least squares techniques for learning have also been utilised in feedforward connectionist neural networks and shown to be very efficient, obtaining smaller training errors and faster training times compared to backpropagation techniques. These methods are based on the observation that the inputs to the neurons of a given layer is a linear function of the outputs of the preceding layer. The nonlinearity arises from the application of the activation function to the input of each neuron in order to obtain its output. Hence, if the outputs of two consecutive layers are known then the optimal weights connecting the two layers can be derived by minimising the Mean Square Error (MSE) between the actual and the desired input to the second layer [29]. One problem with this approach is that it does not take into consideration the scaling effect of the nonlinear activation function. Attempts to improve this deficiency include approximating the activation function with a linear combination of convex functions [30], considering the slope of the activation function at the desired output values to achieve better scaling [31] and restricting the output neuron values in the non-saturation region of the activation function [32], [33].

Least squares have also been considered in hybrid algorithms. One approach is to obtain the weights of all layers by standard backpropagation algorithms, apart from the output layer where least squares are used to exploit the desired output values from the training data [34], [35]. Other hybrid algorithms have sophisticated iterative methods for choosing the desired weights or output values of the non-output neurons such as penalized functions [36] and sensitivity analysis [37], but employ least squares to optimize the performance of the network for given values of those parameters. Extreme learning machines [38] also rely on least squares learning in a feed-forward architecture upon performing random feature mapping, but the activation function can take any nonlinear piecewise continuous form.

This paper revisits the NNLS supervised learning problem in RNN to develop a customized algorithm for the solution of the NNLS learning problem that achieves more than two orders of magnitude execution speedup. This is a result of significant enhancements which regard (a) the development of analytical expressions for the computation of the gradient and objective NNLS functions and (b) a novel limited-memory, quasi-Newton solution algorithm for the NNLS problem. Our approach also differs from existing least squares techniques for connectionist neural networks because it is developed for a different neural network model so that the approximation and solution approach are different; moreover, it is applied to a fully recurrent network with the least squares method applied to the whole network rather than on a layer-by-layer basis.

The paper is organized as follows. Section II introduces some preliminary material on the mathematical representation of RNN and the solution of the NNLS problem. Sections III and IV outline the procedure to transform the supervised RNN learning problem into an NNLS formulation and the overall solution approach. Section V describes the proposed approach for the solution of the NNLS problem, while Section VI presents the derived analytical expressions for the fast computation of the NNLS gradient and objective function. Section VII demonstrates the efficiency of the developed approach and Section VIII concludes the paper.

Notation: All boldface letters indicate vectors (lower case) or matrices (upper case), while calligraphic letters denote sets. The superscript $(\cdot)^T$ denotes the matrix transpose. Matrix $\mathbf{I}$ is the identity matrix. $\text{diag}(\mathbf{x})$ is a diagonal matrix with elements of the main diagonal given by the entries of vector $\mathbf{x}$, and all other elements equal to zero. $\mathbf{1} = [1, ..., 1]^T$ is a vector of all ones, while $\mathbf{e}_i = [0, ..., 1, ..., 0]^T$ is a vector of all zeros apart from a one at position i. $||\mathbf{z}||_2$ denotes the Euclidean norm of a vector $\mathbf{z}$. Operator $\odot$ denotes element-wise multiplication and $\otimes$ denotes the *Kronecker product* which if applied on $\mathbf{x} \in \mathbb{R}^{N\times 1}$ and $\mathbf{y} \in \mathbb{R}^{M\times 1}$ yields:$\mathbf{x}\otimes\mathbf{y} = [x_1\mathbf{y}^T, ..., x_N\mathbf{y}^T]^T \in \mathbb{R}^{NM\times 1}$. $U(a, b)$ and $U^{int}(a, b)$ represent the uniform distribution in the interval $[a, b]$ generating real and integer numbers, respectively.

II. Preliminaries

A. The Random Neural Network Model

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

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

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

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

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

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

Combining Eqs. (1), (2) and (3) an expression which associates r_i with $w^+(i, j)$ and $w^-(i, j)$ is derived:

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

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

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

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

$$\lambda^+(i) = \Lambda_i + \sum_{j=1}^{N} r_j q_j p^+(j,i) \tag{5}$$

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

where

$$q_i = \min\left\{1, \frac{\lambda^+(i)}{r_i + \lambda^-(i)}\right\} \tag{7}$$

If a unique non-negative solution $\{\lambda^-(i),\ \lambda^+(i)\}$ *exists for the non-linear system of Eqs. (5)-(7) such that* $q_i < 1\ \forall i,$ *then*:

$$\pi(\mathbf{k}) = \prod_{i=1}^{N} \pi_i(k_i) = \prod_{i=1}^{N} (1 - q_i)\, q_i^{k_i} \tag{8}$$

The theorem states that whenever a solution to the signal flow Eqs. (5)-(7) can be found such that $q_i < 1, \forall i$, then the stationary joint probability distribution of the network has the simple product form (8) associated with the marginal probabilities of each neuron, $\pi_i(k_i)$. The condition $q_i < 1$ can be viewed as a "stability condition" that guarantees that the excitation level of each neuron remains finite with probability one.

B. Nonnegative Least Squares

NNLS is a convex quadratic programming (QP) optimization problem [39] defined as:

$$\min_{\mathbf{w} \geq 0} f(\mathbf{w}) \ = \ \frac{1}{2}\|\mathbf{B}\mathbf{w} - \mathbf{b}\|_2^2, \tag{9}$$

Due to the special structure of NNLS, apart from classical QP solution approaches, several other approaches has been proposed to solve large scale problems which can generally be classified into *active set algorithms* and *iterative approaches*.

In active set algorithms variables are divided into two sets: the *active set* and the *passive set*. A variable belongs to the active set if it is negative or zero when the unconstrained problem is solved, otherwise it belongs to the passive set. When the unconstrained least squares problem is solved, negative or zero variables do not contribute to the constrained problem; therefore, if the true active set is known then the solution can be found by solving the unconstrained problem for the passive set of variables and setting the active variables equal to zero.

The most widely known active set algorithm is the one proposed by Lawson and Hanson [39]. In this approach, initially all the variables are inserted into the active set. Then an iterative procedure is followed where in each iteration variables that result in a strictly better evaluation of the cost function are identified and removed from the active set. The procedure continues until no more active variables can be freed to reduce further the cost function. Although it is possible to free many variables at a single iteration, general practice has shown that it is better to free only one variable at a time from the active set [40].

A modified version of this algorithm identifies calculations that can be computed beforehand to reduce the computational cost. The algorithm called FNNLS (Fast Nonnegative Least Squares) [41] speeds-up the procedure, but requires the storage of the square matrix $\mathbf{B}^T\mathbf{B}$; thus, it is not suitable for our problem. Active set methods are also not appropriate in our case because they require the solution of the unconstrained least squares problem which involves a matrix inversion operation, that is an operation we want to avoid.

Iterative approaches adhere to nonlinear optimization methods to update $\mathbf{w}$. Usually the update of the current solution is based on *projected gradient methods* which can identify several active set constraints in one iteration. In the projected gradient methods, the update takes place towards the steepest descent direction; nonetheless, by using the projection operation it is ensured that the new point is within the feasible region as shown in Eqs. (10) and (11).

$$\mathbf{w}^{\tau+1} = P[\mathbf{w}^\tau - s^\tau \mathbf{D}^\tau \nabla f(\mathbf{w})], \quad s^\tau \geq 0 \tag{10}$$

$$P[w_i] = \begin{cases} w_i, & w_i \ > \ 0 \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

Projected gradient methods usually differ in the procedure used for the selection of the step-size s^τ and the update of the gradient scaling matrix $\mathbf{D}^\tau$, which must be symmetric and positive definite. They generally require simple matrix vector operations and can perform well in ill-conditioned systems. $\mathbf{D}^\tau$ is computed based on second order gradient information and results in fast convergence to the solution. An efficient and simple projected gradient quasi-Newton method that uses the gradient scaling matrix $\mathbf{D}^\tau$ was proposed in [42]. The method exploits the active and passive variables and also requires only simple matrix-vector operations. Nonetheless, it requires the explicit storage of $\mathbf{D}^\tau$ and cannot be used in our case. In Section V we develop a projected gradient NNLS (PGNNLS) algorithm with all the above desirable characteristics.

III. Nonnegative Least Squares RNN Learning Formulation

In RNN, the kth input training pattern $\mathbf{x}_k$ is represented by the vectors $\boldsymbol{\Lambda_k} = [\Lambda_{1k}, \ \dots \ , \Lambda_{Nk}]$ and $\boldsymbol{\lambda_k} = [\lambda_{1k}, \ \dots \ , \lambda_{Nk}]$, $k = 1, ..., K$. Usually the approach taken is to assign the input training values, x_{ik} to the exogenous arrival rates such that $\Lambda_{ik} = \max\{0, x_{ik}\}$ and $\lambda_{ik} = \max\{0, -x_{ik}\}$. The desired values of the kth pattern, $\mathbf{y}_k$, are represented by the steady-state excitation probabilities of the neurons $\mathbf{q}_k = [q_{1k}, \ \dots \ , q_{Nk}]$ emanating from applying input training pattern k to the network. The RNN weights updated during the learning process are $w^+(i,j)$ and $w^-(i,j)$, $\forall i, j$.

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

$$E = \sum_{k=1}^{K} E_k = \frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N}\bar{c}_i(g_i(q_{ik}) - y_{ik})^2 \tag{12}$$

where E_k is the error function of the kth input-output pair, $\bar{c}_i \in \{0,1\}$ shows whether neuron i belongs to the set of output neurons ($i \in \mathcal{I}_{out}$) or not ($i \in \mathcal{I}_{\overline{out}}$) and $g_i(q_{ik})$ is a differentiable function of neuron i.

Ideally we would like to observe the desired output $\mathbf{y}_k$ for all patterns. This means, that for all patterns we should have: q_{ik} = $g_i^{-1}(y_{ik})$, $i \in \mathcal{I}_{out}$, where $g_i^{-1}(\cdot)$ is the inverse function of $g_i(\cdot)$. This is achieved, if the positive and negative weights are selected so that the appropriate q_{ik} values are achieved for all neurons. Without loss of generality, in the sequel we assume that $g_i(q_{ik}) = q_{ik}$, so that $g_i^{-1}(y_{ik}) = y_{ik}$.

Combining Eqs. (5)-(7) we obtain:

$$q_{ik} = \min\left\{1, \frac{\lambda^+(i,k)}{r_i + \lambda^-(i,k)}\right\} = \min\left\{1, \frac{\Lambda_{ik} + \sum\limits_{j=1}^{N} q_{jk} w^+(j,i)}{r_i + \lambda_{ik} + \sum\limits_{j=1}^{N} q_{jk} w^-(j,i)}\right\} \ \forall i, k \tag{13}$$

If we further assume that $\lambda^+(i,k) < r_i + \lambda^-(i,k) \ \forall i, k$ and also substitute Eq. (4) into Eq. (13) we obtain:

$$q_{ik}(1 - d(i))^{-1}\sum_{j=1}^{N}(w^-(i,j) + w^+(i,j)) + q_{ik}\sum_{j=1}^{N} q_{jk} w^-(j,i) - \sum_{j=1}^{N} q_{jk} w^+(j,i) = \Lambda_{ik} - q_{ik}\lambda_{ik}, \ \forall i, \ k \tag{14}$$

If the network is only composed of output neurons, and if we assume that $q_{ik} = y_{ik}, \ \forall i, k$, then Eq. (14) becomes a linear system of NK equations with $2N^2$ nonnegative unknowns, the weights $w^+(i,j)$ and $w^-(i,j)$. If there are both output and non-output neurons then by selecting appropriate values for the excitation probabilities of the latter we can still obtain a linear system, as discussed in Section IV.

An accurate solution to Eq. (14) may not be available for two reasons. First, the number of equations may be larger that the number of unknowns; this is true when $K > 2N$. Second, the nonnegativity constraints restrict the values of the variables and a solution may not exist even if $K < 2N$. As a result we formulate Eq. (14) as an NNLS problem to approach equality as much as possible in the least square sense yielding (9) with $\mathbf{B} \in \mathbb{R}^{NK \times 2N^2}$, $\mathbf{b} \in \mathbb{R}^{NK \times 1}$ and $\mathbf{w} \in \mathbb{R}^{2N^2 \times 1}$.

