A Training Algorithm and Stability Analysis for Recurrent Neural Networks

Zhao Xu, Qing Song, Danwei Wang and Haijin Fan
School of Electrical and Electronic Engineering
Nanyang Technological University, Singapore 639798

***Abstract*—Training of recurrent neural networks (RNNs) introduces considerable computational complexities due to the need for gradient evaluations. How to get fast convergence speed and low computational complexity remains a challenging and open topic. Besides, the transient response of learning process of RNNs is a critical issue, especially for on-line applications. Conventional RNNs training algorithms such as the backpropagation through time (BPTT) and real-time recurrent learning (RTRL) have not adequately satisfied these requirements because they often suffer from slow convergence speed. If a large learning rate is chosen to improve performance, the training process may become unstable in terms of weight divergence. In this paper, a novel training algorithm of RNN, named robust recurrent simultaneous perturbation stochastic approximation (RRSPSA), is developed with a specially designed recurrent hybrid adaptive parameter and adaptive learning rates. RRSPSA is a powerful novel twin-engine simultaneous perturbation stochastic approximation (SPSA) type of RNN training algorithm. It utilizes specific designed three adaptive parameters to maximize training speed for recurrent training signal while exhibiting certain weight convergence properties with only two objective function measurements as the original SPSA algorithm. The RRSPSA is proved with guaranteed weight convergence and system stability in the sense of Lyapunov function. Computer simulations were carried out to demonstrate applicability of the theoretical results.**

***Index Terms*—recurrent neural networks (RNNs), simultaneous perturbation stochastic approximation (SPSA) training, weight convergence and stability proofs.**

I. Introduction

To maximize potentials and capabilities of the RNNs, one of the preliminary task is to design an efficient training algorithm with fast weight convergence speed and low computational complexity. Gradient descent optimization techniques are often used for neural networks which consist of adjustable weights along the negative direction of error gradient. However, due to dynamic structure of RNNs, the obtained error gradient tends to be very small in a classical BP training algorithm with the fixed learning step, because RNN weight depends not only on the current output but also output of past steps. It is widely acknowledged that recurrent derivative of the multilayered RNNs, i.e. the recurrent training signal (see Fig. 1), which contains information of dynamic time dependence in training, is necessary to obtain good time response of RNNs [1], [2]. From computational efficiency and weigh convergence points of view, Spall proposed a simultaneous perturbation stochastic approximation (SPSA) method to find roots and minima of neural network functions which are contaminated with noise [3]. The name of "simultaneous perturbation" arises from the fact that all elements of unknown weight parameters are being varied simultaneously. SPSA has the potential to be significantly more efficient than the usual p-dimensional algorithms (of Kiefer-Wolfowitz/Blum type) that are based on standard finite-difference gradient approximations and it is shown in [3] that approximately the same level of estimation accuracy can typically be achieved with only l/pth the amount of data needed in the standard approach. The main advantage of SPSA is its simplicity and excellent weight convergence property, in which only two objective function measurements are used. Thus, SPSA is more economical in terms of loss function evaluation which are usually the most expensive part of an optimization problem. SPSA has attracted great attention in application areas such as neural networks training, adaptive control and model parameter estimation [4]–[9].

Conventional training methods of RNNs are usually classified as backpropagation through time (BPTT) [10], [11] and real time recurrent learning (RTRL) [12], [13] algorithms. However, BPTT and RTRL often suffer from drawbacks of slow convergence speed and heavy computational load. In fact, BPTT cannot be used for online application due to computational problems. For a standard RTRL training, recursive weight updating requires a small learning step to obtain weight convergence and system stability of the RNNs. If a large learning rate is selected to speed up weight updating, the training process of RTRL may lead to a large steady-state error or even become unstable [1], [5].

In this paper, we propose a robust recurrent simultaneous perturbation stochastic approximation (RRSPSA) algorithm in the framework of deterministic system with guaranteed weight convergence. RNNs are dynamic systems and contain recurrent connections in their structure. Not only the weights at the current time steps but also those at the previous time steps are to be perturbed, which makes the learning easy to be diverge and the convergence analysis complicated to obtain. The key characteristic of RRSPSA training algorithm is to use a recurrent hybrid adaptive parameter together with adaptive learning rate and dead zone concept. The recurrent hybrid adaptive parameter can automatically switch off recurrent training signal whenever the weight convergence and stability conditions are violated (see Fig. 1 and explanation in sections III and IV). There are several advantages to train RNNs based on RRSPSA algorithm. First, RRSPSA is relatively easy to implement as compared to other RNN training methods such

as BPTT due to its RTRL type of training and simplicity. Second, RRSPSA provides excellent convergence property based on simultaneous perturbation of weight, which is similar to the standard SPSA. Finally, RRSPSA has capability to improve RNN training speed with guaranteed weight convergence over the standard RTRL algorithm by using adaptive learning method.

II. Basics of the Recurrent Neural Network and SPSA

Consider a three-layer m-input and m-output RNN with the output feedback structure as shown in Fig. 1. Output vector of the RNN is $\hat{y}(k)$ which can be presented as

$$\hat{y}(k) = H(\bar{W}(k), x(k))\,\bar{V}(k) = [\hat{y}_1(k), \cdots, \hat{y}_m(k)]^T \in R^m \quad (1)$$

where $\bar{V}(k) \in R^{p^v}$ ($p^v = m \times n_h$ and n_h is hidden neuron number) is the long vector of the output layer weight matrix $\hat{V}(k) \in R^{n_h \times m}$ defined by

$$\bar{V}(k) = \left[\hat{V}_{1,1}(k), \cdots, \hat{V}_{n_h,1}(k), \cdots, \hat{V}_{1,m}(k), \cdots, \hat{V}_{n_h,m}(k)\right]^T \quad (2)$$

and $x(k) \in R^{n_I}$ ($n_I = (l+1) \times m$) is the input vector of the RNN defined by

$$\begin{aligned} x(k) &= [x_1(k), x_2(k), \cdots, x_{n_I}(k)]^T \\ &= \left[\hat{y}^T(k-1), \cdots, \hat{y}^T(k-l), u_1(k), \cdots, u_m(k)\right]^T \end{aligned} \quad (3)$$

