A Novel Weight Initialization Method for the Random Neural Network

Stelios Timotheou

Intelligent Systems and Networks Group
Department of Electrical and Electronic Engineering
Imperial College
London SW7 2BT, UK
{stelios.timotheou}@imperial.ac.uk

Abstract. In this paper, we propose a novel weight initialization method for the Random Neural Network. The method relies on approximating the signal-flow equations of the network to obtain a linear system of equations with nonnegativity constraints. For the solution of the formulated linear Nonnegative Least Squares problem we have developed an improved projected gradient algorithm. It is shown that supervised learning with the developed initialization method has better performance than learning with random initialization for a combinatorial optimization emergency response problem.

1 Introduction

Neural network training is usually approached as a nonlinear optimization problem to minimize an error function, and requires the solution of a hard global optimization problem. Due to the large dimensionality of the problem, iterative gradient descent optimization techniques are often employed. However, these techniques lead to a solution corresponding to a local minimum of the problem. The quality of the local solution is determined not only by the optimization technique employed but by the weight initialization as well. Hence, developing efficient weight initialization techniques is prominent to the determination of a good solution.

In this paper, we propose a novel weight initialization technique for the Random Neural Network (RNN). RNN is a bio-inspired open recurrent neural network reflecting the pulsed behaviour observed in natural neuronal networks [1, 2]. It has a product-form stationary probability distribution while its signal-flow equations are of closed-form and analytically solvable.

A gradient descent supervised learning algorithm for the RNN was introduced in [3] and has been successfully exploited in many applications such as optimization [4], image processing [5] and defending communication networks against denial of service attacks [6]. For a survey of applications see [5].

The RNN is also a prominent modeling tool representing the behaviour of interacting entities. In queueing networks, RNN has inspired a great amount of research related to work removal, resulting in numerous extensions of the

standard Jackson networks, which are called *G-networks* [7]. An extension of RNN have also been used to model the behaviour of interacting agents in gene regulatory networks [8]. In addition, a model and a learning algorithm have been developed for the representation of synchronized firing effects in neuronal networks [9, 10]. Therefore, due to the importance of the RNN both as a neural network and as a modeling tool new learning algorithms or weight initialization techniques are needed to facilitate the learning procedure.

Weight initialization techniques have been extensively examined in feed-forward connectionist neural networks. Some approaches rely on defining an optimal distribution and range for the weights either empirically or based on characteristics of the neurons. In [11] these approaches were compared in 8 benchmark problems and it was concluded that the method proposed in [12] was the most effective. Moreover, a number of techniques rely on either neglecting the scaling effect or approximating the sigmoidal activation function to obtain linear systems of equations for different layers of neurons, which can be solved using least squares techniques [13, 14]. Our method differs from existing techniques because it is developed for a recurrent network using a different approximation approach and with a least squares method applied to the whole network.

The paper is organized as follows: Firstly, a brief presentation of the RNN and its supervised learning algorithm is provided (Section 2). The proposed initialization approach is then analyzed (Section 3). Next, a performance comparison is made between our weight initialization method and random initialization when both schemes are used to facilitate RNN supervised learning (Section 4). Following, is a summary of the conclusions (Section 5).

2 The Random Neural Network

2.1 Mathematical model

RNN is a network of N fully connected neurons. The state of neuron i is described by the potential of the neuron $k_i(t) \geq 0$ as well as the probability of being excited $q_i = Prob[k_i(t) > 0]$; the neuron is quiescent or idle if $k_i(t) = 0$. Neurons exchange excitatory spikes, which increase the internal state of the receiving neuron by 1, and inhibitory spikes, which reduce $k_i(t)$ by 1 if the receiving neuron is excited. Neuron i receives excitatory or inhibitory spikes from the outside world according to Poisson processes of rates Λ_i and λ_i. An excited neuron i, can fire a spike at time t with an independent exponentially distributed firing rate r_i. If this happens $k_i(t^+) = k_i(t) - 1$. The resulting spike reaches neuron j as an excitatory spike with probability $p^+(i,j)$ or as an inhibitory spike with probability $p^-(i,j)$, or it departs from the network with probability $d(i)$. Excitatory or inhibitory spikes arriving at neuron j from another neuron i are treated by j exactly in the same way as if such spikes arrived from the outside world. The probabilities must sum up to 1 and hence $\sum_{j=1}^{N} [p^+(i,j) + p^-(i,j)] + d(i) = 1$. The signal flows in

