Applied Mathematics and Computation 217 (2011) 5421–5441

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Nonmonotone BFGS-trained recurrent neural networks for temporal sequence processing

Chun-Cheng Peng *, George D. Magoulas

Department of Computer Science and Information Systems, Birkbeck College, University of London, Malet Street, London WC1E 7HX, UK

ARTICLE INFO

Keywords:
Recurrent neural networks
Quasi-Newton methods
BFGS updates
Nonmonotone methods
Second-order training algorithms
Temporal sequence

ABSTRACT

In this paper we propose a nonmonotone approach to recurrent neural networks training for temporal sequence processing applications. This approach allows learning performance to deteriorate in some iterations, nevertheless the network's performance is improved over time. A self-scaling BFGS is equipped with an adaptive nonmonotone technique that employs approximations of the Lipschitz constant and is tested on a set of sequence processing problems. Simulation results show that the proposed algorithm outperforms the BFGS as well as other methods previously applied to these sequences, providing an effective modification that is capable of training recurrent networks of various architectures.

1. Introduction

There are several real-world applications, such as robotics, speech recognition, language processing, computer vision, which require processing data sequences. This kind of processing involves tasks like clustering, classification, prediction, and transduction of sequential data which can be symbolic, non-symbolic or mixed. When the sequence elements are time-varying then the sequence is called temporal or time-series, depending on whether its elements are nominal symbols from a particular alphabet, or continuous, real-valued data, respectively. Processing these sequences is a challenging task as the range of data dependencies is usually unknown, and generalisation is highly affected by the samples in the training set.

Recurrent neural networks (RNNs) are well-known for their power to memorise time dependencies and model nonlinear systems. They can be trained from examples to map input sequences to output sequences and in principle they can implement any kind of sequential behaviour. They have been considered to be eminently suitable for this task and indeed several attempts have been made to use them for processing sequences [4,10,11,13,36,54].

Most RNN applications are using first-order learning algorithms despite the drawbacks of the Gradient Descent [9]. Some attempts have been made to propose second-order learning algorithms, e.g. dos Santos and von Zuben [17] proposed a quasi second-order method, and Tsoi [65] examined the Newton approach. Also, simulated annealing has given some promising results but the training time is relatively higher. Nevertheless, first-order methods still remain the most popular choice. This is mainly attributed to the nature of the particular problem and the time dependency of the training data which generate error landscapes that cause instabilities in the calculation of the Hessian matrix, vanishing gradients and convergence to local minima that are far away from any desired minimisers. More details on the use of RNNs in sequence processing and their training can be found in [52].

In this paper, we focus on second-order learning algorithms for RNNs to process temporal sequences, i.e. sequences whose elements are nominal symbols from a particular alphabet. We propose a modified version of BFGS, which combines the usage of adaptive nonmonotone learning and self-scaling utility; the former takes the benefits of Lipschitz constant and

* Corresponding author.
E-mail address: goudapeng@gmail.com (C.-C. Peng).

0096-3003/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2010.12.012

provides more information of the morphology of a given function, while the latter has been proved effective in solving unconstrained nonconvex optimisation problems.

The rest of this paper is organised as follows. Section 2 introduces the basic elements of an RNN and presents some popular RNN architectures. Section 3 formulates the RNN training problem in the context of unconstrained optimisation. It also looks into monotone learning algorithms based on quasi-Newton methods and discusses their use for training artificial neural networks in general and RNNs in particular. Section 4 introduces nonmonotone BFGS learning, discussing the property of global convergence. In Section 5, the proposed algorithm, called adaptive nonmonotone self-scaling BFGS (ANMSCBFGS) is presented. The behaviour of the method is investigated through an empirical study and experimental results are presented in Section 6. The paper concludes with Section 7.

2. Overview of recurrent neural networks

In the area of artificial intelligence (AI) researchers proposed simplified models of the human brain, called artificial neural networks (ANNs) in an attempt to produce computational models that will emulate its operation. ANNs can have a large number of highly interconnected processing units, where each unit, called a *artificial neuron*, *neural node*, or simply *node*, can have the same or different processing abilities. The connections between nodes are used to transmit data from one artificial neuron to the other and carry some weight. The way the nodes are organised and connected defines the so-called *architecture* of an ANN. More details of how the biological models of neurons relate to computational ones and the artificial neurons' processing abilities can be found in the relative literature, such as [31,39,42].

In terms of their temporal characteristics, ANNs can be characterised as static or dynamic. In this paper we focus on networks that belong to the second category, and in particular to *recurrent neural networks* (RNNs).

An RNN is an artificial neural network in which self-loop and backward connections between nodes are allowed [39,60]. One of the first RNNs was the *avalanche network* developed by Grossberg [29] for learning and processing an arbitrary spatiotemporal pattern. Jordan's sequential network [35] and Elman's simple recurrent network [18] were proposed later. The first RNNs did not work very well in practical applications, and their operation was poorly understood. However, several variants of these models were developed for real-world applications, such as robotics, speech recognition, music composition, computer vision, and their potential for solving real-world problems has motivated a lot of research in the area of RNNs. Current research in RNNs has overcome some of the major drawbacks of the first models. This progress has come in the form of new architectures and learning algorithms, and has led in a better understanding of the RNNs' behaviour.

Fig. 1 illustrates a node with recurrent connections – a self-loop in this case – where F is the activation function. The node operates by the following difference equation:

$$y_t = F(a_0 + w y_{t-1} + b), \tag{1}$$

where y_t is the new output at time $t \geqslant 1$, y_{t-1} is the previous output ($\forall t \neq 1; y_0 = 0$), w is the weight of the recurrent connection, b is the bias term, which can be considered as a connection to an input that is always $+1$, and a_0 is the initial input to the node, which is applied at $t = 0$ and then removed. During operation, Eq. (1) is used repeatedly to generate a sequence $\{y\}_t^{\infty}$ of values that correspond to the output, or activation level, of the node. This sequence of activation values may converge to a fixed point, which is stable, or diverge depending on the weight's value. So a main feature of this kind of node during operation is that the change of the output over time will cause a network of these nodes to settle into one of several states depending on the input applied and the set of weight values. Seeking appropriate weights is done during the training phase of the RNN, as discussed below.

In this work, we consider RNNs with nodes organised in three basic layers, i.e. input, hidden and output layers, where each layer consists of a specific number of nodes. Each layer receives the outputs of the previous layer and produces an output which is then sent to the next layer. Backward connections between nodes or layers are allowed. The three kinds of RNNs considered in this work, as shown in Figs. 2–4, are the Feed-Forward Time-Delayed (FFTD) network, the Layer Recurrent Network (LRN) and the Nonlinear Autoregressive Network with Exogenous Inputs (NARX) network. In these figures circles

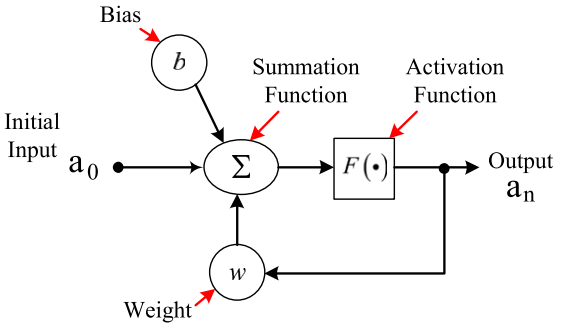

Fig. 1. Neuron with recurrent connection.

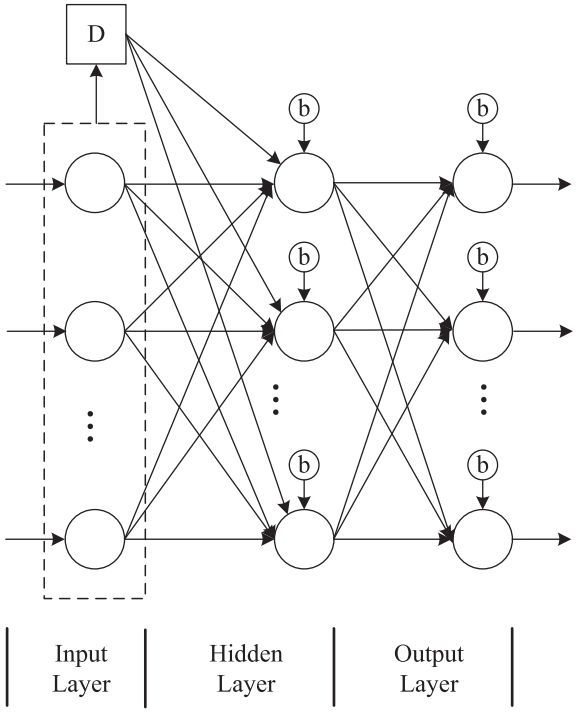

Fig. 2. A three-layer FFTD architecture.

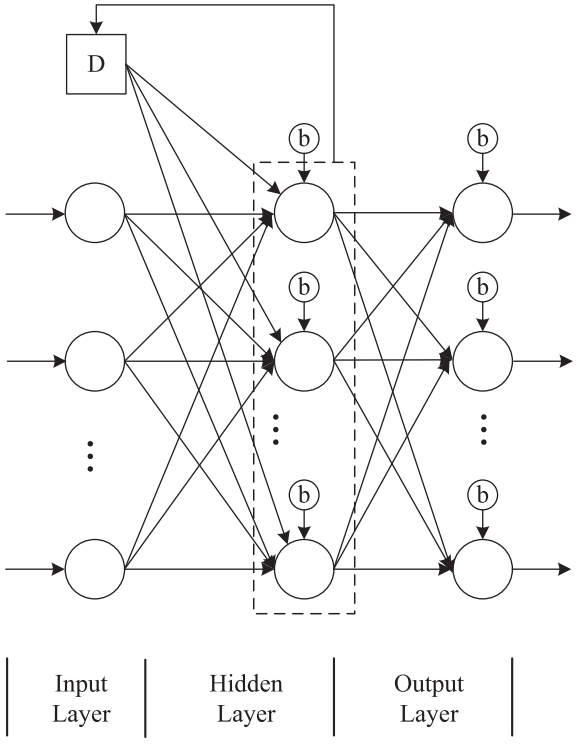

Fig. 3. A three-layer LRN architecture.

denote neurons, squares with solid lines are delay units, while dash-line squares take all outputs of the nodes within and transfer them to the corresponding positions indicated by the weighted arrows. The input layer is normally responsible for receiving the input vectors.