The gradient of the objective function is given by:

$$\nabla f(\mathbf{w}) \ = \ \mathbf{B}^T\mathbf{B}\mathbf{w} - \mathbf{B}^T\mathbf{b} \tag{15}$$

The ik row of matrix $\mathbf{B}$ and vector $\mathbf{b}$ in Eq. (9), which correspond to the ith signal flow equation of the kth pattern, are given by the following expressions:

$$\begin{array}{lll}
B(ik, ij^+) & = q_{ik}(1 - d(i))^{-1}, & \forall \ j \neq i \\
B(ik, ij^-) & = q_{ik}(1 - d(i))^{-1}, & \forall \ j \neq i \\
B(ik, ji^-) & = q_{ik}q_{jk}, & \forall \ j \neq i \\
B(ik, ji^+) & = -q_{jk}, & \forall \ j \neq i \\
B(ik, ii^+) & = q_{ik}(1 - d(i))^{-1} - q_{ik}, & j = i \\
B(ik, ii^-) & = q_{ik}(1 - d(i))^{-1} + q_{ik}^2, & j = i \\
B(ik, otherwise) & = 0, &
\end{array} \tag{16}$$

$$b(ik) = \Lambda_{ik} - q_{ik}\lambda_{ik}, \quad \forall \ i, k \tag{17}$$

The column indices of $\mathbf{B}$, ij^+ and ij^-, indicate the position of the variables $w^+(i,j)$ and $w^-(i,j)$ in $\mathbf{w}$ respectively. Notice that every value of $\mathbf{B}$ can be found by only using matrix $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_k, ..., \mathbf{q}_K]$, $\mathbf{Q} \in \mathbb{R}^{N \times K}$, which holds the q_{ik} values of both output and non-output neurons; the $d(i)$ values are usually constant and for simplicity we assume that $d(i) = 0, \forall i$. Also, despite the fact that every row of matrix $\mathbf{B}$ has $2N^2$ elements, only $4N$ of them are nonzero and hence the density of nonzero elements in $\mathbf{B}$ is $2/N$.

One difficulty associated with the above formulations is the large dimensionality of $\mathbf{B}$ which implies that it may not be possible to be stored in memory. For example, in Section VII we consider supervised RNN problems with dimensions up to $N = 300$ and $K = 1000$, so that matrix $\mathbf{B}$ has dimensions 300000×180000 which prohibits its storage in memory. Moreover, initial experimentation showed that $\mathbf{B}$ is ill-conditioned. Therefore, in Section V we develop a projected gradient NNLS (PGNNLS) algorithm that does not require either storing large matrices or performing matrix inversion operations. It is important that only simple operations are performed, such as matrix-vector products, avoiding inefficient matrix-matrix multiplications or matrix inversion operations. To achieve the requirements of the solution approach, it is also important to consider the sparseness of $\mathbf{B}$.

IV. RNN-NNLS LEARNING ALGORITHM

Solving the NNLS problem can provide good learning performance only when all neurons are output neurons. Nonetheless, because this is not the case we take the following approach: if neuron $i \in \mathcal{I}_{out}$ then we set $q_{ik} = y_{ik},\ \forall k$, while if neuron $i \in \mathcal{I}_{\overline{out}}$ then we set $q_{ik} = U(a,b),\ \forall k$, with $0 \leq a \leq b \leq 1$. Following this approach, we obtain q_{ik} values for all neurons and patterns; thus, an NNLS problem is derived (Eqs. (9),(16),(17)), which can be solved using the PGNNLS algorithm developed in Section V.

As already mentioned, due to the nonnegativity constraints and depending on the dimensions N,K the system of Eqs. (14) may not have a solution; therefore, the obtained weights from the solution of the NNLS optimisation problem will not accurately satisfy Eqs. (5)-(7), and the obtained weights will not result in good performance. To deal with this issue, we use the weights acquired from the execution of Algorithm 2, to compute q_{ik} from Eqs. (5)-(7). Then, a weighted version of the desired values q_{ik}^d and the exact values q_{ik} is computed and used as the new q_{ik}^d values in PGNNLS. Using this iterative procedure, we progressively move towards weights that satisfy $q_{ik}^d \approx q_{ik}$.

To retain our original goal of achieving the desired output values $y_{ik},\ i \in \mathcal{I}_{out}$ we update the output q_{ik}^d values in two different ways: (a) there is a different weighting parameter for these neurons, $0 \leq \alpha_o \leq 1$, typically close to one so that their desired values slowly vary, and (b) we restrict the neuron values within a desired region so that neurons corresponding to "0" decisions must have $q_{ik} \leq 0.4$ and neurons corresponding to "1" decisions must have $q_{ik} \geq 0.6$. By selecting the specific boundary values, we achieve to constrain each q_{ik} in the desired region and to have a large variation range for the parameters.

Algorithm 1 - RNN-NNLS: RNN supervised learning algorithm based on NNLS formulation

```
Input: $\mathbf{x}_k,\ \mathbf{y}_k, \forall k$
Output: $\mathbf{w}$
Initialise $\Lambda_{ik}$ and $\lambda_{ik}\ \forall\ i,\ k$ based on $x_{ik}$;
Set $q_{ik}^d = y_{ik},\ i \in \mathcal{I}_{out}$;
Set $q_{ik}^d = U(a,b),\ i \in \mathcal{I}_{\overline{out}}$;
Form matrix $\mathbf{Q}^d = [\mathbf{q}_1^d, ..., \mathbf{q}_K^d]$, where $\mathbf{q}_k^d(i) = q_{ik}^d,\ \forall i,k$;
for $l = 1$ to $NI_{RNN-NNLS}$ do
  Update $\mathbf{b}$ according to Eq. (17);
  $\mathbf{w} \leftarrow$ PGNNLS$(\mathbf{Q}^d, \mathbf{b})$;
  for all k do
    Obtain $q_{ik},\ i \in \mathcal{I}_{\overline{out}}$ by solving Eqs. (5)-(7);
  end for
  Set $q_{ik}^d \leftarrow \alpha_{no} q_{ik}^d + (1-\alpha_{no}) q_{ik}, i \in \mathcal{I}_{\overline{out}}, \forall k$;
  Set $q_{ik}^d \leftarrow \alpha_o q_{ik}^d + (1-\alpha_o) q_{ik}, i \in \mathcal{I}_{out}, \forall k$;
  for $i \in \mathcal{I}_{\overline{out}},\ k = 1, ..., K$ do
    if $((y_{ik} = 1)$ AND $(q_{ik}^d < 0.6))$ then
      $q_{ik}^d = 0.6$;
    end if
    if $((y_{ik} = 0)$ AND $(q_{ik}^d > 0.4))$ then
      $q_{ik}^d = 0.4$;
    end if
  end for
end for
```

The overall procedure is outlined in Algorithm 1, called **RNN-NNLS**. It is important to note that the NNLS algorithm does not require matrix $\mathbf{B}$ as input, which would be prohibitive for a large network. Due to Eq. (16), we can perform all the computations involving $\mathbf{B}$ using matrix $\mathbf{Q}$ which holds all the q_{ik} values. Thereby, the order of memory required is the same with the standard RNN learning algorithm. The iterative procedure RNN-NNLS needs only a small number of iterations, $NI_{RNN-NNLS}$, before the error stabilises.

V. Projected Gradient Nonnegative Least Squares Algorithm

In this section we develop a Projected Gradient NNLS (PGNNLS) algorithm based on updating the search-direction using a limited-memory BFGS formula and performing an "Armijo rule along the projection arc" (APA) line-search [43]. Our approach is a modified version of the quasi-Newton NNLS algorithm proposed in [42]; nevertheless it is different both in terms of the employed line-search (hyper-exponential instead of standard APA) and the procedure for updating the search direction (limited memory instead of full BFGS); the developed approach is outlined in Algorithm 2.

Algorithm 2 - PGNNLS: Projected Gradient Algorithm for the NNLS problem

```
Input: $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_K]$, $\mathbf{b}$, $M$
Output: $\mathbf{w}^\tau$
Initialise: $\tau \leftarrow 0$; $\mathbf{w}^\tau \leftarrow \mathbf{0}$; $s^{\tau-1} \leftarrow 1$;
Set $f^\tau \leftarrow f(\mathbf{w}^\tau)$ and $\mathbf{g}^\tau \leftarrow \nabla f(\mathbf{w}^\tau)$ using Eqs. (9) and (15);
Find the binding set $\mathcal{B}^\tau$ according to expression (19);
Set $\widetilde{\mathbf{d}}^\tau = \nabla_P f(\mathbf{w}^\tau)$ using Eq. (21);
repeat
  %Perform an APA line-search
  $[\mathbf{w}_{temp},\ s_{temp}] \leftarrow$ lineSearchHE($f^\tau$, $\nabla f(\mathbf{w}^\tau)$, $\widetilde{\mathbf{d}}^\tau$, $\mathbf{w}^\tau$, $s^{\tau-1}$, $\mathbf{Q}$, $\mathbf{b}$);
  $s^\tau \leftarrow s_{temp}$; $\tau \leftarrow \tau + 1$; $\mathbf{w}^\tau \leftarrow \mathbf{w}_{temp}$;
  Set $f^\tau \leftarrow f(\mathbf{w}^\tau)$, $\mathbf{g}^\tau \leftarrow \nabla f(\mathbf{w}^\tau)$ using Eqs. (9) and (15);
  Find the binding set $\mathcal{B}^\tau$ according to expression (19);
  %Update the search direction
  if ($s^\tau \geq s_{min}$) then
    if ($\tau > M$) then
      Discard the vector pair $\{\Delta\mathbf{w}^{\tau-1-M}, \Delta\mathbf{g}^{\tau-1-M}\}$ from storage;
    end if
    Store $\Delta\mathbf{w}^{\tau-1} = \mathbf{w}^\tau - \mathbf{w}^{\tau-1}$; $\Delta\mathbf{g}^{\tau-1} = \mathbf{g}^\tau - \mathbf{g}^{\tau-1}$;
    $\mathbf{d}^\tau \leftarrow$ updateLBFGS($\nabla_P f(\mathbf{w}^\tau), \Delta\mathbf{w}^k, \Delta\mathbf{g}^k, k = \max\{0, \tau - M\}, ..., \tau - 1$);
    Define $\widetilde{\mathbf{d}}^\tau$ according to Eq. (20)
  else
    Discard all stored vector pairs $\{\Delta\mathbf{w}^k, \Delta\mathbf{g}^k\}$;
    Set $\widetilde{\mathbf{d}}^\tau = \nabla_P f(\mathbf{w}^\tau)$ using Eq. (21);
  end if
until a stopping criterion is met
```

The key aspect of Algorithm 2 is that at iteration τ we only perform a line-search for the variables that are in the *free-set* $\mathcal{F}^\tau$ defined as:

$$\mathcal{F}^\tau = \{i | w_i^\tau > 0 \text{ or } (w_i^\tau = 0 \text{ and } [\nabla f(\mathbf{w}^\tau)]_i \leq 0)\} \tag{18}$$

To understand the reason behind this, let us define the complement of $\mathcal{F}^\tau$, called the *binding set* $\mathcal{B}^\tau$:

$$\mathcal{B}^\tau = \{i | w_i^\tau = 0 \text{ and } [\nabla f(\mathbf{w}^\tau)]_i > 0\} \tag{19}$$

For the variables belonging to the binding set there are two possibilities about the search direction: (a) $d_i^\tau \geq 0$, and (b) $d_i^\tau < 0$. In the first case, we have that $w_i^{\tau+1} = P[w_i^\tau - s^\tau d_i^\tau] = P[-s^\tau d_i^\tau] = 0$ so that this variable remains constant and does not affect the cost function. In the second case, we have that $w_i^{\tau+1} = P[w_i^\tau - s^\tau d_i^\tau] = P[-s^\tau d_i^\tau] = -s^\tau d_i^\tau > 0$; however, the fact that $-d_i^\tau[\nabla f(\mathbf{w}^\tau)]_i > 0$ is undesirable, as it contributes negatively towards the condition that guarantees function reduction at the particular direction $(-(\mathbf{d}^\tau)^T \nabla f(\mathbf{w}^\tau) < 0)$.

As a result, variables belonging to $\mathcal{B}^\tau$ should not affect the line-search procedure of iteration τ. This is achieved by considering a modified direction $\widetilde{\mathbf{d}}$ defined as:

$$\widetilde{d}_i^\tau = \begin{cases} \bar{d}_i^\tau, & i \in \mathcal{F}^\tau \\ 0, & i \in \mathcal{B}^\tau \end{cases}, \forall i \tag{20}$$

where $\bar{\mathbf{d}}^\tau = \mathbf{S}^\tau \nabla_P f(\mathbf{w}^\tau)$ and $\nabla_P f(\mathbf{w}^\tau)$ is the *projected gradient* given by Eq. (21).

$$[\nabla_P f(\mathbf{w}^\tau)]_i = \begin{cases} [\nabla f(\mathbf{w}^\tau)]_i, & i \in \mathcal{F}^\tau \\ 0, & i \in \mathcal{B}^\tau \end{cases}, \forall i \tag{21}$$

In this way, Eq. (10) becomes:

$$\mathbf{w}^{\tau+1} = P[\mathbf{w}^\tau - s^\tau \widetilde{\mathbf{d}}^\tau]$$

An equivalent expression can be obtained by updating only the variables belonging to the free-set: $\mathbf{w}_{\mathcal{F}}^{\tau+1} = P[\mathbf{w}_{\mathcal{F}}^{\tau} - s^{\tau}\widetilde{\mathbf{S}}^{\tau}\mathbf{g}_{\mathcal{F}}^{\tau}]$, where $\widetilde{\mathbf{S}}^{\tau} \in \mathbb{R}^{|\mathcal{F}|\times|\mathcal{F}|}$ is the principal submatrix of $\mathbf{S}^{\tau}$ corresponding to the free variables and similarly, $g_{i,\mathcal{F}}^{\tau} = [\nabla f(\mathbf{w}^{\tau})]_{\mathcal{F}(i)}$, $i = 1, ..., |\mathcal{F}|$.

