

01 Jan 2007

# Training Winner-Take-All Simultaneous Recurrent Neural Networks

Xindi Cai

Danil V. Prokhorov

Donald C. Wunsch
*Missouri University of Science and Technology*, dwunsch@mst.edu



## Recommended Citation

X. Cai et al., "Training Winner-Take-All Simultaneous Recurrent Neural Networks," *IEEE Transactions on Neural Networks*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2007.
The definitive version is available at https://doi.org/10.1109/TNN.2007.891685

# Training Winner-Take-All Simultaneous Recurrent Neural Networks

Xindi Cai, *Member, IEEE*, Danil V. Prokhorov, *Senior Member, IEEE*, and Donald C. Wunsch II, *Fellow, IEEE*

***Abstract*—The winner-take-all (WTA) network is useful in database management, very large scale integration (VLSI) design, and digital processing. The synthesis procedure of WTA on single-layer fully connected architecture with sigmoid transfer function is still not fully explored. We discuss the use of simultaneous recurrent networks (SRNs) trained by Kalman filter algorithms for the task of finding the maximum among $N$ numbers. The simulation demonstrates the effectiveness of our training approach under conditions of a shared-weight SRN architecture. A more general SRN also succeeds in solving a real classification application on car engine data.**

***Index Terms*—Backpropagation through time (BPTT), extended Kalman filter (EKF), simultaneous recurrent network (SRN), winner-take-all (WTA).**

## I. Introduction

THE simultaneous (or settling) recurrent network (SRN) is designed to approximate static mappings. (See Fig. 1.) State variables of such a recurrent network are initialized, and the network as a dynamic system is iterated (usually) until its outputs converge. Thus, inputs to the network become mapped to certain outputs, thereby enabling the network to approximate mappings which are more complex than static mappings approximated with conventional feedforward networks [9]. Moreover, other settling network architectures such as the Hopfield network, applied to static optimization tasks, specify all the weights and connections in advance based on the analysis of various energy functions [16]. We contend that, especially for large scale problems, it would be useful to be able to train the weights, rather than obtain them via application of an analytical process. This theoretical consideration is among the motivators for our work.

Backpropagation through time (BPTT) resolves the recurrency by unfolding the temporal operation of the network into a layered feedforward network. Well-organized BPTT can handle more general architecture efficiently and homogenously than other recurrent methods [11]. In literature, BPTT is mistakenly interpreted by many as the learning algorithm. In this paper, it is actually just the method of obtaining temporal derivatives, whereas the extended Kalman filter (EKF) algorithm is the weight update method utilizing these derivatives [20]. The EKF method is formulated in terms of state-space concepts, providing efficient utilization of the information contained in the input data [29]. Here, we train a shared weight version of the SRN. The truncated BPTT with multistream EKF is employed. Very good results are also obtained with the nonlinear Kalman filter algorithm. In addition, we verify that the SRN can be used in a practical engine diagnostic application.

In Section II, we discuss the relationship of trained SRNs to related ideas for WTA, such as MAXNET and the Hopfield network. In Section III, we define the SRN and training algorithm. In Section IV, we describe the shared-weight SRN. We illustrate our simulation results in Section V with conclusions in Section VI.

## II. Related Work

The winner-take-all (WTA) network has played an important role in the design of learning algorithms; in particular, most unsupervised learning algorithms such as competitive learning, self-organizing map and adaptive resonance theory—and has become the fundamental building block of many complex systems. Several methods have been proposed to implement the WTA: MAXNET [1], a discrete-time model, was proposed in order to implement the Hamming network. Several models based on crossbar topology exist [2]–[4]. In very large scale integration (VLSI), a series of compact complementary metal–oxide–semiconductor (CMOS) integrated circuits and cellular-neural-network-based architectures were designed for realizing the WTA function [5]–[8]. While architectures such as the MAXNET and the Hopfield network have been extensively studied in literature, little attention has been given to the issue of training such networks for various problems. Generally, the MAXNET and the Hopfield network are constructed, rather than trained. The simultaneous recurrent networks (SRNs), however, tune weights to learn an optimal iterative approximation to a function by minimizing error. By analyzing the transient states of the dynamic system, the SRN tries to estimate the general functional relationship, rather than building a match-up table based on similarity to previous examples [9]. Research on intelligent control [22] shows that iterative relaxation SRN possesses is crucial in those tasks, where feedforward networks do not appear powerful enough. To the best of our knowledge, there is little existing technique to train

Manuscript received September 26, 2005; revised June 25, 2006 and September 29, 2006; accepted October 4, 2006. This work was supported in part by the National Science Foundation and the M. K. Finley Missouri endowment.

