ELSEVIER

European Journal of Operational Research 126 (2000) 331–339

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Training the random neural network using quasi-Newton methods

Aristidis Likas [a,*], Andreas Stafylopatis [b]

[a] *Department of Computer Science, University of Ioannina, 451 10 Ioannina, Greece*

[b] *Department of Electrical and Computer Engineering, National Technical University of Athens, 157 73 Zographou, Athens, Greece*

Abstract

Training in the random neural network (RNN) is generally specified as the minimization of an appropriate error function with respect to the parameters of the network (weights corresponding to positive and negative connections). We propose here a technique for error minimization that is based on the use of quasi-Newton optimization techniques. Such techniques offer more sophisticated exploitation of the gradient information compared to simple gradient descent methods, but are computationally more expensive and difficult to implement. In this work we specify the necessary details for the application of quasi-Newton methods to the training of the RNN, and provide comparative experimental results from the use of these methods to some well-known test problems, which confirm the superiority of the approach. © 2000 Elsevier Science B.V. All rights reserved.

1. Introduction

The *random neural network* (RNN) model, introduced by Gelenbe [6,7], has been the basis of theoretical efforts and applications during the last years. An important issue is that the model can be constructed in conformity to usual neural network characteristics. Moreover, the output of the model can be expressed in terms of its steady-state solution, that is in terms of quantities obtained through direct numerical computations.

Although the work on the RNN was initially motivated by the behavior of natural neural networks, the model can represent general systems containing processes and resources and supporting some types of control operations. In fact, the model is based on probabilistic assumptions and belongs to the family of Markovian queuing networks. The novelty with respect to usual queueing models lies in the concept of requests for removing work (negative customers) in addition to classical requests for performing work (positive customers). This novel class of models are referred to as *G-networks* in queueing network literature and have been studied extensively during the last years [8,10,17].

Applications of the RNN model have been reported in several fields, including image processing [1,2,13,14], combinatorial optimization [15] and associative memory [9,19,22]. In particular, the *bipolar random network* [18,19,22] is an extension

* Corresponding author.

E-mail addresses: arly@cs.uoi.gr (A. Likas), andreas@soft-lab.ece.ntua.gr (A. Stafylopatis).

of the original RNN adapted to associative memory operation. This extended model includes two types of nodes – positive and negative – and preserves the main characteristics of the original model, which considers one type of nodes.

Many applications of the RNN are based on its capability of learning input–output associations by means of an error-correction algorithm [11]. This algorithm uses *gradient-descent* of a *quadratic error function* to determine the parameters of the network (excitation and inhibition weights). The solution of a system of linear and a system of non-linear equations is required for each training input–output pair. Non-linear equations express the fixed-point solution to the network, whereas linear equations represent the partial derivatives of the fixed-point solution with respect to network parameters. The algorithm is established in terms of theoretical results concerning existence and uniqueness of fixed-point (steady-state) solution to the RNN [11].

In this paper, we propose a learning algorithm for the RNN that is also based on the minimization of the quadratic error function but employs more sophisticated optimization techniques compared to simple gradient-descent. More specifically, we use *quasi-Newton* optimization methods, which exploit gradient information to *approximate the Hessian matrix* of the error function with respect to the parameters of the network. This approximation matrix is subsequently used to determine an effective search direction and update the values of the parameters in a manner analogous to Newton's method [5,16,21]. Quasi-Newton methods are generally considered more powerful compared to gradient-descent and their applications to the training of other neural network methods (multilayer perceptrons) was very successful [3,20]. Therefore, it is reasonable to consider these methods as serious alternatives to gradient methods in the context of RNN training.

In Section 2 we briefly present the main characteristics of the RNN, whereas a description of the gradient-descent learning algorithm is provided in Section 3. Section 4 provides the details of the application of the quasi-Newton methods to RNN training, while Section 5 provides comparative experimental results from the application of the method to parity problems. Finally Section 6 provides conclusions and some directions for future work.

2. The RNN model

The RNN is a model that reproduces the pulsed behavior of natural neural systems. It is based on probabilistic assumptions and is characterized by the existence of signals in the form of spikes of unit amplitude that circulate among nodes. Positive and negative signals represent excitation and inhibition, respectively. The major property of the model is that it accepts a product form solution, i.e., the network's stationary probability distribution can be written as the product of the marginal probabilities of the state of each node. The significant feature of the model is that it is analytically solvable, and therefore computationally efficient, since its application is reduced to obtaining solutions to a system of fixed-point equations. In the remainder of this section we will provide a brief description of the model. A detailed description, along with analytical derivation of its main properties, can be found in [6,7,11].