As mentioned above, Algorithm 2 relies on a limited-memory BFGS update of the scaling matrix. In each iteration, the BFGS formula is updated so that the new matrix is symmetric, satisfies the secant equation and also is the closest to the current approximation matrix in the least squares sense. In addition, if the associated problem is strictly convex and an appropriate line-search is considered, then the updated matrices are also positive definite [44]. The BFGS formula for updating $\mathbf{S}^{\tau}$ is given by:

$$\mathbf{S}^{\tau} = (\mathbf{V}^{\tau-1})^T\mathbf{S}^{\tau-1}\mathbf{V}^{\tau-1} + \rho^{\tau-1}\Delta\mathbf{w}^{\tau-1}(\Delta\mathbf{w}^{\tau-1})^T, \tag{22}$$

where $\rho^k = 1/((\Delta\mathbf{g}^k)^T\Delta\mathbf{w}^k)$, $\mathbf{V}^k = \mathbf{I} - \rho^k\Delta\mathbf{g}^k(\Delta\mathbf{w}^k)^T$, and $\mathbf{S}^0 = \sigma_{BFGS}\mathbf{I}$, $\sigma_{BFGS} > 0$. Notice that $\mathbf{S}^{\tau}$ is a rank-two modification of $\mathbf{S}^{\tau-1}$ which can be obtained using $\Delta\mathbf{w}^{\tau-1}$ and $\Delta\mathbf{g}^{\tau-1}$. Hence, if we store all vectors $\Delta\mathbf{w}^k$ and $\Delta\mathbf{g}^k$ from the start of the algorithm, we can obtain $\mathbf{S}^{\tau}$ without storing any matrix.

In the limited-memory variant of BFGS, instead of storing all vectors, we update $\mathbf{S}^{\tau}$ based on the M most recent $\Delta\mathbf{w}^k$ and $\Delta\mathbf{g}^k$ vector pairs. This is achieved with the use of the following recursive formula which is directly derived from (22) [45].

$$\begin{aligned}\mathbf{S}^{\tau} =&(\mathbf{V}^{\tau-M}\cdots\mathbf{V}^{\tau-1})^T\mathbf{S}_0^{\tau}(\mathbf{V}^{\tau-M}\cdots\mathbf{V}^{\tau-1}) + \rho^{\tau-M}(\mathbf{V}^{\tau-M+1}\cdots\mathbf{V}^{\tau-1})^T\Delta\mathbf{w}^{\tau-M}(\Delta\mathbf{w}^{\tau-M})^T(\mathbf{V}^{\tau-M+1}\cdots\mathbf{V}^{\tau-1})\\ &+\rho^{\tau-M+1}(\mathbf{V}^{\tau-M+2}\cdots\mathbf{V}^{\tau-1})^T\Delta\mathbf{w}^{\tau-M+1}(\Delta\mathbf{w}^{\tau-M+1})^T(\mathbf{V}^{\tau-M+2}\cdots\mathbf{V}^{\tau-1}) + \cdots + \rho^{\tau-1}\Delta\mathbf{w}^{\tau-1}(\Delta\mathbf{w}^{\tau-1})^T.\end{aligned} \tag{23}$$

Using Eq. (23) we can efficiently update the search direction $\mathbf{d}^{\tau} = \mathbf{S}^{\tau}\nabla f(\mathbf{w}^{\tau})$, without storing $\mathbf{S}^{\tau}$ at any iteration. As a result, the required memory for the quasi-Newton update is reduced from $2N^2 \times 2N^2$ to $2M \times 2N^2$. This is a substantial memory saving, as it has been observed in practice that even small values of M (say $M \in [3, 7]$) provide satisfactory results [45]. Nocedal and Wright [46] describe in detail the limited memory BFGS method and outline an iterative procedure for updating the search direction based on (23); we outline this procedure in Algorithm 3.

Algorithm 3 - updateLBFGS: Compute the product of the limited memory scaling matrix and the gradient

```
Input: $\mathbf{g}, \Delta\mathbf{w}^k, \Delta\mathbf{g}^k, k = \max\{0, \tau - M\}, ..., \tau - 1$
Output: $\mathbf{d}$
for ($k = \tau - 1, ..., \max\{0, \tau - M\}$) do
    $\rho^k \leftarrow 1/((\Delta\mathbf{g}^k)^T\Delta\mathbf{w}^k)$;
    $\alpha_1^k \leftarrow \rho^k(\Delta\mathbf{w}^k)^T\mathbf{g}$;
    $\mathbf{g} \leftarrow \mathbf{g} - \alpha_1^k\Delta\mathbf{g}^k$;
end for
$\mathbf{d} \leftarrow \mathbf{S}_0^{\tau}\mathbf{g}$;
for ($k = \max\{0, \tau - M\}, ..., \tau - 1$) do
    $\alpha_2 \leftarrow \rho^k(\Delta\mathbf{g}^k)^T\mathbf{d}$;
    $\mathbf{d} \leftarrow \mathbf{d} + (\alpha_1^k - \alpha_2)\Delta\mathbf{w}^k$;
end for
```

The use of the limited-memory BFGS scheme also provides computational benefits. Note that updating the scaling matrix using the BFGS method requires several matrix-vector operations whose computational complexity is $O((2N^2)^2)$. On the other hand, the use of Algorithm 3 requires $5M$ vector-vector products so that its computational complexity is $O(5M(2N^2))$ which is significantly less than the complexity of a single matrix-vector product.

Let us now turn our attention to the line-search procedure. As mentioned above, the step-size s^{τ} is found by employing the APA line-search [43]. In APA the step-size is chosen to be equal to $s^{\tau} = \beta^m$, where m is the smallest nonnegative integer satisfying the APA condition:

$$f(\mathbf{w}_{cand}^{\tau+1}(\beta^m)) - f(\mathbf{w}^{\tau}) \leq \sigma_{APA}\nabla f(\mathbf{w}^{\tau})^T(\mathbf{w}_{cand}^{\tau+1}(\beta^m) - \mathbf{w}^{\tau}) \tag{24}$$

where $\mathbf{w}_{cand}^{\tau+1}(\beta^m) = P[\mathbf{w}^{\tau} - \beta^m\widetilde{\mathbf{d}}^{\tau}]$, $0 < \sigma_{APA} < 1/2$ and $0 < \beta < 1$. An important advantage of the APA over other step-size rules is that it identifies many active constraints in one iteration. In addition, it is proven that the sequence $\{\mathbf{w}^{\tau}\}$ produced when applying the APA rule, converges to a stationary point $\{\mathbf{w}^*\}$ [43], which in our case is a global minimum. In [47], a more detailed analysis of projected gradient algorithms further relaxed the convergence conditions. The authors showed that convergence to a stationary point can be achieved by choosing any step-size satisfying condition (24), under the assumptions that s^{τ} is not too small, the cost function is bounded below and the gradient is uniformly continuous (Theorem 2.3 in [47]), which are true in the NNLS case. Hence, convergence is guaranteed even if we choose any value of m satisfying (24) rather than the smallest integer, as long as the selected β^m are not too small.

Nevertheless, sequentially identifying the appropriate m value may require a large number of function evaluations and projections. In [28] we proposed to hyper-exponentially alternate s^{τ} for the identification of an appropriate step-size value.

In the hyper-exponential line-search (**lineSearchHE**), the first trial starts from $s^{\tau-1}$ and if the APA condition is satisfied, ($s^{\tau} \geq s^{\tau-1}$), we hyper-exponentially increase the step-size (division of the initial step-size by β^{2^k}, $k = 0, 1, 2...$) until a step-size $s^{\tau}_{init}/\beta^{2^{k_v}}$ violating condition (24) is found; otherwise, we hyper-exponentially decrease the step-size (multiplication of the initial step-size by β^{2^k}, $k = 0, 1, 2...$) until a step-size $s^{\tau}_{init}\beta^{2^{k_s}}$ satisfying condition (24) is found. In this way, an appropriate region for the optimal step-size is identified; then, a divide and conquer procedure is followed to find the largest value β^m satisfying Eq. (24).

Formally the stopping criterion that should be met for the termination of the PGNNLS algorithm is related to the Karush-Kuhn-Tucker (KKT) optimality conditions. However, we do not require the accurate solution of the NNLS problem as it is only used to approximately train RNN. Hence, we employ the maximum number of iterations as stopping criterion.

The most costly operations that need to be performed at each iteration of Algorithm 2 involve the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$, which require matrix-vector product operations. In particular, at the start of each iteration, it is needed to evaluate $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ once. Additionally, each trial of the line-search procedure requires the evaluation of $f(\mathbf{w}^{\tau+1}_{cand})$. In Section VI, we discuss two different approaches for the efficient evaluation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ and derive analytical expressions.

VI. Efficient computation of NNLS costly functions

As already mentioned, the most computationally expensive functions in Algorithm 2 are $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$. However, computing these functions directly is very inefficient, so that the structure and sparsity of matrix $\mathbf{B}$ should be exploited to find efficient ways to compute $\nabla f(\mathbf{w})$. In this section we develop two such approaches. The first is based on the efficient computation of $\mathbf{B}^T\mathbf{z}_1$ and $\mathbf{B}\mathbf{z}_2$ where $\mathbf{z}_1$ and $\mathbf{z}_2$ are vectors of appropriate dimensions. The second is based on first computing and storing $\mathbf{B}^T\mathbf{B}$ in order to compute $(\mathbf{B}^T\mathbf{B})\mathbf{z}$. We show that the computational complexity of the former approach is $O(KN^2)$ per evaluation, while the complexity of the latter is $O(N^3)$ per evaluation plus an initialisation cost of $O(KN^3)$. This indicates that each of the two approaches can be faster than the other depending on the problem dimensions (number of training pairs, K, and number of neurons, N). The second approach is faster that the first if $K \gg N$, otherwise the first approach is better.

A. The structure of matrix $\mathbf{B}$

Matrix $\mathbf{B}$ is composed of many different matrix blocks which correspond to entries associated with positive or negative weights as well as different input-output training pairs. As a result we can represent $\mathbf{B}$ as:

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{+1}, & \mathbf{B}_{-1} \\ \vdots & \vdots \\ \mathbf{B}_{+K}, & \mathbf{B}_{-K} \end{bmatrix}, \mathbf{B} \in \mathbb{R}^{KN\times 2N^2} \text{ and } \mathbf{B}_{\pm k} \in \mathbb{R}^{N\times N^2},\ k = 1, ..., K \tag{25}$$

where $\mathbf{B}_{+k}$ and $\mathbf{B}_{-k}$ indicate the entries associated with the kth input-output training pair of the positive and negative weights respectively. These matrices are sparse and are also of particular structure, as shown below for the case that $N = 3$, when $d(i) = 0$, $i = 1, ..., N$.

$$\mathbf{B}_{+k} = \overbrace{\begin{bmatrix} 0 & q_{1k} & q_{1k} & -q_{2k} & 0 & 0 & -q_{3k} & 0 & 0 \\ 0 & -q_{1k} & 0 & q_{2k} & 0 & q_{2k} & 0 & -q_{3k} & 0 \\ 0 & 0 & -q_{1k} & 0 & 0 & -q_{2k} & q_{3k} & q_{3k} & 0 \end{bmatrix}}^{\text{values corresponding to } w^+(i,j)}$$

$$\mathbf{B}_{-k} = \overbrace{\begin{bmatrix} q_{1k}+q_{1k}^2 & q_{1k} & q_{1k} & q_{1k}q_{2k} & 0 & 0 & q_{1k}q_{3k} & 0 & 0 \\ 0 & q_{1k}q_{2k} & 0 & q_{2k} & q_{2k}+q_{2k}^2 & q_{2k} & 0 & q_{2k}q_{3k} & 0 \\ 0 & 0 & q_{1k}q_{3k} & 0 & 0 & q_{2k}q_{3k} & q_{3k} & q_{3k} & q_{3k}+q_{3k}^2 \end{bmatrix}}^{\text{values corresponding to } w^-(i,j)}$$