X. Cai is with the American Power Conversion Corporation, O'Fallon, MO 63368 USA (e-mail: xindi.cai@apc.com).

D. V. Prokhorov is with Toyota Technical Center—USA, Ann Arbor, MI 48105 USA (e-mail: dvprokhorov@gmail.com).

D. C. Wunsch II is with the Applied Computational Intelligence Laboratory (ACIL), Department of Electrical and Computer Engineering, The University of Missouri—Rolla, Rolla, MO 65409 USA (e-mail: dwunsch@ece.umr.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2007.891685

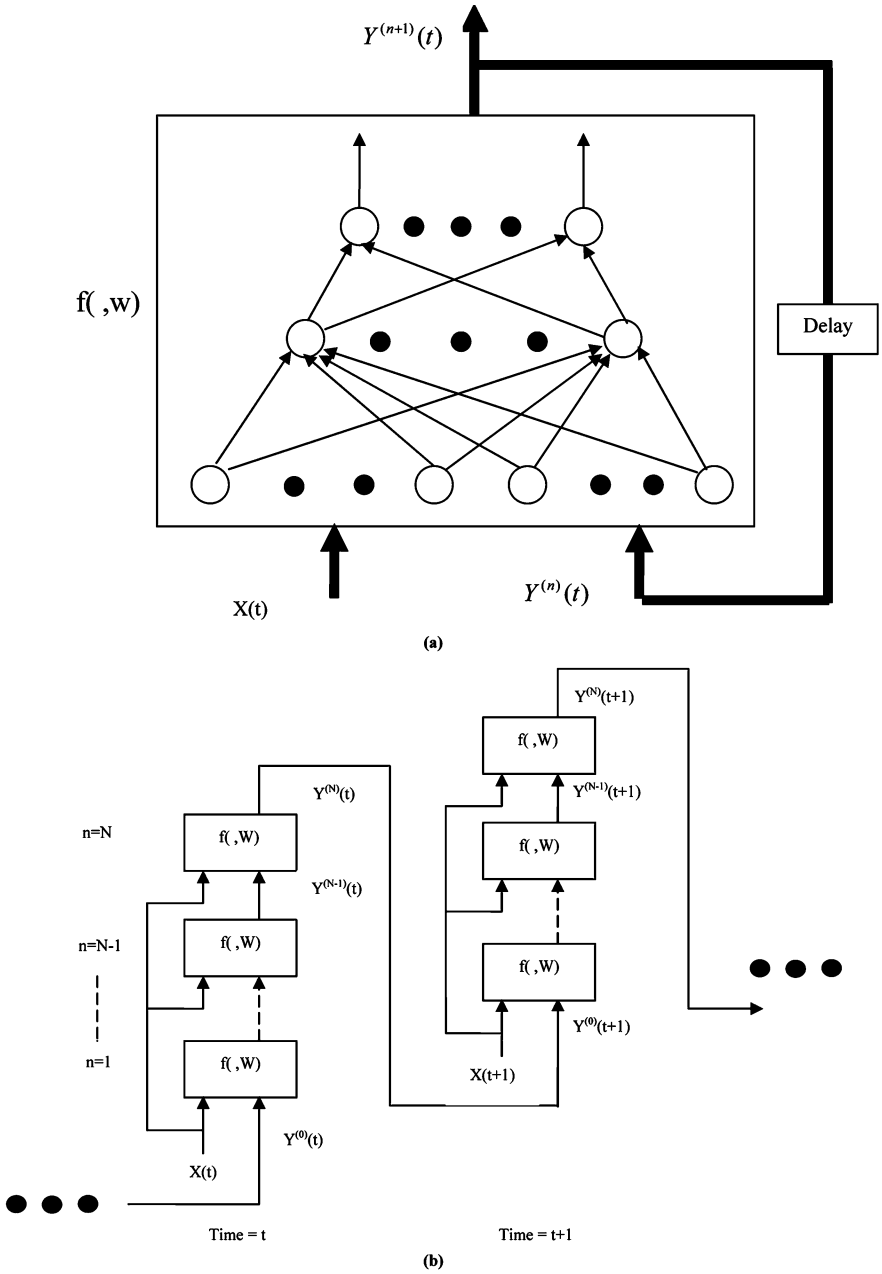



Fig. 1. Architecture of SRN implementing (1). This particular system assumes that only explicit outputs of the network are fed back as additional inputs (output feedback), but other SRN architectures are possible, e.g., those which employ state feedback. In our WTA problem, $f$ is defined as a single-layer fully connected network with a bipolar sigmoid transfer function $S$ [see (4b)]. $Y^{(n)}(t)$ is the transient output vector at time $t$ with $Y^{(0)}(t)$ equal to the previous output vector $Y(t-1)$. (a) General format of SRN. (b) Flow chart of unfolded SRN, in both temporal and spatial directions.

SRN for WTA in the literature. Therefore, study of training methods for SRN, through the WTA problem, will help us to extend SRN to less tractable problems in the future.

The combination of BPTT and EKF algorithms is studied for training weights in WTA networks with a smooth, nonlinear transfer function. We are given $N$ real numbers, $x_1$ to $x_N, N > 1$, and wish to determine the maximum. The competition in an $N$-output network is supposed to result in only one positive output to denote the largest component, the rest of the outputs being negative.