and $H(\bar{W}(k), x(k)) \in R^{m \times p^v}$ is the nonlinear activation block-diagonal matrix given by

$$\begin{aligned} H(\bar{W}(k), x(k)) &= H(k) \\ &= \begin{bmatrix} h_1(k), \cdots, h_{n_h}(k), & 0, \cdots & 0, \cdots \\ 0, \cdots & \ddots & \vdots \\ \vdots & & \\ 0, \cdots & 0, \cdots & h_1(k), \cdots, h_{n_h}(k) \end{bmatrix} \end{aligned} \quad (4)$$

where $h_j(k)$ with $1 \le j \le n_h$ is one of the most popular nonlinear activation functions,

$$h_j(k) = h\left(\hat{W}_{j,:}(k)x(k)\right) = \frac{1}{1 + exp\left(-4\lambda \hat{W}_{j,:}(k)x(k)\right)} \quad (5)$$

with $\hat{W}_{j,:}(k) \in R^{n_I}$ is the jth row vector of the hidden layer weight matrix $\hat{W}(k) \in R^{n_h \times n_I}$, and $\lambda > 0$ is the gain parameter of the threshold function that is defined specifically for ease of use later when deriving the sector condition of the hidden layer. Define the long vector of $\hat{W}(k) \in R^{p^w}$ ($p^w = n_I \times n_h$) as

$$\bar{W}(k) = \left[\hat{W}_{1,:}(k), \hat{W}_{2,:}(k), \cdots, \hat{W}_{n_h,:}(k)\right]. \quad (6)$$

The output vector $\hat{y}(k)$ is to approximate the ideal nonlinear function $y(k) = H(\bar{W}^*, x(k))\bar{V}^*$ with $\bar{V}^*$ and $\bar{W}^*$ being the ideal output layer and hidden layer weight, respectively.

SPSA algorithm is to solve the problem of finding an optimal weight θ^* of the gradient equation of neural networks

$$\hat{g}(\hat{\theta}(k)) = \frac{\partial L(\hat{\theta}(k))}{\partial \hat{\theta}(k)} = 0$$

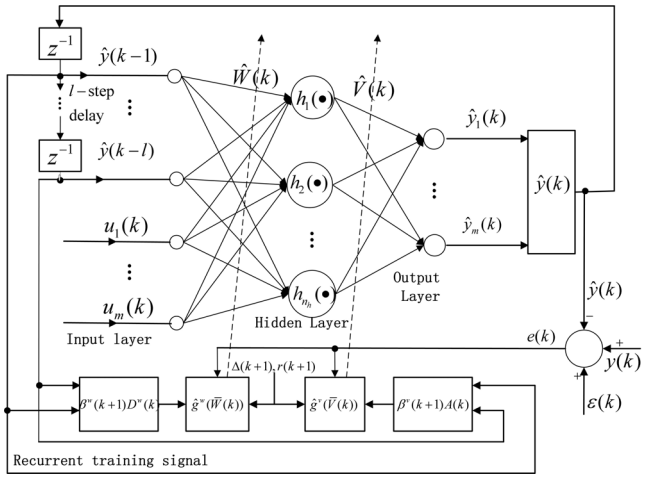

Fig. 1. Block diagram of RRSPSA training for the output feedback based RNN.

for some differentiable loss function $L: R^p \to R^1$, where $\hat{\theta}(k)$ is the estimated weight vector of neural networks. The basic idea is to vary all elements of the vector $\hat{\theta}(k)$ simultaneously and approximate the gradient function using only two measurements of the loss function without requiring exact derivatives or a large number of function evaluation which provides significant improvement in efficiency over the standard gradient decent algorithms. Theory and numerical experience in [3] indicate that SPSA can be significantly more efficient than the standard finite difference-based algorithms in large-dimensional problems.

Inspired by the excellent economical and convergence property of SPSA, we propose the RRSPSA algorithm in the framework of deterministic system based on the idea of simultaneous perturbation with guaranteed weight convergence. The loss function for RRSPSA is defined as

$$L(\hat{\theta}(k)) = \frac{1}{2}\|e(k)\|^2 \quad (7)$$

$$e(k) = y(k) - \hat{y}(k) + \varepsilon(k) \quad (8)$$

where $y(k)$ is the ideal output and a function of θ^*, $\hat{y}(k)$ is a function of $\hat{\theta}(k)$, and $\varepsilon(k)$ is a disturbance term. Similar to SPSA, the RRSPSA algorithm seeks to find an optimal weight vector θ^* of gradient equation, i.e., the weight vector that minimized differentiable $L(\hat{\theta}(k))$. That is, the proposed RRSPSA algorithm for updating $\hat{\theta}(k) \in R^{(p^w + p^v)}$ as an estimation of the ideal weight vector θ^* is of the form

$$\hat{\theta}(k+1) = \hat{\theta}(k) - \alpha(k+1)\hat{g}(\hat{\theta}(k)) \quad (9)$$

where $\alpha(k+1)$ is an adaptive learning rate and $\hat{g}(\hat{\theta}(k))$ is an approximation of the gradient of the loss function. This approximated gradient function (normalized) is of the SPSA

form

$$\hat{g}(\hat{\theta}(k)) = \left(\frac{L\left(\hat{\theta}(k) + c\Delta(k+1)\right)}{2c\rho(k+1)} - \frac{L\left(\hat{\theta}(k) - c\Delta(k+1)\right) - \delta(k)}{2c\rho(k+1)} \right) r(k+1) \quad (10)$$

where $\delta(k)$ is an equivalent approximation error of the loss function, which is also called the measurement noise, and ρ^v, $\rho^w(k+1)$ are the normalization factors for the output and hidden layers, respectively. $\Delta(k+1)$, $r(k+1)$, and $c > 0$ are the controlling parameters of the algorithm defined as

1) The perturbation vector

$$\Delta(k) = [\Delta_1(k), \cdots, \Delta_{p^w+p^v}(k)]^T \in R^{(p^w+p^v)} \quad (11)$$

is a bounded random directional vector that is used to perturb the weight vector simultaneously and can be generated randomly.