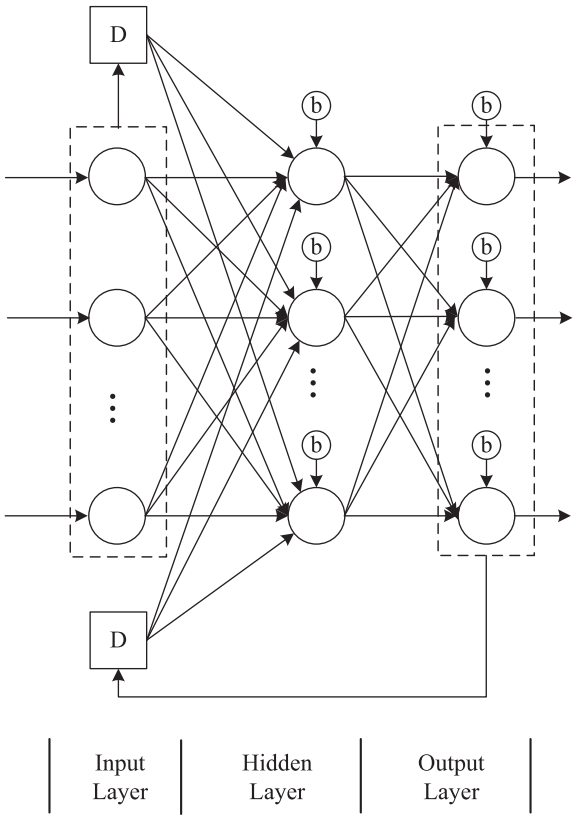

Fig. 4. A three-layer NARX architecture.

Kremer [37] has provided a mathematical formalisation that unifies several RNN models, including the ones considered in this work. Following Kremer's formulation, the FFTD network is described by the following equations with time index t,

$$o(t)^{\mathrm{FFTD}} = \psi\left(w^{out}h(t) + b^{out}\right), \tag{2}$$

$$h(t)^{\mathrm{FFTD}} = \psi\left(w^{hid}s(t) + b^{hid}\right) \tag{3}$$

and

$$s(t)^{\mathrm{FFTD}} = i(t) \oplus i(t-1) \oplus \cdots \oplus i(t-\tau_1), \tag{4}$$

where i is the input vector, w^{hid} and w^{out} the weight matrices of the hidden layer and the output layer, ψ a predefined arbitrary nonlinear activation function, o is the RNN output vector, h is the output vector of the hidden layer, $s(t)$ denotes the state vector at time t, $\oplus$ the Cartesian product, and τ_1 the number of delays.

As shown in Fig. 3, the LRN architecture includes a feedback connection from the hidden layer to the delay unit, and can be formulated as,

$$o(t)^{\mathrm{LRN}} = \psi(w^{out}h(t) + b^{out}), \tag{5}$$

and

$$h(t)^{\mathrm{LRN}} = \phi(\Lambda h(t-1) + w^{hid}i(t) + b^{hid}), \tag{6}$$

where Λ is a diagonal matrix.

Lastly, the NARX network shown in Fig. 4 can be stated as following,

$$o(t)^{\mathrm{NARX}} = \psi(w^{out}h(t) + b^{out}), \tag{7}$$

$$h(t)^{\mathrm{NARX}} = \phi\left(\Lambda h(t-1) + w^{hid}s(t) + b^{hid}\right) \tag{8}$$

and

$$s(t)^{\mathrm{NARX}} = \{i(t) \oplus i(t-1) \oplus \cdots \oplus i(t-\tau_2)\} \oplus \{o(t-1) \oplus o(t-2) \oplus \cdots \oplus o(t-\tau_3)\}, \tag{9}$$

where τ_2 is number of delays at the input and τ_3 is the number of output feedbacks. The details of connection topologies of these RNNs can be found in [51,52].

3. Training of RNNs within a deterministic framework of analysis and the quasi-Newton method

During training an RNN tries to find a set of weight values that would allow it to settle into one of several states for each input presented to it during operation. However, it is not possible to know a priori all possible inputs so RNN training is based on a particular set of data, the so-called training set.

In this work, training is considered within the framework of deterministic optimisation which leads to minimising an objective function, the so-called *error function*, E, and finding an "optimal" set of weights and biases on the basis of some training data. This can be formulated as the following deterministic unconstrained optimisation problem

$$\min E(W), \tag{10}$$

where $E : \Re^n \rightarrow \Re^1,\ E \in \mathcal{C}^2$, W is a set of vectors that consist of weights w between the layers of the RNN, and biases b of its nodes. In general, the objective function $E(W)$ takes the form of the Mean Squared Error (MSE)

$$E(W) = \frac{1}{P}\sum_{p=1}^{P}(y_p - \bar{y}_p)^2, \tag{11}$$

where $\bar{y}$ is the desired output vector, and y is the actual output of the RNN for input p. It is expected that an RNN trained that way would be able to perform well when unknown data are presented at the input by allowing its outputs to settle into the right state during operation.

In the context of deterministic unconstrained optimisation, quasi-Newton methods, sometimes called *variable metric methods*, are well-known algorithms for finding local minima of functions in the form of Eq. (10). Quasi-Newton methods are based on Newton's method to find the stationary point of a function, where the gradient is zero. Newton's method assumes that the function can be locally approximated as a quadratic in the region around the optimum, and requires the first and second derivatives [24], i.e. the gradient vector and the Hessian matrix, to find the stationary point. Moreover, the Newton's method and its variants require that the Hessian is positive definite – a condition that is difficult to guarantee in practice.

Quasi-Newton methods have been proposed in an attempt to alleviate some of the concerns regarding the application of Newton's method in real-world applications. At the k iteration, a quasi-Newton method has the following basic structure:

(1) set $d_k = -H_k g_k$;
(2) apply line search along d_k giving:

$$w_{k+1} = w_k + \alpha_k d_k; \tag{12}$$

(3) update H_k giving H_{k+1};

where d is the search direction, H is the Hessian approximation, g denotes the first derivative, and α the stepsize. The initial H is any given $n \times n$ symmetric positive definite matrix, and $H_k = B_k^{-1}$.

Methods that satisfy the quasi-Newton condition, i.e., $B_{k+1}\hat{s}_k = \hat{y}_k$, where $\hat{s}_k = w_{k+1} - w_k$, and $\hat{y}_k = g_{k+1} - g_k$, can be considered as members of the class of quasi-Newton methods. Some of the most famous approaches for updating B_{k+1} are the Davidon–Fletcher–Powell (DFP) formula:

$$B_{k+1}^{\mathrm{DFP}} = \left(I - \frac{\hat{y}_k\hat{s}_k^T}{\hat{y}_k^T\hat{s}_k}\right)B_k\left(I - \frac{\hat{s}_k\hat{y}_k^T}{\hat{y}_k^T\hat{s}_k}\right) + \frac{\hat{y}_k\hat{y}_k^T}{\hat{y}_k^T\hat{s}_k}, \tag{13}$$

the Broyden–Fletcher–Goldfarb–Shanno (BFGS) formula:

$$B_{k+1}^{\mathrm{BFGS}} = B_k - \frac{B_k\hat{s}_k(B_k\hat{s}_k)^T}{\hat{s}_k^T B_k\hat{s}_k} + \frac{\hat{y}_k\hat{y}_k^T}{\hat{y}_k^T\hat{s}_k}, \tag{14}$$

the Powell–Symmetric–Broyden (PSB) formula:

$$B_{k+1}^{\mathrm{PSB}} = B_k + \frac{(\hat{y}_k - B_k\hat{s}_k)\hat{s}_k^T + \hat{s}_k(\hat{y}_k - B_k\hat{s}_k)^T}{\hat{s}_k^T\hat{s}_k} - \frac{(\hat{y}_k - B_k\hat{s}_k)^T\hat{s}_k}{\left(\hat{s}_k^T\hat{s}_k\right)^2}\hat{s}_k\hat{s}_k^T \tag{15}$$

and the Broyden's class of methods, which uses a linear combination of the DFP and the BFGS updates

$$B_{k+1}^{Broyden} = (1-\xi)B_{k+1}^{\mathrm{BFGS}} + \xi B_{k+1}^{\mathrm{DFP}}, \quad \xi \in [0,1]. \tag{16}$$

There is a relatively large number of applications of quasi-Newton methods in training static neural networks, such as [1,3,30,38,46,47,55,56,66]. This approach exploits the idea of building up curvature information as the iterations of a training

method are progressing, and several techniques to update B_{k+1} have been applied to neural networks [26], such as Eq. (13) [15,16,21], Eq. (15) [22–24], and Eq. (14) [12,23,25,63].

As mentioned in [34,61,62] quasi-Newton methods for training static neural networks are fast to converge but in many cases converge to local minima. Although several efforts have been made to reduce the memory requirement of updating the Hessian approximation [5,32,38,45,59,65], the need of using a monotone line search and the drawback of getting trapped in neighbourhoods of local minimum points limit the application of these methods in real-world applications. Another problem in neural networks applications is that quasi-Newton methods suffer from large eigenvalues in the approximated Hessian matrices of the objective function [58].

Despite the emergence of the self-scaling approaches for the Hessian approximation in the field of numerical optimisation [66] (the fundamental concept of self-scaling is to accommodate the change of target variables efficiently), self-scaling is rarely introduced when training neural networks [47]. In addition, the literature of RNNs includes only very few attempts to train RNNs with quasi-Newton methods with limited results [5,7,8,17,34,37].

The self-scaling techniques can resolve problems caused by larger eigenvalues by scaling the Hessian approximation before it is updated at each iteration to keep the eigenvalues of the approximated Hessian matrix within a suitable range [66]. This technique was first proposed in [49,50], and was used to update the approximated Hessian as follows:

$$B_{k+1}^{Oren} = \left(B_k - \frac{B_k \hat{y}_k \hat{y}_k^T B_k}{\hat{y}_k^T B_k \hat{y}_k} + \theta v v^T\right)\gamma + \frac{pp^T}{p^T \hat{y}_k}, \tag{17}$$

where

$$\begin{aligned} p &= -\alpha B_k g, \\ v &= (\hat{y}_k^T B_k \hat{y}_k)^{1/2}\left(\frac{p}{p^T \hat{y}_k} - \frac{B_k \hat{y}_k}{\hat{y}_k^T B_k \hat{y}_k}\right), \\ \gamma &= \varphi \frac{g' p}{g' B_k \hat{y}_k} + (1-\varphi)\frac{p^T \hat{y}_k}{\hat{y}_k^T B_k \hat{y}_k} \end{aligned} \tag{18}$$

and $\varphi, \theta \in [0,1]$. After these attempts, more relative works have been developed, such as in [2,48].

