Improved Generalization in Recurrent Neural Networks Using the Tangent Plane Algorithm

P May
K College, Brook Street,
Tonbridge, Kent, UK

E Zhou
Applied Engineering and Science
Academic Group,University of
Bolton, UK

C. W. Lee
Applied Engineering and Science
Academic Group University of
Bolton, UK

***Abstract*—The tangent plane algorithm for real time recurrent learning (TPA-RTRL) is an effective online training method for fully recurrent neural networks. TPA-RTRL uses the method of approaching tangent planes to accelerate the learning processes. Compared to the original gradient descent real time recurrent learning algorithm (GD-RTRL) it is very fast and avoids problems like local minima of the search space. However, the TPA-RTRL algorithm actively encourages the formation of large weight values that can be harmful to generalization. This paper presents a new TPA-RTRL variant that encourages small weight values to decay to zero by using a weight elimination procedure built into the geometry of the algorithm. Experimental results show that the new algorithm gives good generalization over a range of network sizes whilst retaining the fast convergence speed of the TPA-RTRL algorithm.**

Keywords—real time recurrent learning; tangent plane; generalization; weight elimination; temporal pattern recognition; non-linear process control

I. INTRODUCTION

It is usually the case that smaller networks generalize better than larger ones. To limit the size of the network, it can either use additive [1 - 3], subtractive [4 - 6] or weight decay techniques [7 - 9]. A common feature is that they try to balance the representational capacity of the network against the information criterion in the training data. Weight decay techniques are considered here.

The principal idea of weight decay is to have the network remove the superfluous weights by itself. This can be achieved by giving each weight connection a tendency to decay to zero so that connections disappear unless they are reinforced. The simplest method is to subtract a small proportion of a weight after it has been updated. This is equivalent to adding a penalty term to the original error function and performing gradient descent on the resulting total error. While this method clearly penalizes more connection weights than necessary, it overly discourages large weights. May et al [7] have shown that using a weight elimination procedure which forces small weight values to decay faster than the large ones is an effective method for removing superfluous weights from a neural network whilst causing minimal disturbance to the learning process. Simulation results show that it out performs weight decay in back propagation learning. Williams [8] have shown that the method of maximum entropy indicates a Laplace prior and proposes a penalty term based on the L1 norm of weights. A further refinement of this approach involves using a sparseness measure based on the L1 and L2 norm of weights [9]. Experiments with Hoyer's method indicate that it performs well in comparison with weight decay and weight elimination.

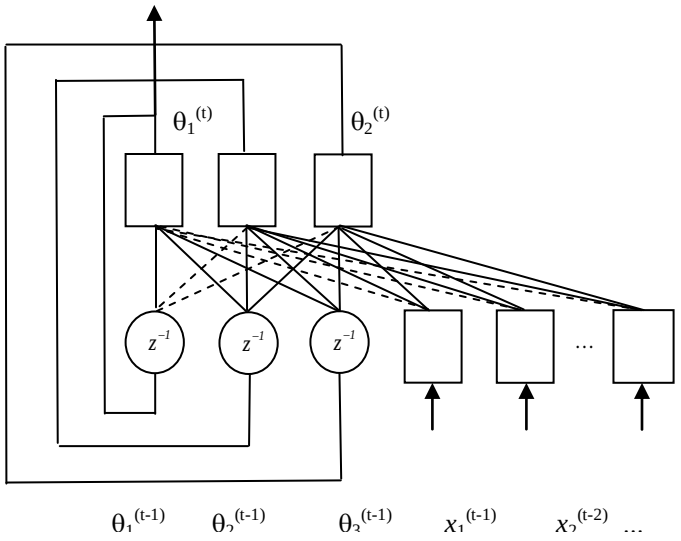

Fig. 1. An example of a fully recurrent neural network with one output unit, two hidden units, and two input units. In this figure the dotted lines represent connections that have been removed from the network

Pruning techniques reduce the number of free parameters in a neural network by removing redundant units and connections. If applied properly this approach often improves generalization. Giles et al [4] have made a comparison of pruning and weight decay in a second order recurrent neural network. Simulations were carried out on strings generated by two regular grammars, a randomly generated 10 state grammar and an 8 state triple parity grammar. These experiments show that pruned networks outperform networks with weight decay in cases where the starting weights were close to a solution. However, in situations where the original network was not well trained weight decay was shown to improve generalization. The convergence time for training with weight decay increased with the learning rate. Leung et al [5] have used a recursive least squares (RLS) algorithm to train the weights of a recursive neural network (RNN). After training the error covariance matrix of the RLS algorithm was used to remove unimportant weights from the network. Simulations show that this new approach is an effective joint learning-

pruning method for recurrent neural networks. Ahmed et al [6] have used the 'Lempel-Ziv' complexity (LZC) measure to prune artificial neural networks (ANNs). The silent pruning algorithm (SPA) prunes ANNs causing minimal disturbance to the network. SPA prunes hidden units during the training process according to their ranking computed from the LZC. Simulation tests carried out on standard benchmarking neural network problems show that SPA can produce simplified ANNs with good generalization ability.

Other techniques for improving generalization in a neural network include injecting synaptic noise [10], regularisation [11] and early stopping [12]. Hirasawa [11] have used a regularisation term for calculating second order derivatives in a Universal Learning Network (ULN) that decreases the degrees of freedom of the network. A ULN is a fully connected recurrent neural network with multiple nodes and multiple branches with arbitrary time delays. Simulation results for a hydraulically controlled robot arm have shown that the proposed method can improve generalization and avoids problems like local minima. In Giles et al [10] synaptic noise was injected into a high order recurrent neural network. Additive, multiplicative and cumulative noise was injected into the weights of a neural network where cumulative is taken to mean accumulated over time. Simulation results on the dual parity automaton problem [4] show that these methods can improve generalization and convergence simultaneously

II. OBJECTIVES

In this paper a weight elimination procedure is used to improve generalization in recurrent neural networks. The algorithm has been developed from one described elsewhere and referred to as iTPA [7]. Unlike other implementations of the weight elimination procedure, the method used here is built into the geometry of the algorithm. There are currently no implementations of the weight elimination procedure for recurrent neural networks.

The rest of the paper is organized as follows: in section III and IV, a detailed derivation of the algorithm and an evaluation of weight sensitivity methods are presented. In section V and VI, the results of computer simulations are considered and the differences in the results tested for statistical significance. Finally, the conclusions are presented in section VII.