Note that the structure of the above matrices allows their further decomposition into:

$$\mathbf{B}_{+k} = \mathbf{C}_k + \mathbf{D}_{+k} \tag{26}$$

$$\mathbf{B}_{-k} = \mathbf{C}_k + \mathbf{D}_{-k} \tag{27}$$

For example, for the case that $N = 3$ matrices $\mathbf{C}_k$, $\mathbf{D}_{+k}$ and $\mathbf{D}_{-k}$ take the form:

$$\mathbf{C}_k = \begin{bmatrix} q_{1k} & q_{1k} & q_{1k} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_{2k} & q_{2k} & q_{2k} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & q_{3k} & q_{3k} & q_{3k} \end{bmatrix}$$

$$\mathbf{D}_{+k} = \begin{bmatrix} -q_{1k} & 0 & 0 & -q_{2k} & 0 & 0 & -q_{3k} & 0 & 0 \\ 0 & -q_{1k} & 0 & 0 & -q_{2k} & 0 & 0 & -q_{3k} & 0 \\ 0 & 0 & -q_{1k} & 0 & 0 & -q_{2k} & 0 & 0 & -q_{3k} \end{bmatrix}$$

$$\mathbf{D}_{-k} = \begin{bmatrix} q_{1k}q_{1k} & 0 & 0 & q_{2k}q_{1k} & 0 & 0 & q_{3k}q_{1k} & 0 & 0 \\ 0 & q_{1k}q_{2k} & 0 & 0 & q_{2k}q_{2k} & 0 & 0 & q_{3k}q_{2k} & 0 \\ 0 & 0 & q_{1k}q_{3k} & 0 & 0 & q_{2k}q_{3k} & 0 & 0 & q_{3k}q_{3k} \end{bmatrix}$$

Notice that matrices $\mathbf{C}_k$, $\mathbf{D}_{+k}$ and $\mathbf{D}_{-k}$ also have a special structure while all can be decomposed further into $N \times N$ sized submatrices such that $\mathbf{C}_k = [\mathbf{C}_{k1}, \ldots, \mathbf{C}_{kN}]$, $\mathbf{D}_{+k} = [\mathbf{D}_{+k1}, \ldots, \mathbf{D}_{+kN}]$, and $\mathbf{D}_{-k} = [\mathbf{D}_{-k1}, \ldots, \mathbf{D}_{-kN}]$. Sub-matrices $\mathbf{C}_{ki}$, $\mathbf{D}_{+ki}$, $\mathbf{D}_{-ki} \in \mathbb{R}^{N\times N}$ are given by:

$$\mathbf{C}_{ki} = q_{ik}(\mathbf{e}_i \mathbf{1}^T) \tag{28}$$

$$\mathbf{D}_{+ki} = \text{diag}([-q_{ik}, -q_{ik}, \ldots, -q_{ik}]) = -q_{ik}\mathbf{I} \tag{29}$$

$$\mathbf{D}_{-ki} = q_{ik}\text{diag}([q_{1k},\, q_{2k}, \ldots, q_{Nk}]) = q_{ik}\text{diag}(\mathbf{q}_k). \tag{30}$$

B. The first approach for the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$

As already mentioned, functions $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ can be computed according to Eqs. (9) and (15) respectively, which can be written as:

$$f(\mathbf{w}) = \frac{1}{2}(\mathbf{B}\mathbf{w} - \mathbf{b})^T(\mathbf{B}\mathbf{w} - \mathbf{b}) = \frac{1}{2}(\hat{\mathbf{z}} - \mathbf{b})^T(\hat{\mathbf{z}} - \mathbf{b}) = \frac{1}{2}\mathbf{z}^T\mathbf{z} \tag{31}$$

$$\nabla f(\mathbf{w}) = \mathbf{B}^T(\mathbf{B}\mathbf{w} - \mathbf{b}) = \mathbf{B}^T\mathbf{z} \tag{32}$$

where we have defined $\hat{\mathbf{z}} = \mathbf{B}\mathbf{w}$, $\hat{\mathbf{z}} \in \mathbb{R}^{NK\times 1}$ and $\mathbf{z} = \hat{\mathbf{z}} - \mathbf{b}$. As a result, for the computation of $f(\mathbf{w})$ the only expensive step is the calculation of $\hat{\mathbf{z}} = \mathbf{B}\mathbf{w}$. Similarly, the expensive steps for the computation of $\nabla f(\mathbf{w})$ are the calculation of $\hat{\mathbf{z}} = \mathbf{B}\mathbf{w}$ and $\widetilde{\mathbf{z}} = \mathbf{B}^T\mathbf{z}$, where $\widetilde{\mathbf{z}} \in \mathbb{R}^{2N^2\times 1}$. Note that the matrix-vector product $\mathbf{B}\mathbf{w}$ appears in both terms. As a result, only two matrix-vector products are needed for the evaluation of both functions at the same point $\mathbf{w}_c$: $\hat{\mathbf{z}}_c = \mathbf{B}\mathbf{w}_c$ and $\widetilde{\mathbf{z}} = \mathbf{B}^T\mathbf{z}_c$, where $\mathbf{z}_c = \hat{\mathbf{z}}_c - \mathbf{b}$. As the naive calculation of these matrix-vector products is not efficient, we manipulate the special structure and sparsity of matrix $\mathbf{B}$ to derive expressions of low computational complexity.

Let us first examine the term $\hat{\mathbf{z}} = \mathbf{B}\mathbf{w}$. Expanding $\mathbf{B}$ and $\mathbf{w}$ we obtain:

$$\hat{\mathbf{z}} = \mathbf{B}\mathbf{w} = \begin{bmatrix} \mathbf{C}_1 + \mathbf{D}_{+1}, & \mathbf{C}_1 + \mathbf{D}_{-1} \\ \vdots & \vdots \\ \mathbf{C}_K + \mathbf{D}_{+K}, & \mathbf{C}_K + \mathbf{D}_{-K} \end{bmatrix} \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \end{bmatrix} = \begin{bmatrix} \mathbf{C}_1\mathbf{w}^+ + \mathbf{C}_1\mathbf{w}^- + \mathbf{D}_{+1}\mathbf{w}^+ + \mathbf{D}_{-1}\mathbf{w}^- \\ \vdots \\ \mathbf{C}_K\mathbf{w}^+ + \mathbf{C}_K\mathbf{w}^- + \mathbf{D}_{+K}\mathbf{w}^+ + \mathbf{D}_{-K}\mathbf{w}^- \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{z}}_1 \\ \vdots \\ \hat{\mathbf{z}}_K \end{bmatrix}, \tag{33}$$

where $\hat{\mathbf{z}}_k \in \mathbb{R}^{N\times 1}$ and $\mathbf{w}^+$ represents the positive weights so that value $w^+(iN - N + j) \equiv w^+(i, j)$ and $\mathbf{w}^-$ represents the negative weights such that $w^-(iN - N + j) \equiv w^-(i, j)$. Note that to evaluate $\hat{\mathbf{z}}$ it is sufficient to derive expressions for terms $\hat{\mathbf{z}}_k$:

$$\hat{\mathbf{z}}_k = \mathbf{C}_k\mathbf{w}^+ + \mathbf{C}_k\mathbf{w}^- + \mathbf{D}_{+k}\mathbf{w}^+ + \mathbf{D}_{-k}\mathbf{w}^- \tag{34}$$

Hence, the computation of $\hat{\mathbf{z}}_k$ requires the efficient evaluation of $\mathbf{C}_k\mathbf{w}^+$, $\mathbf{C}_k\mathbf{w}^-$, $\mathbf{D}_{+k}\mathbf{w}^+$ and $\mathbf{D}_{-k}\mathbf{w}^-$. Manipulation of these terms using Eqs. (28) - (30) and matrix algebra yields

$$\hat{\mathbf{z}}_k = \mathbf{q}_k \odot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) - (\mathbf{W}^+)^T\mathbf{q}_k + \mathbf{q}_k \odot \left((\mathbf{W}^-)^T\mathbf{q}_k\right), \tag{35}$$

where the $N \times 1$ vectors $\boldsymbol{\sigma}_{W^+}$ and $\boldsymbol{\sigma}_{W^-}$ are given by

$$\boldsymbol{\sigma}_{W^+} = \mathbf{W}^+\mathbf{1}, \tag{36}$$

$$\boldsymbol{\sigma}_{W^-} = \mathbf{W}^-\mathbf{1}. \tag{37}$$

This definition implies that the ith element of $\boldsymbol{\sigma}_{W^+}$ or $\boldsymbol{\sigma}_{W^-}$ is equal to the sum of the elements belonging to the ith row of the associated matrix.

Having computed $\hat{\mathbf{z}}$ and hence $\mathbf{z}$, we can now proceed with the computation of $\widetilde{\mathbf{z}} = \mathbf{B}^T\mathbf{z}$. If we define $\mathbf{z}^T = \left[\mathbf{z}_1^T, ..., \mathbf{z}_K^T\right]$, where $\mathbf{z}_k \in \mathbb{R}^{N\times 1}$, and use Eq. (25) to expand matrix $\mathbf{B}$ we obtain

$$\widetilde{\mathbf{z}} = \mathbf{B}^T\mathbf{z} = \begin{bmatrix} \mathbf{B}_{+1}^T & \cdots & \mathbf{B}_{+K}^T \\ \mathbf{B}_{-1}^T & \cdots & \mathbf{B}_{-K}^T \end{bmatrix} \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_K \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^K \mathbf{B}_{+k}^T\mathbf{z}_k \\ \sum_{k=1}^K \mathbf{B}_{-k}^T\mathbf{z}_k \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^K (\mathbf{C}_k^T\mathbf{z}_k + \mathbf{D}_{+k}^T\mathbf{z}_k) \\ \sum_{k=1}^K (\mathbf{C}_k^T\mathbf{z}_k + \mathbf{D}_{-k}^T\mathbf{z}_k) \end{bmatrix}. \tag{38}$$

By exploiting the structure of $\mathbf{B}$ and using matrix algebra for each of the appearing terms $\mathbf{C}_k^T\mathbf{z}_k$, $\mathbf{D}_{+k}^T\mathbf{z}_k$ and $\mathbf{D}_{-k}^T\mathbf{z}_k$ yields the following expression:

$$\widetilde{\mathbf{z}} = \begin{bmatrix} \left(\sum_{k=1}^K (\mathbf{q}_k \odot \mathbf{z}_k)\right) \otimes \mathbf{1} - \sum_{k=1}^K (\mathbf{q}_k \otimes \mathbf{z}_k) \\ \left(\sum_{k=1}^K (\mathbf{q}_k \odot \mathbf{z}_k)\right) \otimes \mathbf{1} - \sum_{k=1}^K (\mathbf{q}_k \otimes (\mathbf{q}_k \odot \mathbf{z}_k)) \end{bmatrix}. \tag{39}$$

Having derived expressions to efficiently derive functions $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ the computational complexity of this approach is now examined. For the computation of $\mathbf{Bw}$ the most costly operations are the evaluation of the matrix-vector products $(\mathbf{W}^+)^T\mathbf{q}_k$ and $(\mathbf{W}^-)^T\mathbf{q}_k$ that appear in vectors $\hat{\mathbf{z}}_k$ in Eq. (35). The time complexity of these operations is $O(N^2)$, and as there are K such terms to be computed, the total complexity of evaluating $\mathbf{Bw}$ is $O(KN^2)$. For the computation of term $\mathbf{B}^T\mathbf{z}$, for each k we need to evaluate $\mathbf{q}_k \odot \mathbf{z}_k$, $\mathbf{q}_k \otimes \mathbf{z}_k$ and $\mathbf{q}_k \otimes \boldsymbol{\delta}_k$ which have $O(N)$, $O(N^2)$ and $O(N^2)$ complexity, respectively. In addition, summation of the latter two terms for all k requires $O(KN^2)$. If the required multiplications are performed naively, then the computation of both $\mathbf{Bw}$ and $\mathbf{B}^T\mathbf{z}$ matrix-vectors products would require $O(2KN^3)$, as the dimensions of $\mathbf{B}$ are $KN \times 2N^2$, while the dimensions of $\mathbf{w}$ and $\mathbf{z}$ are $2N^2 \times 1$ and $KN \times 1$ respectively. Hence this approach provides an $O(N)$ complexity reduction compared to naive matrix-vector multiplication.

With respect to memory requirements, this approach involves the storage of the necessary vectors i.e. matrices $\mathbf{W}^+$ and $\mathbf{W}^-$ which have N^2 elements, and matrix $\mathbf{Q}$ which have KN elements in total, as well as a small number of auxiliary vectors. Naively storing $\mathbf{B}$ requires memory for $2KN^3$ elements which is $\min\{KN, 2N^2\}$ times larger than the memory required by our approach.