the network are described by the following system of nonlinear equations:

$$q_i = \min\left(1, \frac{\lambda^+(i)}{r_i + \lambda^-(i)}\right) = \min\left(1, \frac{\Lambda_i + \sum_{j=1}^{N} r_j q_j p^+(j,i)}{r_i + \lambda_i + \sum_{j=1}^{N} r_j q_j p^-(j,i)}\right),\ \forall\ i \quad (1)$$

It has been proven that a solution to the nonlinear system of equations (1) always exists and it is unique[1–3], whilst the stationary probability distribution of the system is of product form if $q_i < 1$.

2.2 Gradient descent learning in RNN

A gradient descent supervised learning algorithm for the recurrent RNN has been developed in [3]. In the RNN, the $k-th$ input training pattern $\mathbb{X}_k$ is represented by the vectors $\underline{\Lambda}_k = [\Lambda_{1k},\ \ldots\ , \Lambda_{Nk}]$ and $\underline{\lambda}_k = [\lambda_{1k},\ \ldots\ , \lambda_{Nk}]$. The positive and negative values of the input pattern are initialized using the excitatory $\underline{\Lambda}_k$ and inhibitory $\underline{\lambda}_k$ external arrival rates respectively. The desired values of the k-th pattern $\underline{y_k}$ are represented by the steady-state values of the network, such as the N probabilities $\underline{q}_k = [q_{1k},\ \ldots\ , q_{Nk}]$ emanating from applying an input training pattern to the network. The weights of the RNN that are updated during the learning phase are the excitatory and inhibitory firing rates of signals from neuron i to neuron j, $w^+(i,j)=r_i p^+(i,j) \geq\ 0$ and $w^-(i,j) = r_i p^-(i,j) \geq\ 0$ respectively. Furthermore, because the sum of probabilities associated with neuron i must sum up to 1 the following expression about r_i is derived:

$$r_i = \sum\nolimits_{j=1}^{N} [w^+(i,j) + w^-(i,j)]/(1 - d(i)) \quad (2)$$

The error function to be minimized is $E = \sum_k E_k$ where E_k corresponds to the quadratic error function of the $k-th$ input-output pair, $E_k = 1/2 \sum_{i=1}^{N} c_i (q_{ik} - y_{ik})^2$, $c_{ik} \geq 0$. If, a particular neuron i is not considered as an output then c_i can be set to 0. The approach taken in [3] is to sequentially process the patterns and update the weights of the network. Thereby, by denoting by the generic term $w(u,v)$ any weight of the network, the rule for updating the weights of the $k-th$ input-output pair at the $n-th$ step is:

$$w^{n+1}(u,v) = w^n(u,v) - \eta \frac{\partial E_k}{\partial w(u,v)} = w^n(u,v) - \eta \sum_{i=1}^{N} c_i\,(q_{ik} - y_{ik})\,\frac{\partial q_i}{\partial w(u,v)} \quad (3)$$

where q_{ik} are calculated by solving the system of equations (1) using the values $w^n(u,v)$ and values for $\underline{\Lambda}_k$ and $\underline{\lambda}_k$ corresponding to the particular input pattern.