2) The sequence of $r(k)$ is defined as

$$r(k) = \left[\frac{1}{\Delta_1(k)}, \cdots, \frac{1}{\Delta_{p^w+p^v}(k)} \right]^T \in R^{(p^w+p^v)}. \quad (12)$$

3) $c(k) > 0$, a constant $k \geq 1$ is a sequence of positive numbers [3], [7], [8]. $c(k)$ can be chosen as a small constant number for a slow time-varying system as pointed in [8].

III. Output Layer Training and Stability Analysis

The output layer weight $\bar{V}(k)$ and hidden layer weight $\bar{W}(k)$ are to be updated separately using different learning rates as shown in Fig. 1 as in the standard BP training algorithm [1]. During the updating of the output layer weights, the hidden layer weights are fixed. We are now ready to propose the RRSPSA algorithm to update the estimated weight vector $\bar{V}(k)$ of output layer. That is,

$$\bar{V}(k+1) = \bar{V}(k) - \alpha^v \hat{g}^v(\bar{V}(k)) \quad (13)$$

where $\hat{g}^v(\bar{V}(k)) \in R^{p^v}$ is the normalized gradient approximation that uses simultaneous perturbation vectors $\Delta^v \in R^{p^v}$ and $r^v \in R^{p^v}$ to stimulate weight. Δ^v and r^v are, respectively, the first p^v components of perturbation vectors defined in (11) and (12).

$$\hat{g}^v(\bar{V}(k)) = -e^T(k)H(k)\Delta^v r^v (\rho^v)^{-1} - \beta^v e^T(k)A(k)r^v (2c\rho^v)^{-1} \quad (14)$$

where $A(k) \in R^m$ is the extended recurrent gradient defined as

$$A(k) = H\left(\bar{W}(k), \hat{D}^v(k)(\bar{V}(k) + c\Delta^v)\right)\bar{V}(k) - H\left(\bar{W}(k), \hat{D}^v(k)(\bar{V}(k) - c\Delta^v)\right)\bar{V}(k) \quad (15)$$

$$\hat{D}^v(k) = \left[H^T(k-1), \cdots, H^T(k-l), 0, \cdots, 0 \right]^T \in R^{n_I \times p^v}. \quad (16)$$

Remark 1: Our purpose is to find the desired output layer weight vector $\bar{V}^*$ of the gradient equation

$$\hat{g}(\bar{V}(k)) = \frac{\partial L(\bar{V}(k))}{\partial \bar{V}(k)} = 0.$$

We obtain $\hat{g}(\bar{V}(k))$ by varying all elements of unknown weight parameters simultaneously as shown in the proof of Lemma 1 in the framework of deterministic system. We will show that RRSPSA has the same form as SPSA and only two objective function measurements are used at each iteration which maintains the efficiency of SPSA.

Lemma 1: The normalized gradient approximation in (14) is an equivalent presentation of the standard SPSA algorithm in equation (10).

Proof: We will proof that (14) is obtained by perturbing all the weights simultaneously as SPSA did. Rewrite the loss function defined in (7) as

$$L\left(\hat{\theta}(k)\right) = L(\bar{V}(k), \bar{W}(k)) = \frac{1}{2}\|y(k) - \hat{y}(k) + \varepsilon(k)\|^2. \quad (17)$$

Further, we use $\hat{y}(k) = H(k)\bar{V}(k)$ in (1) to define

$$\begin{aligned}
&\hat{y}^{v+}(\bar{V}(k)) \\
=&H(k)(\bar{V}(k) + c\Delta^v) \\
&+ H(\bar{W}(k), (x(k) + \Delta x(k+1)))\bar{V}(k) \\
=&H(k)(\bar{V}(k) + c\Delta^v) \\
&+ H\Big(\bar{W}(k), \Big(\left[\hat{y}^T(k-1), \cdots, \hat{y}^T(k-l), 0, \cdots, 0\right]^T \\
&+ \Delta x(k+1)\Big)\Big)\bar{V}(k) \\
=&H(k)(\bar{V}(k) + c\Delta^v) \\
&+ H\Big(\bar{W}(k), \Big(\Big[\bar{V}^T(k-1)H^T(k-1), \cdots, \bar{V}^T(k-l)H^T(k-l), \\
&0, \cdots, 0\Big]^T + \Delta x(k+1)\Big)\Big)\bar{V}(k) \\
=&H(k)(\bar{V}(k) + c\Delta^v) \\
&+ H\Big(\bar{W}(k), \Big[(\bar{V}(k-1) + c\Delta^v)^T H^T(k-1), \\
&\cdots, (\bar{V}(k-l) + c\Delta^v)^T H^T(k-l), \\
&0, \cdots, 0\Big]^T\Big)\bar{V}(k)
\end{aligned} \quad (18)$$

The input disturbance vector $\Delta x(k+1)$ is necessary for RNN training since the recurrent connections. The disturbance $\Delta x(k+1)$ is actually to perturb the weights at previous time steps which contain in $x(k)$ as seen in (3). However, the last equation of (18) is difficult to implement. Here we assume the model parameters do not change apparently between each iteration,i.e. $\bar{V}(k) \approx \bar{V}(k-1) \cdots \approx \bar{V}(k-l)$. We can derive a similar approach as RTRL introduced by Williams and Ziper [12], [13]. Moreover, such approximation makes the proposed

algorithm real time and adaptive in a recursive fashion. Under the approximation,

$$\begin{aligned}
&\hat{y}^{v+}\left(\bar{V}(k)\right)\\
\approx&H(k)\left(\bar{V}(k)+c\Delta^{v}\right)\\
&+H\left(\bar{W}(k),\left[\left(\bar{V}(k)+c\Delta^{v}\right)^{T}H^{T}(k-1),\right.\right.\\
&\left.\left.\cdots,\left(\bar{V}(k)+c\Delta^{v}\right)^{T}H^{T}(k-l),0,\cdots,0\right]^{T}\right)\bar{V}(k)\\
=&H(k)\left(\bar{V}(k)+c\Delta^{v}\right)\\
&+H\left(\bar{W}(k),\left[H^{T}(k-1),\cdots,H^{T}(k-l),0,\cdots,0\right]^{T}\right.\\
&\left.\times\left(\bar{V}(k)+c\Delta^{v}\right)\right)\bar{V}(k)\\
=&H(k)\left(\bar{V}(k)+c\Delta^{v}\right)\\
&+H\left(\bar{W}(k),\hat{D}^{v}(k)\left(\bar{V}(k)+c\Delta^{v}\right)\right)\bar{V}(k)
\end{aligned}\tag{19}$$

where $\hat{D}^{v}(k)$ is defined in (16). By the approximation $\bar{V}(k)\approx\bar{V}(k-1)\cdots\approx\bar{V}(k-l)$, the recurrent term $H^{T}(k-1),\cdots,H^{T}(k-l)$ of (16) can be iteratively obtained from the previous steps to speed up the calculation of RRSPSA (normally, the input $u(k)$ is independent from weight so that the last entries of $\hat{D}^{v}(k)$ are zero vectors). Weight convergence and system stability can be guaranteed by using the three adaptive learning rates which will be explained detaily later.