In sum, the computational complexity of computing $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ is $O(KN^2)$, while the approach does not require the storage of additional matrices other than the necessary $\mathbf{W}^+$, $\mathbf{W}^-$ and $\mathbf{Q}$ which require $KN + 2N^2$ memory.

C. The second approach for the computation of $\nabla f(\mathbf{w})$

A second approach for the evaluation of functions $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ is based on computing (during the initialisation phase) the quantities $\boldsymbol{\Gamma} = \mathbf{B}^T\mathbf{B}$ and $\boldsymbol{\beta} = \mathbf{B}^T\mathbf{b}$. Then, functions $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ can be expressed using these quantities as:

$$f(\mathbf{w}) = \frac{1}{2}(\mathbf{Bw} - \mathbf{b})^T(\mathbf{Bw} - \mathbf{b}) = \frac{1}{2}\mathbf{w}^T\mathbf{B}^T\mathbf{Bw} - \mathbf{w}^T\mathbf{B}^T\mathbf{b} + \frac{1}{2}\mathbf{b}^T\mathbf{b} = \mathbf{w}^T(\frac{1}{2}\boldsymbol{\Gamma}\mathbf{w} - \boldsymbol{\beta}) + \frac{1}{2}\mathbf{b}^T\mathbf{b} \tag{40}$$

$$\nabla f(\mathbf{w}) = \mathbf{B}^T\mathbf{Bw} - \mathbf{B}^T\mathbf{b} = \boldsymbol{\Gamma}\mathbf{w} - \boldsymbol{\beta} \tag{41}$$

Based on the above expressions only the matrix-vector product $\boldsymbol{\Gamma}\mathbf{w}$ is required for their evaluation and hence at a particular point both functions can be computed by just evaluating $\boldsymbol{\Gamma}\mathbf{w}$. Notice that $\mathbf{B} \in \mathbb{R}^{KN\times 2N^2}$ and $\boldsymbol{\Gamma} \in \mathbb{R}^{2N^2\times 2N^2}$ so that the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ are depended both on K, N in the first approach and only on N in the second. Expansion of matrix $\mathbf{B}$ according to Eq. (25) yields:

$$\boldsymbol{\Gamma} = \mathbf{B}^T\mathbf{B} = \begin{bmatrix} \mathbf{B}_{+1}^T & \cdots & \mathbf{B}_{+K}^T \\ \mathbf{B}_{-1}^T & \cdots & \mathbf{B}_{-K}^T \end{bmatrix} \begin{bmatrix} \mathbf{B}_{+1} & \mathbf{B}_{-1} \\ \vdots & \vdots \\ \mathbf{B}_{+K} & \mathbf{B}_{-K} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^K \mathbf{B}_{+k}^T\mathbf{B}_{+k} & \sum_{k=1}^K \mathbf{B}_{+k}^T\mathbf{B}_{-k} \\ \sum_{k=1}^K \mathbf{B}_{-k}^T\mathbf{B}_{+k} & \sum_{k=1}^K \mathbf{B}_{-k}^T\mathbf{B}_{-k} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Gamma}_{11} & \boldsymbol{\Gamma}_{12} \\ \boldsymbol{\Gamma}_{21} & \boldsymbol{\Gamma}_{22} \end{bmatrix} \tag{42}$$

The terms $\boldsymbol{\Gamma}_{lm}$, $l, m = 1, 2$, can be further decomposed into expressions involving $\mathbf{C}_k$, $\mathbf{D}_{+k}$ and $\mathbf{D}_{-k}$ through substitution of Eqs. (26) and (27), associated with $\mathbf{B}_{+k}$ and $\mathbf{B}_{-k}$, into (42). For example, $\boldsymbol{\Gamma}_{11}$ yields:

$$\boldsymbol{\Gamma}_{11} = \sum_{k=1}^K \mathbf{C}_k^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{C}_k^T\mathbf{D}_{+k} + \sum_{k=1}^K \mathbf{D}_{+k}^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{D}_{+k}^T\mathbf{D}_{+k}. \tag{43}$$

Further substitution of Eqs. (28) - (30) into each of the subsequent terms of $\boldsymbol{\Gamma}_{lm}$ and matrix algebra yields that these terms can be reproduced by only storing five vectors/matrices computed during the initialisation phase. These include vector $\boldsymbol{\sigma}_q \in \mathbb{R}^{N\times 1}$ and matrices $\mathbf{M} \in \mathbb{R}^{N\times N}, \mathbf{M}^s \in \mathbb{R}^{N\times N}, \mathbf{R}^i \in \mathbb{R}^{N\times N}$ and $\mathbf{R}^{s,i} \in \mathbb{R}^{N\times N}$, $i = 1, ..., N$ with elements:

$$\sigma_q(i) = \sum_{k=1}^K q_{ik}^2,\ M_{i,j} = \sum_{k=1}^K q_{ik}q_{jk},\ M_{i,j}^s = \sum_{k=1}^K q_{ik}^2 q_{jk},\ R_{j,l}^i = \sum_{k=1}^K q_{ik}q_{jk}q_{lk},\ R_{j,l}^{s,i} = \sum_{k=1}^K q_{ik}q_{jk}q_{lk}^2. \tag{44}$$

For the derivation of $\mathbf{\Gamma w}$ it is true that:

$$\overline{\mathbf{z}} = \mathbf{\Gamma w} = \begin{bmatrix} \mathbf{\Gamma}_{11} & \mathbf{\Gamma}_{12} \\ \mathbf{\Gamma}_{21} & \mathbf{\Gamma}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \end{bmatrix} = \begin{bmatrix} \mathbf{\Gamma}_{11}\mathbf{w}^+ + \mathbf{\Gamma}_{12}\mathbf{w}^- \\ \mathbf{\Gamma}_{21}\mathbf{w}^+ + \mathbf{\Gamma}_{22}\mathbf{w}^- \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{z}}_1 \\ \overline{\mathbf{z}}_2 \end{bmatrix} \tag{45}$$

where vectors $\mathbf{w}^+$and $\mathbf{w}^-$ have already been defined in the first approach, while vectors $\overline{\mathbf{z}}_l = \mathbf{\Gamma}_{l1}\mathbf{w}^+ + \mathbf{\Gamma}_{l2}\mathbf{w}^-$, $\overline{\mathbf{z}}_l \in \mathbb{R}^{N^2\times 1}, l = 1, 2$ can be further decomposed into $\overline{\mathbf{z}}_l^T = [\overline{\mathbf{z}}_{l1}^T, ..., \overline{\mathbf{z}}_{lN}^T]$ with elements $\overline{\mathbf{z}}_{li} \in \mathbb{R}^{N\times 1}$. In order to obtain low complexity expressions for these terms, we take advantage of the expressions derived for the composing matrices of $\mathbf{\Gamma}_{lm}$, and of Eqs. (44), yielding the following expressions for $\overline{\mathbf{z}}_{1i}$ and $\overline{\mathbf{z}}_{2i}$, $i = 1, ..., N$

$$\overline{\mathbf{z}}_{1i} = \sigma_{\overline{z}}(i)\mathbf{1} - \mathbf{m}_i^c \odot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) + (\mathbf{W}^+)^T\mathbf{m}_i^r - (\mathbf{R}^i \odot \mathbf{W}^-)^T\mathbf{1} \tag{46}$$

$$\overline{\mathbf{z}}_{2i} = \sigma_{\overline{z}}(i)\mathbf{1} + \mathbf{m}_i^{s,c} \odot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) - (\mathbf{R}^i \odot \mathbf{W}^+)^T\mathbf{1} + (\mathbf{R}^{s,i} \odot \mathbf{W}^-)^T\mathbf{1} \tag{47}$$

where vectors $\boldsymbol{\sigma}_{W^+}$ and $\boldsymbol{\sigma}_{W^-}$ have already been defined in (36)-(37), vectors $\mathbf{m}_i^c$, $\mathbf{m}_i^r \in \mathbb{R}^{N\times 1}$ are the ith column and row of matrix $\mathbf{M}$, while the vector $\boldsymbol{\sigma}_{\overline{z}} \in \mathbb{R}^{N\times 1}$ is defined as

$$\boldsymbol{\sigma}_{\overline{z}} = \boldsymbol{\sigma}_q \odot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) - (\mathbf{M} \odot (\mathbf{W}^+)^T)\mathbf{1} + (\mathbf{M}^s \odot (\mathbf{W}^-)^T)\mathbf{1}. \tag{48}$$

The computational complexity of computing $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ is dominated by the computation of $\mathbf{\Gamma w}$. In this approach, we need to examine both the initialization cost and the cost per function evaluation. The initialization cost is dominated by the derivation of matrices $\mathbf{R}^{s,i}$, $i = 1, ..., N$ which is of computational complexity $O(KN^3)$. The computational cost per objective function or gradient evaluation involves the computation of vectors $\overline{\mathbf{z}}_{1i}$ and $\overline{\mathbf{z}}_{2i}$, for $i = 1, ..., N$ given by Eqs. (46) and (47), as well as vector $\boldsymbol{\sigma}_{\overline{z}}$ according to Eq. (48), which are of computational complexity $O(N^3)$, $O(N^3)$ and $O(N^2)$. Hence, the computational complexity of evaluating $\mathbf{\Gamma w}$ is also $O(N^3)$. If the derivation is performed naively, then the computation of $\mathbf{\Gamma} = \mathbf{B}^T\mathbf{B}$ and $\mathbf{\Gamma w}$ are of complexity $O(KN^5)$ and $O(N^4)$ respectively. Hence, our approach achieves an $O(N^2)$ complexity reduction of the initialisation phase and an $O(N)$ complexity reduction per objective function or gradient evaluation.

In terms of memory requirements, this approach requires the storage of $\boldsymbol{\sigma}_q$, $\mathbf{M}$, $\boldsymbol{M}^s$, $\mathbf{R}^i$ and $\mathbf{R}^{s,i}$ for $i = 1, ..., N$ apart from the necessary $\mathbf{W}^+$, $\mathbf{W}^-$ and $\mathbf{q}_k$, $k = 1, ..., K$. As each matrix $\mathbf{R}^i$ or $\mathbf{R}^{s,i}$ has N^2 elements, the total storage space required for this approach is $O(N^3 + KN)$ which is limiting for large values of N. As a result, this approach is more suitable for cases that $K > N$ and N is small enough so that we can store at least $2N^3 + KN$ elements. Notice that if the above matrices are not used, then $\mathbf{\Gamma}$ requires the storage of $4N^4$ elements.

In sum, the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ has time complexity $O(KN^3)$ for initialisation and $O(N^3)$ per evaluation, while it requires the storage of approximately $O(2N^3 + KN)$ elements.

VII. Simulation results

In this section we evaluate the performance of the developed RNN-NNLS and PGNNLS algorithms in the context of emergency management optimization by considering the problem of assigning emergency units to incidents (AEUI), a problem first discussed in [48].

A. Problem description

Consider that 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 $c_i \in \mathbb{N}^+$ injured and having response time to incident j given by $T_{ij} > 0$. We also assume that decisions are irrevocable so that after a unit is allocated to some incident, it cannot be re-assigned to some other incident.

The objective is to collect all injured at the minimum possible response time to ensure the quick collection and treatment of the civilians. The problem can be defined using mathematical programming as:

$$\min_{\mathbf{X}} \sum_{i=1}^{N_U}\sum_{j=1}^{N_L} T_{ij}X_{ij} \tag{49a}$$

$$\text{s.t.} \sum_{j=1}^{N_L} X_{ij} = 1,\ \forall i, \tag{49b}$$

$$\sum_{i=1}^{N_U} c_i X_{ij} \geq I_j,\ \forall j, \tag{49c}$$

$$X_{ij} \in \{0, 1\},\ \forall i, j. \tag{49d}$$

Constraint (49b) indicates that an emergency unit must be allocated to exactly one incident, while (49c) 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 allocation matrix $\mathbf{X}$ with elements X_{ij} indicates whether agent i has been allocated to incident j ($X_{ij} = 1$) or not ($X_{ij} = 0$). The above problem is NP-hard in the strong sense since it is a generalisation of the 0-1 Multiple Knapsack Problem which is of the same complexity class [49]. This means that no known algorithms exist to solve the problem in polynomial time. For this reason we rely upon heuristic algorithms that can provide fast and close to optimal solutions to the problem. In the next section we discuss a heuristic method based on supervised learning that will be used to obtain fast and decentralized decision making, as well as close to optimal results.