III. DESCRIPTION OF THE ALGORITHM

In May et al [7], a fast tangent plane method is described for training feed-forward multilayered neural networks. This method uses the training data to define a surface in weight space. The weights are updated by moving from the current position to a point nearby the foot of the perpendicular to this surface, biased in the direction that forces small weights values to decay faster than large ones. The principal advantage of the algorithm is that it self regulates the size of the network by removing superfluous weights which can be harmful to generalization. Unlike other implementations of the weight elimination procedure, the method used here is built into the geometry of the algorithm and causes minimal disruption to the learning process. Experimental results show that a weight elimination strategy is a more effective method for improving the generalization performance of the tangent plane algorithm than a weight growth strategy. This paper describes an equivalent implementation of the tangent plane algorithm for recurrent neural networks. A tangent plane variant of the real time recurrent learning algorithm referred to as TPA-RTRL has already been described elsewhere [13]. The detailed derivation of the algorithm follows

Consider a FRNN of units $\{u_j\}$ (see Fig 1). For unit u_j, $\mathbf{w}_j^T = [w_{j1}, w_{j2}, \ldots, w_{j,(n+m+1)}]$ denotes a $(n+m+1)\times 1$ vector of weights, where n are the number of processing units, m the number of external inputs, with one remaining input being for the fixed input bias. Let ϕ_j and θ_j denote the net input and output of u_j, and f the unit's activation function, typically $tanh(x)$. The following equations describe the FRNN at time instant t

$$\theta_j^{(t)} = f(\phi_j^{(t)}),\ j = 1, 2, \ldots, n \quad (1)$$

$$\phi_j^{(t)} = \sum_{i=1}^{n+m+1} w_{ji}^{(t)} z_i^{(t)} \quad (2)$$

$$[z_i^{(t)}]^{\mathrm{T}} = [\theta_1^{(t-1)}, \ldots, \theta_n^{(t-1)}, +1, x_1^{(t-1)}, \ldots, x_m^{(t-m)}] \quad (3)$$

For the non-linear time series prediction paradigm, there is only one output unit of the FRNN. Let this output unit be denoted by u_1, with $\theta_1^{(t)}$ at time step t being trained to mimic the teaching value $y_1^{(t)}$. For a given set of inputs $\{x_i^{(t-i)}, i = 1, \cdots, m\}$, we can consider $\phi_1^{(t)}$ to be a function of the weights, $\phi_1^{(t)} : R^{n\times(n+m+1)} \to R$. Thus the equation $\phi_1^{(t)} = f^{-1}(y_1^{(t)})$ defines a $n\times(n+m+1)-1$ surface in $R^{n\times(n+m+1)}$. The aim of this training procedure is to move from the current position $\boldsymbol{a} \in R^{n\times(n+m+1)}$ in weight space to the foot of the perpendicular to the tangent plane of this surface (see Fig 2)

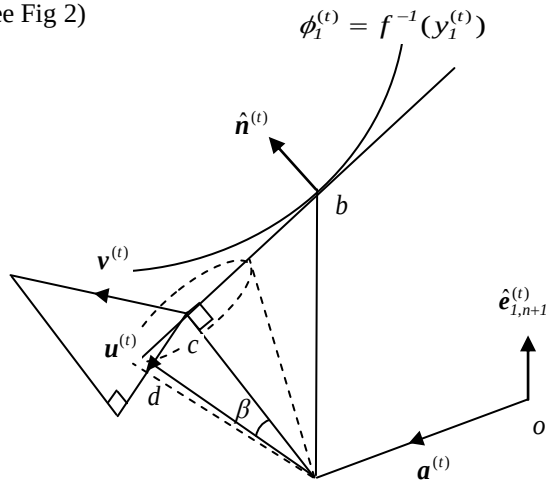

Fig. 2. Movement from the present position a to the point d inclined at an angle β to the perpendicular ac to the tangent plane to the constraint surface

$\phi_1^{(t)} = f^{-1}(y_1^{(t)})$ at point b in weight space $R^{n\,x\,(m+n+1)}$. The vector $\boldsymbol{u}$ represents the projection of the weight elimination vector $\boldsymbol{v}$ orthogonally onto the normal $\boldsymbol{n}$ to the constraint surface at b

Let $\boldsymbol{a}^{(t)} = \sum_{j,i} w_{ji}^{'(t)} \hat{\boldsymbol{e}}_{ji}$ be the current vector of weights, where $\hat{\boldsymbol{e}}_{ji}$ is a unit vector in the direction of the w_{ji} axis. Use the equation $f^{-1}(y_1^{(t)}) = w_{1,n+1}^{(t)} + \sum_{i \neq n+1} w_{1,i}^{(t)} z_i^{(t)}$ to find a particular value $w_{1,n+1}^{''(t)}$ for the constant input weight $w_{1,n+1}$ from the values $w_{ji}^{'(t)}$ of the other weights, so that the constraint surface $\phi_1^{(t)} = f^{-1}(y_1^{(t)})$ contains the point $\boldsymbol{b}^{(t)} = w_{1,n+1}^{''(t)} \hat{\boldsymbol{e}}_{1,n+1} + \sum_{j,i \neq 1,n+1} w_{ji}^{'(t)} \hat{\boldsymbol{e}}_{ji}$. Now, if we use the equations $f^{-1}(y_1^{(t)}) = w_{1,n+1}^{''(t)} + \sum_{i \neq n+1} w_{1i}^{'(t)} z_i^{(t)}$ and $f^{-1}(\theta_1^{(t)}) = w_{1,n+1}^{'(t)} + \sum_{i \neq n+1} w_{1i}^{'(t)} z_i^{(t)}$, and from the definition of $\boldsymbol{b}^{(t)}$, we have

$$\begin{aligned} \boldsymbol{b}^{(t)} - \boldsymbol{a}^{(t)} &= [\, w_{1,n+1}^{''(t)} - w_{1,n+1}^{'(t)} \,]\, \hat{\boldsymbol{e}}_{1,n+1} \\ &= [\, f^{-1}(y_1^{(t)}) - f^{-1}(\theta_1^{(t)}) \,]\, \hat{\boldsymbol{e}}_{1,n+1} \end{aligned} \quad (4)$$