Similar to (18), we can get

$$\begin{aligned}
&\hat{y}^{v-}\left(\bar{V}(k)\right)\\
=&H(k)\left(\bar{V}(k)-c\Delta^{v}\right)\\
&+H\left(\bar{W}(k),\left(x(k)-\Delta x(k+1)\right)\right)\bar{V}(k)\\
\approx&H(k)\left(\bar{V}(k)-c\Delta^{v}\right)\\
&+H\left(\bar{W}(k),\hat{D}^{v}(k)\left(\bar{V}(k)-c\Delta^{v}\right)\right)\bar{V}(k)
\end{aligned}\tag{20}$$

so that

$$\begin{aligned}
&\hat{y}^{v+}\left(\bar{V}(k)\right)-\hat{y}^{v-}\left(\bar{V}(k)\right)\\
\approx&2H(k)c\Delta^{v}+A(k)
\end{aligned}\tag{21}$$

where the extended recurrent gradient $A(k)$ is defined as (15). $A(k)$ in (21) is the result of recurrent connections. To guarantee weight convergence and system stability during training, we insert an adaptive recurrent hybrid parameter $\beta^{v}(k+1)$ (see Fig. 1) to control the recurrent training signal. Thus, (21) can be rewritten as follows

$$\begin{aligned}
&\hat{y}^{v+}\left(\bar{V}(k)\right)-\hat{y}^{v-}\left(\bar{V}(k)\right)\\
\approx&2H(k)c\Delta^{v}+\beta^{v}A(k).
\end{aligned}\tag{22}$$

According to (10), we can ge

$$\begin{aligned}
&\hat{g}^{v}\left(\bar{V}(k)\right)\\
=&\frac{L(\bar{W}(k),\bar{V}(k)+c\Delta^{v})}{2c\rho^{v}}r^{v}\\
&-\frac{L(\bar{W}(k),\bar{V}(k)-c\Delta^{v})-\delta^{v}(k)}{2c\rho^{v}}r^{v}\\
=&\frac{\left\|y(k)-\hat{y}^{v+}\left(\bar{V}(k)\right)\right\|^{2}-\left\|y(k)-\hat{y}^{v-}\left(\bar{V}(k)\right)\right\|^{2}+2\delta^{v}(k)}{4c\rho^{v}}\\
&\times r^{v}\\
=&\left\{\left(y(k)-\hat{y}^{v+}\left(\bar{V}(k)\right)+y(k)-\hat{y}^{v-}\left(\bar{V}(k)\right)\right)^{T}\right.\\
&\left.\times\left(\hat{y}^{v-}\left(\bar{V}(k)\right)-\hat{y}^{v+}\left(\bar{V}(k)\right)\right)+2\delta^{v}(k)\right\}r^{v}\\
&\times\left(4c\rho^{v}\right)^{-1}\\
=&\frac{\left(y(k)-\hat{y}(k)+\varepsilon(k)\right)^{T}\left(\hat{y}^{v-}\left(\bar{V}(k)\right)-\hat{y}^{v+}\left(\bar{V}(k)\right)\right)}{2c\rho^{v}}r^{v}\\
=&\frac{e^{T}(k)\left(\hat{y}^{v-}\left(\bar{V}(k)\right)-\hat{y}^{v+}\left(\bar{V}(k)\right)\right)}{2c\rho^{v}}r^{v}\\
\approx&-\frac{e^{T}(k)H(k)\Delta^{v}}{\rho^{v}}r^{v}\\
&-\frac{\beta^{v}e^{T}(k)A(k)}{2c\rho^{v}}r^{v}
\end{aligned}\tag{23}$$

$e(k)$ in the fifth equality of (23) comes from (8). The fourth equality of (23) reveals the relationship between gradient approximation error $\delta^{v}(k)$ and system disturbance $\varepsilon(k)$.

$$\begin{aligned}
\delta^{v}(k)=&\left(-2\hat{y}(k)+\hat{y}^{v+}\left(\bar{V}(k)\right)+\hat{y}^{v-}\left(\bar{V}(k)\right)+\varepsilon(k)\right)^{T}\\
&\times\left(\hat{y}^{v-}\left(\bar{V}(k)\right)-\hat{y}^{v+}\left(\bar{V}(k)\right)\right)\\
=&\varepsilon^{T}(k)\left(\hat{y}^{v-}\left(\bar{V}(k)\right)-\hat{y}^{v+}\left(\bar{V}(k)\right)\right).
\end{aligned}\tag{24}$$

On the fifth equality of (23), the gradient approximation presentation appears exactly the same as the original one in [3] (see [3, eq. (2.2)]). Only two objective function measurements are used at each step, which maintains the efficiency of SPSA. ■

For stability analysis, the training error $e(k)$ will be linked to the parameter estimating error of the output layer and the disturbance. Define the weight estimating errors of the output layer and hidden layer as

$$\tilde{\bar{V}}(k)=\bar{V}^{*}-\bar{V}(k)\tag{25}$$

$$\tilde{\bar{W}}(k)=\bar{W}^{*}-\bar{W}(k)\tag{26}$$

where $\bar{W}^{*}$ and $\bar{V}^{*}$ are the ideal (optimal) weight vectors of $\bar{W}(k)$ and $\bar{V}(k)$. According to (8), the training error can be rewritten as

$$\begin{aligned}
&e(k)=y(k)-\hat{y}(k)+\varepsilon(k)\\
&=H\left(\bar{W}^{*},x(k)\right)\bar{V}^{*}-H(k)\bar{V}(k)+\varepsilon(k)\\
&=H\left(\bar{W}^{*},x(k)\right)\bar{V}^{*}-H(k)\bar{V}^{*}+H(k)\bar{V}^{*}-H(k)\bar{V}(k)+\varepsilon(k)
\end{aligned}\tag{27}$$