In our approach, presented in detail in the following sections, we use the scaling factor ρ_k which was introduced for the BFGS method by [66]. This is defined as

$$B_{k+1}^{SCBFGS} = \rho_k \left[B_k - \frac{B_k \hat{s}_k \hat{s}_k B_k^T}{\hat{s}_k^T B_k \hat{s}_k}\right] + \frac{\hat{y}_k \hat{y}_k^T}{\hat{y}_k^T \hat{s}_k}, \tag{19}$$

where

$$\rho_k = \frac{\hat{y}_k^T \hat{s}_k}{\hat{s}_k^T B_k \hat{s}_k}. \tag{20}$$

Numerical evidence has shown that methods that apply a scaling factor for B_{k+1} are superior to the original quasi-Newton methods. Especially in real-world applications the scaling factor could potentially play an important role: when ρ_k is sufficiently large, the eigenvalues of B_{k+1} are relative small, with strong self-correcting property [66]. Despite this looks particularly appealing for training RNNs, to the best of our knowledge it has not been exploited at all in this area to improve the effectiveness of second-order training algorithms, and in [53] we conducted a preliminary study to explore its potential with promising results. Another useful characteristic of the factor ρ_k, which makes it useful in RNN training, is that it takes only the information of the most current point to scale the Hessian approximation and no user-defined parameters, compared to the factor γ in Eq. (17). This is particularly helpful when dealing with high-dimensional search spaces, such as the ones encountered in the applications discussed in this paper.

4. Nonmonotone BFGS training for recurrent neural networks

Typically, deterministic optimisation methods for RNN training require monotonicity of the error values, i.e. they reduce the error function at each iteration. Nevertheless, enforcing monotonicity does not guarantee that a method will efficiently explore the search space in the sense that it may be trapped in a local minimum point early on (e.g. when poorly initialised weights are used) and never jump out to reach a global one under ill conditions.

We propose here a nonmonotone approach to RNN training inspired by some of the characteristics that learning exhibits during cognitive development [20]. In the cognitive process, learning is typically defined as systematic change of behaviour, resulting from experiences, and is characterised by frequent errors and "U-shaped" patterns of behaviour in which good performance is sometimes followed by periods where performance deteriorates [19]. At the same time nonmonotone RNN training is consistent with recent research in nonlinear optimisation aiming at better exploring the search space, and can potentially enhance the convergence behaviour of RNN training methods.

From a deterministic optimisation perspective, nonmonotonicity can be introduced through conditions, such as those initially proposed by Grippo et al. [27], for finding a stepsize that occasionally permits an increase in the function value while retaining global convergence of the minimisation method:

$$E(w_k + \alpha_k d_k) \leqslant \max_{0 \leqslant j \leqslant m(k)} \{E(w_{k-j})\} + \epsilon_1 \alpha_k g_k^T d_k \tag{21}$$

and

$$g(w_k + \alpha_k d_k)^T d_k \leqslant -\epsilon_2 g_k^T d_k, \tag{22}$$

where $0 < \epsilon_1 < \epsilon_2 < 1/2$, $m(0) = 0$, and m is updated by the rule:

$$m(k) = \min\{m(k-1) + 1, M\}. \tag{23}$$

The parameter $m(k)$ plays the role of a memory element, or buffer, and is typically a nondecreasing integer (cf. with Eq. 23), bounded by a nonnegative prefixed integer M. Another approach proposed recently is to replace *max* in Eq. (21) by an average of function values [67]. Lastly, Grippo et al. [28] proposed the use of a slightly different approach that employs the following condition instead of Eq. (22)

$$\|d_k\| \leqslant \alpha \beta^h, \tag{24}$$

where $\alpha > 0$ and $\beta \in (0, 1)$ are user-defined real numbers and h is an integer that increases by one unit whenever the condition is satisfied. If Eq. (24) is satisfied, $\alpha_k = 1$ and the new point, w_{k+1}, is accepted without evaluating the objective function. If Eq. (24) does not hold then Eq. (21) is used to determine the stepsize α_k.

Grippo et al. also proved the following theorem that describes the convergence properties of quasi-Newton algorithms that adopt a nonmonotone strategy.

Theorem 4.1 [28]. *Let $\{w_k\}$ be a sequence produced by an iterative scheme of the form (12), where the stepsize α_k is computed by Eqs. (21) and (24). Assume that:*

(H4.1) *the level set $\Phi = \{w | E(w) \leqslant E(w_0)\}$ is compact;*
(H4.2) *positive numbers ϵ, c, p_1 and p_2 exist such that the following conditions hold:*

$$g_k^T d_k \leqslant -\epsilon \|g_k\|^{p_1},$$
$$\|d_k\|^{p_2} \leqslant c \|g_k\|.$$

Then either the algorithm terminates at some point w_μ such that $g(w_\mu) = 0$, or it generates an infinite sequence such that:

(1) *the sequence $\{w_k\}$ remains in a compact set and every limit point w^* belongs to the level set and satisfies $g(w^*) = 0$;*
(2) *no limit point of $\{w_k\}$ is a local maximum of E;*
(3) *if the number of stationary points of E in Φ is finite or there exists a limit point where H is nonsingular, the sequence $\{w_k\}$ converges.*

Although Grippo et al. [28] do not directly make any assumptions about the convexity of the objective function, they assume that the search direction is computed by minimising the quadratic approximation of the objective function at the current point. Furthermore, they demonstrate that this scheme works well even when the search direction is computed approximately by means of a truncated quasi-Newton algorithm with finite difference approximations of second-order derivatives.

Theorem 4.1 can be specialised to algorithms of the Newton class, such as those employing the updates defined in Eqs. (13)–(16), by imposing appropriate form in the conditions H4.2. Thus, when the search direction is defined by

$$d_k = -B_k^{-1} g_k \tag{25}$$

and $\{B_k\}$ is a sequence of symmetric positive definite matrices with uniformly bounded eigenvalues $\lambda(B_k)$, i.e. there exist λ, Λ such that for all k:

$$0 < \lambda \leqslant \lambda_i(B_k) \leqslant \Lambda. \tag{26}$$

Then

$$g_k^T d_k \leqslant -\Lambda^{-1} \|g_k\|^2, \tag{27}$$
$$\|d_k\| \leqslant \lambda^{-1} \|g_k\|. \tag{28}$$

Since RNNs' error functions are nonconvex, we present and discuss below the main theoretical results for global convergence of nonmonotone BFGS methods that hold in this case. It is worth mentioning that proving global convergence for nonconvex objective functions is a very challenging problem that has not been explored totally yet. Also it is important to distinguish between the notion of *global convergence* and that of *global optimisation*: a globally convergent algorithm always reaches a

minimiser (not necessarily the global minimiser) starting from almost any initial weight [41]. Here we are based on the work of Yin and Du [66] which applies the nonmonotone technique of Han and Liu [33] stated as follows:

$$E(w_k + \alpha_k d_k) \leqslant \max_{0 \leqslant j \leqslant M}\{E(w_{k-j})\} - \delta \min\{\sigma_1(\mu_k), \sigma_2(v_k)\} \tag{29}$$

and

$$g(w_k + \alpha_k d_k)^T d_k \geqslant \beta g_k^T d_k, \tag{30}$$

where

$$\mu_k = -\frac{g_k^T d_k}{\|d_k\|} \tag{31}$$

and $v_k = -\alpha g_k^T d_k,\ 0 < \delta < \beta < 1$, and σ_1 and σ_2 are two forcing functions, which are used to measure the sufficiency of descent and prove convergence. As shown in [33], Eqs. (29) and (30) formulate one of the most general types of line search, which has as special cases many monotone and nonmonotone techniques. Furthermore, Sun et al. [64] have shown that the nonmonotone Armijo rule, the nonmonotone Goldstein rule, and the nonmonotone Wolfe rule employ special forms of forcing functions.

Before presenting the main theorem for global convergence, the following Assumptions (**H5**) are needed.

(H5.1) The level set $\Psi = \{w \in \Re^n : E(w) \leqslant E(w_0)\}$ is bounded.
(H5.2) In some neighbourhood $\aleph(\Psi)$ of Ψ, the gradient of $E(w), g(w)$ is Lipschitz continuous, that is, there exists a constant $L > 0$ such that for all $w, \bar{w} \in \aleph(\Psi)$,

$$\|g(w) - g(\bar{w})\| \leqslant L\|w - \bar{w}\|.$$

Below we present the theorem of Yin and Du that needs Assumptions (**H5**). It makes use of the BFGS property to generate positive definite matrices B_k^{BFGS} [24], and exploits their own result that the update Eq. (19) preserves the positive definiteness of the matrices B_k^{SCBFGS}.

Theorem 4.2 [66]. *Suppose that Assumptions (**H5**) hold, and let us assume that w_0 is any starting point, B_0 is any symmetric positive definite matrix, and that the sequence $\{w_k\}$ is generated by the iterative scheme (12), where $d_k = -(B_k^{\mathrm{SCBFGS}})^{-1} \cdot g_k$, and the stepsize α_k is determined by Eqs. (29) and (30). If there exists a positive constant $K \geqslant 1$ for which*

$$\|\hat{y}_k\| \leqslant (1 - \beta)\|g_k\|$$

for all $k \geqslant K$, then

$$\liminf_{k \to \infty} \|g_k\| = 0. \tag{32}$$

Assumption (H5.1) holds for training RNNs of a fixed architecture on a finite set of training patterns because the error function is bounded below in $\Re^n$ since $E \geqslant 0$: if a w^* exists such that $E(w^*) = 0$ then w^* is the global minimum; otherwise the vector w with the smallest available value is the global minimiser. Assumption (H5.2) also holds for RNNs that use smooth enough activation functions (the derivatives of order p are available and continuous), such as the logistic function that is used in our experiments later in the paper. Moreover, H5.2 implies that there exists a constant c such that $\|g_k\| \leqslant c,\ \forall w \in \aleph(\Psi)$. A detailed proof is provided in [66], which shows that the limit (32) is the best type of global convergence result that can achieved for nonconvex functions.

5. Self-scaling BFGS with adaptive nonmonotone strategy

In this section we present the proposed algorithm, named *Adaptive Self-scaling NonMonotone BFGS* (ASCNM-BFGS), through the high-level description presented below.