B. *Supervised learning solution approach*

The approach taken for the solution of AEUI problem is to train a random neural network using numerous instances of the optimisation problem with exact solutions which are obtained off-line. Then, if a problem instance is presented to the trained neural network, it will be able to provide a solution that is close to optimal, due to its generalization ability. As a result, the trained RNN can be "handed out" to all decision agents (emergency units) to serve as an "oracle" for decision making. When the emergency happens, each individual agent uses its "oracle" to obtain fast and decentralized decisions. Since all agents have the same "oracle", if they have the same information there will be no conflicts in their decisions; the "oracle" provides the same allocation matrix $\mathbf{X}$ to each agent, so that agent i is allocated to incident j' with $X_{ij'} = 1$.

C. *Training Architecture*

By fixing the T_{ij} and c_i parameters, the problem can be mapped to a supervised learning context by representing the inputs to the network by I_j and the outputs by X_{ij}. Because $I_j \geq 0 \ \forall j$, in the RNN they will be represented by the parameters Λ_j of the input neurons. The output variables are associated with the excitation probabilities of output neurons. Specifically, output neuron with index (i, j) represents decision variable X_{ij}. During the training phase, we represent decision $X_{ij} = 1$ with $q_{(i,j)} = 1 - \epsilon_q$, $0 \leq \epsilon_q < 1/2$ and decision $X_{ij} = 0$ with $q_{(i,j)} = \epsilon_q$. During the testing phase, if the value of the particular neuron is $q_{(i,j)} > 0.5$ then we assume that $X_{ij} = 1$, otherwise we take $X_{ij} = 0$.

With respect to the number of hidden neurons, we consider a configuration where the number of hidden neurons is twice the number of output neurons, i.e. $N_H = 2N_O$. Furthermore, we always assume that our network is fully connected in terms of the $\mathbf{W}^+$ and $\mathbf{W}^-$ weight matrices. For the solution of the problem we considered two general NN architectures. In the "collective" NN architecture we construct a single neural network for all decisions which is comprised of $N_O = N_U N_L$ output neurons. As the output of the neural network provides the actions for all agents, each agent only performs the action corresponding to him/her. In the "individual" NN architecture we construct and train a different NN for each agent's decision, so that we need to train N_U architectures of N_L output neurons. In this case, the ith NN is trained using as outputs only the variables $X_{ij}, j = 1, ..., N_L$ to advise agent i. Despite the fact that each NN provides a single action, decision making is still consistent because training is performed using the optimal solutions to the problem instances which are globally consistent.

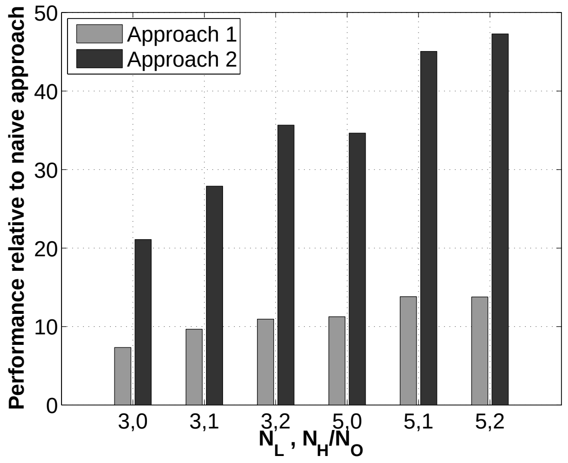
To train the NNs, we have first generated at random 1000 problem instances for different numbers of emergency units and locations of incidents. The remaining parameters have been chosen at random with $T_{ij} = U(0, 1)$ and $c_i = U^{int}(1, 4)$. For each problem instance, the number of injured at location I_j is also chosen from the uniform distribution such that $I_j = U^{int}(0.5c_t/N_L, c_t/N_L)$, where $c_t = \sum_i c_i$ is the total capacity of the emergency units.

To evaluate the developed approach 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 only chose those whose required capacity was within the total available capacity of the emergency units. The optimal solution in each case was then obtained accurately by solving the combinatorial optimisation problem in Matlab using function *bintprog* which employs a branch and bound procedure for the solution of binary combinatorial optimisation problems. Testing after training was performed using a distinct but similarly generated set of 250 test cases so that the training and testing were disjoint, but with the same probability distributions for all parameters. The effectiveness of the learning algorithms was evaluated on the basis of the following metrics:

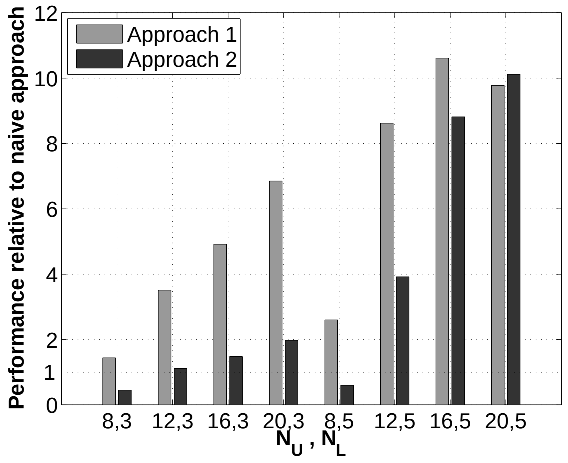
- The percentage of instances that were solved so that all of the injured were evacuated
- The percentage of people collected averaged over all testing instances
- The average relative percentage deviation from the optimal, σ_{opt},which evaluates the closeness of the solution to optimality, taken over the number of problem instances that the emergency units covered all casualties N_F, defined as:

$$\sigma_{opt} = \frac{1}{N_F} \sum_{i=1}^{N_F} \frac{f^i_{NN}(\mathbf{X}) - f^i_{opt}(\mathbf{X})}{f^i_{opt}(\mathbf{X})} \times 100 \tag{50}$$

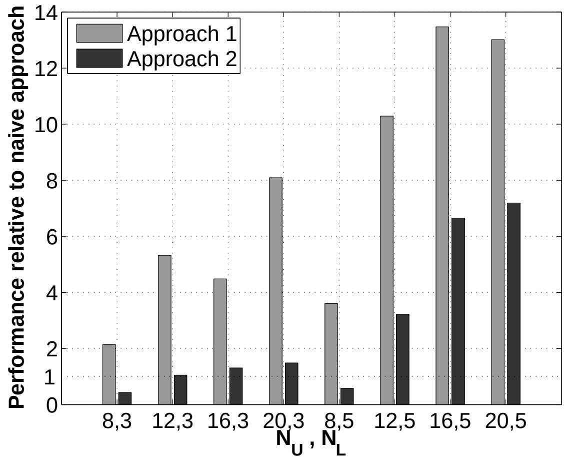
where $f^i_{NN}(\mathbf{X})$ and $f^i_{opt}(\mathbf{X})$ are the cost function values obtained from the heuristic neural network learning algorithm and the exact algorithm for instance i respectively.



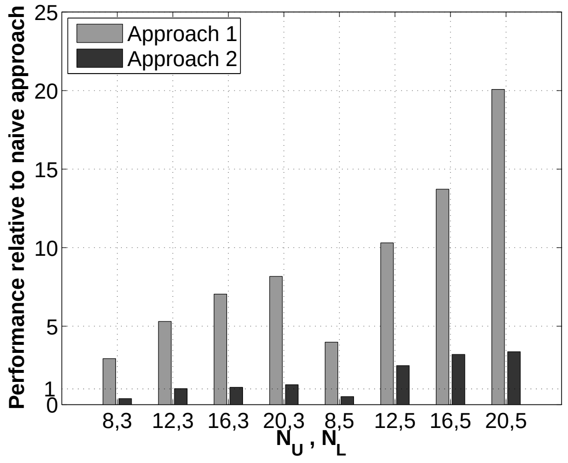
(a) “Inidividual” RNN



(b) “Collective” RNN with $N_H = 0$



(c) “Collective” RNN with $N_H = N_O$



(d) “Collective” RNN with $N_H = 2N_O$

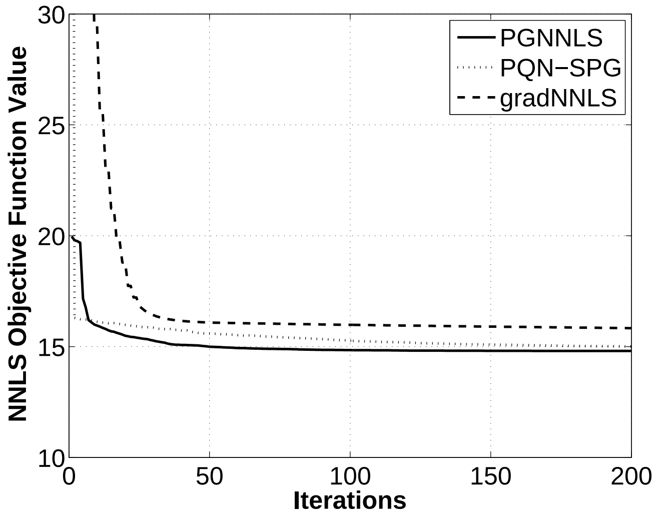
Fig. 1. Performance of approaches for computing the objective and gradient NNLS function compared to a “naive” one; the metric used is the ratio of execution times between the naive and another approach

D. Performance evaluation of PGNNLS

Before discussing the effectiveness of the RNN-NNLS algorithm for the solution of the investigated problem, results are presented concerning the computational efficiency and convergence speed of the developed PGNNLS algorithm. Specifically, the empirical computational performance of PGNNLS is examined for: (a) the fast computation of the objective and gradient NNLS functions, and (b) the convergence speed of PGNNLS compared to two other approaches.

To evaluate the efficiency of the two developed approaches for the computation of the costly NNLS functions in section VI, we have measured the execution time required for the evaluation of 200 $\mathbf{B}^T(\mathbf{B}\mathbf{w})$ operations which involve two matrix-vector products; the execution time also includes the initialisation time. To demonstrate the benefit from using these approaches, a “naive” method for the computation of these products was implemented, that takes into consideration the sparsity of $\mathbf{B}$, but performs no analytical manipulation.

Fig. 1, illustrates the execution time ratio of the “naive” against the developed approaches (speedup) for different RNN architectures. In the figures, Approaches 1 and 2 correspond to the computation methods discussed in Sections VI-B and VI-C, respectively. Fig. 1(a), shows the results of the “individual” RNN architecture when $N_L = \{3, 5\}$ and various ratios of hidden to output neurons. In this case the N_U parameter is not important as the size of the network for each emergency unit depends only on N_L. Because the constructed neural network for the “individual” architecture is small, Approach 2 is significantly better than Approach 1, while both approaches have an order of magnitude speedup compared to the “naive” implementation. In fact, Approach 2 reaches an overall speedup of fifty for $N_L = 5$ and $N_H/N_O = 2$. On the contrary, for the “collective” architecture the number of neurons is significantly larger that the “individual” one, which is in favour of Approach 1. Indeed, this is verified by the results which show that Approach 1 is better than Approach 2 by up to seven times. Also as the network size increases, with the addition of more hidden neurons, Approach 1 becomes more efficient and Approach 2 less efficient.



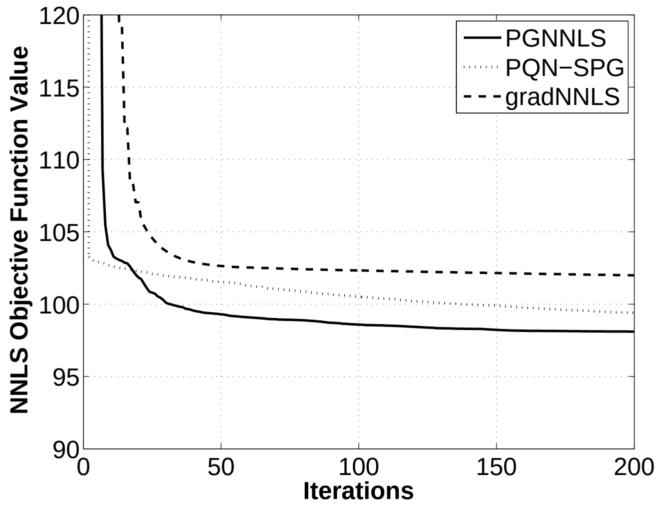
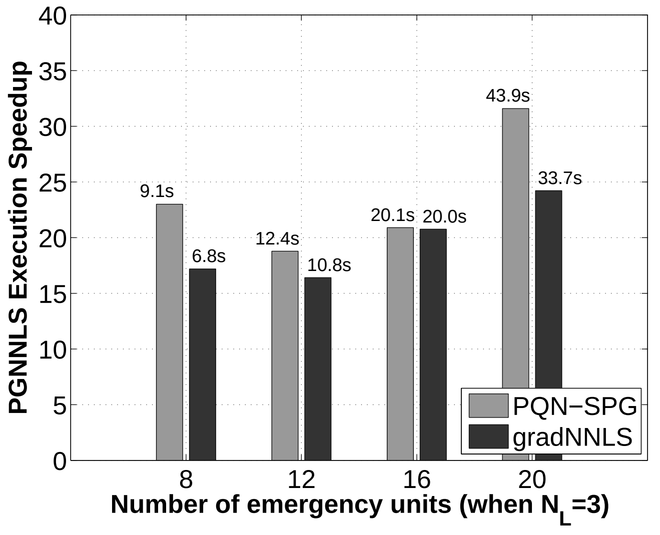


Fig. 2. Comparison of iteration convergence between algorithms gradNNLS, PQN-SPG and PGNNLS for N_U = 20 and a) $N_L = 3$ and b) NL=5.