Because the term $H(\bar{W}^*,x(k))\bar{V}^* - H(k)\bar{V}^*$ is temporarily bounded during output layer training (hidden layer weights are fixed), we define

$$\tilde{e}^v(k) = H(\bar{W}^*,x(k))\bar{V}^* - H(k)\bar{V}^* + \varepsilon(k) \tag{28}$$

The training error caused by the weight estimation error is defined as

$$\phi^v(k) = -H(k)\tilde{\bar{V}}(k). \tag{29}$$

Then, (27) can be rewritten as

$$e(k) = -\phi^v(k) + \tilde{e}^v(k) \tag{30}$$

Hence, the training error is directly linked to the parameter estimating error of the output layer $\phi^v(k)$ and the disturbance $\tilde{e}^v(k)$.

Theorem 1: If the output layer of the RNN is trained by RRSPSA algorithm (13), $\bar{V}(k)$ is guaranteed to be convergent in the sense of Lyapunov function $\left\|\tilde{\bar{V}}(k+1)\right\|^2 - \left\|\tilde{\bar{V}}(k)\right\|^2 \leqslant 0$, $\forall k$ with $\tilde{\bar{V}}(k) = \bar{V}^* - \bar{V}(k)$ and the training error will be L_2-stable in the sense of $\tilde{e}^v(k) \in L_2$ and $\phi^v(k) \in L_2$.

Proof: The proof is omitted due to space limitation. ■

The conditions for the three adaptive learning rates in (13) and (14) are as follows.

1) ρ^v is the normalization factor defined by

$$\rho^v = \tau\rho^v(k) + max\left\{\frac{\alpha^v Z(k)}{p^v + 0.5\beta^v Y(k)} + 1, \bar{\rho}\right\} \tag{31}$$

where $0 < \tau < 1$, $0 < \alpha^v \leq 1$, and $\bar{\rho}$ are positive constants and

$$Y(k) = \frac{(r^v)^T\left(\eta I + H^T(k)H(k)\right)^{-1}H^T(k)A(k)}{c}$$

where η is a small perturbation parameter.

$$Z(k) = \|(2H(k)c\Delta^v + \beta^v A(k))\|^2 \times \|r^v\|^2\left(4(c)^2\right)^{-1}.$$

2) α^v is an adaptive learning rate similar to dead zone approach in the classical neural network and adaptive control

$$\begin{cases} \alpha^v = \alpha^v, \text{ if } \|e(k)\| \geqslant \Lambda(k) \\ \alpha^v = 0, \text{ if } \|e(k)\| < \Lambda(k) \end{cases} \tag{32}$$

with $\Lambda(k) = \tilde{e}^v_m(k)/\sqrt{1 - \frac{\alpha^v\rho^v - 1Z(k)}{p^v + 0.5\beta^v Y(k)}}$ and $\tilde{e}^v_m(k)$ being the maximum value of $\tilde{e}^v(k)$ defined in (28).

3) β^v is the recurrent hybrid adaptive parameter determined by

$$\begin{cases} \beta^v = 1, \text{if } Y(k) \geq 0 \\ \beta^v = 0, \text{ if } Y(k) < 0. \end{cases} \tag{33}$$

Remark 2: According to the theoretical analysis, three parameters β^v, α^v and ρ^v play a key role for the proof of weight convergence and stability analysis. They are designed to guarantee the negativeness of Lyapunov function. The function of β^v is to automatically switch off the important recurrent training signal as at the lower left corner of Fig.1 when convergence and stability requirements caused by recurrent training signal are violated. The normalization factor ρ^v is also used to bound the signals in learning algorithms as traditionally did in adaptive control system [5], [14]. α^v is to guarantee weight convergence during training, which means that weights are only updated when convergence and stability requirements are met. Besides, α^v needs not to be small as in RTRL training algorithm which will make the training by RRSPSA faster than that by RTRL.

IV. Hidden Layer Training and Stability Analysis

The algorithm for updating estimated weight vector $\bar{W}(k)$ of the hidden layer is

$$\bar{W}(k+1) = \bar{W}(k) - \alpha^w \hat{g}^w(\bar{W}(k)) \tag{34}$$

where $\hat{g}^w(\bar{W}(k)) \in R^{p^w}$ is the normalized gradient approximation that uses simultaneous perturbation vectors $\Delta^w(k+1) \in R^{p^w}$ and $r^w \in R^{p^w}$ to stimulate weight of the hidden layer. $\Delta^w(k+1)$ and r^w are, respectively, the last p^w components of perturbation vectors defined in (11) and (12).

$$\hat{g}^w(\bar{W}(k)) = -\frac{e^T(k)\left(B^+(k) - B^-(k) + \beta^w\hat{D}^w(k)\right)}{2c\rho^w(k+1)}r^w \tag{35}$$

where

$$B^+(k) = H(\bar{W}(k) + c\Delta^w(k+1), x(k))\bar{V}(k) \in R^m \tag{36}$$

$$B^-(k) = H(\bar{W}(k) - c\Delta^w(k+1), x(k))\bar{V}(k) \in R^m \tag{37}$$

$$\begin{aligned}&\hat{D}^w(k)\\ &= \left(H\left(\bar{W}(k), \left[\left(B^+(k-1)\right)^T, \cdots, \left(B^+(k-l)\right)^T, 0, \cdots, 0\right]^T\right)\right.\\ &\left. - H\left(\bar{W}(k), \left[\left(B^-(k-1)\right)^T, \cdots, \left(B^-(k-l)\right)^T, 0, \cdots, 0\right]^T\right)\right)\\ &\times \bar{V}(k) \in R^m.\end{aligned} \tag{38}$$

Remark 3: The same as the output layer, our purpose is to find the desired output layer weight vector $\bar{W}^*$ of the gradient equation

$$\hat{g}(\bar{W}(k)) = \frac{\partial L(\bar{W}(k))}{\partial \bar{W}(k)} = 0.$$

The detailed mathematical deduction is given in the proof of Lemma 2. $B^+(k) - B^-(k)$ is the result of the perturbations of the feedforward signals and $\hat{D}^w(k)$ is the result of the perturbations of the recurrent signals in RNNs. The function of β^w is to automatically switch off recurrent contribution when the convergence and stability requirements caused by the recurrent signals are violated. The conditions for the key parameters appeared in (34) and (35) are shown in (42), (45), and (46).