Most of the work on WTA is based on the saturation transfer function $\mathrm{sat}()$. In such a system, the "winning" component usually has the value of "1", while all the "losing" components have the value of "$-1$". Let $\mathbf{P}^N = \{\mathbf{x} \in \mathbf{Z}^N : x_i = 1 \text{ or } -1, i = 1, \ldots, N\}$ and $\mathbf{C}(\boldsymbol{\alpha}) = \{\mathbf{x} \in \mathbf{R}^N : x_i \alpha_i > 1,\ i = 1, \ldots, N\}$ for $\boldsymbol{\alpha} = [\alpha_1, \alpha_{2,\ldots,} \alpha_N]^T \in \mathbf{P}^N$. In an $N$-neuron WTA system, $\boldsymbol{\alpha}^k = [\alpha_1^k, \alpha_{2,}^k, \ldots, \alpha_N^k]^T \in \mathbf{P}^N$, where $\alpha_k^k = 1, \alpha_j^k = -1, \forall j \neq k,\ k = 1, \ldots, N$, is the desired output. Due to the properties of the saturation function, the equilibrium of the system must be in the region of $\mathbf{C}(\boldsymbol{\alpha})$ [18]. We interpret the equilibrium the same way as it is done in the Hopfield network literature, namely, the fixed point at which the neural activations arrive during their evolution in time.

To illustrate the challenges involved in even a simple SRN architectures, consider the performance of MAXNET with a saturation transfer function. The single-layer fully connected

MAXNET, with the following connection weight matrix, can be described as:

$$W_{ij} = \begin{cases} 1, & i = j \\ -\varepsilon, & i \neq j \end{cases} \qquad 0 < \varepsilon < 1/N, \quad 1 \le i, \quad j \le N$$

$$X_i(t+1) = \mathrm{Sat}\left(\sum_{j=1}^{N} W_{ij} X_j(t) + B_i\right), \qquad 1 \le i \le N.$$

Such a network must have a nonzero bias to converge to the desired output $\boldsymbol{\alpha}^k$. Also, the MAXNET cannot achieve the desired output $\alpha^k$ in a sparsely connected situation, i.e., when some of the nondiagonal elements of the weight matrix are zero. In [18], Michel *et al.* proposed a synthesis procedure for the sparsely connected situation. However, [19] shows that the synthesis procedure results in a trivial solution for the WTA problem since the desired outputs $\alpha^k$ are linearly independent. Furthermore, there is a lack of procedure for constructing or training networks with a sigmoid transfer function, the most frequently used one, for the WTA problem.

## III. SRN and Training Algorithm

### A. SRN

SRN is an architecture used for function approximation problems [9], [10]. The SRN uses recurrence to approximate a *static* deterministic mapping $X(t) \to Y(t)$ (Fig. 1). This mapping is computed by iterating the following:

$$Y^{(n+1)}(t) = f\left(Y^{(n)}(t), X(t), W\right) \tag{1}$$

where $f$ is any feedforward network or system, and $Y(t)$ is defined as

$$Y(t) = \lim_{n\to\infty} Y^{(n)}(t). \tag{2}$$

The recurrence in SRN is invoked at each time $t$, but it is not visible from the outside of the network. In applications, the number $Y^{(n+1)}(t) = f(Y^{(n)}(t), X(t), W)$ of iterations $n$ can be limited to a reasonably large number $N$.

### B. BPTT

BPTT can be viewed as unfolding a recurrent network from a time evolving architecture into its multilayer counterpart, thereby translating time into space. The relevant history of input data and network state is saved only for a fixed number of time steps, defined as the truncation depth and denoted as $h$. With the truncated depth $h$ well selected, BPTT [11] produces accurate derivatives with reduced complexity and computational expense than real-time recurrent learning. In BPTT, computing the derivatives of error with respect to weights, in a recurrent network, is reduced to computing the derivatives in each layer, in a feedforward network, and adding them [12]. Derived in reverse order of forward propagation, the ordered derivatives are appropriately distributed, using the chain rule, from a given node to all nodes and weights that connect it in the forward direction [13]. Efficient implementations of BPTT($h$) are described in [14], [20].

### C. EKF

The EKF [14], [20] is used for learning the weights of an SRN. At the $n$th time step, the algorithm of EKF to train the network with $m$ tunable parameters and $l$ outputs is performed by the following recursion:

$$\begin{aligned} A(n) &= [(\eta(n)I(n))^{-1} + H(n)^T P(n) H(n)]^{-1} \\ K(n) &= P(n)H(n)A(n) \\ \hat{w}(n+1) &= \hat{w}(n) + K(n)\xi(n) \\ P(n+1) &= P(n) - K(n)H(n)^T P(n) + Q(n). \end{aligned}$$