The challenging step in the derivation of the updated weights is the calculation of the partial derivative $\partial q_i\ /\ \partial w(u,v)$ because it depends on the nonlinear equations (1). After algebraic calculation an expression for the aforementioned quantity yields the following easily computable and efficient ($O(N^3)$ complexity) expession:

$$\frac{\partial \underline{q}}{\partial w(u,v)} = \underline{\gamma}(u,v)\,(\mathbb{I} - \mathbb{B})^{-1},\ \mathbb{B}, \mathbb{I} \in \mathbb{R}^{N \times N},\ \underline{\gamma}(u,v) \in \mathbb{R}^{1 \times N}\ \forall\ u,v \quad (4)$$

where $\mathbb{I}$ is the identity matrix; $\mathbb{B}$ and $\underline{\gamma}(u,v)$ are computed at every iteration of the learning algorithm according to the current values of the network parameters.

The gradient descent RNN learning algorithm can now be outlined; following the initialization of the weights the input-output patterns are sequentially presented to the network. To update the weights, the nonlinear system of equations (1) is solved and the values obtained are utilized to calculate the new weights according to (3) and (4). The procedure is repeated for all patterns and several times until convergence. Hence the complexity of the algorithm is $\#\text{iterations} \times O(KN^3)$.

3 The proposed weight initialization method

In this section, we propose a novel weight initialization method for the RNN. The approach relies on the fact that if $0 \leq q_{ik} < 1 \; \forall \; i,k$ then equations (1) and (2) for the $k-th$ input-output pair yield:

$$\begin{array}{l} \frac{q_{ik}}{1-d(i)} \left(\sum_{j=1}^{N} w^{-}(i,j) + \sum_{j=1}^{N} w^{+}(i,j) \right) + q_{ik} \sum_{j=1}^{N} q_{jk} w^{-}(j,i) \\ - \sum_{j=1}^{N} q_{jk} w^{+}(j,i) = \Lambda_{ik} - q_{ik}\lambda_{ik}, \; \forall i, \; k \text{ and } w^{\pm}(i,j) \; \geq \; 0 \; \forall \; i, \; j \end{array} \tag{5}$$

Close observation of the above equations shows that if all q_{ik} are known then the above system is a linear system composed of NK equations with $2N^2$ nonnegative unknowns; the unknowns are the weights $w^{+}(i,j)$ and $w^{-}(i,j)$.

A unique solution to system (5) may not be available for two reasons. First, the number of equations may be larger that the number of unknowns ($K > 2N$). Second, the present constraints restrict the values of the variables and a solution may not exist even if ($K < 2N$). As a result we formulate the system (5) as a Nonnegative Least Squares (NNLS) problem in order to approach equality as much as possible in the least square sense.

$$\min_{\underline{w} \geq 0} f(\underline{w}) \; = \; \min_{\underline{w} \geq 0} 1/2 \| \mathbb{A}\underline{w} - \underline{b} \|_2^2, \quad \mathbb{A} \in \mathbb{R}^{NK \times 2N^2}, \; \underline{b} \in \mathbb{R}^{NK \times 1}, \; \underline{w} \in \mathbb{R}^{2N^2 \times 1} \tag{6}$$

The gradient of $f(\underline{w})$ is $\nabla f(\underline{w}) \; = \; \mathbb{A}^T(\mathbb{A}\underline{w}) - \mathbb{A}^T\underline{b}$. Under the assumption that matrix $\mathbb{A}$ is a full rank matrix, NNLS is a strictly convex quadratic optimization problem which has a unique optimal solution $\underline{w}^*$. Matrix $\mathbb{A}$ and vector $\underline{b}$ are given by the following expression:

$$\begin{array}{lll} \mathbb{A}(ik, ij^{+}) & = q_{ik}, & j = 1,...,N, \; j \neq i \\ \mathbb{A}(ik, ij^{-}) & = q_{ik}, & j = 1,...,N, \; j \neq i \\ \mathbb{A}(ik, ji^{-}) & = q_{ik}q_{jk}, & j = 1,...,N, \; j \neq i \\ \mathbb{A}(ik, ji^{+}) & = -q_{ik}, & j = 1,...,N, \; j \neq i \quad \forall \; i,k \\ \mathbb{A}(ik, ii^{+}) & = 0, & j = i \\ \mathbb{A}(ik, ii^{-}) & = q_{ik} + q_{ik}^2, & j = i \\ \mathbb{A}(ik, otherwise) & = 0, & \end{array} \tag{7}$$