Lemma 2: The normalized gradient approximation in (35) is an equivalent presentation of the standard SPSA algorithm in equation (10).

Proof: The basic idea of the proof is similar to Lemma 1. The proof here is omitted due to space limitation. ■

We can rewrite the estimation error as

$$\begin{aligned} e(k) &= y(k) - \hat{y}(k) + \varepsilon(k) \\ &= H(\bar{W}^*, x(k))\bar{V}^* - H(\bar{W}^*, x(k))\bar{V}(k) \\ &\quad + H(\bar{W}^*, x(k))\bar{V} - H(k)\bar{V}(k) + \varepsilon(k) \\ &= -\phi^w(k) + \tilde{e}^w(k) \end{aligned} \tag{39}$$

where

$$\phi^w(k) = -H(\bar{W}^*, x(k))\bar{V} + H(k)\bar{V}(k) \tag{40}$$

and the bounded equivalent disturbance $\tilde{e}^w(k)$ of the hidden layer is defined as

$$\tilde{e}^w(k) = H(\bar{W}^*, x(k))\tilde{\bar{V}}(k) + \varepsilon(k) \tag{41}$$

Note that $\tilde{e}^w(k)$ is a bounded signal because $\tilde{\bar{V}}(k)$ has been proven to be bounded in Section III.

Theorem 2: If the hidden layer of the RNN is trained by the adaptive RRSPSA algorithm (34), weight $\bar{W}(k)$ is guaranteed to be convergent in the sense of Lyapunov function

$$\left\|\tilde{\bar{W}}(k+1)\right\|^2 - \left\|\tilde{\bar{W}}(k)\right\|^2 \leqslant 0,\ \forall k$$

with $\tilde{\bar{W}}(k) = \bar{W}^* - \bar{W}(k)$ and the training error will be L_2-stable in the sense of $\tilde{e}^w(k) \in L_2$ and $\phi^w(k) \in L_2$.

Proof: The proof is omitted due to space limitation. ■

The three adaptive learning rates appeared in (34) and (35) are as follows.

1) The normalization factor is defined as

$$\rho^w(k+1) = \tau\rho^w(k) + max\left\{1 + \alpha^w M(k)\|r^w\|^2 \times \Upsilon^{-1}(k), \bar{\rho}\right\} \tag{42}$$

where

$$M(k) = \left\|\frac{B^+(k) - B^-(k) + \beta^w \hat{D}^w(k)}{2c}\right\|^2$$

$$\Upsilon(k) = \frac{\lambda_{min}}{\lambda}p^w + \frac{1}{2\lambda c}\beta^w \times (r^w)^T\left(\eta I + \bar{\Omega}^T(k)\bar{\Omega}(k)\right)^{-1}\bar{\Omega}^T(k)\hat{D}^w(k)$$

and $0 < \alpha^w \leq 1$ being a positive constant, λ being the maximum value of the derivative $\dot{h}_j(k)$ of the activation function in (5), and $\lambda_{min} \neq 0$ being the minimum nonzero value of the mean value $\tilde{\mu}_j(k)$ defined as

$$0 \leq \tilde{\mu}_j(k) \leq \lambda\ (1 \leq i \leq n_h) \tag{43}$$

and

$$\begin{aligned} \bar{\Omega}(k) &= \begin{bmatrix} \bar{V}_1(k)x^T(k) & \cdots & \bar{V}_{n_h}(k)x^T(k) \\ \bar{V}_{n_h+1}(k)x^T(k) & \cdots & \bar{V}_{2n_h}(k)x^T(k) \\ \vdots & & \vdots \\ \bar{V}_{(m-1)n_h+1}(k)x^T(k) & \cdots & \bar{V}_{p^v}(k)x^T(k) \end{bmatrix} \\ &= \left[\hat{V}^T_{1,:}(k)x^T(k) \cdots \hat{V}^T_{n_h,:}(k)x^T(k)\right] \in R^{m \times p^w} \end{aligned} \tag{44}$$

where $\bar{V}_j(k)$ is the jth element of $\bar{V}(k)$, $1 \leq j \leq p^v$.

2) The adaptive dead zone learning rate of hidden layer is defined as

$$\begin{cases} \alpha^w = \alpha^w, \text{ if } \|e(k)\| \geqslant \Gamma(k) \\ \alpha^w = 0, \text{ if } \|e(k)\| < \Gamma(k) \end{cases} \tag{45}$$

with $\Gamma = \tilde{e}^w_m(k)/\sqrt{\left(1 - \frac{\alpha^w\|r^w\|^2}{\rho^w(k+1)\Upsilon(k)}M(k)\right)}$ and $\tilde{e}^w_m(k)$ being the maximum value of $\tilde{e}^w(k)$ defined in (41).

3) The recurrent hybrid adaptive parameter of hidden layer β^w is defined as

$$\begin{cases} \beta^w = 1, \text{if } (r^w)^T\chi(k) \geqslant 0 \\ \beta^w = 0, \text{if } (r^w)^T\chi(k) < 0 \end{cases} \tag{46}$$

with $\chi(k) = \left(\eta I + \bar{\Omega}^T(k)\bar{\Omega}(k)\right)^{-1}\bar{\Omega}^T(k)\hat{D}^w(k)$. β^w can automatically cut off the recurrent training signal as in the lower right corner of Fig.1 whenever weight convergence and stability conditions of the RNN are violated as shown in Theorem 2 of section IV.

Remark 4: Summary of the RRSPSA algorithm for RNNs.