Let $\hat{\boldsymbol{n}}$ be the unit normal to the surface at $\boldsymbol{b}$, so $\hat{\boldsymbol{n}} = \nabla\phi_1 / \|\nabla\phi_1\|$. The length of the perpendicular from $\boldsymbol{a}$ to the tangent plane at $\boldsymbol{b}$ is $(\boldsymbol{b} - \boldsymbol{a}) . \hat{\boldsymbol{n}}$. If $\boldsymbol{c}$ is the foot of the perpendicular from $\boldsymbol{a}$ to the tangent plane at $\boldsymbol{b}$,

$$\boldsymbol{c}^{(t)} - \boldsymbol{a}^{(t)} = (f^{-1}(y_1^{(t)}) - f^{-1}(\theta_1^{(t)}))(\hat{\boldsymbol{e}}_{1,n+1} . \hat{\boldsymbol{n}}^{(t)})\hat{\boldsymbol{n}}^{(t)} \quad (5)$$

The vector that is directed towards the origin and biased along the axes of the weights w_{ji} that have small weight values relative to some small positive constant w_a is $\boldsymbol{v}^{(t)} = -\sum_{j,i} (w_{ji}^{(t)} / w_a)\, \hat{\boldsymbol{e}}_{ji}^{(t)} / (1 + (w_{ji}^{(t)} / w_a)^2)$. The projection of $\boldsymbol{v}^{(t)}$ onto the tangent plane is given by

$$\boldsymbol{u}^{(t)} = \boldsymbol{v}^{(t)} - (\boldsymbol{v}^{(t)} . \hat{\boldsymbol{n}}^{(t)})\hat{\boldsymbol{n}}^{(t)} \quad (6)$$

Thus, if $\boldsymbol{d}^{(t)}$ is the point of intersection with the tangent plane of a line from $\boldsymbol{a}^{(t)}$ inclined at angle β to the perpendicular *ac*, then

$$\boldsymbol{d}^{(t)} - \boldsymbol{a}^{(t)} = \left\| \boldsymbol{c}^{(t)} - \boldsymbol{a}^{(t)} \right\| \tan\beta \frac{\boldsymbol{u}^{(t)}}{\left\| \boldsymbol{u}^{(t)} \right\|} + \boldsymbol{c}^{(t)} - \boldsymbol{a}^{(t)} \quad (7)$$

Let $\delta^{(t)} = f^{-1}(y_1^{(t)}) - f^{-1}(\theta_1^{(t)})$ be the error in the input to u_1 at time *t*. Hence, using equations (5), (6) and (7) yields

$$\boldsymbol{d}^{(t)} - \boldsymbol{a}^{(t)} = \delta^{(t)}(\hat{\boldsymbol{e}}_{1,n+1} . \hat{\boldsymbol{n}}^{(t)})\hat{\boldsymbol{n}}^{(t)} + \left|\delta^{(t)}\right|(\hat{\boldsymbol{e}}_{1,n+1} . \hat{\boldsymbol{n}}^{(t)}) \tan\beta \frac{\boldsymbol{u}^{(t)}}{\left\| \boldsymbol{u}^{(t)} \right\|} \quad (8)$$

However

$$\begin{aligned} \hat{\boldsymbol{e}}_{1,n+1} . \hat{\boldsymbol{n}} &= \hat{\boldsymbol{e}}_{1,n+1} . \frac{1}{\left\| \nabla\phi_1^{(t)} \right\|} \sum_{j,i} \frac{\partial \phi_1^{(t)}}{\partial w_{ji}} \hat{\boldsymbol{e}}_{ji} \\ &= \frac{1}{\left\| \nabla\phi_1^{(t)} \right\|} \frac{\partial \phi_1^{(t)}}{\partial w_{1,n+1}} = \frac{1}{\left\| \nabla\phi_1^{(t)} \right\|} \frac{\partial}{\partial w_{1,n+1}} \sum_{j,i} w_{ji}^{(t)} z_i^{(t)} \\ &= \frac{1}{\left\| \nabla\phi_1^{(t)} \right\|} \end{aligned} \quad (9)$$

Therefore,

$$\boldsymbol{d}^{(t)} - \boldsymbol{a}^{(t)} = \frac{1}{\left\| \nabla\phi_1^{(t)} \right\|^2} \delta^{(t)} \nabla\phi_1^{(t)} + \frac{\left|\delta^{(t)}\right|}{\left\| \nabla\phi_1^{(t)} \right\|} \tan\beta \frac{\boldsymbol{u}^{(t)}}{\left\| \boldsymbol{u}^{(t)} \right\|} \quad (10)$$

Thus, the adjustment to weight w_{ji} is given by

$$\begin{aligned} \Delta w_{ji}^{(t)} &= \frac{1}{\left\| \nabla\phi_1^{(t)} \right\|^2} \delta^{(t)} \frac{\partial \phi_1^{(t)}}{\partial w_{ji}} + \frac{\left|\delta^{(t)}\right|}{\left\| \nabla\phi_1^{(t)} \right\|} \tan\beta \frac{1}{\left\| \boldsymbol{u}^{(t)} \right\|} \times \\ &\left(v_{ji}^{(t)} - \frac{1}{\left\| \nabla\phi_1^{(t)} \right\|^2} \sum_{p,q} v_{pq}^{(t)} \frac{\partial \phi_1^{(t)}}{\partial w_{pq}} \frac{\partial \phi_1^{(t)}}{\partial w_{ji}} \right) \end{aligned} \quad (11)$$

where

$$\left\| \nabla\phi_1^{(t)} \right\|^2 = \sum_{j,i} \left(\frac{\partial \phi_1^{(t)}}{\partial w_{ji}} \right)^2 \quad (12)$$

and

$$\left\| \boldsymbol{u}^{(t)} \right\|^2 = \sum_{j,i} \left(v_{ji}^{(t)} - \frac{1}{\left\| \nabla\phi_1^{(t)} \right\|^2} \frac{\partial \phi_1^{(t)}}{\partial w_{ji}} \sum_{p,q} \left(v_{pq}^{(t)} \frac{\partial \phi_1^{(t)}}{\partial w_{pq}} \right) \right)^2 \quad (13)$$

and from May (2012)

$$\frac{\partial \phi_1^{(t)}}{\partial w_{ji}^{(t)}} = \sum_{m=1}^{n} f'(\phi_m^{(t-1)}) \frac{\partial \phi_m^{(t-1)}}{\partial w_{ji}^{(t-1)}} w_{1m}^{(t)} + \delta_{j1} z_i^{(t)} \quad (14)$$

where

$$\delta_{j1} = \begin{cases} 1, & \text{if } j = 1 \\ 0, & \text{otherwise} \end{cases}$$

Equation (14) holds for all units u_k , $k = 1 \cdots n$. Thus, we can create a dynamical system with the dynamics given by

$$\frac{\partial \phi_k^{(t)}}{\partial w_{ji}^{(t)}} = \sum_{m=1}^{n} f'(\phi_m^{(t-1)}) \frac{\partial \phi_m^{(t-1)}}{\partial w_{ji}^{(t)}} w_{km}^{(t)} + \delta_{jk} z_i^{(t)} \quad (15)$$

Since we assume that the initial state of the recurrent neural network has no functional dependence on the weights, we have $\partial \phi_k^{(t=t_0)} / \partial w_{ji} = 0,\ k \le n,\ j \le n,\ 1 \le i \le n+m+1$.