$$b_{ik} = \Lambda_{ik} - q_{ik}\lambda_{ik}, \quad \forall \; i,k \tag{8}$$

The row index ik denotes the row corresponding to the $i-th$ equation of system (1) $k-th$ training pair, while the column indices of $\mathbb{A}$, ij^+ and ij^-, indicate the position of the variables $w^+(i,j)$ and $w^-(i,j)$ in $\underline{w}$ respectively. Observing an $\mathbb{A}$ row reveals that albeit composed of $2N^2$ elements, only $4N-2$ of them are nonzero, making $\mathbb{A}$ highly sparse. Furthermore, the value of every element of $\mathbb{A}$ can be found by only using vector $\underline{q} \in \mathbb{R}^{NK\times 1}$ which holds the q_{ik} values. Thus, simple matrix-vector multiplications can be performed without explicitly storing $\mathbb{A}$ and efficiently. For example, the calculation of $\mathbb{A}\underline{w}$ requires approximately $4KN^2$ multiplications and hence it is of complexity $O(KN^2)$.

The large dimensionality of $\mathbb{A}$ implies that it may not be possible to be stored in memory. Moreover, initial experimentation showed that $\mathbb{A}$ is ill-conditioned. Therefore, the developed approach for the solution of the NNLS problem must not require the storage of any large matrix or any matrix inversion.

The algorithms proposed for the solution of the NNLS problem can generally be classified into *active set algorithms* and *iterative approaches*. In active set algorithms [15, 16] an iterative procedure is followed which involves the solution of unconstrained least squares problems and separation of variables that take positive values from the rest. These approaches require matrix inversions and are not appropriate for our case.

Nonetheless, iterative methods adhere to nonlinear optimization methods to update $\underline{w}$. Usually the update of the current solution is based on *projected gradient methods* which can identify several active set constraints at one iteration.

$$\underline{w}^{\tau+1} = P[\underline{w}^{\tau} - s^{\tau}\mathbb{D}^{\tau}\nabla f(\underline{w})], \quad s^{\tau} \geq 0,\ \mathbb{D}^{\tau} \in \mathbb{R}^{2N^2\times 2N^2} \tag{9}$$

$$P[w_i] = \begin{cases} w_i, & w_i > 0 \\ 0, & w_i \leq 0 \end{cases} \tag{10}$$

Projected gradient methods differ in the selection procedure of the step size s^{τ} and the gradient scaling matrix $\mathbb{D}^{\tau}$, but they generally require simple matrix vector operations and can perform well for ill-conditioned problems. $\mathbb{D}^{\tau}$ is computed based on second order gradient information and results in fast convergence; nonetheless, it needs storage and will not be utilized.

We have developed a first order projected gradient NNLS algorithm, which is an improvement of the one proposed by Lin [17]. Lin employed an efficient modification of the "Armijo rule along the projection arc" (APA) [18] for finding an optimal update value for the step size s^{τ}. In each iteration of APA, s^{τ} starts from a relatively large value and exponentially decays until condition (11) is satisfied. Lin's approach manipulates the fact that the value of s^{τ} is similar to $s^{\tau-1}$ to start the search from $s^{\tau-1}$. Then, s^{τ} is either increased or decreased accordingly.

$$f(\underline{w}_{cand}^{\tau+1}) - f(\underline{w}^{\tau}) \leq \sigma\nabla f(\underline{w})^T(\underline{w}_{cand}^{\tau+1} - \underline{w}^{\tau}) \tag{11}$$

Although Lin's algorithm is quite efficient after the first iteration, no suggestion has been made in efficiently obtaining the value of s^1. In a typical execution of this algorithm, if 100 iterations are undertaken, the first may require $NT_1 = 90$ trials while the rest $NT_{rest} = 250$ in total. Hence, the first iteration requires approximately one third of the simulation time.