1) Set a maximum training step as the stop criterion;
2) Initialize the weights for the RNN;
3) Form new input $x(k)$ as defined in equation (3);
4) Perform forward computation and calculate the output $\hat{y}(k)$ and error $e(k)$ with the input and current estimated weights;
5) Calculate ρ^v, α^v, and β^v using (31), (32), and (33), respectively;
6) Update the output layer weights $\bar{V}(k+1)$ via the learning law (13) for the next iteration;
7) Calculate $\rho^w(k+1)$, α^w, and β^w using (42), (45), and (46), respectively;
8) Update the hidden layer weights $\bar{W}(k+1)$ via the learning law (34) for the next iteration;
9) Go back to step 3 to continue the iteration until the stop criterion is reached.

V. Examples

In this section, the proposed RRSPSA training algorithm is tested on two application problems. To obtain a comparative idea about performance of the algorithm, we have compared the RRSPSA with a standard BP and RTRL training algorithms using the fixed learning rate, respectively.

We propose to compare the three algorithms in terms of the mean squared error (MSE) and the standard deviation (STD), which are defined, respectively, as:

$$MSE \triangleq \frac{1}{n}\sum_{k=1}^{n} e^2(k)$$

$$STD \triangleq \left(\frac{1}{n}\sum_{k=1}^{n}(e(k)-\bar{e})^2\right)^{1/2}$$

where $\bar{e} \triangleq \frac{1}{n}\sum_{k=1}^{n} e(k)$.

A. Time series prediction

Sunspots are dark blotches on the sun which are caused by magnetic storms on the surface of the sun. The underlying mechanism for sunspot appearances is not exactly known. The sunspots data is a classical example of a combination of periodic and chaotic phenomena which has served as a benchmark in the statistics literature of time series and much work has been done in trying to analyze the sunspot data using neural networks [16]. One set of the high quality America Sunspot Numbers, monthly sunspot numbers from December 1949 to October 2009 in the database, is used as an example to minimize unnecessary human measurement errors. The total America sunspot number in the monthly database is 778 points. The output sequence is scaled by dividing by 100, to get into a range suitable for the network. The hidden neuron number of the RNN is ten. The selected learning rates for BP and RTRL are 0.09 and 0.01, respectively. Perturbation step size for RRSPSA is $c = 0.01$. Fig. 2 shows outputs of the plant $y^*(k)$ and estimated outputs $y(k)$ of the RNN trained by the three different methods. Table 1 gives comparison results of the three methods in terms of MSE and STD.

TABLE I
COMPARISON OF MEAN SQUARED ERROR AND STANDARD DEVIATION OF THE TRAINING ERROR

Squared Error	RRSPSA	RTRL	BP
MSE	$6.01e^{-4}$	$1.30e^{-3}$	$4.30e^{-3}$
STD	$2.45e^{-2}$	$3.61e^{-2}$	$6.57e^{-2}$

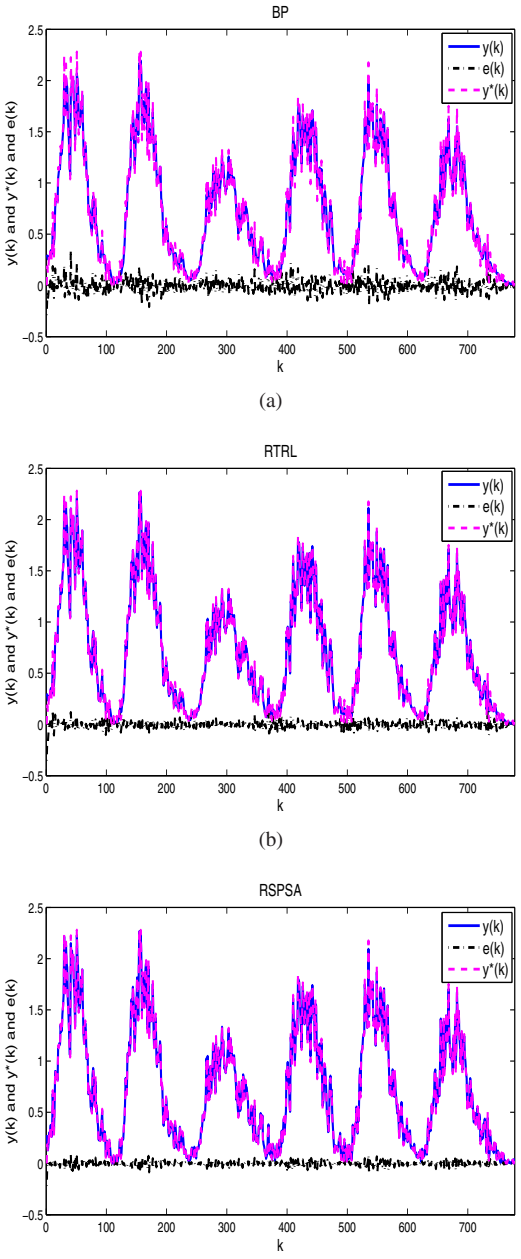

Fig. 2. Output $y(k)$ and reference signal $y^*(k)$ and error $e(k)$ using BP and RTRL and RRSPSA training algorithms for time series prediction.

B. Michaelis-Menten equation

Consider a continue time Michaelis-Menten equation [17]

$$\dot{y}(t) = 0.45\frac{y(t)}{2.27+y(t)} - y(t) + u(t) \qquad (47)$$

Given an unit impulse as input $u(t)$, a fourth-order Runge-Kutta algorithm is used to simulate this model with integral step size $\triangle t = 0.01$, and 1000 equi-spaced samples are obtained from input and output with a sampling interval of $T = 0.02$ time units. The measured noise is white Gaussian noise with mean 0 and variance 0.05. An RNN is utilized to emulate the dynamics of the given system and approximate the system output in (47). Gradient function of this model has many local minima and the global minimum has a very narrow domain of attraction [17]. We compare performance of the RRSPSA to that of both BP and RTRL training algorithms. The hidden neuron number of RNNs in this example is selected as ten. The fixed learning rates for BP and RTRL are 0.03 and 0.01, respectively. The perturbation size $c = 0.01$ is selected for RRSPSA. Fig. 3 shows outputs of the plant $y^*(k)$ and estimated outputs $y(k)$ of the RNN trained by the three different methods. Table 2 shows the comparison results of the three methods in terms of MSE and STD.