In the RNN, each node accumulates signals that either arrive from the outside of the network or from other nodes. External positive and negative signal arrivals to each node i are considered Poisson with rates $\Lambda(i)$ and $\lambda(i)$, respectively. If the total signal count of a node at a given time instant is strictly positive, the node fires and sends out spikes to other nodes or to the outside of the network. The intervals between successive firing instants at node i are random variables following an exponential distribution with mean $1/r(i)$. Nodes accumulate and emit only positive signals. The role of negative signals is purely suppressive, i.e., they simply cancel positive signals if there are any.

Connections between nodes can be positive or negative, so that a signal leaving a node can move to another node as a signal of the same or the opposite sign, respectively. More specifically, a signal leaving node i arrives to node j as a positive signal with probability $p^+(i,j)$ and as a negative signal with probability $p^-(i,j)$. Also, a signal leaving node i departs from the network with probability $d(i)$. Obviously, for a network with n nodes we shall have

$$\sum_{j=1}^{n}[p^+(i,j)+p^-(i,j)]+d(i)=1,$$

for $i=1,\dots,n$.

As already stated, the above described Markovian network has product form solution. This property has been shown in [6] for the original version of the RNN and in [22] for the extended version including positive and negative nodes (bipolar random network). Results concerning the existence and uniqueness of the solution can be found in [7,11]. Additional properties of the RNN model have been established in [12].

The steady-state characteristics of the RNN can be summarized as follows. Considering a network with n nodes its stationary probability distribution is given by $p(\hat{k})=\lim_{t\to\infty}\text{Prob}[\hat{K}(t)=\hat{k}]$, whenever this limit exists, where $\hat{K}(t)$ is the state vector at time t representing the signal count at each node of the network, and $\hat{k}=(k_1,\dots,k_n)$ denotes a particular value of the vector. The flow of signals in the network can be described by the following equations in terms of the arrival rates $\lambda^+(i)$ and $\lambda^-(i)$ of positive and negative signals to node i:

$$\lambda^+(i)=\Lambda(i)+\sum_j q_j r(j)p^+(j,i), \tag{1}$$

$$\lambda^-(i)=\lambda(i)+\sum_j q_j r(j)p^-(j,i), \tag{2}$$

where

$$q_i=\frac{\lambda^+(i)}{r(i)+\lambda^-(i)}. \tag{3}$$

It can be shown [7] that, if a unique non-negative solution $\{\lambda^+(i),\lambda^-(i)\}$ exists to the above equations such that $q_i<1$, then the steady-state network probability distribution has the form

$$p(\hat{k})=\prod_{i=1}^{n}[1-q_i]q_i^{k_i}.$$

Clearly, q_i is the steady-state probability that node i is excited and is the key quantity associated with application of the model. It represents the level of excitation as an analog rather than as a binary variable, thus leading to more detailed information on system state. In a sense, each node acts as a non-linear frequency demodulator, since it transforms the frequency of incoming spike trains into an 'amplitude' value q_i. We take advantage of this feature when employing the steady-state solution of the network. An analytical solution of the set of Eqs. (1)–(3) is not always feasible. However, necessary and sufficient conditions for the existence of the solution can be established [7,11].

An analogy between usual neural network representation and the random neural network model can be constructed [6,11]. Considering neuron i, a correspondence of parameters can be established as follows:

$$w^+(i,j)=r(i)p^+(i,j)\geqslant 0,$$
$$w^-(i,j)=r(i)p^-(i,j)\geqslant 0,$$
$$r(i)=\sum_j[w^+(i,j)+w^-(i,j)],$$

where the quantities $w^+(i,j)$ and $w^-(i,j)$ represent rates (or frequencies) of spike emission, but clearly play a role similar to that of synaptic weights. Nevertheless, these weight parameters have a somewhat different effect in the RNN model than weights in the conventional connectionist framework. In the RNN model, all the $w^+(i,j)$ and $w^-(i,j)$ are nonnegative, and for a given pair (i,j) it is possible that both $w^+(i,j)$ and $w^-(i,j)$ be positive, therefore they must both be learned.

3. The RNN learning algorithm

We now provide a concise presentation of the algorithm developed in [11] for determining the network parameters in order to learn a set of input–output pairs.