What we propose, is to hyper-exponentially alternate s^τ as shown in Algorithm 1. The proposed approach requires approximately $2log_2(NT_1)$ to compute s^1. The algorithm presented here, finds s^τ based on hyper-exponential search $\forall\tau$, but because after iteration 1 the number of trials is small the approach could be used only for the first iteration. The proposed algorithm satisfies all the requirements set for the desired approach.

In each trial of the algorithm only matrix-vector products involving $\mathbb{A}$ have to be calculated so the complexity of a sub-iteration is $O(KN^2)$. Thus, the complexity of the algorithm is #iterations $\times$ #trials $\times O(KN^2)$. For large problems, #iterations $\times$ #trials is of the same order as N and so the algorithm is of the same complexity as a RNN gradient descent iteration.

Algorithm 1 Improved Projected Gradient Algorithm for the NNLS problem

Set $\sigma \leftarrow 0.01$; $\beta \leftarrow 0.1$; $s^0 \leftarrow 1$; $\underline{w}^1 \leftarrow 0$;
repeat
  $s^\tau \leftarrow s^{\tau-1}_{des}$; $s^\tau_{init} \leftarrow s^{\tau-1}_{des}$; $k \leftarrow -1$; $\underline{w}^{\tau+1}_{cand} \leftarrow P[\underline{w}^\tau - s^\tau \nabla f(\underline{w}^\tau)]$;
  if ((11) is satisfied) **then**
    repeat
      $k \leftarrow k+1$; $s^\tau \leftarrow s^\tau_{init}/\beta^{2^k}$; $\underline{w}^{\tau+1}_{cand} \leftarrow P[\underline{w}^\tau - s^\tau \nabla f(\underline{w}^\tau)]$;
    until ((11) is not satisfied)
    $low \leftarrow \lfloor 2^{k-1}+1 \rfloor$; $high \leftarrow 2^k$; $s^\tau_{des} \leftarrow s^\tau$;
    while ($low < high$) **do**
      $mid \leftarrow \lfloor (low+high)/2 \rfloor$; $s^\tau \leftarrow s^\tau_{init}/\beta^{mid}$; $\underline{w}^{\tau+1}_{cand} \leftarrow P[\underline{w}^\tau - s^\tau \nabla f(\underline{w}^\tau)]$;
      if ((11) is not satisfied) **then**
        $high \leftarrow mid$; $s^\tau_{des} \leftarrow s^\tau$;
      else
        $low \leftarrow mid+1$;
      end if
    end while
  else
    repeat
      $k \leftarrow k+1$; $s^\tau \leftarrow s^\tau_{init} \cdot \beta^{2^k}$; $\underline{w}^{\tau+1}_{cand} \leftarrow P[\underline{w}^\tau - s^\tau \nabla f(\underline{w}^\tau)]$;
    until ((11) is satisfied)
    $low \leftarrow \lfloor 2^{k-1}+1 \rfloor$; $high \leftarrow 2^k$; $s^\tau_{des} \leftarrow s^\tau$;
    while ($low < high$) **do**
      $mid \leftarrow \lfloor (low+high)/2 \rfloor$; $s^\tau \leftarrow s^\tau_{init} \cdot \beta^{mid}$; $\underline{w}^{\tau+1}_{cand} \leftarrow P[\underline{w}^\tau - s^\tau \nabla f(\underline{w}^\tau)]$
      if ((11) is satisfied) **then**
        $high \leftarrow mid$; $s^\tau_{des} \leftarrow s^\tau$;
      else
        $low \leftarrow mid+1$;
      end if
    end while
  end if
  $\underline{w}^{\tau+1} \leftarrow P[\underline{w}^\tau - s^\tau_{des} \nabla f(\underline{w}^\tau)]$;
until {Convergence}