5.1. Algorithm model of the self-scaling BFGS with adaptive nonmonotone strategy

STEP 0. Initialise $w_0, k = 0$, a symmetric positive definite matrix $B_0, M^0 = 0, M^{max}$ is an upper boundary for $M^k, \alpha_0 \in (\lambda_1, \lambda_2),\ 0 < \lambda_1 < \lambda_2$ are positive constants, $d_0 = -g(w_0)$ and $\sigma, \delta \in (0, 1)$.
STEP 1. If $g(w_k) = 0$, stop.
STEP 2. If $k \geqslant 1$, Calculate a local approximation of the Lipschitz constant $\Lambda^k = \frac{\|g(w_k) - g(w_{k-1})\|}{\|w_k - w_{k-1}\|}$ and adapt M^k using the scheme:

$$M^k = \begin{cases} M^{k-1} + 1, & \text{if } \Lambda^k < \Lambda^{k-1} < \Lambda^{k-2}, \\ M^{k-1} - 1, & \text{if } \Lambda^k > \Lambda^{k-1} > \Lambda^{k-2}, \\ M^{k-1}, & \text{otherwise}, \end{cases}$$

where $M^k = \min\{M^k, M^{\max}\}$;

STEP 3. $\forall k \geqslant 1$, set $\alpha_k = \max\{\alpha_{k-1}, \bar{\alpha}_k\}$, where $\bar{\alpha}_k = \frac{|2[E(w_k)-E(w_{k-1})]|}{g^T(w_k)\cdot d_k}$, and check that α_k satisfies the nonmonotone condition

$$E(w_k + \alpha_k d_k) \leqslant \max_{0\leqslant j\leqslant M^k}\{E(w_{k-j})\} + \delta\cdot\alpha_k\cdot g^T(w_k)\cdot d_k;$$

otherwise find stepsize $\alpha_k := \alpha_k \cdot \sigma^{l_q}$ that satisfies the nonmonotone condition, setting each time $l_q := l_q + 1$.
STEP 4. Generate a new weight vector $w_{k+1} = w_k + \alpha_k d_k$.
STEP 5. Update the search direction d_k using the Hessian approximation B_k calculated by the self-scaling BFGS formula

$$B_{k+1} = \rho_k\left[B_k - \frac{B_k\hat{s}_k\hat{s}_k B_k^T}{\hat{s}_k^T B_k\hat{s}_k}\right] + \frac{\hat{y}_k\hat{y}_k^T}{\hat{y}_k^T\hat{s}_k},$$

where $\hat{s}_k = w_{k+1} - w_k, \hat{y}_k = g_{k+1} - g_k$ and the self-scaling factor is $\rho_k = \frac{\hat{y}_k^T\hat{s}_k}{\hat{s}_k^T B_k\hat{s}_k}$; set $d_k = -B_k^{-1}g_k$.
STEP 6. Set $k := k + 1$ and $l_q = 0$, go to STEP 1.

A feature of the ASCNM-BFGS method is the use of an adaptive memory element M_k, called *nonmonotone learning horizon*, instead of a fixed heuristic value. To this end, it calculates in Step 2 a local estimation of the Lipschitz constant, which could provide helpful information on the morphology of a function, and uses it to automatically adapt the size of M (see also [56]

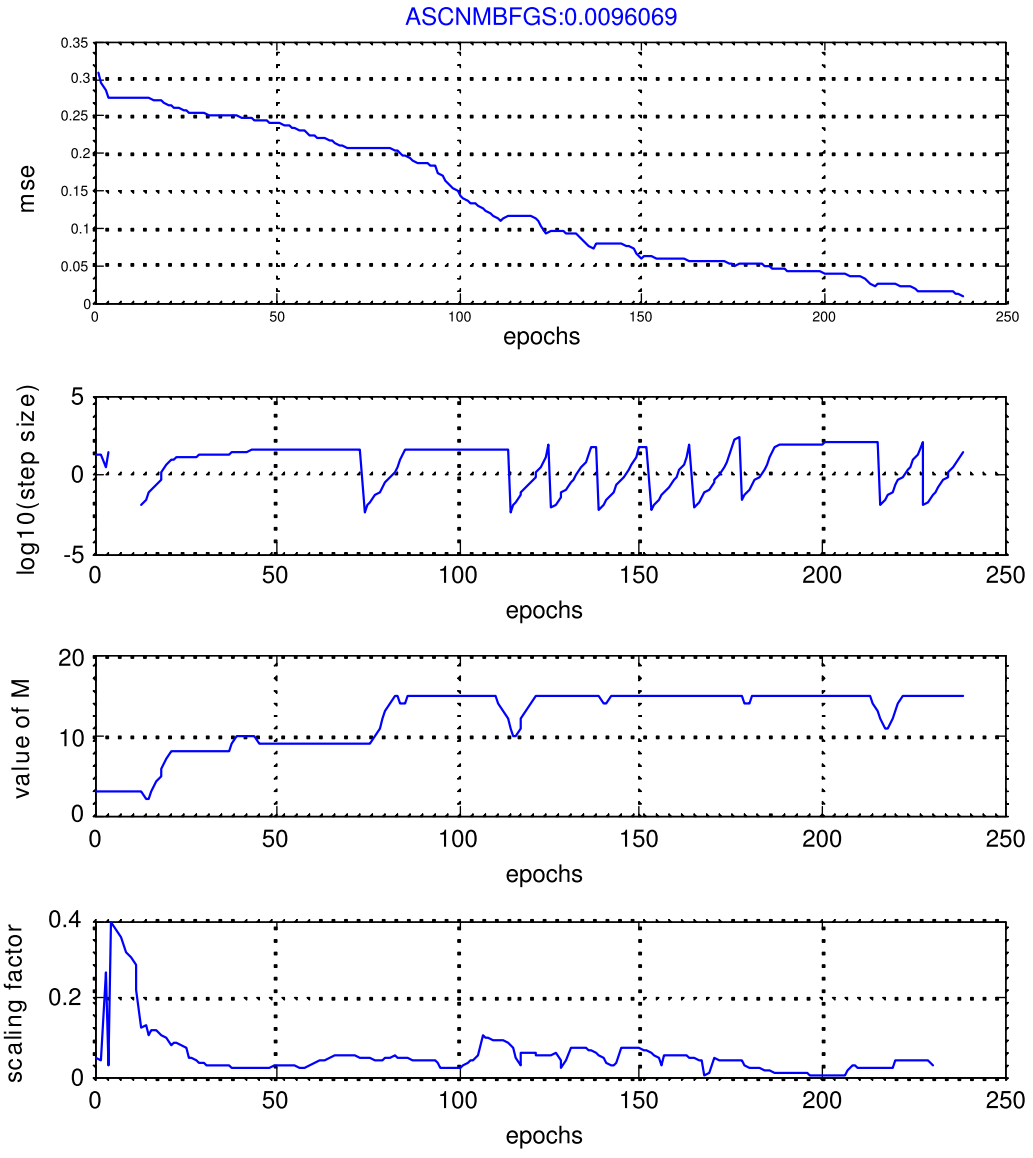

Fig. 5. Convergence behaviour of FFTD network trained with the ASCNM-BFGS method in the P5 problem.

for the usefulness of this estimate). The local estimation of the Lipschitz constant gets large values in steep regions of the search space and small values in flat areas. At the beginning, i.e. $k < 3$, there is not enough information to adapt M through the local estimation of the Lipschitz, and as a result the nonmonotone conditions in Step 3 actually operates as a monotone one comparing the new function value against the previous one.

Also the initial choice of the stepsize merits some attention. At $k = 0$ the stepsize is an arbitrary positive real number randomly chosen in the interval (λ_1, λ_2) and the algorithm operates in the direction of the negative of the gradient, $d_0 = -g(w_0)$. That quickly changes, $\forall k \geqslant 1$, as the search direction is updated through the self-scaling BFGS update equation, which tunes the Hessian approximations at every iteration the eigenvalues possess large values; when ρ_k is sufficiently large, then the eigenvalues of B_k^{SCBFGS} are small. The stepsize is then initialised through $\bar{\alpha}_k$, following a technique suggested by Charalambous [14], and constantly tuned to ensure that, whilst it is not smaller than the stepsize of the previous iteration, it satisfies the nonmonotone condition in Step 3. This condition regulates the sufficient decrease of the error function through the forcing function $+\delta \cdot \alpha_k \cdot g^T(w_k) \cdot d_k$, whilst for $k = 1, 2$ this condition is reduced to the monotone Armijo rule (cf. with Theorem 2, [43]).

The algorithm also employs some heuristic parameters: an upper bound for M^k to help the algorithm concentrate on the recent past, while, in Step 4, σ regulates the stepsize, i.e. the larger σ the smaller trial stepsize is used, while δ controls the amount of change. The error function E is calculated through the Mean Squared Error (MSE) formula (11), while the gradient is calculated using the Backpropagation-Through-Time (BPTT) formulae [6].

To illustrate the behaviour of the method we provide below some examples of convergence behaviour from learning the parity-5 problem, [61], using RNNs of the three types discussed above, namely the FFTD network, the LRN and the NARX network. Figs. 5–7 illustrate the behaviour of the MSE, the stepsize, the value of M, and the scaling factor. Despite the nonmonotonone behaviour that one can observe in the MSE values, it is clear that there is a trend toward smaller learning errors,

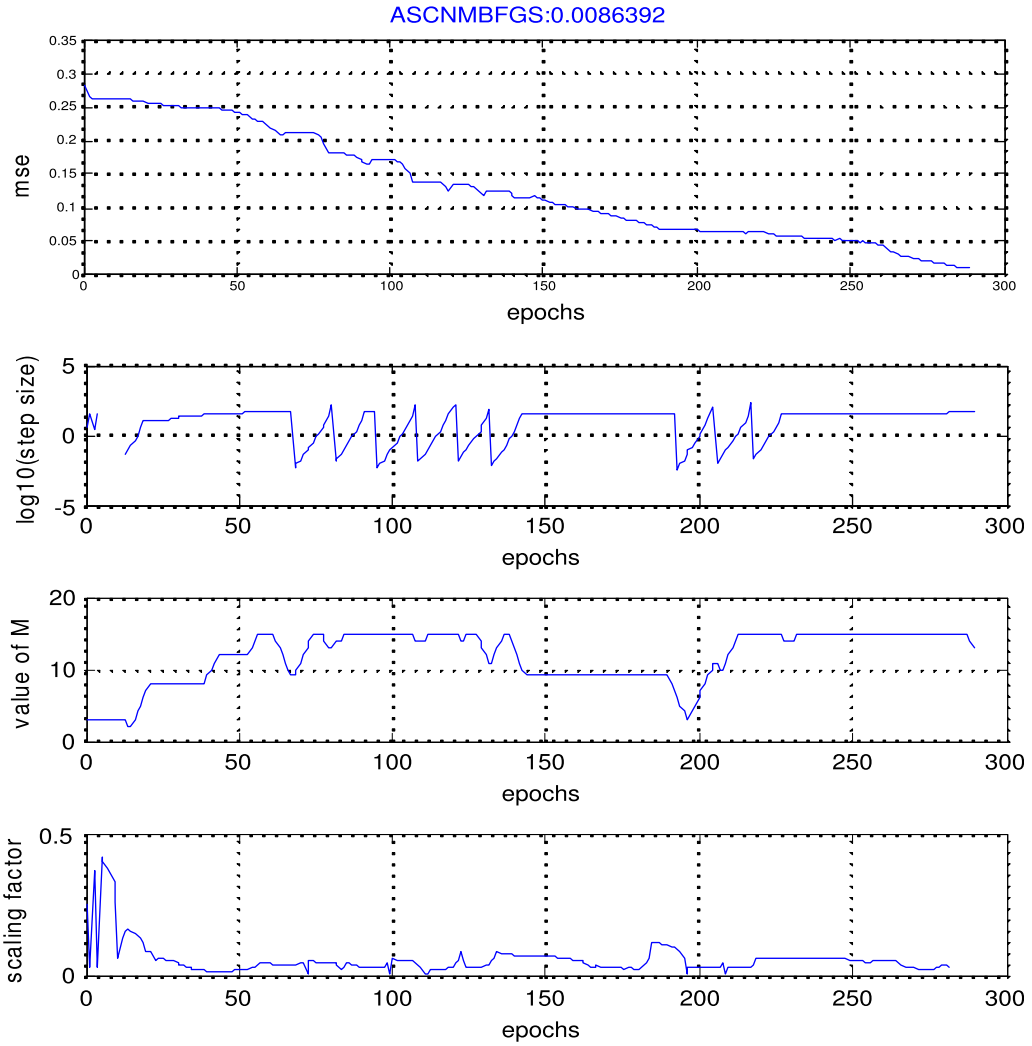

Fig. 6. Convergence behaviour of LRN trained with the ASCNM-BFGS method in the P5 problem.