For convenience we can use the notation

$$N(i)=\sum_j q_j w^+(j,i)+\Lambda(i),$$

$$D(i)=r(i)+\sum_j q_j w^-(j,i)+\lambda(i),$$

and write

$$q_i = N(i)/D(i). \tag{4}$$

The training set $(\imath, Y)$ consists of K input–output pairs, where $\imath = \{\imath_1, \dots, \imath_K\}$ denotes successive inputs and $Y = \{y_1, \dots, y_K\}$ are successive desired outputs. Each input is a pair $\imath_k = (\Lambda_k, \lambda_k)$ of positive and negative signal flow rates entering each neuron: $\Lambda_k = [\Lambda_k(1), \dots, \Lambda_k(n)]$, $\lambda_k = [\lambda_k(1), \dots, \lambda_k(n)]$. Each output vector $y_k = (y_{ik}, \dots, y_{nk})$ has elements $y_{ik} \in [0,1]$ that correspond to the desired values of each neuron. The desired output vectors are approximated in a manner that minimizes a cost function

$$E = \sum_{k=1}^{K} E_k, \tag{5}$$

where

$$E_k = (1/2) \sum_{i=1}^{n} a_i (q_i - y_{ik})^2, \quad a_i \geqslant 0. \tag{6}$$

The aim is to let the network learn the two $n \times n$ weight matrices $W_k^+ = \{w_k^+(i,j)\}$ and $W_k^- = \{w_k^-(i,j)\}$, by computing new values of the matrices, after presentation of each input $\imath_k$, using gradient-descent.

The rule for weight update can take the generic form

$$w_k(u,v) = w_{(k-1)}(u,v) - \eta \sum_{i=1}^{n} a_i (q_{ik} - y_{ik}) \times [\partial q_i / \partial w(u,v)]_k, \tag{7}$$

where $\eta > 0$ is a learning coefficient, and the term $w(u,v)$ denotes any weight in the network (either $w^-(u,v)$ or $w^+(u,v)$).

To compute the partial derivatives let us first define the vector $q = (q_1, \dots, q_n)$ and the $n \times n$ matrix

$$W = \{[w^+(i,j) - w^-(i,j) q_j]/D(j)\}, \quad i,j = 1, \dots, n.$$

Also, define the n-vectors $\gamma^+(u,v)$ and $\gamma^-(u,v)$ with elements

$$\gamma_i^+(u,v) = \begin{cases} -1/D(i) & \text{if } u = i,\ v \neq i, \\ +1/D(i) & \text{if } u \neq i,\ v = i, \\ 0 & \text{for all other values of } (u,v); \end{cases}$$

$$\gamma_i^-(u,v) = \begin{cases} -(1+q_i)/D(i) & \text{if } u = i,\ v = i, \\ -1/D(i) & \text{if } u = i,\ v \neq i, \\ -q_i/D(i) & \text{if } u \neq i,\ v = i, \\ 0 & \text{for all other values of } (u,v). \end{cases}$$

Using the above notation, we can derive from Eq. (4) the following vector equations:

$$\partial q / \partial w^+(u,v) = \partial q / \partial w^+(u,v) W + \gamma^+(u,v) q_u,$$
$$\partial q / \partial w^-(u,v) = \partial q / \partial w^-(u,v) W + \gamma^-(u,v) q_u,$$

which can be written equivalently

$$\partial q / \partial w^+(u,v) = \gamma^+(u,v) q_u [\mathbf{I} - W]^{-1}, \tag{8}$$

$$\partial q / \partial w^-(u,v) = \gamma^-(u,v) q_u [\mathbf{I} - W]^{-1}, \tag{9}$$

where $\mathbf{I}$ denotes the $n \times n$ identity matrix.

The complete learning algorithm can now be sketched. Starting from some appropriately chosen initial values for the matrices W_0^+ and W_0^-, we proceed with successive presentation of input values $\imath_k = (\Lambda_k, \lambda_k)$. For each k, first solve the system of nonlinear Eqs. (1)–(3) and then, using the obtained results, solve the system of linear Eqs. (8) and (9). Through substitution in (7) update the matrices W_k^+ and W_k^-.

In [11], different technical options are considered as to the exact procedure of implementing the algorithm. Also, as we are seeking for nonnegative weights, appropriate alternatives are prescribed in case a negative term is obtained during the iteration.