The training procedure described above is a tangent plane variant of the real time recurrent learning algorithm (RTRL) proposed by Williams and Zipser [14] for online training of recurrent neural networks. The GD-RTRL algorithm utilises the gradient information to guide the search towards the minimum training error. However, it is sensitive to the choice of a learning rate that requires careful tuning. The TPA-RTRL algorithm proposed by May et al [13] differs to the original algorithm by automatically calculating the correct step size in weight space using the method of approaching tangent planes. The improved TPA-RTRL algorithm described here includes a weight elimination procedure built into its geometry that suppresses the formation of large weight values

The angle parameter β, which gives the angle between the movement vector and the perpendicular to the tangent plane, requires setting to an appropriate value. Its value is preferred to be small, typically $tan\beta \approx 0.005$. Tests showed that the performance of the algorithm deteriorated rapidly when $tan\ \beta$ was greater than 0.1. Network training times are much longer and the number of failures to converge more frequent. The reason for the failure to converge was that the weights became clustered too closely about the origin with average weight values < 0.01. In these circumstances individual weight updates $\Delta w_{ji}^{(t)}$ are no longer based on the values of previous derivatives $\partial \phi_k^{(t)} / \partial w_{ji}$, $k = 1, \ldots, n$ over the whole trajectory from $t = t_0$ to t_1 but are driven by the training data, thus the trajectory will not follow a steepest descent path

The weight sensitivity parameter w_a, which determines the size of the push towards the origin that an individual weight $w_{ji}^{(t)}$ will receive, also requires setting to an appropriate value. w_a is preferred to be small, typically 0.5, so that weights with small values are selected for removal from the network. An individual term $v_{ji}^{(t)} = -(w_{ji}^{(t)} / w_a) / (1 + (w_{ji}^{(t)} / w_a)^2)$ in $\boldsymbol{v}^{(t)}$ varies according to $(w_{ji}^{(t)} / w_a)$ in an anti-symmetric fashion. When $|w_{ji}^{(t)}| < w_a$, the directional term for that weight is approximately linear. On the other hand, when $|w_{ji}^{(t)}| > w_a$, the directional term quickly approaches to zero. Thus a weight will receive a large push when $w_{ji}^{(t)}$ equals w_a.

The iTPA-RTRL algorithm uses the gradient vector to do a linear extrapolation of the constraint surface $\phi_1^{(t)} = f^{-1}(y_1^{(t)})$ in order to gain a new weight vector that is hoped to be on, or at least close, to this surface. However, $\phi_1^{(t)} = f^{-1}(y_1^{(t)})$ is a non-trivial function of the weights, the recursive feedback activations $\theta_i^{(t-1)}$, $i = 1, \cdots, n$, themselves being a function of the weights. Thus the basic approximation that the surface can be locally approximated may only be limited to certain regions of weight space close to $\phi_1^{(t)} = f^{-1}(y_1^{(t)})$. Removing superfluous weight from the network using a weight elimination procedure will have the effect of constraining the weight vector to a subspace of $R^{n \times (n+m+1)}$. This in turn will have a smoothing effect on the constraint surface $\phi_1^{(t)} = f^{-1}(y_1^{(t)})$ making a tangent plane approximation to this surface viable.

A potential difficulty with the iTPA-RTRL algorithm is that while learning from a large amount of data the weight change between the start of the learning phase ($t = t_0$) and the end of the learning phase ($t = t_1$) will not be small. The reason is that the weight update $\Delta w_{ji}^{(t)}$ is based upon the value of the derivatives $\partial \phi_k^{(t)} / \partial w_{ji}$, $k = 1, \ldots, n$ over the whole trajectory from $t = t_0$ to t_1. These derivatives are calculated recursively using a relationship that is dependent on the weights and the weights are not constant. Thus, the iTPA-RTRL algorithm can move far from the steepest descent trajectory and never return. Constraining the FRNN to small weight values might actually improve convergence behaviour as small weight values will lead to small fluctuations in the gradient vector. This in turn might lead to a more robust implementation of the TPA-RTRL algorithm as the computation of the partial derivatives $\partial \phi_k^{(t)} / \partial w_{ji}$ is prone to arithmetic overflow errors

IV. Estimating Weight Sensitivity Values

The effectiveness of the weight elimination procedure can be measured by calculating the importance of each weight with the expectation that weights with low importance are redundant in the network. There are several methods for calculating the importance of the weights [15, 16], optimal brain damage [17], and optimal brain surgeon [18]. In the case of OBD, the saliency of removal of a weight is estimated by using the second derivative of the error function. Low saliency means low importance of the weights. OBS avoids the drawbacks of approximating the second derivatives by computing them exactly [18]. The last two methods have the disadvantage of requiring training down to the error minimum. The method adopted here is autoprune [7, 19], which avoids the disadvantage of training down to the error minimum. Autoprune uses a statistic t to allocate an importance coefficient to each weight based upon the assumption that a weight becomes zero. It can be computed at any time during the training process

$$t(w_{ji}) = log\left(\frac{\left| \sum_t w_{ji} - \Delta w_{ji}^{(t)} \right|}{\sum_t (\Delta w_{ji}^{(t)} - (\Delta \overline{w}_{ji}))^2} \right) \quad (16)$$

In the above formula, sums are over all training examples t of the training set, and the overline means arithmetic mean over all examples. A large value of t_{ji} indicates high importance of weight w_{ji}. May et al [7] have demonstrated using autoprune the effectiveness of the weight elimination procedure in a multi-layer feed forward neural network.