In the recursion, $I(n)$ is a $l \times l$ user-specified nonnegative definite weighting matrix usually chosen to be the identity matrix, $\eta(n)$ is the learning rate usually chosen between $10^{-4}$ and 1, and $\xi(n)$ is the $l \times 1$ error vector, which represents the difference between the desired and actual outputs. The network's trainable weights are arranged into an $m \times 1$ vector $\hat{w}(n)$ (where $\hat{w}(0)$ has small random values, e.g., of magnitude less than 0.1). The $m \times m$ approximate error covariance matrix $P(n)$ is used to model the correlations or interactions between each pair of weights in the network. $P(0)$ is initialized as a diagonal matrix with large values (e.g., magnitude around 100). The $m \times l$ matrix $H(n)$ forms the derivatives of a function of the network's outputs with respect to its weights. (The derivatives of one output with respect to all $m$ weights are listed in one column.) Finally, $Q(n)$ is an $m \times m$ diagonal covariance matrix that provides a mechanism by which the effect of artificial process noise is included in the Kalman recursion to avoid poor local minima and divergence. $Q(0)$ is usually set to the identity matrix with a scaling factor chosen between $10^{-4}$ and $10^{-8}$ (these values are problem-dependent, but typical).

The multistream EKF is designed to update the weights simultaneously satisfying the demands from multiple input–output pairs [14]. The dimensionality of the derivative matrix $H$ is increased proportionally to the number of streams, and the corresponding columns in $H$ denote the derivatives of outputs in different streams with respect to the trainable weights. Similarly, the dimensionality of the error vector $\xi(n)$ is also proportionally increased. The rest is left unchanged as in the single-stream EKF. The only extra computational cost is the inversion of the $A$ matrix.

For the recurrent network, computing dynamic derivatives [23] requires backpropagation of partial derivatives combined with dynamic derivatives from previous steps. BPTT computes dynamic derivatives which consist of the $H$ matrix in EKF. Using these derivatives, EKF is applied for the weights update. For discussions on convergence of gradient-based algorithms and stability of recurrent network, the reader is referred to [12], [13], and [24]–[28].

## IV. Shared-Weight Network Model

We consider an $N$-neuron fully connected WTA network. Its system dynamic can be defined as the following SRN network:

$$\begin{aligned} Y^{(n+1)}(t) &= S\left(WY^{(n)}(t) + B\right) \\ Y(t) &= \lim_{n\to\infty} Y^{(n)}(t) \end{aligned} \tag{3}$$

where $Y(t) \in R^N$ is the converged output vector of the network at time $t$, and $Y^{(n)}(t)$ is the transient output vector at time $t$ with $Y^{(0)}(t)$ equal to the input vector $X(t)$, and $X(t) \in R^N$. The variable "$n$" is an iteration variable for the SRN, while $t$ is the time step governing presentations of external inputs. (In practice, the sequence $\{Y^{(0)}(t), Y^{(1)}(t), \ldots, Y^{(n)}(t)\}$ is assumed converged if $Y^{(n)}(t)$ reaches a limit $Y(t)$ for sufficiently large $n$.) $W$ is a real $N \times N$ matrix, $B$ is a real $N \times 1$ vector of constant bias values, and the real $N$-dimensional vector-valued function $S(x)$ can be one of the following: 1) saturation function in (4a) or 2) bipolar sigmoid function in (4b)

$$s_i(x_i) = \mathrm{sat}(x_i) = \begin{cases} +1, & \text{if } x_i > +1 \\ -1, & \text{if } x_i < -1 \\ x_i, & \text{otherwise} \end{cases} \tag{4a}$$

$$s_i(x_i) = \tanh(\gamma x_i) = \frac{e^{\gamma x_i} - e^{-\gamma x_i}}{e^{\gamma x_i} + e^{-\gamma x_i}} \tag{4b}$$

where $s_i(x_i)$ is the $i$th component of $S(x)$.

We would like to illustrate our SRN-WTA training procedure for the task of finding the maximum among $N$ numbers. [Numbers are required to be different only for the purpose of unique winner. Generally, two or more identical numbers, e.g., (0.8 0.7 0.7 0.6) for $N = 4$, will not affect the final output of the model.] A shared-weight SRN is more suitable for this problem due to invariance with respect to the position of the maximum. Similar with the MAXNET model, we assume that our SRN weights and biases are defined as follows:

$$\begin{bmatrix} 1 & -p & \cdots & -p \\ -p & 1 & \cdots & -p \\ \vdots & \vdots & \ddots & \vdots \\ -p & -p & \cdots & 1 \end{bmatrix} \quad \begin{bmatrix} B \\ B \\ \vdots \\ B \end{bmatrix} \tag{5}$$

where $p$ is a positive constant and $B$ is a constant bias. The initial values of $p$ and $B$ can be randomly selected in the range of $[-1, 1]$. From (A2), we know that there exist some equilibrium points within the hypercube if $p$ meets certain conditions (see Appendix). The inputs of neurons at iteration $n$ are defined as

$$\mathrm{net}_k = y_k^{(n)} - p \cdot \sum_{j \neq k} y_j^{(n)} + B, y_k^{(0)} = x_k, \quad \text{for } k = 1, \ldots, N. \tag{6}$$

Clearly, the $\mathrm{net}_k$, for $k = 1, \ldots, N$, keeps the same order, in terms of magnitude, as those of $y_k^{(n)}$. The sigmoid transfer function is a monotonically increasing function. Therefore, the outputs of the active neurons, i.e., $y_k^{(n+1)}$, also keep the same order, which is set by the original inputs $x_k$.