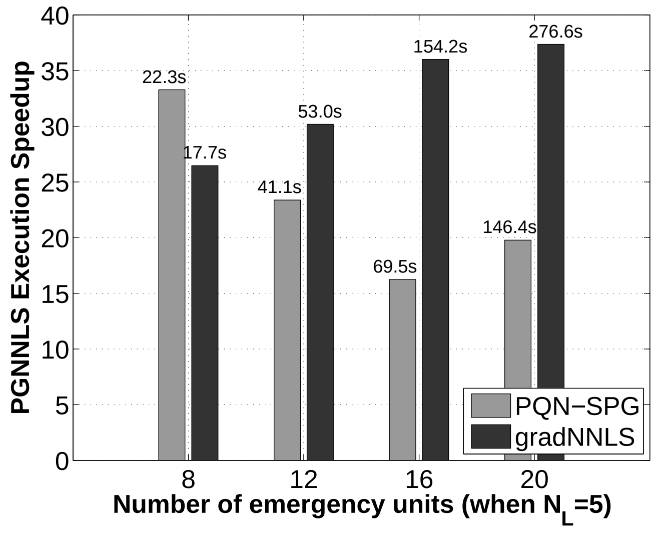


Fig. 3. PGNNLS execution speed compared to gradNNLS and PQN-SPG for N_U = 20 and a) $N_L = 3$ and b) NL=5; the number on each bar indicates the execution time of the corresponding algorithm.

These results show that both architectures are useful, as they perform better under different conditions, while they both provide a significant speedup over a "naive" implementation, as discussed in the derivation of these approaches.

For the PGNNLS algorithm, we have chosen to perform 200 iterations with five correction vectors ($M = 5$) and hyper-exponential line search with parameters $\beta = 0.4$, $\sigma_{APA} = 0.25$ and $\mathbf{S}_0^\tau = \mathbf{I}$, $\forall\tau$ (see Eqs. (24) and (23)). To examine the efficiency of the proposed PGNNLS algorithm, we have compared its convergence in terms of iterations and execution time with two other algorithms, gradNNLS [28], and PQN-SPG [44]. The former is a projected gradient algorithm with first-order information ($\mathbf{S}^\tau = \mathbf{I}$) which was developed in conjuction with the introduction of the NNLS learning approach for RNN. The latter is a state-of-the-art limited-memory projected quasi-Newton algorithm for box constrained problems, such as NNLS. It is evident from Fig. 2 that the PGNNLS algorithm outperforms gradNNLS and PQN-SPG both in terms of the attained NNLS objective function value and the number of iterations to achieve convergence. In fact, for $N_L = 5$ where the training architecture is larger, the attained results are even better compared to the two other algorithms.

Fig. 3 depicts the execution speedup attained by the PGNNLS algorithm compared to the two other algorithms to attain the objective NNLS value achieved by running PGNNLS for 200 iterations. In all cases examined the speedup compared to PQN-SPG and gradNNLS is in the range $[\times 16, \times 34]$ and $[\times 16, \times 37]$, respectively. Interestingly, Fig. 2 indicates that gradNNLS convergences faster for smaller problems (with $N_L = 3$), while for larger problems PQN-SPG is up to 2.5 times faster. Note that all three NNLS algorithms have employed Approach 1 for the evaluation of the costly NNLS functions, which implies that the combined benefit obtained from the developed approach is more that two orders of magnitude.

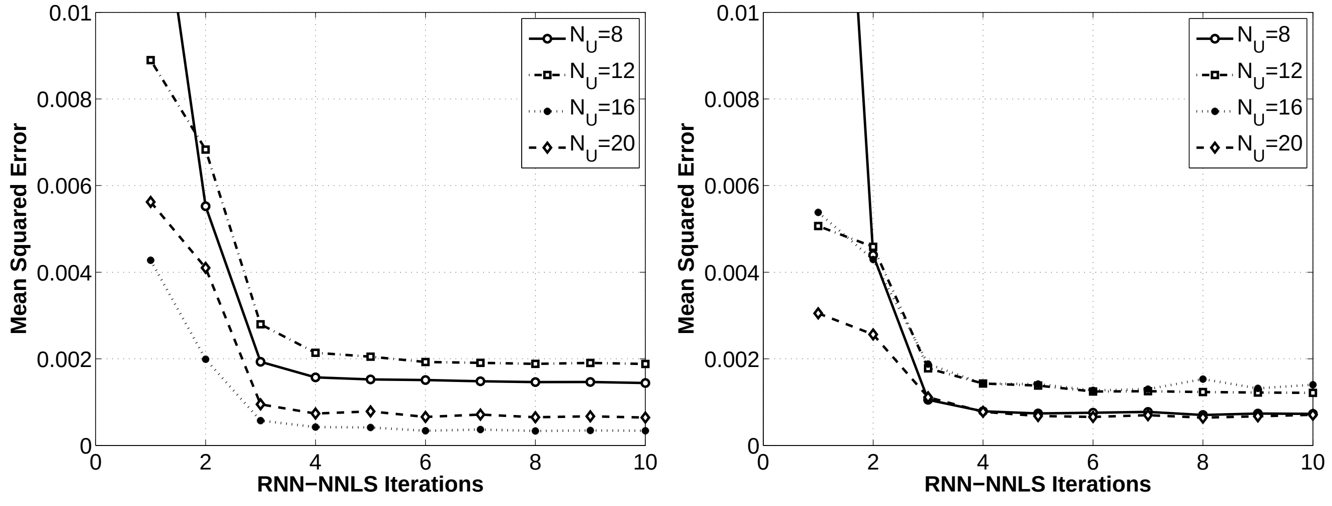


Fig. 4. Convergence of RNN-NNLS algorithm for $N_L = 3$ and $N_L = 5$.

E. Solving the AEUI problem

In this section the performance of the RNN-NNLS algorithm for the solution of the AEUI problem is evaluated. To solve AEUI using the RNN-NNLS learning algorithm, we have employed the algorithm developed in IV. Specifically, we perform ten iterations ($NI_{RNN-NNLS} = 10$), checking the solution quality after each iteration and storing the weights corresponding to the largest percentage of instances where all injured were collected. For updating the desired values of the non-output and output weights we have set $\alpha_{no} = 0.75$ and $\alpha_o = 0.9$ respectively. We have also normalised the inputs of the RNN so that $\Lambda_i \in [0.2, 1]$, while for the output neurons we have chosen $\epsilon_q = 1/3$, so that "low" and "high" neurons take values $1/3$ and $2/3$ respectively. The initial desired excitation probabilities of the non-output neurons are generated according to $U(0.25, 0.75)$.

Fig. 4 depicts the mean squared error (MSE) with respect to the desired and attained excitation probabilities for the output neurons for ten iterations of the RNN-NNLS algorithm. It is clear that the MSE error decreases for subsequent iterations leading to the converge of the algorithm. In fact, stabilisation of the MSE is accomplished after a very small number of iterations (around five). Although monotonic convergence cannot be guaranteed, the observed behaviour is sufficient to produce good trained weights that will derive high quality solutions.

Finally, Fig. 5 summarizes the best results for the "collective" and "individual" NN architectures of the RNN-NNLS and RNN approaches. It is evident that the "collective" RNN-NNLS algorithm yields the best results in terms of percentage of instances were all injured were collected, as it is the most effective for $N_L = 5$ and highly competitive for $N_L = 3$. On the other hand, the "individual" RNN architecture produces the best results in terms of deviation from the optimal but has the worst performance in terms of the other two metrics. Among the other three architectures the "collective" RNN is the one with the best performance in terms of σ_{opt} but it is not as effective in collecting injured, especially for larger problems.

VIII. Conclusions

In this work, we have studied nonnegative least squares learning in the context of the random neural network. For this problem, a solution algorithm has been developed that achieves two orders of magnitude speedup compared to other state-of-the-art approaches. The speedup is a result of a customized limited-memory quasi-Newton method for the solution of the problem, as well as two efficient approaches for the computation of the gradient and objective functions that appear in the problem. Apart from improved learning efficiency, the developed approach has been shown to provide very good results for the solution of an emergency management optimization problem.

References

[1] E. Gelenbe, "Random Neural Networks with Negative and Positive Signals and Product Form Solution," *Neural Computation*, vol. 1, no. 4, pp. 502–510, 1989.

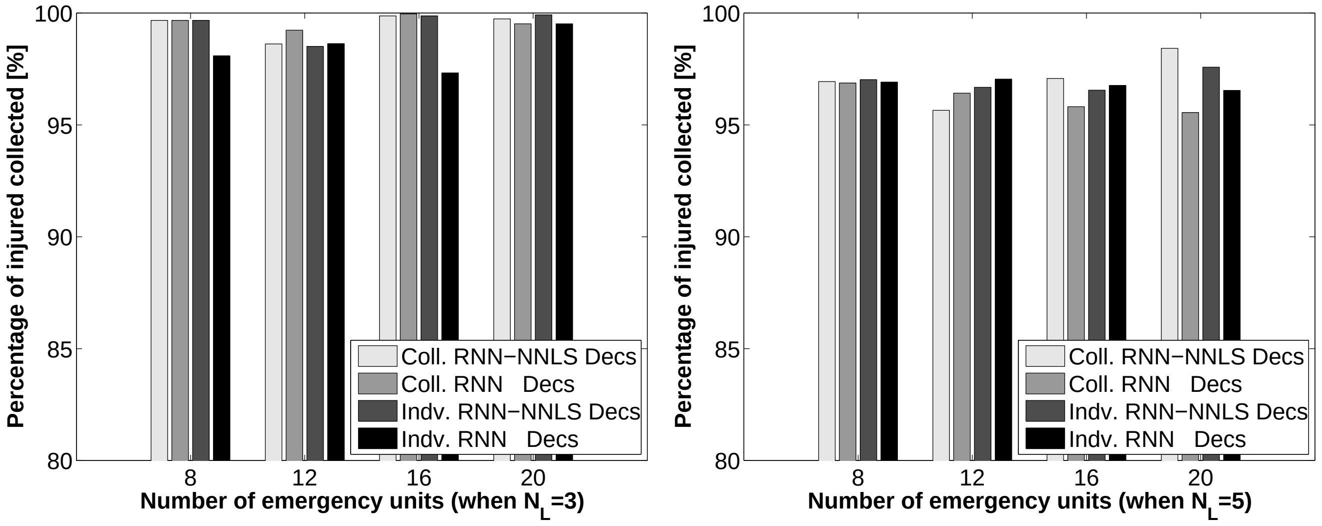
[2] E. Gelenbe, "Stability of the random neural network," *Neural Computation*, vol. 2, no. 2, pp. 239–247, 1990.

[3] E. Gelenbe, "Theory of the Random Neural Network," in *NEURAL NETWORKS: Advances and Applications* (E. Gelenbe, ed.), pp. 1–20, Elsevier Science Publishers B.V., 1991.

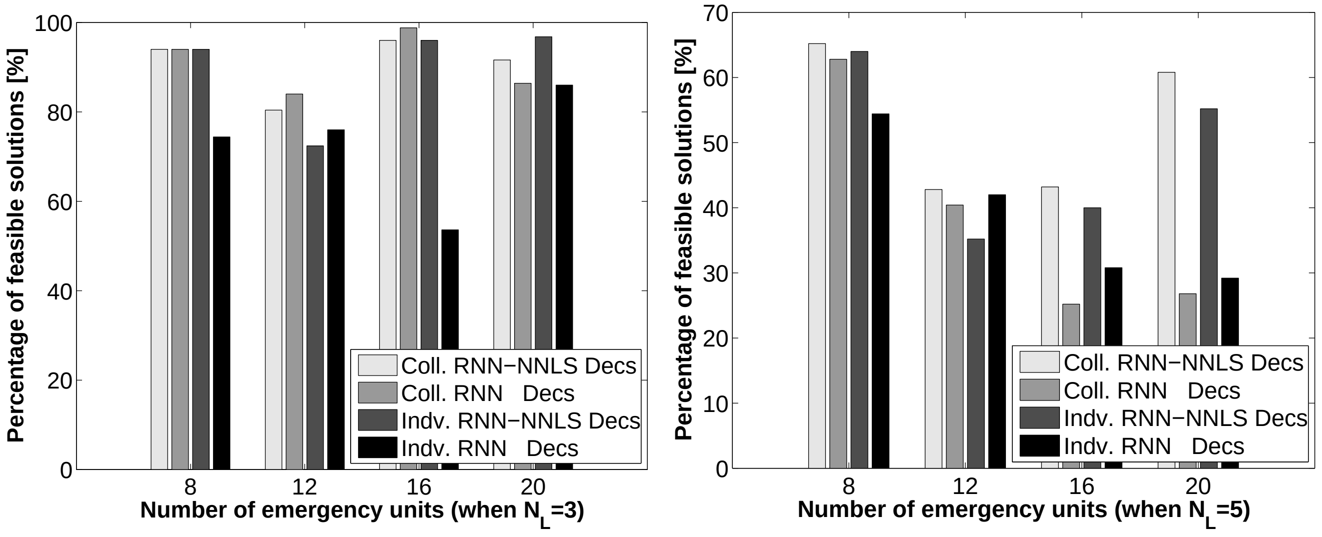
[4] E. Gelenbe, A. Ghanwani, and V. Srinivasan, "Improved neural heuristics for multicast routing," *IEEE Journal of Selected Areas of Communications*, vol. 15, no. 2, pp. 147–155, 1997.