V. Computer Simulations

Comparative tests were performed on the iTPA-RTRL and TPA-RTRL algorithms, and the original GD-RTRL algorithm using different network sizes and initial conditions. In order to assess the generalization performance of the iTPA-RTRL algorithm, a weight decay procedure was utilized with the original GD-RTRL algorithm, which has been shown to improve generalization in second order recurrent neural networks [4].

The comparison was done based on the performance for the following benchmark neural network datasets; the Henon map time series [20], the continuous stirred tank reactor [21], and the non-linear dynamic plant [22]. The Henon map time series was used to analyse the effect of changing the weight sensitivity parameter w_a on the ability of a single layer FRNN to generalize. The continuous stirred tank and non-linear dynamic plant problems were chosen to establish the degree to which each learning algorithm used in this study has succeeded in removing superfluous weights from the network.

A. Simulation Problems

The Henon map problem is a chaotic time-series prediction problem. The time series is computed by

$$x^{(t+1)} = 1 - c(x^{(t)})^2 + b x^{(t-1)} \quad (17)$$

Where $b = 0.3$, $c = 1.4$, and $x^{(1)} = x^{(2)} = 0.6313354$. The objective of the simulation is to train a single layer FRNN with one input and one output to model the chaotic series generated by (17). Since $x_{max} = 1.272967$ and $x_{min} = -1.284657$, the input values were scaled in the range [-1, 1].

The non-linear dynamic plant problem is a high order non-linear system introduced in Narendra and Parthasarathy [22]. It is modeled by the following discrete time equation

$$y^{(t)} = \frac{y^{(t-1)} y^{(t-2)} y^{(t-3)} u^{(t-1)} [y^{(t-3)} - 1] + u^{(t)}}{1 + [y^{(t-2)}]^2 + [y^{(t-3)}]^2} \quad (18)$$

Where $y^{(t)}$ is the model output at time t. A single layer FRNN with one output unit and two input units was trained. The training data was generated using a random input signal uniformly distributed over the interval [-1, 1].

The Van de Vusse reaction in a continuous stirred tank reactor (CSTR) can be modelled by the following discrete-time nonlinear system introduced by Hernandez and Arkun [21]

$$y^{(t)} = c_1 + c_2 u^{(t-1)} + c_3 y^{(t-1)} + c_4 [u^{(t-1)}]^3 + c_5 y^{(t-2)} u^{(t-1)} u^{(t-2)} \quad (19)$$

Here $y^{(t)}$ is the product concentration and $u^{(t)}$ the scaled reactant at time step t. The input $u^{(t)}$ has been normalized to $0 \le u^{(t)} \le 1$, and the parameters of the system are $c_1 = 0.558$, $c_2 = 0.538$, $c_3 = 0.116$, $c_4 = -0.127$, and $c_5 = -0.034$. A single layer FRNN with one output unit and two input units was trained. The training data was generated using a random input signal uniformly distributed over the interval [0, 1].

B. Network Initialization

The GD-RTRL algorithm requires two parameters that need to be set, the learning rate η and weight decay rate λ. Preliminary tests showed that the best results with the Henon map problem were obtained with $\eta = 0.01$ and $\lambda = 0.000001$. For the continuous stirred tank and non-linear plant problems, $\eta = 0.1$ and $\lambda = 0.00001$. The iTPA-RTRL algorithm also requires two parameters to be set: the angle parameter β and the weight sensitivity parameter w_a. For the Henon map problem: $tan\ \beta = 0.01$. The weight sensitivity parameter was set to different values in {0.5, 1.0, 2.0}. For the continuous tank and non-linear plant problems, $tan\ \beta = 0.05$ and $w_a = 1.0$. Both algorithms require the weight connections of the neural network to be set. In all the tests carried out the weight connections were initialized to random values in the range [-0.5, 0.5].

C. Discussion of Results

Henon map time series. The first test is a classical deterministic one-step-ahead prediction problem. The network used was a single layer FRNN with the number of processing units varied according to [8, 21, 26]. 20 trials were made with any failed trials excluded from the results. The error metric used was the mean square error obtained by averaging the square error over 1000 time steps. Network training was terminated when the mean square error on the training set was reduced to below 0.001 or 500,000 time steps trained. The ability of the network to generalize was measured over 1000 time steps after convergence had occurred.

TABLE I. Final Test Error and Number of Steps To Convergence For Different Values of The Parameter Wa

	Units = 6		Units = 9		Units = 12	
w_a	MSE x10^2	Steps x10^3	MSE x10^2	Steps x10^3	MSE x10^2	Steps x10^3
0.5	1.27	65	1.05	63	1.26	61
1.0	0.98	65	1.07	63	1.33	64
2.0	1.02	60	1.36	66	1.39	76

Table 1 shows the mean square error and the average number of steps to converge for different values of the weight sensitivity parameter w_a in the iTPA-RTRL algorithm. It was found that the generalization performance of the FRNN improved with decreasing values of w_a, except in the smallest network where it was found to be worse. This result is not surprising as selective pruning of weights in a small network structure is unlikely to improve the representational capacity of the network. It was also found that generalization performance declined with the size of the network. This was particularly noticeable for values of the weight sensitivity parameter greater than 0.5. Generalization was found to be

independent of the size of the network for the smallest value of w_a. This result suggests that selective pruning of weights implemented using small values of the parameter w_a have produced parsimonious networks capable of good generalization behavior.

A further test was carried out to assess the effectiveness of using a weight elimination procedure to produce 'good' network architectures. Weight sensitivity values based on t_{ji} statistic were computed during the last 1,000 time steps trained. Weights were considered to have low sensitivity values if their corresponding t_{ji} statistic was 50% of the mean value, which has been adapted from the pruning schedule iPrune [19]. Network training was terminated after 5,000,000 time steps. The ability of the network to generalize was measured over 1,000 time steps after convergence had occurred

TABLE II. FINAL TEST ERRORS AND % NUMBER OF WEIGHTS WITH LOW SENSITIVITY VALUES IN DIFFERENT SIZE NETWORKS FOR THE HENON MAP.