The shared-weight model can be described using a network connectivity table similar to that in [20]. The network is represented by a combination of summation and product nodes. In the network description, outputs of two *bias* nodes with weights $p$ and $B$ are distributed.

The desired equilibrium point is the point which has only one component of the output vector $Y$ at its maximum ($y_{\max} > 0$) whereas all other components should be at their minima. Assuming the same negative minimum for all such components ($y_{\min} < 0$) and some values for $p$ and $B$, we can compute coordinates of the candidate equilibrium point for our shared-weight SRN by solving the following system of equations (cf. $C(\alpha)$ in Section II):

$$\begin{cases} y_{\max} = \tanh(\gamma(y_{\max} - p(N-1)y_{\min} + B)) \\ y_{\min} = \tanh(\gamma(y_{\min}(1 - p(N-2)) - py_{\max} + B)) \end{cases}. \tag{7}$$

After solving (7), one also has to check whether the SRN linearization at the equilibrium point $(y_{\max}, y_{\min}, \ldots, y_{\min})$ (the Jacobian of the SRN at this point), where location of the maximum $y_{\max}$ is arbitrary between 1 and $N$, is locally stable. This is equivalent to making sure that all eigenvalues of the matrix product as in (A1) are within the unit circle. Unfortunately, the ability to obtain a solution of (7) is not sufficient because it does not provide a recipe for constructing the desired SRN. We resort to training an SRN for this task.

## V. Experimental Results

### A. Experiment on WTA Problem

Simulations were run on SRN for the WTA problem with various numbers of neurons $N$. The training data are generated by a random uniform distribution in $[0, 1]$. The data are divided into three sets. The easy set contains training samples that satisfy two conditions

$$x_L = \max[x_1 \ldots x_N]^T \text{ and } x_L - x_i \geqslant 0.2, \qquad \text{for } L \neq i.$$

The tough set consists of data that satisfy

$$x_L = \max[x_1 \cdots x_N]^T \text{ and } 0 < x_L - x_i \leqslant 0.2, \qquad \text{for } L \neq i.$$

The random set is produced from the random uniform distribution without any limitations. Another random set is also generated as the test set. All previous training sets are composed of $N$ streams of 600 vectors each. Each stream denotes a pattern of the maximum index, i.e., the $i$th stream has $x_i = \max[x_1, \ldots, x_N]^T$ with the target $y(x_i) = 1$ and the target for the rest of the outputs $= -1$. The maximum number of iterations of SRN is 50 [instead of $\infty$ in $n \to \infty$ in (3)]. In (4b), $\gamma$ represents the slope of the bipolar sigmoid function at the point where the sigmoid's total input is zero. Large $\gamma$, i.e., $\gamma > 1$, amplifies the differences among the inputs and facilitates the classification, but expands the saturation region (small $\gamma$, i.e., $0 < \gamma < 1$, does the opposite). During the training process, $\gamma$ is fixed as a constant so that it will not affect the convergence. We choose $\gamma = 1.2$, rather than 1, because it improved the training process.