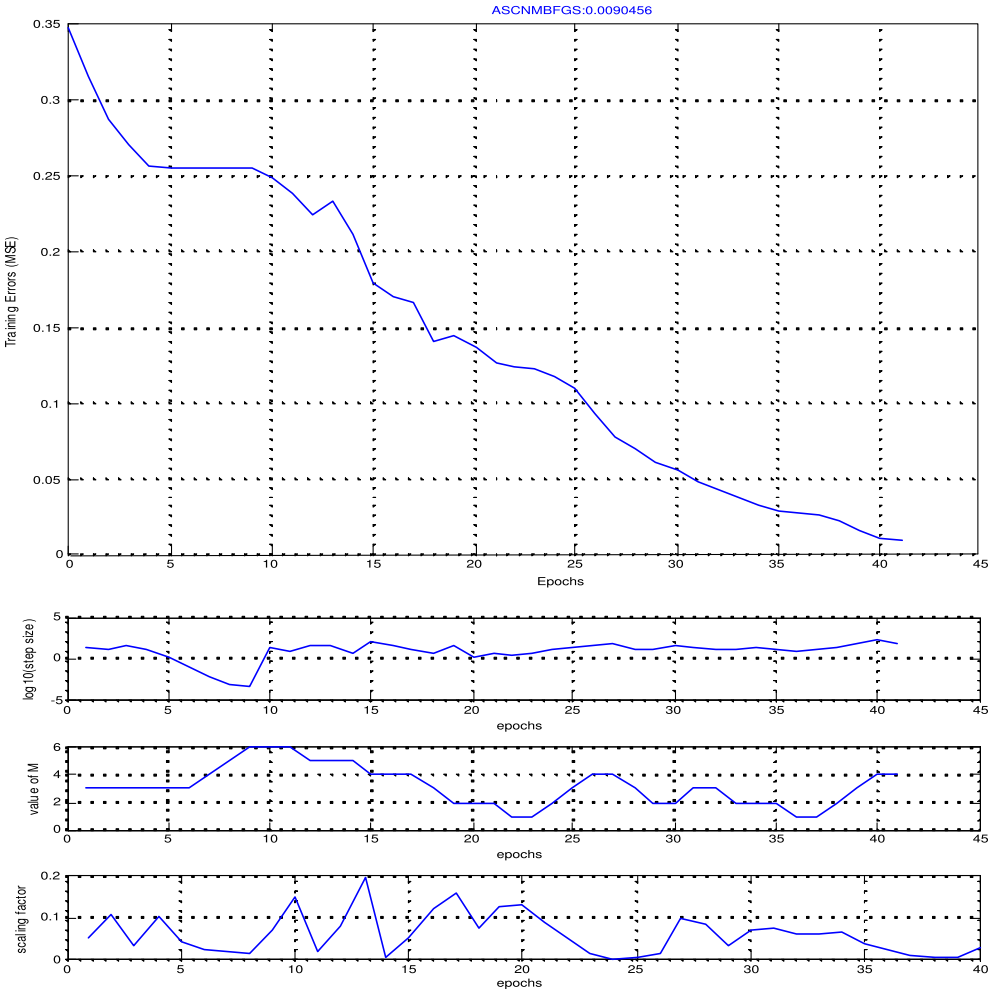

Fig. 7. Convergence behaviour of NARX network trained with the ASCNM-BFGS method in the P5 problem.

whilst in all cases, the use of adaptive M does not affect the stability of the method. The scaling factor behaviour indicates the self-correcting property of the method, which results in smaller eigenvalues for B_k^{SCBFGS} for relatively larger ρ values.

6. Experiments and results

In this section, we present simulation results on the following problems, parity-5 and parity-10, Sequence Classification, Sequence Learning, and Reading Aloud. Learning the parity-N sequence has been treated as a standard test problem for new algorithms operating in the presence of strong local minima and saddle points, while the other problems are used as examples of temporal sequence applications in order to assess the generalisation performance of the trained RNNs. The description of each problem is followed by the setting of training parameters and the numerical results. All simulations in this paper were coded in Matlab 7.2 running on Windows XP platform. The notation used in the tables below is as follows: *#hid* indicates the number of hidden nodes used; *Conv* indicates the percentage of runs that met the MSE condition within predefined epochs, if applicable; *MSE* gives the average mean-squared-error (in %); *Ave.* provides the average number of epochs achieved by each method at the end of training; *Min.*, *Max.* and *Std.* give the minimum, the maximum and the standard deviation of epochs for runs that converged to the MSE condition; while *CE* (in %) represents the classification error in the testing set (i.e. generalisation performance). All results reported in this paper are averaged over 100 runs using randomly initialised weights and biases.

6.1. Parity-N problem

The N-bit parity problem has been widely used to verify the performance of novel training algorithms [55]. We consider here two instances: the parity-5 (P5) and the parity-10 (P10). The stopping criterion is set to a Mean-Squared-Error = 0.01

within 2000 epochs for the P5 and 4000 epochs for the P10. In both cases, the heuristic parameters are set as $3 \leqslant M^k \leqslant 15$, $\sigma = 0.9$ and $\delta = 0.01$. Numerical results for the three RNNs architectures in the P5 and P10 problems are shown in Tables 1–6, respectively, while Figs. 8–13 provide examples of learning behaviours using the BFGS and the ASCNM-BFGS.

As shown in Tables 1–3, the performance of the new method for the parity-5 problem employing three different neural architectures, i.e. Feedforward Time Delay (FFTD), Elman's Recurrent Network (LRN) and Nonlinear Autoregressive Network with Exogenous Inputs (NARX), using 1, 2, 5 or 7 hidden nodes is always better than the original BFGS. For example, BFGS-trained NARX networks using 5 hidden nodes converged in 59 out of 100 runs (see Table 3), exhibiting a 100-run average

Table 1
Average performance of FFTD networks in the P5 problem.

Algorithm	#Hid	Conv.	MSE	Ave.	Min.	Max.	Std.
BFGS	1	0	23.825	–	–	–	–
	2	0	20.862	–	–	–	–
	5	4	9.229	1933	53	1171	331
	7	19	5.131	1679	33	1719	682
ASCNM–BFGS	1	0	23.794	–	–	–	–
	2	0	17.487	–	–	–	–
	5	30	3.336	1616	52	1974	653
	7	74	1.772	803	61	1983	327

Table 2
Average performance of LRNs in the P5 problem.

Algorithm	#Hid	Conv.	MSE	Ave.	Min.	Max.	Std.
BFGS	1	0	23.195	–	–	–	–
	2	0	19.978	–	–	–	–
	5	5	9.205	1928	53	1568	339
	7	15	5.635	1173	34	1927	573
ASCNM–BFGS	1	0	22.849	–	–	–	–
	2	0	17.699	–	–	–	–
	5	30	3.316	1617	54	1978	652
	7	77	2.029	758	63	1951	762

Table 3
Average performance of NARX networks in the P5 problem.

Algorithm	#Hid	Conv.	MSE	Ave.	Min.	Max.	Std.
BFGS	1	15	24.979	903	254	1983	1003
	2	31	11.370	688	121	1537	892
	5	59	3.862	373	72	1244	538
	7	68	1.979	146	63	893	301
ASCNM–BFGS	1	100	0.551	14	3	62	10
	2	100	0.586	17	5	53	10
	5	100	0.543	16	3	39	6
	7	100	0.595	15	5	33	5

Table 4
Average performance of FFTD networks in the P10 problem.

Algorithm	#Hid	Conv.	MSE	Ave.	Min.	Max.	Std.
BFGS	1	0	24.977	–	–	–	–
	2	0	23.307	–	–	–	–
	5	0	14.067	–	–	–	–
	7	4	9.612	2022	1047	3091	859
	10	12	6.428	3697	327	3949	912
ASCNM–BFGS	1	0	24.925	–	–	–	–
	2	0	23.349	–	–	–	–
	5	1	11.253	1520	1520	1520	0
	7	15	2.938	1848	711	3743	856
	10	57	1.786	2798	671	3983	1198

Table 5
Average performance of LRNs in the P10 problem.

Algorithm	#Hid	Conv.	MSE	Ave.	Min.	Max.	Std.
BFGS	1	0	24.862	–	–	–	–
	10	0	5.873	–	–	–	–
ASCNM–BFGS	1	0	24.794	–	–	–	–
	10	100	0.964	1736	889	1970	841

Table 6
Average performance of NARX networks in the P10 problem.

Algorithm	#Hid	Conv.	MSE	Ave.	Min.	Max.	Std.
BFGS	1	0	24.998	–	–	–	–
	2	0	17.632	–	–	–	–
	5	34	7.659	1352	591	3439	1387
	7	52	4.205	855	273	2155	601
	10	67	3.837	492	107	1563	489
ASCNM–BFGS	1	100	0.788	13	4	121	12
	2	100	0.739	21	6	91	14
	5	100	0.742	18	6	40	6
	7	100	0.726	17	5	30	5
	10	100	0.730	18	5	28	4

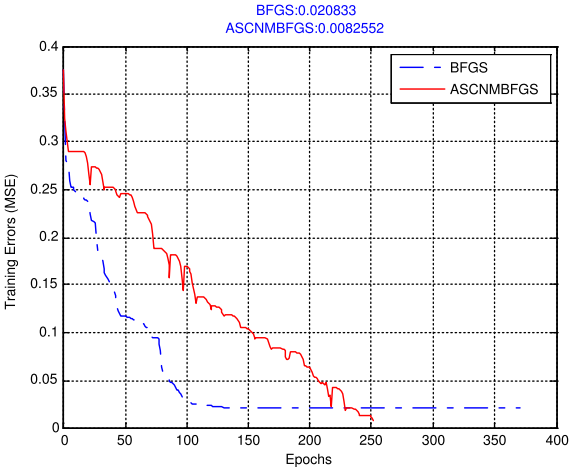

Fig. 8. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained FFTD networks in the P5 problem.