	Units = 6		Units = 9		Units = 12	
	MSE x10^2	n%	MSE x10^2	n%	MSE x10^2	n%
iTPA-RTRL	0.039	21.87	0.008	13.03	0.013	12.28
TPA-RTRL	0.039	24.46	0.012	13.15	0.036	13.80
GD-RTRL	0.024	26.49	0.013	12.68	0.022	12.61

Table 2 shows the mean square error and the percentage number of weights with low sensitivity values eligible for removal from the network. It was found that the new iTPA-RTRL algorithm gave the best generalization performance except in the smallest network where it was no better than the TPA-RTRL algorithm. The TPA-RTRL algorithm gave the worst performance. The poor generalization of both tangent plane algorithms in the smallest network is probably due to oscillatory behavior near a solution caused by a large step size. It was also found that networks trained by the iTPA-RTRL algorithm had the smallest percentage number of weights with low sensitivity values. The results suggest that a weight elimination strategy effectively discriminates between active and inactive weights in the network thus improving generalization performance. Figures 3 and 4 show the fit for the iTPA-RTRL and GD-RTRL algorithms using an FRNN with 12 processing units after 500,000 time steps. Clearly the fit is not exact but this is quite reasonable considering the type of input data used.

Continuous stirred tank reactor. The second test is a discrete-time nonlinear system introduced by Hernandez and Arkun [21]. The network used was a single layer FRNN with the number of processing units varied according to [8, 21, 26]. The error metric used was the mean square error obtained by averaging the square error over 100 time steps. Weight sensitivity values based on the t_{ji} statistic measured over the last 100 time steps trained. Weights were considered to have low sensitivity values if their corresponding t_{ji} statistic was 50% of the mean value. Network training was terminated after 50,000 time steps.

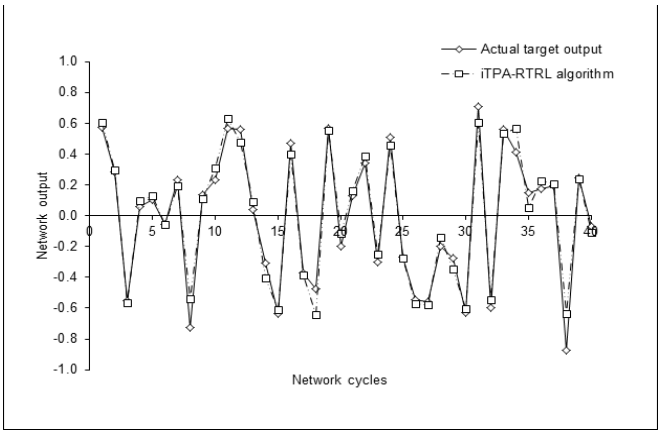

Fig. 3. Typical convergence behavior of the new iTPA-RTRL algorithm on the Henon map time series problem.

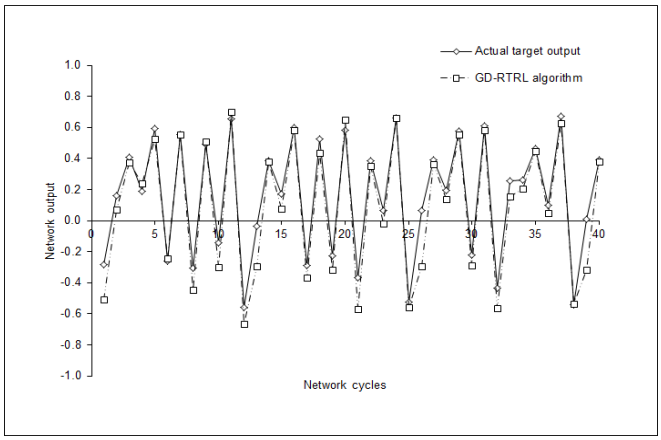

Fig. 4. Typical convergence behavior of the original GD-RTRL algorithm on the Henon map time series problem

Table 3 shows the mean square error and the percentage number of weights with low sensitivity values. It was found that the iTPA-RTRL algorithm gave improved generalization relative to the GD-RTRL algorithm. The TPA-RTRL algorithm gave the worst performance. Generalization was found to be independent of the size of the network. It was also found that the iTPA-RTRL and GD-RTRL algorithms produced networks with the smallest number of weights with low sensitivity values. The results suggest that training networks with weight regularisation is better than training without weight regularisation. Fig 5 and 6 show the fit for the iTPA-RTRL and GD-RTRL algorithms using an FRNN with 12 processing units after 50,000 time steps. Clearly the fit of the iTPA-RTRL algorithm is very good despite the type of the input data used.

TABLE III. FINAL ERROR AND % NUMBER OF WEIGHTS WITH LOW SENSITIVITY VALUES IN DIFFERENT SIZE NETWORKS FOR THE CONTINUOUS STIRRED TANK

	Units = 6		Units = 9		Units = 12	
	MSE x10^2	n %	MSE x10^2	n %	MSE x10^2	n %
iTPA-RTRL	0.037	20.87	0.035	9.97	0.036	7.05
TPA-RTRL	0.054	21.69	0.055	13.27	0.054	9.25
GD-RTRL	0.041	17.75	0.040	9.58	0.043	7.96

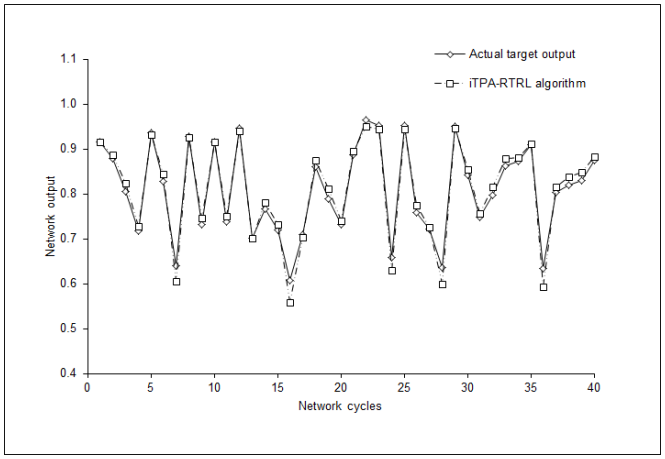

Fig. 5. Typical convergence behavior of the new iTPA-RTRL algorithm on the continuous stirred tank problem