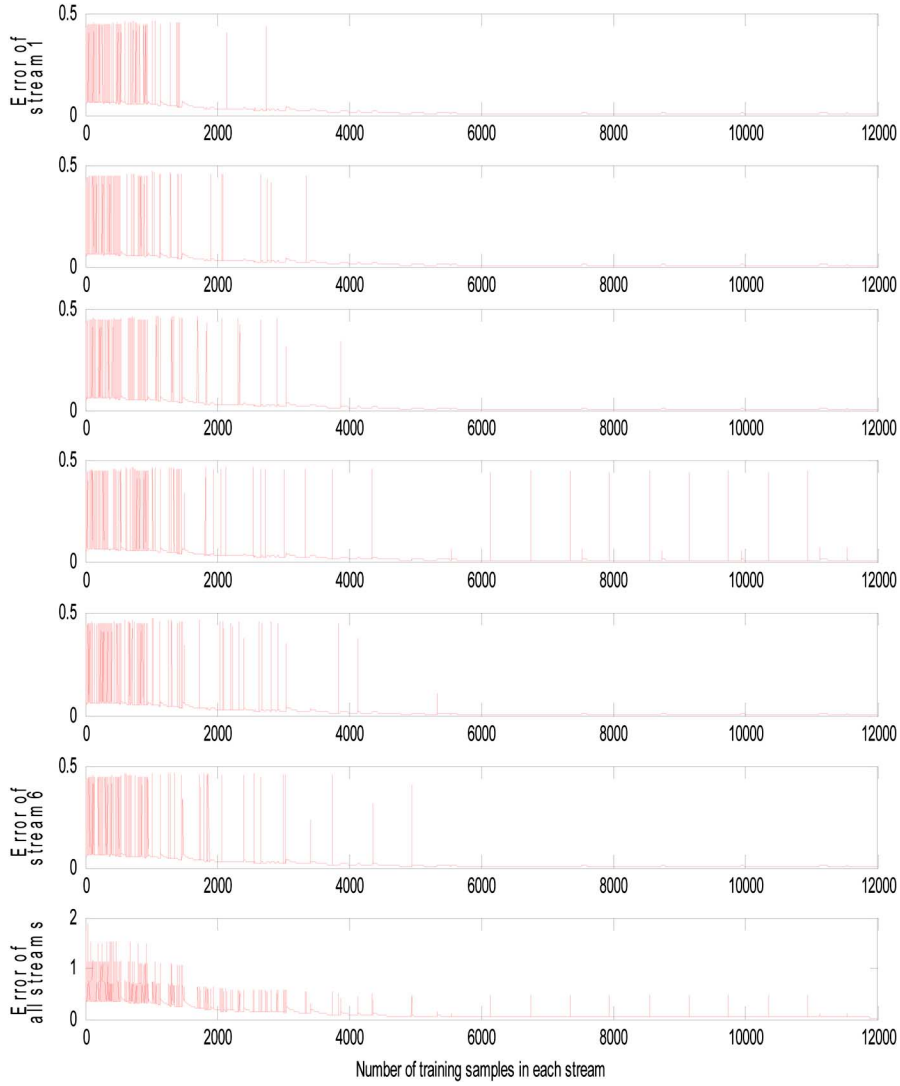



Fig. 2. Example of error plots for all 20 epochs of training our SRN for WTA problem on $N = 6$. (The first six panels from top to bottom are rmse of individual streams.)

Random initial weights work well for this task, although we used the technique of [18] for choosing initial values. During one epoch of training, all $N$ streams of inputs are presented to the network, with updates of the network weights ($p$ and $B$) being carried out by the $N$-stream EKF algorithm. The derivatives of the network outputs are computed according to the BPTT($h$) algorithm with $h = 50$. One epoch of training corresponds to $N \times 600$ updates of weights. The network is first trained on the easy set to find a near-optimal solution (for five epochs), and then, the tough set is used to help the network classify close inputs in order to refine the near-optimal solution obtained from previous training samples (ten additional epochs). Finally, the random set improves the generalization ability and helps the network avoid local minima (additional five epochs). Thus, we train a total of 20 epochs. The training on each set is terminated when the error reaches a predefined threshold (see Fig. 2). The error of each stream is not approaching zero but a small constant (see Fig. 3), because the targets are not the real equilibrium points of the network outputs, which are not available at the beginning of the training. We choose targets that provide sufficient margins between the winner and losers [e.g., output of winner neuron $\approx 0.95$ and that of loser $\approx -0.86$; so our $\xi(n) = [0.95, -0.86, \ldots, -0.86] - Y(n)$].

The algorithm has found some near-optimal solutions for $N = 4$ and $N = 6$. In most of the misclassified cases, the difference between the maximum and the second maximum is less than 1% of the maximum, suggesting that the trained network still cannot distinguish between two (or more) nearly identical input numbers. In the $N = 6$ case, a solution of 99.7% accuracy (i.e., the percentage of correctly recognized test inputs) is found on 3600 randomly generated inputs (see Table I). Among the misclassified cases, the difference between the maximum and the second maximum is less than 0.1% of the maximum. (In such situations, when the network misclassifies, it usually outputs two positive values for both the maximum and the second maximum.) Some other solutions are also more than 99% accurate, as checked on a large number, e.g., 3600, of testing inputs [see Table II; the equilibrium point has only one positive component corresponding to the maximum location, and the rest of components are as specified and confirmed