To obtain the initial weights the procedure outlined in Algorithm 2 is followed. Note that the NNLS algorithm does not require matrix $\mathbb{A}$ as input; $\underline{q}$ is sufficient to form the rows and columns of $\mathbb{A}$ and obtain matrix-vector products involving $\mathbb{A}$. Thereby, the order of memory required is the same as for the standard RNN learning algorithm. The procedure is repeated for a small number of iterations and the best acquired set of weights is employed to execute the supervised learning algorithm outlined in Section 2.2. Since the complexity of the initialization method is similar to a RNN gradient descent iteration, the running time is not negatively affected; in fact, it is improved because a substantially smaller number of gradient iterations has to be performed.

Algorithm 2 NNLS weight initialization algorithm for the RNN

Initialize Λ_{ik} and λ_{ik} $\forall$ $i,$ k based on the input patterns
Set $0 < q_{i_{out}k} < 1$ $\forall,$ $k,$ i_{out}, according to the y_{ik} where i_{out} is an output neuron
Randomly initialize the non-output neurons $0 < q_{\overline{i_{out}}k} < 1$ $\forall$ $i,$ k and form $\underline{q}$
for a number of iterations **do**
  Update $\underline{b}$ according to (8)
  $\underline{w} \leftarrow$ ImprovedProjectedGradientNNLS($\underline{q}, \underline{b}$)
  for all k **do**
    Update $q_{\overline{i_{out}}k}$ by solving the nonlinear system of equations (1)
  end for
end for

4 Experimental Results

In this section, the developed initialization method is used in accordance with the supervised learning algorithm to a combinatorial optimization problem arising in emergency response; its performance is compared to the standard supervised RNN learning algorithm with random weight initialization [3].

Emergency response problems require real-time and close to optimum solutions. The approach taken is to train the RNN off-line with numerous instances of the optimization problem in the same physical context and use the trained network to solve any problem instances that emerge in real-time.

We consider an emergency problem where N_L incidents occur simultaneously at different locations with I_j people injured at incident j. N_U emergency units are spatially distributed at the time of the incident; unit i can collect $c_i > 0$ injured and respond to incident j in time $T_{ij} > 0$. It is assumed that one unit can only be allocated to one incident and that there is a possible allocation to collect all the injured. The goal of the problem is two-fold: to collect all the injured and minimize the response time. Formulating the problem yields:

$$min\ f(\underline{x}) = \sum_{i=1}^{N_U}\sum_{j=1}^{N_L} T_{ij}x_{ij} \tag{12}$$

$$\text{subject to } \sum_{j=1}^{N_L} x_{ij} = 1 \ \forall \ i, \quad \sum_{i=1}^{N_U} c_i x_{ij} \geq I_j \ \forall \ j, \quad x_{ij} \epsilon \{0,1\} \forall \ i, \ j \tag{13}$$

The binary decision variables x_{ij} are equal to 1 if unit i is allocated to incident j and 0 otherwise. The above optimization problem is NP-hard.

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}. The output variables are represented by two neurons of opposite polarity, a "positive" and a "negative" one. If $x_{ij} = 1$, then the excitation level of the corresponding "positive" output neuron is high (close to 1) and the excitation level of the "negative" output neuron is low (close to 0). If $x_{ij} = 0$, then the reverse is true. The neural network architecture is comprised of a recurrent neural network with N_L input neurons and $2N_U N_L$ output neurons as well as $2N_U N_L$ hidden neurons. It is assumed that there is no self-feedback for any of the neurons and that no neuron interacts with input neurons.