Remark 5: Simulation results presented in this paper for the BP training algorithm performs worst as compared to other methods. The reason is that the second partial derivatives sim-

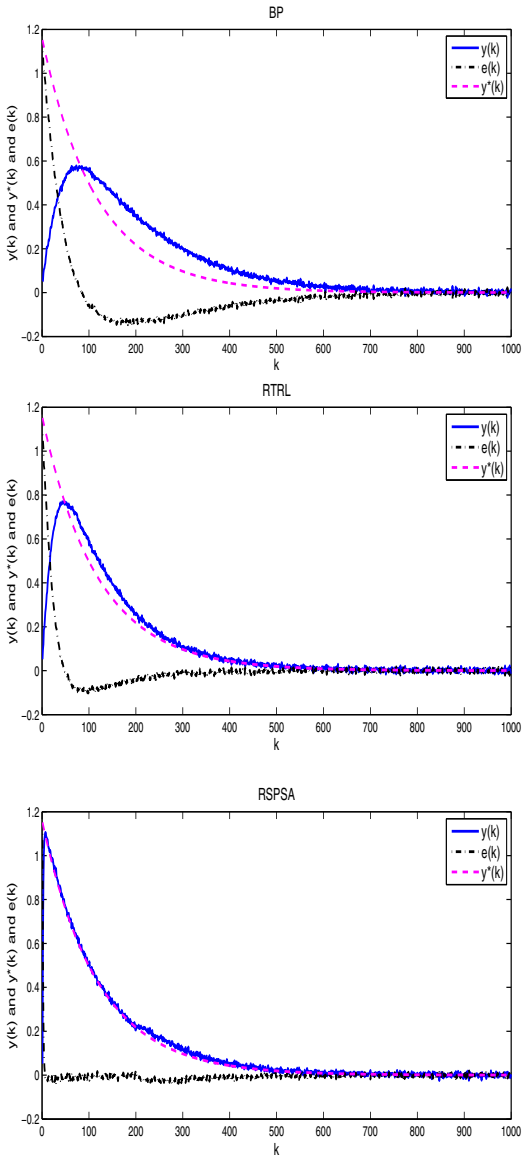

Fig. 3. Output $y(k)$ and reference signal $y^*(k)$ and error $e(k)$ using BP and RTRL and RRSPSA training algorithms for simulation of Michaelis-Menten equation.

TABLE II
COMPARISON OF THE MEAN SQUARED ERROR AND STANDARD DEVIATION OF THE ERROR

	RRSPSA	RTRL	BP
MSE	$4.2e^{-3}$	$1.64e^{-2}$	$2.98e^{-2}$
STD	$6.46e^{-2}$	$1.28e^{-1}$	$1.72e^{-1}$

ilar to that of as in (14) are not considered in the BP training. For RTRL, the slower convergence compared with RRSPSA is mainly caused by the small fixed learning rate and absent of the recurrent hybrid adaptive parameter for guaranteed weight convergence in the presence of noise. RRSPSA is able to use relatively larger effective learning rate to accelerate system response with guaranteed weight convergence.

VI. CONCLUSION

In this paper, a RRSPSA algorithm is proposed to train RNNs with guaranteed weight convergence and stability in the sense of Lypunov function. In addition to the inherent weight convergence property of a standard SPSA, the RRSPSA uses three specific designed adaptive parameters, including the important recurrent hybrid adaptive parameter, to optimize convergence speed and computational efficiency. Two experiments were carried out to verify the derived theoretical results, which show clearly that the learning performance of RRSPSA is improved in terms of convergence speed and mean squared error as compared to that of the conventional gradient-type RNN algorithms.

REFERENCES

[1] S. Haykin, *Neural networks: a comprehensive foundation.* Prentice Hall, New Jersey, USA, 1999.

[2] D. Mandic and J. Chambers, *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. Wiley, New York, USA, 2001.

[3] J. C. Spall, "Multivariate stochastic approximation using a simultaneous-perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, no. 3, pp. 332–341, 1992.

[4] Y. Maeda and M. Wakamura, "Simultaneous perturbation learning rule for recurrent neural networks and its FPGA implementation," *IEEE Transactions on Neural Networks*, vol. 16, no. 6, pp. 1664–1672, 2005.

[5] Q. Song, J. C. Spall, Y. C. Soh, and J. Ni, "Robust neural network tracking controller using simultaneous perturbation stochastic approximation," *IEEE Transactions on Neural Networks*, vol. 19, no. 5, pp. 817 –835, 2008.

[6] A. Trunov and M. Polycarpou, "Automated fault diagnosis in nonlinear multivariable systems using a learning methodology," *IEEE Transactions on Neural Networks*, vol. 11, no. 1, pp. 91–101, 2000.

[7] J. C. Spall and J. A. Cristion, "Model-free control of nonlinear stochastic systems with discrete-time measurements," *IEEE Transactions on Automatic Control*, vol. 43, no. 9, pp. 1198 –1210, Sept. 1998.

[8] ——, "A neural network controller for systems with unmodeled dynamics with applications to wastewater treatment," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 27, no. 3, pp. 369–375, 1997.

[9] Y. Maeda and R. De Figueiredo, "Learning rules for neuro-controller via simultaneous perturbation," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1119–1130, 1997.

[10] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 4, pp. 339–356, 1988.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error backpropagation," in *Parallel Distributed Processing*, vol. 1. Cambridge, MA: MIT Press, pp. 318–362, 1986.

[12] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.

[13] ——, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Back-Propagation: Theory, Architectures and Applications*. NJ: Lawrence Erlbaum, pp. 433-486, 1995.

[14] Q. Song, J. Xiao, and Y. C. Soh, "Robust backpropagation training algorithm for multilayered neural tracking controller," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1133–1141, 1999.

[15] T. Lin, C. Giles, B. Horne, and S. Kung, "A delay damage model selection algorithm for narx neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2719–2730, 1997.

[16] Y. Park, T. Murray, and C. Chen, "Predicting sun spots using a layered perceptron neural network," *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 501–505, 2002.

[17] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, 2010.