In the approach presented in this paper, to ensure nonnegativity of the weights, we have introduced the variables $u^+(i,j)$ and $u^-(i,j)$ defined such that $w^+(i,j) = (u^+(i,j))^2$ and $w^-(i,j) = (u^-(i,j))^2$. The parameters $u^+(i,j)$ and $u^-(i,j)$ constitute the *actual adaptable parameters* of the network and their derivatives are given by

$$\partial q / \partial u^+(i,j) = 2u^+(i,j) \partial q / \partial w^+(i,j), \tag{10}$$

$$\partial q / \partial u^-(i,j) = 2u^+(i,j) \partial q / \partial w^-(i,j). \tag{11}$$

The complexity of the learning algorithm can be considered in terms of the complexity of each weight update. The complexity of solving the system of nonlinear Eqs. (1)–(3) is $O(mn^2)$ if a relaxation method with m iterations is used. The main computational task in solving the system of linear Eqs. (8) and (9) is to obtain $[\mathbf{I} - W]^{-1}$, which can be done in time complexity $O(n^3)$ (or $O(mn^2)$ if a relaxation method is adopted).

4. Training using quasi-Newton methods

Once the derivatives of the error function with respect to the weights w have been computed, it is possible, instead of using naive gradient-descent, to minimize the training error using the more sophisticated *quasi-Newton* optimization methods. These methods exploit the derivative information to *approximate* the Hessian matrix required in the Newton optimization formulas. Among the several quasi-Newton methods we have selected to implement two quasi-Newton variants, the BFGS (named for its inventors Broyden, Fletcher, Goldfarb and Shanno) method and the DFP (named for Davidon, Fletcher and Powell) method described below [5,16].

In analogy with most neural network models, the problem of training the RNN to perform static mappings from the input space to the output space can be specified as a parameter optimization problem of the error function E (Eq. (4)) with the values of $u^+(i,j)$ and $u^-(i,j)$ as adjustable parameters. This error function is continuous and differentiable with respect to the parameters, i.e., the gradient of the error function with respect to any of the parameters can be *analytically* specified and computed. Once the gradients have been specified, the most straightforward approach for error minimization is gradient-descent, where at each point the update direction of the parameter vector is the negative of the gradient at this point. This approach, due to its simplicity, has several drawbacks, as for example the *zig-zag behavior* and the well-known problem related to the specification of the value of the *learning rate* parameter. Moreover, it is very slow, usually requiring a large number of training steps. To alleviate these problems, several modifications of the basic gradient descent strategy have been proposed [4], mainly in the context of training multilayer perceptrons (MLPs), but none of them has received widespread acceptance and use.

Apart from simple gradient-descent, several more sophisticated optimization techniques have been developed for the minimization of a function with known derivatives. A popular category of techniques of this kind are quasi-Newton methods [5,16], which have been shown to perform significantly better than gradient-descent methods in many optimization problems. They have also been examined in the context of MLP training yielding results of better quality (in terms of error function minimization) in less training steps [3]. For this reason, we consider quasi-Newton optimization methods as tools for training recurrent random neural networks and examine their effectiveness compared with the gradient descent method considered so far [11].

Let the generic vector p contain the adjustable parameters $u^+(i,j)$ and $u^-(i,j)$ of the RNN according to the specification described in the previous section and let $p^{(k)}$ denote the value of the parameter vector at step k of the optimization process.

Let also $g^{(k)}$ be the gradient vector at the point $p^{(k)}$, that is the lth component of vector g is $g_l = \partial E/\partial p_l$. Let also $H^{(k)}$ be the Hessian matrix of the RNN at step k of the optimization process, that is $h(l,m) = \partial E/\partial p_l \partial p_m$. In case this matrix can be computed analytically, the Newton method suggests that the new parameter vector $p^{(k+1)}$ be given by

$$p^{(k+1)} = p^{(k)} + s^{(k)}, \tag{12}$$

where $s^{(k)}$ is *Newton's direction* obtained by solving the system

$$H^{(k)} s^{(k)} = -g^{(k)}. \tag{13}$$

Since in most cases it is difficult and expensive to compute the exact matrix H, quasi-Newton methods have been developed which, instead of directly computing the Hessian matrix H, consider at each iteration k an approximation $B^{(k)}$ of $H^{(k)}$ that is suitably updated at each step. More specifically, quasi-Newton methods suggest at each step k the following operations:

1. Compute a *search direction* $s^{(k)}$ by solving the system $B^{(k)}s^{(k)} = -g^{(k)}$.
2. Perform *line search* along the direction $s^{(k)}$ to select a new point $p^{(k+1)}$. This means that we perform one dimensional minimization with respect to the parameter λ to minimize the error $E(p^{(k)} + \lambda s^{(k)})$ along the direction $s(k)$. If $\lambda^{\star}$ is the value provided by line search, we set $p^{(k+1)} = p^{(k)} + \lambda^{\star} s^{(k)}$.
3. Compute the new gradient vector $g^{(k+1)}$ at the point $p^{(k+1)}$.
4. Update $B^{(k)}$ to $B^{(k+1)}$ using an appropriate formula.
5. Check the termination criteria (such as maximum number of steps, minimum value of error function, convergence to a local minimum etc.). If they are not satisfied set $k := k + 1$ and return to step 1.

Initially we consider $B^{(0)} = \mathbf{I}$.

The two steps that need to be specified in the above scheme are the update formula for $B^{(k)}$ (step 4) and the details of line search (step 2).

We shall describe two popular formulas for updating $B^{(k)}$, the BFGS update formula and the DFP update formula [5,21]. Let $\delta = p^{(k+1)} - p^{(k)}$ and $\gamma = g^{(k+1)} - g^{(k)}$. To implement the update of $B^{(k)}$, first a *Choleski factorization* is required to compute the matrix L: $B^{(k)} = LL^{\top}$. Then we proceed as follows depending on the applied update formula.

- The BFGS update formula requires the computation of the quantities a (scalar) and v (vector) as

  $$a^2 = \frac{\delta^{\top}\gamma}{\delta^{\top}B\delta}$$

  and

  $$v = aL^{\top}\delta,$$

  from which we obtain

  $$M = L + \frac{(\gamma - Lv)v^{\top}}{v^{\top}v}. \tag{14}$$

- For application of the DFP update formula, first the quantity β must be computed as

  $$\beta^2 = \frac{\delta^{\top}\gamma}{\gamma^{\top}B^{-1}\gamma}.$$

  The next step is the solution of the linear system

  $$Lw = \beta\gamma,$$

  yielding the vector w, from which we obtain

  $$M = L - \frac{\gamma(\delta^{\top}L - w^{\top})}{\delta^{\top}\gamma}. \tag{15}$$

Finally, the new approximation $B^{(k+1)}$ to the Hessian is given by

$$B^{(k+1)} = MM^{\top}, \tag{16}$$

using the matrix M from Eq. (14) or (15) according to the selected approach.

In what concerns line search implemented in step 2, minimization is performed with respect to λ using a procedure which requires only function evaluations (computations of E) and no additional gradient computations (only the already computed gradient $g^{(k)}$ is used). The exact implementation of the line search method is described in [21, p. 237].

Given the parameter vector $p = (p_1, \ldots, p_M)$ (namely the parameters $u^+(i,j)$ and $u^-(i,j)$), we summarize below the operations required to perform a function evaluation (i.e., computation of the total error E) and a *gradient evaluation* (i.e. computation of the vector g, where $g_l = \partial E/\partial p_l$).

- Set $E = 0$ and $g_l = 0$ for all parameters $l = 1, \ldots, M$ (M is the number of parameters $u^+(i,j)$ and $u^-(i,j)$).
- For $k = 1, \ldots, K$ (K number of training patterns)
  1. Specify the values $\Lambda_k(i)$ and $\lambda_k(i)$ for each network node i ($i = 1, \ldots, n$) and the desired outputs y_{ik} for the output nodes.
  2. Solve the system of non-linear equations (1)–(3) to obtain the values q_{ik} ($i = 1, \ldots, n$).
  3. Compute the matrix $(\mathbf{I} - W)^{-1}$.
  4. Compute the gradients $\partial q_{ik}/\partial p_l$ for $i = 1, \ldots, n$ and $l = 1, \ldots, M$.
  5. Compute the error

     $$E_k = (1/2)\sum_{i=1}^{n} a_i(q_{ik} - y_{ik})^2.$$

  6. Compute for $l = 1, \ldots, M$,

     $$\frac{\partial E_k}{\partial p_l} = 2p_l \sum_{i=1}^{n} a_i(q_{ik} - y_{ik})\frac{\partial q_{ik}}{\partial p_l}.$$

7. Set $E := E + E_k$.
8. Update for $l = 1, \dots, M$,

$$g_l := \frac{\partial E}{\partial p_l} + \frac{\partial E_k}{\partial p_l}.$$

5. Experimental results

To assess the effectiveness of the proposed learning algorithms for the RNN, we have implemented the BFGS and the DFP quasi-Newton techniques following the specifications described in the previous section. For the purpose of comparison we have also implemented the gradient-descent method, where at step k each adjustable parameter p_l is updated as follows:

$$p_l^{k+1} = p_l^k - \eta \frac{\partial E}{\partial p_l}, \tag{17}$$

η being the learning rate.