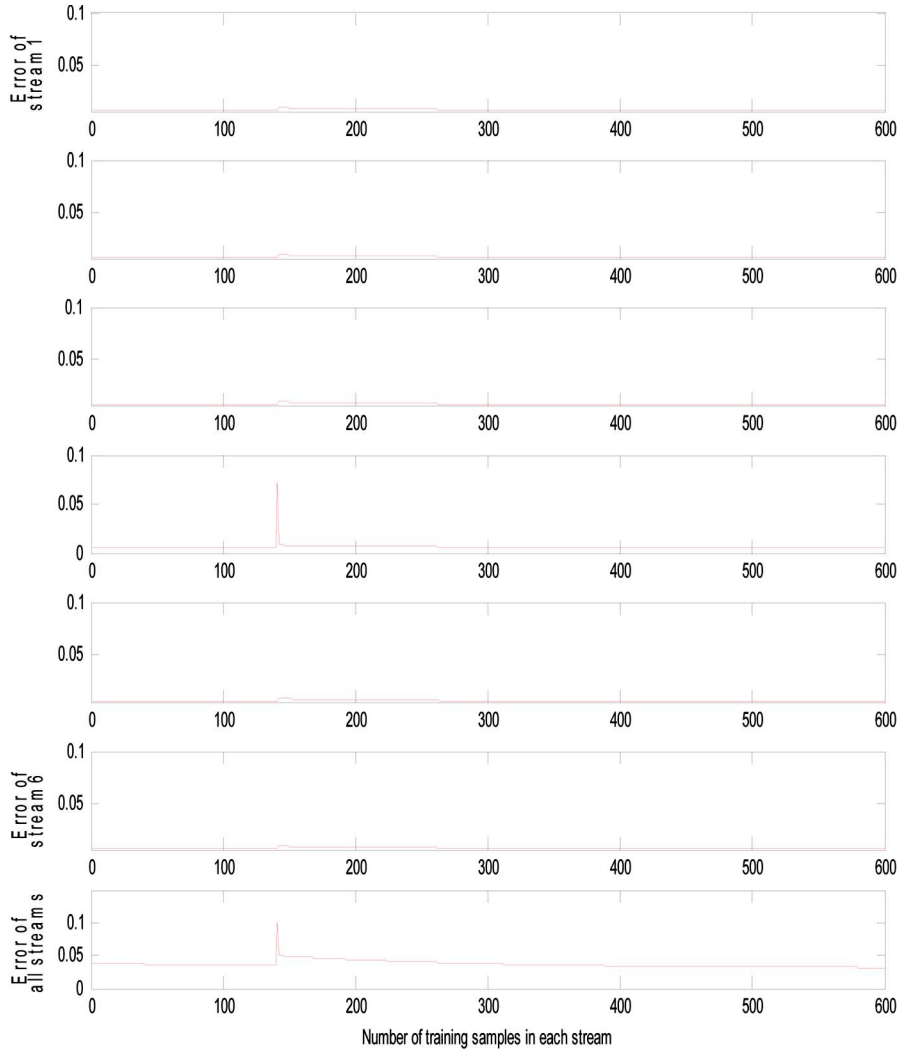



Fig. 3. Error plots of the last epoch of training our SRN for WTA problem on $N = 6$ (the last 600-sample-long segment of Fig. 2).

TABLE I
BEST RESULT OF WTA PROBLEM ON $N = 6$ (STREAM I CONSISTS OF VECTORS THAT THE $i$th ELEMENT IS THE MAXIMUM. ACCURACY IS BASED ON 600 SAMPLE VECTORS PER STREAM)

| Stream i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Number of misclassified | 2 | 3 | 2 | 2 | 0 | 3 |
| Accuracy (%) | 99.7 | 99.5 | 99.7 | 99.7 | 100 | 99.5 |

by solving the system (7)]. This demonstrates that the model produces many high-quality solutions.

Further experiments using the method described in [21] allow use of a single stream of data from the random set only. We train the shared-weight SRN [limiting the number of iterations $n$ in (3) to 100] in one pass through 10 000 samples and test on an independently generated test set of 100 000 samples, confirming the same high accuracy described previously. Furthermore, we enable training the self-feedback weight $\mathrm{Wd}$. [The diagonal element of the matrix in (5) is a constant value of 1, but we call it $\mathrm{Wd}$ and allow its training in this set of experiments.] This results in nearly perfect solutions for $N = 4$ and 5, and is 99.9% accurate for $N = 6$, (and also obtains better than 99% accurate solutions for $N = 7$ and 8). For example, for $N = 6$ and $\gamma = 1.2$, where the weights of a trained SRN are $(\mathrm{Wd} = 3.4490, p = 1.0714, B = -2.9065)$, we achieve 99.9% accuracy, with equilibrium coordinates less than 0.001 away from $\pm 1$.

The method of [21] also makes it possible to train a *nondifferentiable* network, because the referred method is derivative-free. Our preliminary results with the shared-weight SRN-WTA with $\mathrm{sat}()$ nonlinearities indicate that nearly perfect solutions for $N$

TABLE II
SOME OTHER GOOD SOLUTIONS FOR WTA PROBLEM ON $N = 6$. ACCURACY IS BASED ON $6 \times 600 = 3600$ SAMPLE VECTORS

| p | B | Equilibrium | Accuracy (%) |
|---|---|---|---|
| 0.3458 | -1.0241 | (0.9258, -0.8415) | 99.7 |
| 0.3444 | -1.0044 | (0.9268, -0.8365) | 99.2 |
| 0.4376 | -1.3049 | (0.9498, -0.8593) | 99.5 |

from four to eight are achievable, but they are more dependent on initial weights and training parameters than those for the networks with $\tanh()$ of (4b). For example, for $N = 8$ the weights of a trained SRN $(\mathrm{Wd} = 1.0322,\ p = 0.1664,\ B = -1.0164)$ result in 99.99% accuracy with equilibrium coordinates at $\pm 1$.