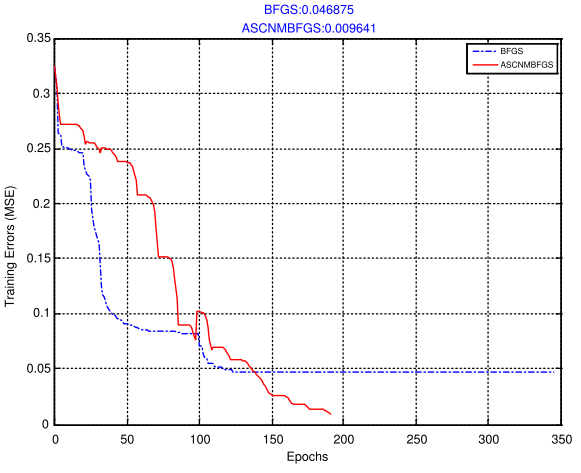

Fig. 9. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained LRN in the P5 problem.

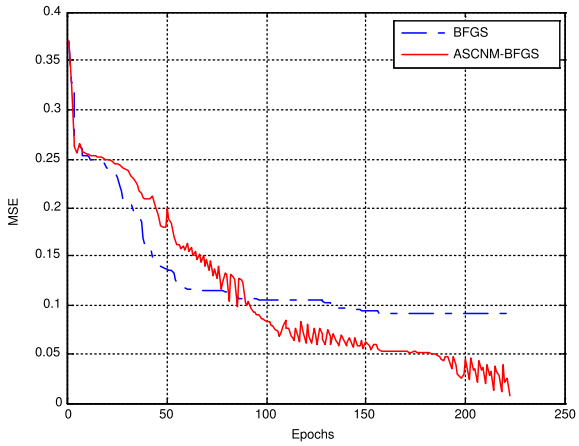

Fig. 10. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained NARX network in the P5 problem.

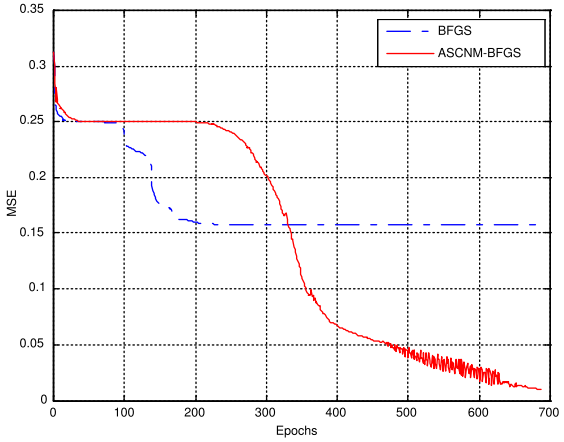

Fig. 11. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained FFTD network in the P10 problem.

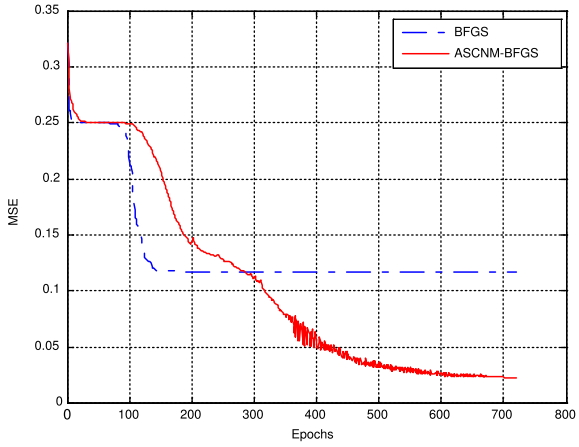

Fig. 12. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained LRN in the P10 problem.

MSE of 0.09229, while the proposed method reaches 100% convergence rate with improvements that are 6 times better in terms of MSE, 22 times smaller in the average and minimum number of training epochs, 30 times smaller in the maximum number of training epochs, and a 88 times smaller in the value of standard deviation for the converged runs.

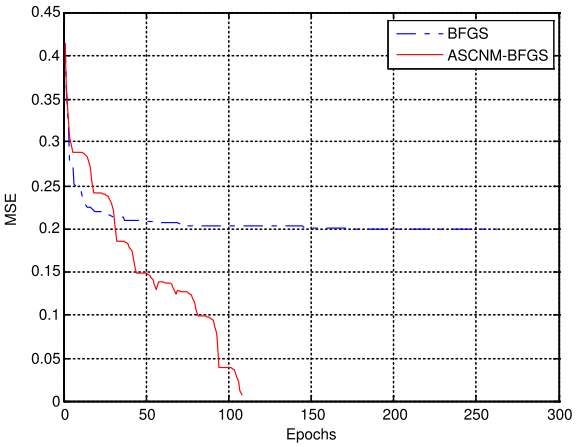

Fig. 13. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained NARX network in the P10 problem.

In general, experimental results in Tables 1–6 provide evidence that the new method is able to locate minimisers with smaller function values than the original method, which is important in certain real-world problems to provide good generalisation. For example, in Table 4, 12% of the BFGS-trained FFTD networks reached an MSE = 0.01 in a maximum of 3949 epochs, while the average MSE achieved by BFGS in that case was 0.06428. That was caused by the fact that the majority of the BFGS-trained networks did not reach the MSE goal within 4000 training epochs; some of them stuck to minima with higher function values while others failed to converge because of instabilities in the Hessian. When BFGS fails to reach the error goal we only provide the average error obtained. Also a 0% convergence in Tables 1, 2, 4, 5 and 6 indicates that not a single run of the BFGS method converged within the predefined number of epochs, and since only epochs of the converged runs are reported, we enter the symbol "–" in the corresponding cells. We observed that the ASCNM-BFGS method provided consistently a stable behaviour with the use of the scaling factor and a better ability to escape from swallow local minima, which could be attributed to its nonmonotone behaviour; some examples of improved learning behaviour are illustrated in Figs. 8–13.

6.2. Sequence Classification

This problem [40] concerns recognising the task from a sequence of events generated by end-users of a personalised system as they perform various tasks. A task consists of a sequence of events, such as the ones generated when opening a browser window, searching for information, saving information from the search results or storing bookmarks, and users may execute it in slightly different ways depending on their preferred way of seeking or processing information.

The dataset consists of a sequence of 203 training and 73 testing patterns (events), which are represented by 36-dimensional binary patterns for the inputs and 3-dimensional binary patterns for the outputs representing three different tasks. As the patterns represent events and are collected over time, they generate a sequence that is particularly noisy because some events can occur during execution of more than one task, and tasks can have varying lengths.

For this problem, 10-hidden node FFTD, LRN and NARX networks, with 763, 503 and 433 of weights and biases, respectively, were trained 200 epochs and the training goal is set to MSE = 0.01. The amounts of input/output delays are 5 for FFTD and 1-input-1-output delays for NARX. All other training parameters are the same as the parity-*N* problem. Table 7 shows the average performance in terms of MSE (%) achieved in training and CE (%) in testing. In all cases, the proposed algorithm achieves better MSE, from 0.1% to 12%, and CE, from 0.2% to about 20%, with LRNs producing better generalisation (i.e. lower CE) than the other RNNs. Examples of learning behaviours are in Figs. 14–16, showing how the nonmonotone strategy helps locating minimisers with lower error values, which leads to lower average classification error in testing (cf. with Table 7).

Table 7
Average performance of the three RNNs architectures in the Sequence Classification problem.

RNN	Algorithm	MSE% (training)	CE% (testing)
FFTD	BFGS	21.484	33.363
	ASCNM–BFGS	20.326	32.041
LRN	BFGS	21.518	32.301
	ASCNM–BFGS	9.175	11.534
NARX	BFGS	7.496	27.247
	ASCNM–BFGS	7.100	27.082

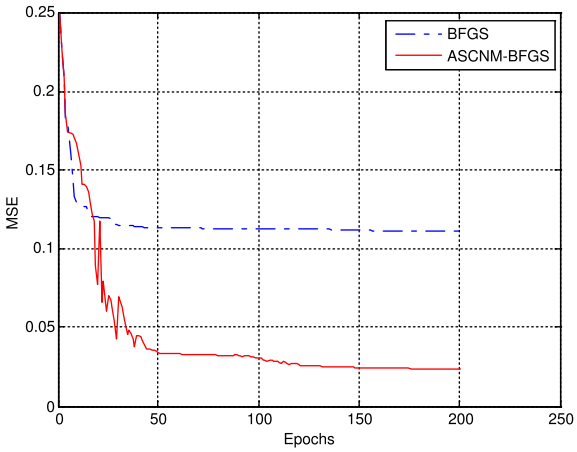

Fig. 14. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained FFTD network in the Sequence Classification problem.

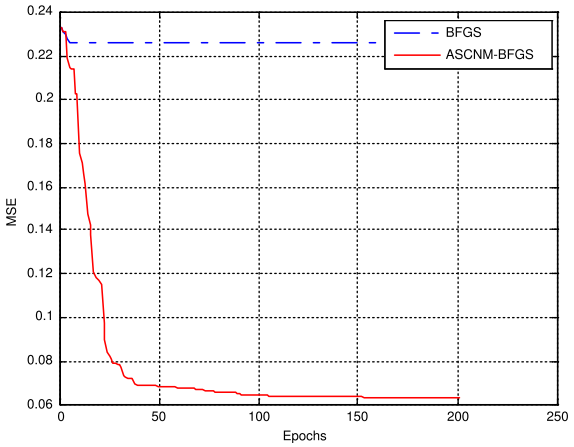

Fig. 15. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained LRN in the Sequence Classification problem.

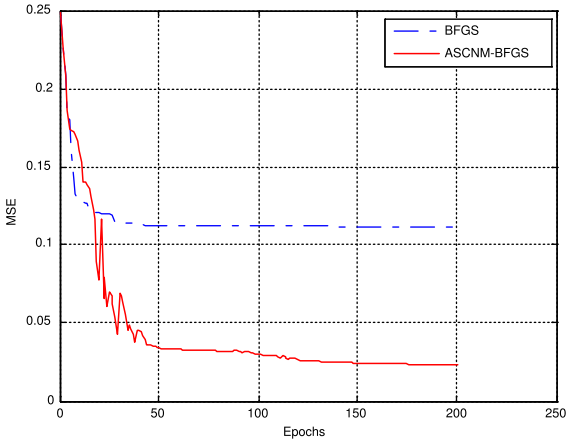

Fig. 16. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained NARX network in the Sequence Classification problem.

6.3. Sequence Learning

In this application [44], a grammar is given that consists of a set of six letters {a, b, d, i, g, u} that are used to generate letter sequences. A sequences consists of combinations of strings {ba}, {dii} and {guuu} and all possible permutations of these 3 strings are legal, with each letter represented by 4-bit binary codes. The task concerns predicting successive letters in a random sequence of 1000 words and each word consists of one of the above 3 consonant–vowel combinations. A sequence of length 2993 is used for training and a sequence of length 9 for testing, as in the original work [44].