This approach constitutes in essence a *batch* version of the training algorithm proposed in [11] with the additional characteristic that the adjustable parameters are the $u^+(i,j)$ and $u^-(i,j)$ to ensure nonnegativity of the actual network weights $w^+(i,j)$ and $w^-(i,j)$.

The experiments were conducted on classification problems, specifically on *parity* datasets with the input dimension ranging from $b = 2$ (XOR problem) to $b = 7$ (7-bit parity problem). In these problems the inputs are considered to be binary (0 or 1) and the output is the parity bit of the b inputs. For each value of b, the number of training pairs is equal to 2^b. The objective of training is the construction of networks that correctly yield the value of the parity bit for all 2^b input combinations. It is well known that, as the value of b increases, the corresponding mappings are difficult to implement by neural architectures.

In what concerns the topology of the employed RNN, we first considered *fully recurrent* networks with no self-feedback (i.e., $w_{ii}^+ = w_{ii}^- = 0$). For a network with n nodes, the first b of them were considered as input nodes, and the last of them was considered as output node, i.e. $a_i = 0$ for $i = 1, \dots, n-1$ and $a_n = 1$. For each training pattern k, input representation was specified as follows. For an input value of 0 we set $\Lambda_k(i) = 0$ for the corresponding input node, while for a value of 1 we set $\Lambda_k(i) = 1$. For all other network nodes, except the input nodes, we set $\Lambda_k(i) = 1$. Also, we set $\lambda_k(i) = 0$ for all nodes i of the network. In what concerns the output node, if the desired output is 0 we set $y_{nk} = 0.1$, while for desired output 1 we set $y_{nk} = 0.9$. For a given input pattern, the output of the network was considered to be 1 if $q_n > 0.5$ and 0 if $q_n < 0.5$. Experiments using the fully recurrent architecture yielded very poor results, independently of the method that was used for training. This means that fully recurrent random networks are difficult to train on parity problems. For this reason we have considered *feedforward* architectures and specifically a feedforward RNN with b input nodes, one hidden layer with 15 hidden nodes and finally one node in the output layer. The same architecture was used in all experiments. The specifications of the values of $\Lambda_k(i)$ and $\lambda_k(i)$ were the same as described above for the recurrent network case.

Table 1 summarizes results concerning the training effectiveness of the three examined learn-

Table 1
Comparative results of the three training methods on parity problems

b	Gradient-descent			BFGS			DFP		
	Steps	E	Wrong	Steps	E	Wrong	Steps	E	Wrong
2	2140	5×10^{-5}	0	152	2×10^{-20}	0	175	5×10^{-10}	0
3	3138	0.015	0	283	8×10^{-4}	0	350	3×10^{-4}	0
4	5673	0.530	1	952	7×10^{-4}	0	1220	6×10^{-3}	0
5	5832	1.34	3	3682	0.17	0	3986	0.13	0
6	7421	4.42	10	3543	1.56	3	4012	2.1	4
7	8733	10.38	18	3755	3.21	8	3905	4.12	10

ing algorithms. For each algorithm we specify: (i) the final error value E (local minimum), (ii) the number of input patterns that the network was not able to learn and (iii) the required number of training steps (epochs). In all methods, the complexity of each step stems mainly from the computation of the gradient method. Due to line search and additional computations, each step of the quasi-Newton methods is slightly more computationally intensive compared to gradient-descent.

As Table 1 indicates, quasi-Newton techniques are superior to simple 'batch' gradient-descent and lead to significantly better solutions requiring fewer training steps. Therefore, they constitute serious alternatives to gradient-descent methods. In addition, these methods do not suffer from the problem related to the specification of the learning rate parameter which is crucial for the performance of the gradient-descent method. On the other hand, a drawback of quasi-Newton methods is that they are difficult to implement. A solution to this problem is the exploitation of powerful software packages for multidimensional minimization. Such packages provide a variety of optimization techniques that the user may employ to solve minimization problems. In our work we used the *Merlin* optimization package [21] that we had also employed previously for effective training of multilayer perceptrons [20]. Another drawback of the quasi-Newton methods is that they cannot be used for on-line learning problems, since they operate in a 'batch' mode.