[5] E. Gelenbe, S. Timotheou, and D. Nicholson, "Fast distributed near-optimum assignment of assets to tasks," *The Computer Journal*, vol. 53, no. 9, pp. 1360–1369, 2010.

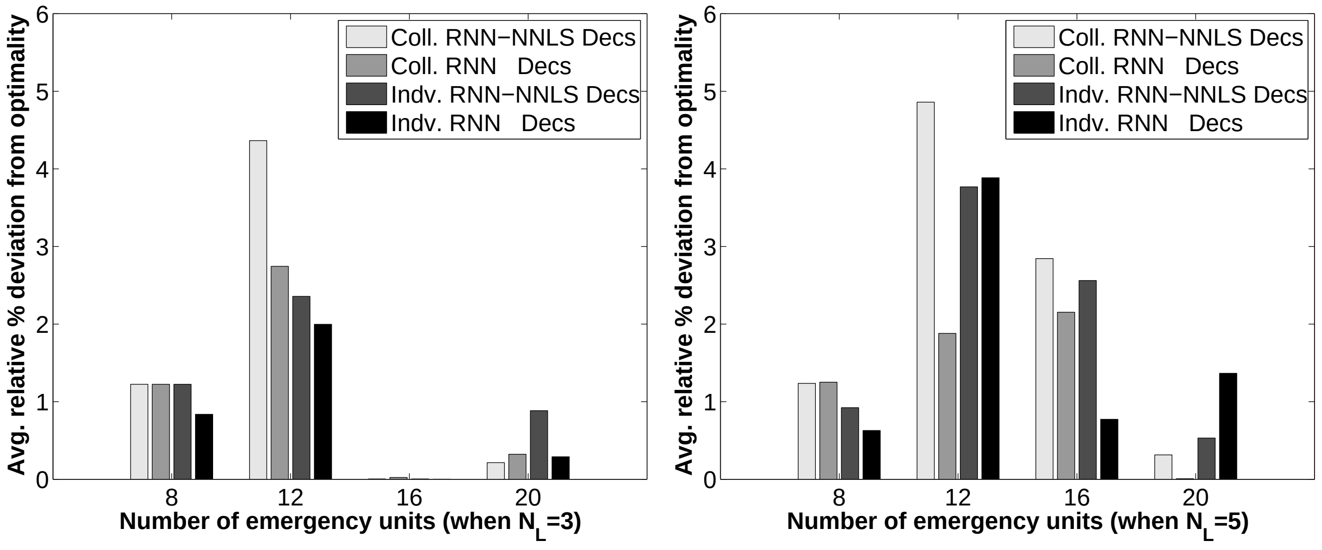
[6] J. Aguilar and E. Gelenbe, "Task assignment and transaction clustering heuristics for distributed systems," *Information Sciences – Informatics and Computer Science*, vol. 97, no. 1 & 2, pp. 199–221, 1997.



(a) Percentage of injured that are collected



(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term



(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

Fig. 5. Comparison between the RNN-NNLS and RNN learning algorithms. The four architectures considered are: (a) "Collective" RNN-NNLS, (b) "Collective" RNN, (c) "Individual" RNN-NNLS, and (d) "Individual" RNN

[7] E. Gelenbe and Q. Han, “Near-optimal emergency evacuation with rescuer allocation,” in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pp. 314–319, IEEE, 2014.
[8] A. N. E. Gelenbe, R. Lent, “Self-aware networks and QoS,” *Proceedings of the IEEE*, vol. 92, no. 9, pp. 1478 – 1489, 2004.
[9] E. Gelenbe, “Product-Form Queueing Networks with Negative and Positive Customers,” *Journal of Applied Probability*, vol. 28, no. 3, pp. 656–663, Sep. 1991.
[10] E. Gelenbe, “G-networks: a unifying model for neural and queueing networks,” *Annals of Operations Research*, vol. 48, no. 5, pp. 433–461, 1994.
[11] E. Gelenbe, “G-networks with triggered customer movement,” *Journal of Applied Probability*, vol. 30, no. 3, pp. 742–748, 1993.
[12] J. R. Artalejo, “G-networks: A versatile approach for work removal in queueing networks,” *European Journal of Operational Research*, vol. 126, no. 2, pp. 233–249, 2000.
[13] E. Gelenbe, “Steady-state solution of probabilistic gene regulatory networks,” *Physical Review E*, vol. 76, no. 1, p. 031903, 2007.
[14] H. Phan, M. Stemberg, and E. Gelenbe, “Aligning protein-protein interaction networks using random neural networks,” in *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 1–6, Oct 2012.
[15] K. Hussain and G. S. Moussa, “Laser intensity vehicle classification system based on random neural network,” in *Proceedings of the 43rd Annual Southeast Regional Conference, Kennesaw, Georgia, Alabama, USA, 18-19 March*, pp. 31–35, ACM, New York, 2005.
[16] E. Gelenbe and T. Kocak, “Wafer surface reconstruction from top-down scanning electron microscope images,” *Microlectronic Engineering*, vol. 75, pp. 216–233, 2004.
[17] H. Abdelbaki, E. Gelenbe, and T. Kocak, “Neural Algorithms and Energy Measures for EMI Based Mine Detection,” *Journal of Differential Equations and Dynamical Systems*, vol. 13, no. 1-2, pp. 63–86, 2005.
[18] G. Oke and G. Loukas, “A Denial of Service Detector based on Maximum Likelihood Detection and the Random Neural Network,” *The Computer Journal*, vol. 50, no. 6, pp. 717–727, 2007.
[19] E. Gelenbe, “Learning in the recurrent random network,” *Neural Computation*, vol. 5, pp. 154–164, 1993.
[20] E. Gelenbe and K. Hussain, “Learning in the multiple class random neural network,” *IEEE Trans. on Neural Networks*, vol. 13, no. 6, pp. 1257–1267, 2002.
[21] E. Gelenbe and S. Timotheou, “Synchronised Interactions in Spiked Neuronal Networks,” *The Computer Journal*, vol. 51, no. 4, pp. 723–730, 2008.
[22] M. Georgiopoulos, C. Li, and T. Kocak, “Learning in the feed-forward random neural network: A critical review,” *Performance Evaluation*, vol. 68, no. 4, pp. 361–384, 2011.
[23] A. Romariz and E. Gelenbe, “Contrastive learning in random neural networks and its relation to gradient-descent learning,” in *Computer and Information Sciences II* (E. Gelenbe, R. Lent, and G. Sakellari, eds.), pp. 511–517, Springer London, 2012.
[24] A. Likas and A. Stafylopatis, “Training the random neural network using quasi-Newton methods,” *European Journal of Operational Research*, vol. 126, no. 2, pp. 331–339, 2000.
[25] S. Basterrech, S. Mohammed, G. Rubino, and M. Soliman, “Levenberg-marquardt training algorithms for random neural networks,” *The Computer Journal*, vol. 54, no. 1, pp. 125–135, 2011.
[26] S. Timotheou, “The random neural network: a survey,” *The computer journal*, vol. 53, no. 3, pp. 251–267, 2010.
[27] S. Timotheou, “Nonnegative Least Squares Learning for the Random Neural Network,” in *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN 2008), Prague, Czech Republic, 3-6 September*, pp. 195–204, Springer-Verlag, Berlin, Heidelberg, 2008.
[28] S. Timotheou, “A novel weight initialization method for the random neural network.” to appear in *Neurocomputing*, 2009.
[29] F. Biegler-Konig and F. Barmann, “A learning algorithm for multilayered neural networks based on linear least squares problems,” *Neural Networks*, vol. 6, no. 1, pp. 127–131, 1993.
[30] E. Castillo, O. Fontenla-Romero, B. Guijarro-Berdinas, and A. Alonso-Betanzos, “A global optimum approach for one-layer neural networks,” *Neural Computation*, vol. 14, no. 6, pp. 1429–1449, 2002.
[31] D. Erdogmus, O. Fontenla-Romero, J. Principe, A. Alonso-Betanzos, and E. Castillo, “Linear-least-squares initialization of multilayer perceptrons through backpropagation of the desired response,” *IEEE Transactions on Neural Networks*, vol. 16, no. 2, pp. 325–337, 2005.
[32] Y. F. Yam, T. W. S. Chow, and C. T. Leung, “A new method in determining initial weights of feedforward neural networks for training enhancement,” *Neurocomputing*, vol. 16, no. 1, pp. 23 – 32, 1997.
[33] J. Y. F. Yam and T. W. S. Chow, “A weight initialization method for improving training speed in feedforward neural network,” *Neurocomputing*, vol. 30, no. 1-4, pp. 219–232, 2000.
[34] O. Fontenla-Romero, D. Erdogmus, J. C. Principe, A. Alonso-Betanzos, and E. Castillo, “Linear least-squares based methods for neural networks learning,” in *Proceedings of the International Conference on Artificial Neural Networks and Neural Information Processing, Istanbul, Turkey, June 26-29*, (Heidelberg), pp. 84–91, Springer-Verlag, 2003.
[35] B. Guijarro-Berdinas, O. Fontenla-Romero, B. Perez-Sanchez, and P. Fraguela, “A fast semi-linear backpropagation learning algorithm,” in *Proceedings of the International Conference on Artificial Neural Networks, Porto, Portugal, September 9-13*, (Heidelberg), pp. 190–198, Springer-Verlag, 2007.
[36] Siu-yeung Cho and Tommy W. S. Chow, “Training multilayer neural networks using fast global learning algorithm - least-squares and penalized optimization methods,” *Neurocomputing*, vol. 25, no. 1-3, pp. 115 – 131, 1999.
[37] E. Castillo, B. Guijarro-Berdinas, O. Fontenla-Romero, and A. Alonso-Betanzos, “A very fast learning method for neural networks based on sensitivity analysis,” *Journal of Machine Learning Research*, vol. 7, pp. 1159–1182, 2006.
[38] G. Huang, G.-B. Huang, S. Song, and K. You, “Trends in extreme learning machines: A review,” *Neural Networks*, vol. 61, pp. 32–48, 2015.
[39] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. Prentice Hall, 1987.
[40] A. Dax, “On computational aspects of bounded linear least squares problems,” *ACM Transactions on Mathematical Software*, vol. 17, pp. 64–73, Mar. 1991.
[41] R. Bro and S. D. Long, “A fast non-negativity-constrained least squares algorithm,” *Journal of Chemometrics*, vol. 11, no. 5, pp. 393–401, 1997.
[42] D. Kim, S. Sra, and I. Dhillon, “A new projected quasi-newton approach for the nonnegative least squares problem,” tech. rep., Dept. of Computer Sciences, The University of Texas at Austin, Dec. 2006.
[43] D. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1995.
[44] M. Schmidt, E. van den Berg, M. P. Friedlander, and K. Murphy, “Optimizing Costly Functions with Simple Constraints: A Limited-Memory Projected Quasi-Newton Algorithm,” in *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS), Florida, US 16-18 April 2009*, pp. 456–463, 2009.
[45] R. H. Byrd, J. Nocedal, and R. B. Schnabel, “Representations of quasi-Newton matrices and their use in limited memory methods,” *Mathematical Programming*, vol. 63, pp. 129–156, 1994.
[46] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer-Verlag, 1999.
[47] P. Calamai and J. Moré, “Projected gradient methods for linearly constrained problems,” *Mathematical Programming*, vol. 39, pp. 93–116, 1987.
[48] E. Gelenbe and S. Timotheou, “Random Neural Networks with Synchronised Interactions,” *Neural Computation*, vol. 20, pp. 2308–2324, 2008.
[49] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.