We adopted the approach of [44] and trained the RNNs for 23 epochs measuring the MSE in training and testing. We kept the same settings as in the parity-*N* problem for the BFGS and ASCNM-BFGS heuristic parameters in order to test the robustness of the method. We made 100 independent runs for each of the three RNN architectures to estimate the average generalisation performance of the two algorithms.

Table 8
Average performance of the three RNNs architectures in the Sequence Learning problem.

RNN	Algorithm	MSE% (Training)	MSE% (Testing)
FFTD	BFGS	21.485	17.437
	ASCNM–BFGS	17.275	15.457
LRN	BFGS	23.347	32.301
	ASCNM–BFGS	17.854	15.070
NARX	BFGS	8.991	9.846
	ASCNM–BFGS	7.584	8.313

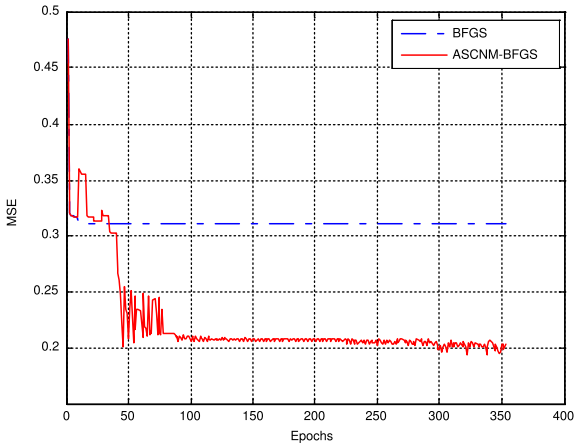

Fig. 17. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained FFTD network in the Sequence Learning problem.

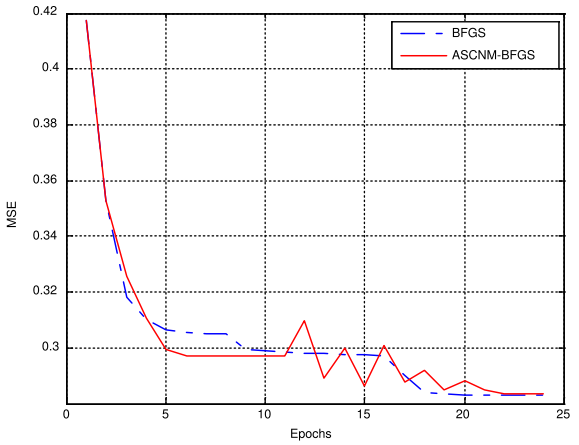

Fig. 18. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained LRN in the Sequence Learning problem.

It is worth mentioning that the average performance presented in Table 8 was achieved using RNNs with 10 hidden nodes (i.e., numbers of weights and biases are: 134 for FFTD, 194 for LRN, and 214 for NARX), as in [44]. For comparison we should mention that in [44] LRNs are only used and the MSE obtained is 25% in training (using a first-order training method) and 22% in testing. As shown in Figs. 17–19, the nonmonotone strategy helps to located solutions that lead to low errors, while the monotone BFGS suffers from convergence to minimisers with higher values. We also investigated the performance of NARX networks with smaller number of hidden nodes. Results in Table 9 show that the new method is able to train networks with 2 and 5 hidden nodes better than the BFGS.

6.4. Reading Aloud

This task concerns learning the mapping of a set of orthographic representation to their phonological forms [57]. Both subsets of orthography and phonology have 3 different parts, i.e. onset, vowel and coda, with 30, 27 and 48, and 23, 14 and 24 possible characters, respectively. Examples of words included in the dataset are: bed, keep, bike, boy. Data are in the form of a temporal sequence which consists of 105-dimensional input patterns and 61-dimensional output patterns, and the training dataset has 2998 patterns. Ideally, a specially designed RNN architecture with 100 hidden nodes (26582 adjustable weights), which is described in detail in [57], is needed to learn this dataset. Although in the original work there is no special testing dataset, we choose 30 words which are not included in the original training set from an online dictionary in order to verify our algorithm's generalisation ability.

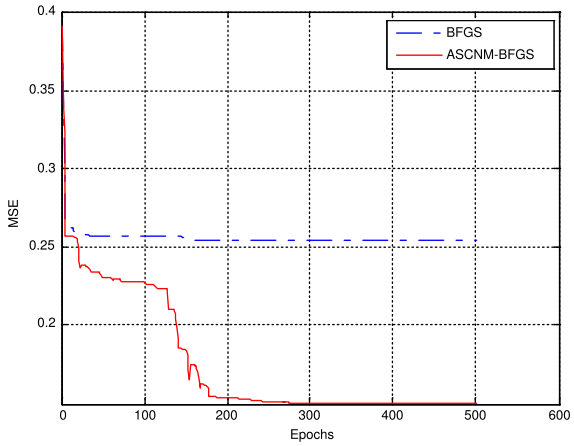

Fig. 19. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained NARX network in the Sequence Learning problem.

Table 9
Average MSE for NARX networks in the Sequence Learning problem.

Algorithm	#Hid	MSE% (training)	MSE% (testing)
BFGS	2	20.196	20.824
	5	11.029	12.119
	10	8.991	9.846
ASCNM–BFGS	2	19.972	20.792
	5	9.740	10.360
	10	7.584	8.313

Table 10
Average performance of FFTD and NARX networks in the Reading Aloud problem.

Net	Algorithm	#Hid	MSE% (Training)	MSE% (Testing)
FFTD	BFGS	5	10.665	18.716
		10	6.982	15.840
	ASCNM–BFGS	5	9.558	18.143
		10	6.081	15.652
NARX	BFGS	5	8.589	16.625
		10	6.248	15.407
	ASCNM–BFGS	5	7.080	16.106
		10	5.184	14.547

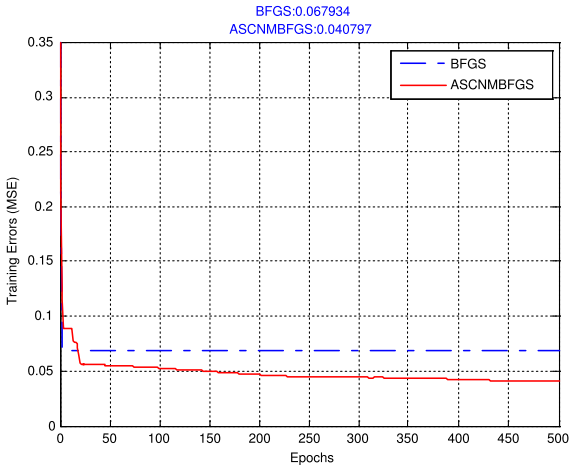

Fig. 20. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained FFTD network in the Reading Aloud problem.

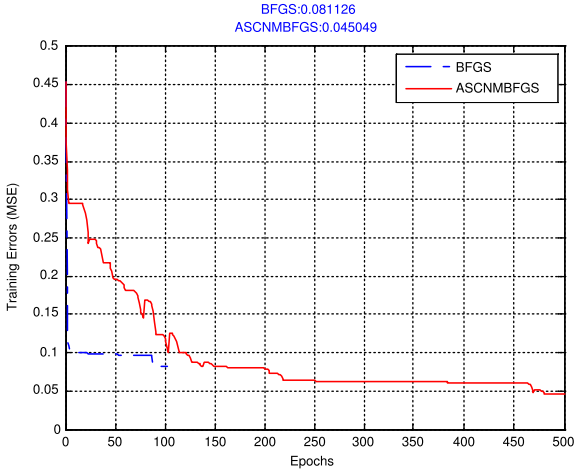

Fig. 21. Example of convergence behaviour for BFGS (dashed line) and ASCNM-BFGS (solid line) trained NARX network in the Reading Aloud problem.

We used FFTD and NARX networks with 5 and 10 hidden nodes (for FFTD networks, the numbers of adjustable parameters are 1421 and 2781, while 2031 and 4001 for NARX) and the maximum number of epochs was set to 300. Two delays were applied, i.e. 2 input delays for the FFTD networks, and 2-input-2-output delays for the NARX networks. The heuristic parameters are $3 \leqslant M^k \leqslant 15,\ \sigma = 0.5$ and $\delta = 0.9$. The results in Table 10 provide numerical evidence that the nonmonotone BFGS exhibits better ability to produce solutions with lower errors on the average. Both methods are able to train networks to reach small training errors. For comparison we should mention that in [57] 1900 epochs are needed to produce similar training error values. Figs. 20 and 21 provide some examples of learning behaviours, showing the ability of the ASCNM-BFGS to reach desirable solutions with smaller error values than then the BFGS method.

7. Conclusions

The paper investigated the problem of training recurrent neural networks in the framework of deterministic nonlinear optimisation. The proposed approach is based on the BFGS method, a well-known quasi-Newton method, and employs self-scaling of the approximations of the Hessian matrix. Furthermore, it is equipped with an adaptive nonmonotone strategy to better exploit information collected as it searches the space of the adjustable parameters. Comparing to the traditional monotone learning approach, our experiments using various data sequences provide evidence that the self-scaling BFGS with adaptive nonmonotone strategy enhances the convergence behaviour of RNNs and is more effective for training networks of various RNN architectures than the BFGS, even when hidden nodes numbers are smaller than the ones typically reported in the literature for these problems.