As far as the relative performance of the two quasi-Newton methods is concerned, the BFGS technique seems to be faster, although both methods finally lead to solutions of comparable quality. Finally, it must be stressed that all experiments were conducted using the same number of hidden nodes. By increasing the number of hidden nodes it is expected that training will lead to perfect classification.

6. Conclusions

A new method is proposed for training the RNN that is based on quasi-Newton optimization techniques. Such techniques exploit derivative information as happens with gradient-descent methods, but are more sophisticated since at each step they compute an approximation to the Hessian matrix. This approximation is subsequently used to obtain a search direction using Newton's method and, finally, the new point in parameter space is located through line search along this direction. As expected, this training method is more powerful compared to gradient-descent but has the drawback that it is more difficult to implement. Moreover, it is a *batch* technique requiring a pass through all training patterns before an update takes place. Therefore it cannot be used for problems requiring on-line learning. As a future work, one may consider other optimization techniques as training alternatives, as for example the well known family of conjugate gradient methods and the Levenberg–Marquardt method [16].

References

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

[2] V. Atalay, E. Gelenbe, Parallel algorithm for colour texture generation using the random neural network model, International Journal of Pattern Recognition and Artificial Intelligence 6 (1992) 437–446.

[3] E. Barnard, Optimization for training neural nets, IEEE Transactions on Neural Networks 3 (2) (1992) 232–240.

[4] C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[5] R. Fletcher, Practical Methods of Optimization, Wiley, New York, 1987.

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

[7] E. Gelenbe, Stability of the random neural network model, Neural Computation 2 (1990) 239–247.

[8] E. Gelenbe, Product form queueing networks with negative and positive customers, Journal of Applied Probability 28 (1991) 656–663.

[9] E. Gelenbe, A. Stafylopatis, A. Likas, Associative memory operation of the random network model, in: T. Kohonen, et al. (Eds.), Artificial Neural Networks, vol. 1, North-Holland, Amsterdam, 1991, pp. 307–312.

[10] E. Gelenbe, G-Networks with triggered customer movement, Journal of Applied Probability 30 (1993) 742–748.

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

[12] E. Gelenbe, Hopfield energy of the random neural network, in: Proceedings of the IEEE International Conference on Neural Networks, vol. VII, Orlando, FL, June 1994, pp. 4681–4686.

[13] E. Gelenbe, Y. Feng, K. Ranga, R. Krishnan, Neural network methods for volumetric magnetic resonance imaging of the human brain, Proceedings of the IEEE 84 (1996) 1488–1496.

[14] E. Gelenbe, M. Sungur, C. Cramer, P. Gelenbe, Traffic and video quality with adaptive neural compression, Multimedia Systems 4 (1996) 357–369.

[15] E. Gelenbe, A. Ghanwani, V. Srinivasan, Improved neural heuristics for multicast routing, IEEE Journal on Selected Areas in Communications 15 (1997) 147–155.

[16] P. Gill, W. Murray, M. Wright, Practical Optimization, Academic Press, New York, 1997.

[17] W. Henderson, Queueing networks with negative customers and negative queue lengths, Journal of Applied Probability 30 (1993) 931–942.

[18] A. Likas, A. Stafylopatis, An investigation of the analogy between and the random network and the Hopfield network, in: Proceedings of the ISCIS VI, North-Holland, 1991.

[19] A. Likas, A. Stafylopatis, High capacity associative memory based on the random neural network model, International Journal of Pattern Recognition and Artificial Intelligence 10 (1996) 919–937.

[20] A. Likas, D.A. Karras, I.E. Lagaris, Neural network training and simulation using a multidimensional optimization system, International Journal of Computer Mathematics 67 (1998) 33–46.

[21] D.G. Papageorgiou, I.N. Demetropoulos, I.E. Lagaris, Merlin 3.0, a multidimensional optimization environment, Computer Physics Communications 109 (1998) 227–249.

[22] A. Stafylopatis, A. Likas, Pictorial information retrieval using the random neural network, IEEE Transactions on Software Engineering 18 (1992) 590–600.