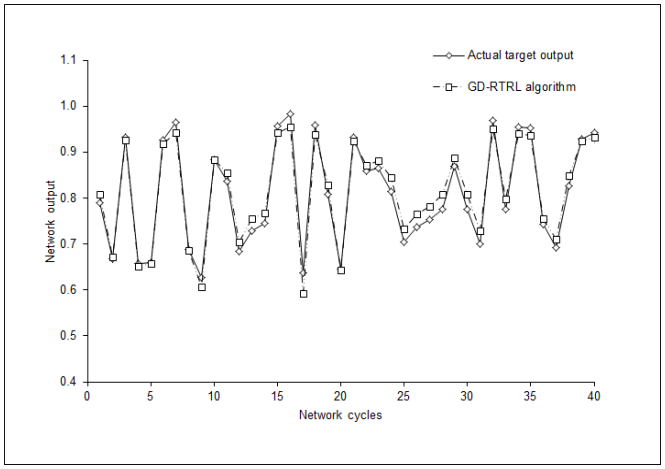

Fig. 6. Typical convergence behavior of the original GD-RTRL algorithm on the continuous stirred tank problem

Non-linear dynamic plant. The final test is a discrete-time nonlinear system introduced by Narendra and Parthasarathy [22]. Once again the network used was a single layer FRNN with the processing units varied according to [8, 21, 26]. The error metric used was the mean square error obtained by averaging the square error over intervals of 100 time steps. Network training was terminated after 50,000 time steps trained. Weight sensitivity values based on t_{ji} statistic were computed during the last 100 time steps of the training phase.

Table 4 shows the mean square error and percentage number of weights with low sensitivity values. It was found that the iTPA-RTRL algorithm gave good generalization performance across a range of network sizes. The GD-RTRL algorithm gave the worst performance. Generalization performance was found to deteriorate with the size of the network. This was particularly noticeable in networks trained by the TPA-RTRL algorithm. It was also found that the iTPA-RTRL algorithm produced networks with fewer redundant weights compared with the original algorithm. The results suggest that using a weight elimination procedure during training is better than not using it at all. Fig 7 and 8 show the fit for the iTPA-RTRL and GD-RTRL algorithms using an FRNN with 12 processing units after 50,000 time steps. Clearly the fit of both algorithms is very good considering the input data used (high spectral content).

TABLE IV. FINAL ERROR AND % NUMBER OF WEIGHTS WITH LOW SENSITIVITY VALUES IN DIFFERENT SIZE NETWORKS FOR THE NON-LINEAR DYNAMIC PLANT

	Units = 6		Units = 9		Units = 12	
	MSE x10^2	n%	MSE x10^2	n%	MSE x10^2	n%
iTPA-RTRL	4.39	14.33	4.34	7.85	4.75	6.15
TPA-RTRL	4.33	15.42	4.35	9.78	4.85	7.74
GD-RTRL	4.65	13.11	4.88	7.99	4.74	6.02

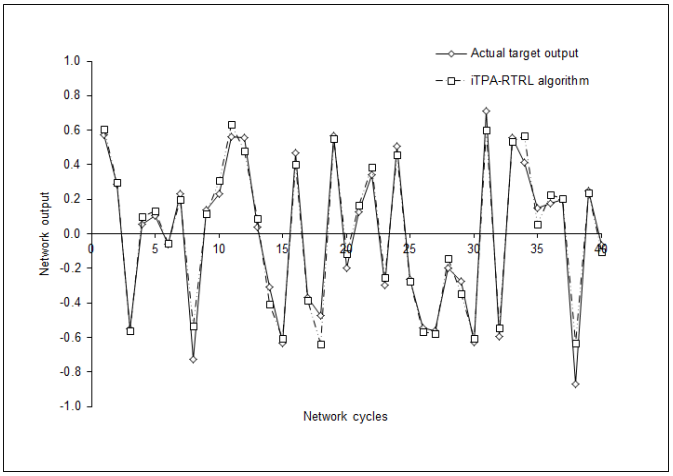

Fig. 7. Typical convergence behavior of the new iTPA-RTRL algorithm on the non-linear dynamic plant problem

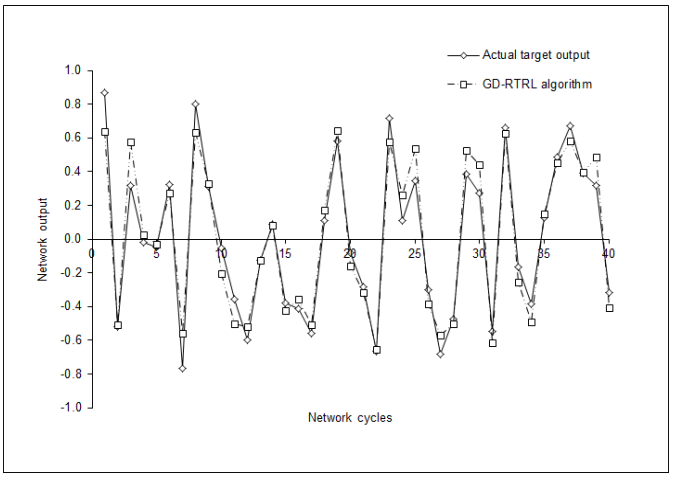

Fig. 8. Typical convergence behavior of the original GD-RTRL algorithm on the non-linear dynamic plant problem

VI. COMPARISON OF THE DIFFERENT ALGORITHMS

In order to determine whether the difference in the results is statistically significant, we perform some hypothesis tests. The test used was a standard t-test with the sample of test errors from the iTPA-RTRL algorithm compared with the corresponding sample from the original TPA-RTRL algorithm for each dataset used in the study. A second test was carried

out by comparing these test results with the GD-RTRL algorithm on the same set of problems. For the correct application of the t-test, it was necessary to take the logarithm of the test errors (since the test errors have log-normal distribution) and remove any outliers, following the same procedure in [19]. The resulting samples were tested for normality using the Kolmogorov-Smirnov test.

TABLE V. RESULTS OF A T-TEST COMPARING THE MEAN TEST ERRORS OF THE DIFFERENT ALGORITHMS

Problem	Processing units	O	I	(a)	(b)	(c)
Henon map	9	1	1	L 3.54	L 2.33	-
Stirred tank	9	1	2	L 11.52	L 5.73	G 6.20
Non-linear plant	9	1	2	-	-	-

Note: The entries show differences that are statistically significant on a 10% level and dashes mean no significance found. Column (a): iTPA-RTRL ("L") vs. TPA-RTRL ("T"). Column (b): iTPA-RTRL vs. GD-RTRL ("G"). Column. (c): TPA-RTRL vs. GD-RTRL

The results are tabulated in Table 5. Dashes mean differences that are not significant at the 10% level i.e. the probability that the differences are purely accidental. Other entries indicate the superior algorithm (e.g. new iTPA-RTRL algorithm - L, TPA-RTRL algorithm – T, GD-RTRL algorithm - G), and the value of the t statistic. Column (a) gives a comparison between the new iTPA-RTRL algorithm and the TPA-RTRL algorithm. The results show two times L is better (Henon map and continuous stirred tank) and once no statistical difference (non-linear dynamic plant). This suggests that training using weight elimination is better than training with no weight regularisation at all. Column (b) and (c) give comparisons between the new iTPA-RTRL and original TPA-RTRL algorithms, and the GD-RTRL algorithm. The results show three times no statistical difference, twice L is better and once G is better. This suggests that the generalization performance of the new iTPA-RTRL algorithm is superior, and that training RNN using weight elimination or weight decay is better than training with none at all, which is the situation with the TPA-RTRL algorithm.