At first, 250 training instances for different numbers of emergency units and locations of incidents were generated. The remaining parameters have been chosen at random with $T_{ij} \in \mathbb{R}$ and $c_i \in \mathbb{N}$ being uniformly distributed in the intervals $0 - 1$ and $1 - 4$, respectively. For each of the training patterns the $I_j \in \mathbb{N}$ are also uniformly distributed in the interval $0.5 \cdot c_t/N_L, 1.1 \cdot c_t/N_L$ 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: $N_U = 8, 12, 16, 20$ and $N_L = 3, 5$. Testing was performed using a distinct generated set of 200 test cases, with the same probability distributions for all parameters, so that the training and testing sets were disjoint. To emphasize the fact that $\mathbb{A}$ cannot be stored notice that for $N_U = 20$ and $N_L = 5$ a network of $N = 405$ neurons is produced and $\mathbb{A}$ is a 101250×328050 matrix.

The evaluation of the attained results was undertaken on the basis of the ratio $f_{NN}(\underline{x})/f_{opt}(\underline{x})$ that shows how close to optimality the results are (Fig. 1), the percentage of people collected (Fig. 3) and the percentage of instances solved so that all of the injured were evacuated (feasible solutions)(Fig. 2). All metrics were averaged over the testing instances.

As illustrated in Fig. 1 both algorithms approach equally well the optimal solutions especially for $N_L = 3$. However, using the NNLS initialization approach equal or better results are always obtained. In fact for the case of $N_L = 5$ the results are substantially better. The same conclusion can be drawn for the metrics depicted on Figures 2 and 3; namely, the percentage of casualties collected and the percentage of instances where all people are collected. Actually, the proposed approach is better in almost all the cases, except from the case $N_U = 8$ and $N_L = 5$ where the random initialization leads to better results. This may be the case because for small networks there are less local minima and it is more probable that gradient descent with random initialization will reach a better solution. In addition, note that the performance improvement is higher as the dimensionality of the network increases which illustrates that the higher the dimensionality the better the approximation.

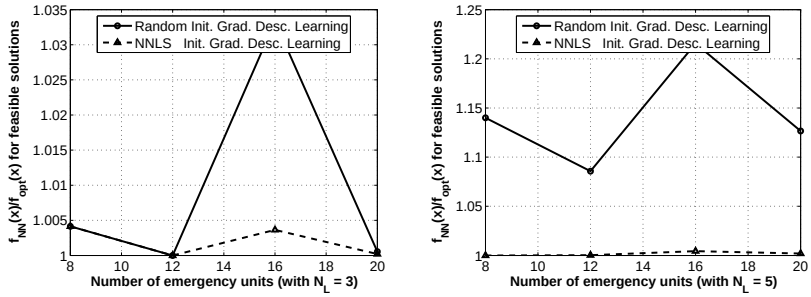

Fig. 1. Average of $f_{NN}(\underline{x})/f_{opt}(\underline{x})$ for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

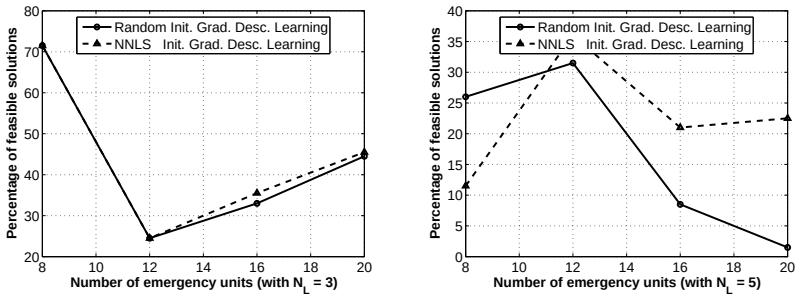

Fig. 2. Percentage of solutions in which all casualties are evacuated; these solutions are called "feasible" in the graphs, for want of a better term