References

[1] A. Abraham, Meta learning evolutionary artificial neural networks, Neurocomputing 56 (2004) 1–38.
[2] M. Al-Baali, Numerical experience with a class of self-scaling quasi-Newton algorithms, Journal of Optimization Theory and Applications 96 (1998) 533–553.
[3] V.S. Asirvadam, S.F. McLoone, G.W. Irwin, Memory efficient BFGS neural-network learning algorithms using MLP-network: a survey, in: Proceedings of the IEEE International Conference on Control Application, 2004, pp. 586–591.
[4] M. Assaad, R. Bone, H. Cardot, Study of the behaviour of a new boosting algorithm for recurrent neural networks, in: Proceedings of the 15th International Conference on Artificial Neural Networks (ICANN), 2005, pp. 169–174.
[5] F. Bajramovic, C. Gruber, B. Sick, A comparison of first-and second-order training algorithms for dynamic neural networks, in: Proceedings of the IEEE International Joint Conference on Neural Networks, 2004, pp. 837–842.
[6] P. Baldi, Gradient descent learning algorithm overview: a general dynamical systems perspective, IEEE Transactions on Neural Networks 6 (1) (1993) 182–195.
[7] Y. Becerikli, A.F. Konar, T. Samad, Intelligent optimal control with dynamic neural networks, Neural Networks 16 (2003) 251–259.
[8] Y. Becerikli, Y. Oysal, A.F. Konar, Trajectory priming with dynamic fuzzy networks in nonlinear optimal control, IEEE Transactions on Neural Networks 15 (2) (2004) 383–394.
[9] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (2) (1994) 157–166.
[10] R. Bone, H. Cardot, Time delay learning by gradient descent in recurrent neural networks, in: Proceedings of the 15th International Conference on Artificial Neural Networks (ICANN), 2005, pp. 175–180.
[11] R.K. Brouwer, Training of a discrete recurrent neural network for sequence classification by using a helper FNN, Soft Computing 9 (2005) 749–756.
[12] C.G. Broyden, The convergence of a class of double-rank minimization algorithms, Journal of the Institute of Mathematics and Its Applications 6 (1970) 76–90.
[13] P. Campolucci, A. Uncini, F. Piazza, B.D. Rao, On-line learning algorithms for locally recurrent neural networks, IEEE Transactions on Neural Networks 10 (2) (1999) 253–271.
[14] C. Charalambous, A conjugate gradient algorithm for the efficient training of artificial neural networks, IEE Proceedings Part G 139 (1992) 301–310.
[15] W.C. Davidon, Variable metric method for minimization, A.E.C. Research Develop. Report ANL-5990, Argonne National Laboratory, Argonne, Illinois, 1959.
[16] W.C. Davidon, Variable metric method for minimization, SIAM Journal on Optimization 1 (1) (1991) 1–17.
[17] E.P. dos Santos, F.J. von Zuben, Efficient Second-Order Learning Algorithms for Discrete-Time Recurrent Neural Networks, Recurrent Neural Networks: Design and Applications, CRC Press, New York, 2000.
[18] J.L. Elman, Finding structure in time, Cognitive Science 14 (1990) 179–211.
[19] J.L. Elman, Connectionist models of cognitive development: where next?, Trends to Cognitive Science 9 (3) (2005) 111–117
[20] J.L. Elman, E.A. Bates, M.H. Johnson, A. Karmiloff-Smith, D. Parisi, K. Plunkett, The Shape of Change, in Rethinking Innateness: A Connectionist Perspective on Development, MIT Press, MA, 1997 (Chapter 6).
[21] R. Fletcher, M.J.D. Powell, A rapid convergent descent method for minimization, Computer Journal 6 (1963) 163–168.
[22] R. Fletcher, A Review of Methods for Unconstrained Optimization, Optimization, Academic Press, London, 1969.
[23] R. Fletcher, A new approach to variable metric algorithms, Computer Journal 13 (1970) 317–322.
[24] R. Fletcher, Practical Methods of Optimization, second ed., Wiley, 2000.
[25] D. Goldfarb, A family of variable metric updates derived by variational means, Mathematics of Computation 24 (1970) 23–26.
[26] P. Gordienko, Construction of efficient neural networks: algorithms and tests, in: Proceedings of IEEE International Joint Conference on Neural Networks, 1993, pp. 313–316.
[27] L. Grippo, F. Lampariello, S. Lucidi, A nonmonotone line search technique for Newton's method, SIAM Journal on Numerical Analysis 23 (1986) 707–716.
[28] L. Grippo, F. Lampariello, S. Lucidi, A quasi-discrete Newton algorithm with a nonmonotone stabilization technique, Journal of Optimization Theory and Applications 64 (1990) 495–510.
[29] S. Grossberg, Some networks that can learn, remember, and reproduce any number of complicated space-time patterns, International Journal of Mathematics and Mechanics 19 (1969) 53–91.
[30] C. Gruber, B. Sick, Fast and efficient second-order training of the dynamic neural network paradigm, in: Proceedings of IEEE International Joint Conference on Neural Networks, 2003, pp. 2482–2487.
[31] S. Haykin, Neural Networks: A Comprehensive Foundation, McMillan College Publishing Company, 1994.
[32] T. Ishikawa, Y. Tsukui, M. Matsunami, Optimization of electromagnetic devices using artificial neural network with quasi-Newton algorithm, IEEE Transactions on Magnetics 32 (3) (1996) 1226–1229.
[33] J. Han, G. Liu, General form of stepsize selection rule of line search and relevant analysis of global convergence of BFGS algorithm, Acta Mathematicae Applicatae Sinica 18 (1995) 112–122.
[34] L.C.K. Hui, K.-Y. Lam, C.W. Chea, Global optimisation in neural network training, Neural Computing and Applications 5 (1997) 58–64.
[35] M.I. Jordan, Attractor dynamics and parallelism in a connectionist sequential machine, in: Proceedings of the 8th Annual Conference on the Cognitive Science Society, 1986, pp. 531–546.
[36] S.C. Kremer, J.F. Kolen, Dynamical Recurrent Networks for Sequential Data Processing, Hybrid Neural Systems, Revised Papers From A Workshop, December 04–05, 1998, Lecture Notes in Computer Science,vol. 1778, 1998, pp. 107–122.
[37] S.C. Kremer, Spatiotemporal connectionist networks: a taxonomy and review, Neural Computation 13 (2001) 249–306.
[38] A. Likas, A. Stafylopatis, Training the random neural network using quasi-Newton methods, European Journal of Operational Research 126 (2000) 331–339.
[39] C.-T. Lin, C.S.G. Lee, Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems, Prentice-Hall, Singapore, 1996.
[40] G.D. Magoulas, S.Y. Chen, D. Dimakopoulos, A personalised interface for web directories based on cognitive styles, in: Proceedings of the 8th ERCIM Workshop on User Interfaces for All, Lecture Notes in Computer Science, vol. 3196, 2004, pp. 159–166.
[41] G.D. Magoulas, V.P. Plagianakos, M.N. Vrahatis, Globally convergent algorithms with local learning rates, IEEE Transactions oon Neural Networks 13 (3) (2002) 774–779.
[42] G.D. Magoulas, M.N. Vrahatis, Adaptive algorithms for neural network supervised learning: a deterministic optimization approach, International Journal Bifurcation and Chaos 16 (7) (2006) 1929–1950.
[43] G.D. Magoulas, M.N. Vrahatis, G.S. Androulakis, Effective back-propagation training with variable stepsize, Neural Networks 10 (1997) 69–82.
[44] P. McLeod, K. Plunkett, E.T. Rolls, Introduction to Connectionist Modelling of Cognitive Processes, Oxford University Press, Oxford, 1998.
[45] S. McLoone, G.W. Irwin, A variable memory quasi-Newton training algorithm, Neural Processing Letters 9 (1999) 77–89.
[46] S. McLoone, V.S. Asirvadam, G.W. Irwin, A memory optimal BFGS neural network training algorithm, in: Proceedings of the IEEE International Joint Conference on Neural Networks, 2002, pp. 513–518.
[47] N.M. Nawi, M.R. Ransing, R.S. Ransing, An improved learning algorithm based on the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method for back propagation neural networks, in: Proceedings of the 6th International Conference on Intelligent Systems Design and Applications, 2006, pp. 152–157.
[48] J. Nocedal, Y. Yuan, Analysis of a self-scaling quasi-Newton method, Mathematical Programming 61 (1993) 19–37.
[49] S.S. Oren, Self-scaling variable metric algorithms for unconstrained minimization, Ph.D Thesis, Standford University, California, USA, 1972.

[50] S.S. Oren, D.G. Luenberger, Self-scaling variable metric (SSVM) algorithms. Part I: Criteria and sufficient conditions for scaling a class of algorithms, Management Science 20 (1974) 845–862.
[51] C.-C. Peng, G.D. Magoulas, Adaptive nonmonotone conjugate gradient training algorithm for recurrent neural networks, in: IEEE Proceedings of the International Conference on Tools. Artificial Intelligence (ICTAI), 2007, pp. 374–381.
[52] C.-C. Peng, G.D. Magoulas, Sequence processing with recurrent neural networks, Encyclopedia of artificial intelligence, Information science reference, 2008, pp. 1411–1417.
[53] C.-C. Peng, G.D. Magoulas, Adaptive self-scaling nonmonotone BFGS training algorithm for recurrent neural networks, in: Proceedings of the 17th International Conference on Artificial Neural Networks (ICANN), 2007, pp. 259–268.
[54] J.A. Perez-Ortiz, J. Calera-Rubio, M.L. Forcada, Online symbolic-sequence prediction with discrete-time recurrent neural networks, in: International Conference ANNs, Lecture Notes in Computer Science, vol. 2130, 2001, pp. 719–724.
[55] P.K.H. Phua, D. Ming, Parallel nonlinear optimization techniques for training neural networks, IEEE Transactions on Neural Networks 14 (6) (2003) 1460–1468.
[56] V.P. Plagianakos, G.D. Magoulas, M.N. Vrahatis, Deterministic nonmonotone strategies for effective training of multi-layer perceptrons, IEEE Transactions on Neural Networks 13 (6) (2002) 1268–1284.
[57] D. Plaut, J. McClelland, M. Seidenberg, K. Patterson, Understanding normal and impaired reading: computational principles in quasi-regular domains, Psychological Review 103 (1996) 56–115.
[58] M.J.D. Powell, How bad are the BFGS and DPF methods when the objective function is quadratic?, Mathematical Programming 34 (1986) 34–47
[59] L.M. Saini, M.K. Soni, Artificial neural network based peak load forecasting using Levenberg–Marquardt and quasi-Newton methods, in: Proceedings of Generation, Transmission and Distribution, 2002, pp. 578–584.
[60] R.J. Schalkoff, Artificial Neural Networks, McGraw-Hill, New York, 1997.
[61] R. Setiono, L.C.K. Hui, Some *n*-bit parity problems are solvable by feed-forward networks with less than *n* hidden units, in: Proceedings of the IEEE International Joint Conference on Neural Networks, 1993, pp. 305–308.
[62] R. Setiono, L.C.K. Hui, Use of a quasi-Newton in a feedforward neural network construction algorithm, IEEE Transactions on Neural Networks 6 (1) (1995) 273–277.
[63] D.F. Shanno, Conditioning of quasi-Newton methods for function minimization, Mathematics of Computation 24 (1970) 647–655.
[64] W. Sun, J. Han, J. Sun, Global convergence of nonmonotone descent methods for unconstrained optimization problems, Journal of Computational and Applied Mathematics 146 (2002) 89–98.
[65] A.C. Tsoi, Gradient Based Learning Algorithms, Adaptive Processing of Sequences and Data Structures, Springer-Verlag, Berlin, 1998.
[66] H.X. Yin, D.L. Du, The global convergence of self-scaling BFGS algorithm with nonmonotone line search for unconstrained nonconvex optimization problems, Acta Mathematica Sinica (2006).
[67] H. Zhang, W.W. Hager, A nonmonotone line search technique and its application to unconstrained optimization, SIAM J. Optim. 14 (2004) 1043–1056.