VII. CONCLUSIONS

A new variant of the tangent plane algorithm referred to as iTPA-RTRL is proposed for online training of recurrent neural networks. This algorithm automatically adjusts the step size by approaching tangent planes to constraint surfaces. A weight elimination vector is projected onto the tangent plane with the expectation that the algorithm will prune superfluous weights from the network without causing much disturbance during network training. The iTPA-RTRL algorithm requires two parameter to set manually; the angle parameter β and the weight sensitivity parameter w_a. Small values of w_a which implement a weight elimination procedure are preferred in large network structures. Increasing the value of $w_a > 1.0$ has a deleterious effect on generalization and produces slower convergence.

Comparative tests were carried out using the new iTPA-RTRL and TPA-RTRL algorithms and the GD-RTRL algorithm with weight decay. The neural network benchmark datasets used were the Henon map [23], the continuous stirred tank [24] and the non-linear dynamic plant [25].

The results show that the iTPA-RTRL algorithm was two times better (Henon map and continuous stirred tank) than the TPA-RTRL algorithm, and two times better (Henon map and continuous stirred tank) than the GD-RTRL algorithm with weight decay. It was also found that the iTPA-RTRL algorithm pruned the smallest percentage of weights from the network. This result suggests that a weight elimination strategy is an effective method for discriminating between active and inactive weights and actually results in better generalization performance.

VIII. FUTURE WORK

This paper shows that the newly developed improved tangent plane algorithm for recurrent neural networks gives improved generalization performance relative to the gradient descent real time recurrent learning algorithm with weight decay. In situations where time varying signals are required, such as grammatical inference or process control modelling, the sequential learning ability of the improved tangent plane algorithm might be the preferred method.

REFERENCES

[1] Shaohua Tanb, Ah Chung Tsoi, "Recurrent neural networks: a constructive algorithm and its properties" , Neurocomputing, vol. 15, 1997, pp. 309 - 326

[2] Subrahmanya, N., Yung, C. Shin, "Constructive training of recurrent neural networks using hybrid optimization", Neurocomputing, vol. 73, 2010, pp. 2624 – 2631

[3] Puma-Villanuevaa, W.J., Santos, E.P., Zuben, F.J., "A constructive algorithm to synthesize connected feedforward neural networks", Neurocomputing, vol. 75, 2012, pp. 14 – 32

[4] Giles, C.L., Omlin, C.W., "Pruning recurrent neural networks for improved generalization performance", IEEE transactions on neural networks, 1992, vol. 5, no. 5, pp. 848

[5] Chi-Sing Leung, Lai-Wan Chan, "Dual extended Kalman filtering in recurrent neural network," Neural Networks, vol. 16, 2003, pp. 223-239

[6] Ahmed, S.U., Shahjahan, M.D., Kazuyuki, M., "A Limpel-Ziv complexity based neural network pruning algorithm," *Int. J. Neur. Syst.* 2011 (5), pp. 427 – 441

[7] May, P, Zhou, E., and Lee, C.W., "A comprehensive evaluation of weight growth and weight elimination methods using the tangent plane algorithm", *Advanced computer science and applications*, 2013, vol. 4. no. 6

[8] P.M. Williams, "Baysian regularisation and pruning using a Laplacian prior," *Technical report,* (312), 1994

[9] P.O. Hoyer, "Non-negative matrix factorisation with sparseness constraints," *Journal of machine learning research,* (5): 1457 – 1469. 2004

[10] Jim, K., K., Giles, C.L., Horne, B.G., "Synaptic noise in dynamically driven recurrent neural networks: convergence and generalization", Technical report, UMIACS-TR-94-89 and CS-TR-3322 Institute for advanced computer science, 1994, University of Maryland, College Park

[11] Hirasawa, K., Kim, S., Hu, J., Han, M., and Jin, C., "Improvements of generalization ability for identifying dynamical systems by using universal learning networks", *Neural Networks*, vol 14 2001, pp1389 - 1404

[12] Giles, C.L, Lawrence, S, Ah Chung Tsio, "Noisy time series prediction using recurrent neural networks and grammatical inference", *Journal of machine learning*, (44), 2001, pp 161-183

[13] May, P, Zhou, E., and Lee, C.W, "Learning in fully recurrent neural networks by approaching tangent planes to constraint surfaces", *Neural Networks*, vol 34, 2012, pp. 72-79.

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

[15] Mozer, M. C, and Smolensky, P., "Skeletonisation: a technique for trimming the fat from a neural network via relevance assessment", *Advances in neural information processing*, 1989, (1), pp. 107-115

[16] Karnin, E., "A simple procedure for pruning back-propagation trained neural networks", *IEEE trans on neural networks*, 1990, vol. 1, no. 2, pp. 329 – 242

[17] B. Hassibi, and D.G. Stork, "Second order derivatives for network pruning," *Advances in neural information processing systems*, vol.5, 1993, pp. 164-171.

[18] Y.L, LeCun, J.S., Denker,and S.A. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol.2, 1990, pp. 598-605.

[19] L. Prechelt, "Connection pruning with static and adaptive pruning schedules," *Neurocomputing*, Volume 16, Issue 1, 1997, pp. 49-61

[20] Mak, M.W., Ku, K.W., and Lu, Y.L., "On the improvement of the real time recurrent learning algorithm for recurrent neural networks", *Neurocomputing*, 1999, vol 24, issues 1-3, pp. 13-36.

[21] Hernandez, E., and Arkun, T., "Stability of non-linear polynomial ARMA models and their inverse", *International journal of control*, 1996, 63, 885-906

[22] K.S Narandra, and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE transactions on neural networks*, 1 (1), 4 1990.