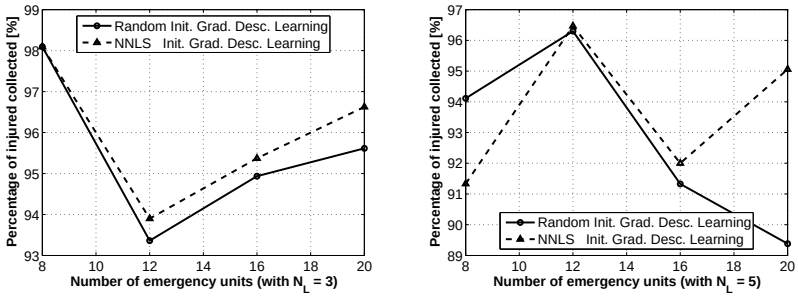

Fig. 3. Percentage of casualties that are collected

5 Conclusions

We have developed a new weight initialization method for supervised learning in the RNN. The core of the algorithm is the solution of a NNLS problem, which is formulated by approximating the equations of the network as a linear system. To solve NNLS, we have improved a projected gradient algorithm and shown that it is appropriate for ill-conditioned and large-scale problems. We illustrate that supervised learning with the proposed initialization method has substantially better performance than with random weight initialization when applied to a combinatorial optimization problem arising in emergency response.

References

1. Gelenbe, E.: Random neural networks with positive and negative signals and product form solution. Neural Computation **1**(4) (1989) 502–510
2. Gelenbe, E.: Stability of the random neural network. Neural Computation **2**(2) (1990) 239–247
3. Gelenbe, E.: Learning in the recurrent random network. Neural Computation **5** (1993) 154–164
4. Gelenbe, E., Ghanwani, A., Srinivasan, V.: Improved neural heuristics for multicast routing. IEEE Journal of Selected Areas of Communications **15**(2) (1997) 147–155
5. Bakircioglu, H., Kocak, T.: Survey of random neural network applications. European Journal of Operational Research **126**(2) (2000) 319–330
6. Oke, G., Loukas, G.: A denial of service detector based on maximum likelihood detection and the random neural network. The Computer Journal **50**(6) (2007) 717–727
7. Artalejo, J.R.: G-networks: A versatile approach for work removal in queueing networks. European Journal of Operational Research **126**(2) (2000) 233–249
8. Gelenbe, E.: Steady-state solution of probabilistic gene regulatory networks. Physical Review E **76**(1) (2007) 031903
9. Gelenbe, E., Timotheou, S.: Random Neural Networks with Synchronised Interactions. accepted for publication in *Neural Computation* (2008)
10. Gelenbe, E., Timotheou, S.: Synchronised Interactions in Spiked Neuronal Networks. accepted for publication in *The Computer Journal* (2008)
11. Thimm, G., Fiesler, E.: High-order and multilayer perceptron initialization. IEEE Transactions on Neural Networks **8**(2) (1997) 349–359
12. Wessels, L., Barnard, E.: Avoiding false local minima by proper initialization of connections. IEEE Transactions on Neural Networks **3**(6) (1992) 899–905
13. Fontenla-Romero, O., Erdogmus, D., Principe, J.C., Alonso-Betanzos, A., Castillo, E.: Linear least-squares based methods for neural networks learning. In: Artificial Neural Networks and Neural Information Processing. (2003) 84–91
14. Yam, Y.F., Chow, T.W.S., Leung, C.T.: A new method in determining initial weights of feedforward neural networks for training enhancement. Neurocomputing **16**(1) (1997) 23–32
15. Lawson, C.L., Hanson, R.J.: Solving Least Squares Problems. PrenticeHall (1987)
16. Bro, R., Long, S.D.: A fast non-negativity-constrained least squares algorithm. Journal of Chemometrics **11**(5) (1997) 393–401
17. Lin, C.J.: Projected gradient methods for nonnegative matrix factorization. Neural Computation **19**(10) (2007) 2756–2779
18. Bertsekas, D.: Nonlinear Programming. Athena Scientific (1995)