### B. Engine Data Classification

The final example is application of an SRN to an engine data classification. There are four classes in both training and testing data sets, with different numbers of samples in each class. Each sample consists of 14 elements obtained from a car engine (factors in an engine diagnostic experiment). Each element represents a certain diagnostic parameter from a test run on a partially assembled engine. Various unknown combinations of parameter values indicate normal engines, or they may be indicative of different defects. It is known that the classification problem (whether the engine is normal or defective, and if so what kind of defect it is likely to have) is not linearly separable. A 14-10R-4 network with bipolar sigmoid transfer functions in the hidden and output layers is used. Neurons between different layers are fully connected. Weights in this SRN are not required to be symmetric and shared, as they were in the WTA problem of Section V-A.

The original data are normalized before training and testing. There are 41, 14, 13, and 10 sample vectors for each class in the training set, respectively, and 4, 3, 2, and 2 in the testing set. We arrange training into five streams. Each stream consists of a sample vector randomly selected from those 78 and changed for every epoch. After the five streams are presented to the SRN network, weights are updated at the end of the epoch. A root-mean-squared error (rmse) is then calculated for all 78 sample vectors to monitor the training. The training strategy for BPTT(30) and global EKF (GEKF) is to train for the first 200 epochs with $\eta = 0.1$ and $Q = 10^{-2}I$, then 800 epochs with $\eta = 0.1$ and $Q = 10^{-3}I$ (1000 epochs total). After 1000 epochs, the rmse drops to 0.019 (see Fig. 4) and the SRN network identifies all four classes with 100% accuracy, in both training and testing sets (see Table III). In addition to the previously mentioned bootstrap like training, a leave-one-out cross validation is also employed to evaluate the quality of the learning algorithm. The SRN network is stable in classifying all four classes, with the mean error vectors $(0.1061, -0.0303, -0.0269, -0.0638)$, $(-0.1421, 0.1227, -0.0739, -0.0771)$, $(-0.0627, -0.0583, 0.1336, -0.0987)$, $(-0.0817, -0.0561, -0.098, 0.1596)$, and the standard deviation of error vectors $(0.0857, 0.0227, 0.0274, 0.0821)$, $(0.2993, 0.1107, 0.1382, 0.0822)$, $(0.0659, 0.0992, 0.1223, 0.1201)$, $(0.0879, 0.0731, 0.1027, 0.1154)$.

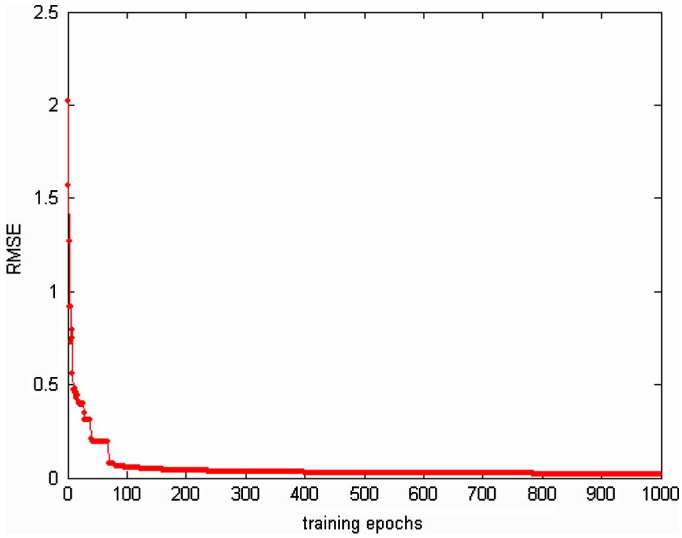



Fig. 4. The rmse of SRN training on engine data classification. The rmse is calculated based on all 78 training samples after SRN weights are updated by EKF at each epoch.

TABLE III
SRN ENGINE DATA CLASSIFICATION ON THE TESTING SET

| Testing data index | Network output | Target/Class |
|---|---|---|
| $X_1$ | (0.9999, -0.9772, -0.9980, -0.9956) | (1, -1, -1 , -1)/1 |
| $X_2$ | (0.9999, -0.9786, -0.9974, -0.9975) | (1, -1, -1 , -1)/1 |
| $X_3$ | (0.9998, -0.9592, -0.9986, -0.9956) | (1, -1, -1 , -1)/1 |
| $X_4$ | (0.9993, -0.8565, -0.9994, -0.9836) | (1, -1, -1 , -1)/1 |
| $X_5$ | (-0.9873, 0.9882, -0.9918, -0.9893) | (-1, 1, -1 , -1)/2 |
| $X_6$ | (-0.9699, 0.9799, -0.9873, -0.9978) | (-1, 1, -1 , -1)/2 |
| $X_7$ | (-0.6719, 0.8587, -0.9980, -0.9833) | (-1, 1, -1 , -1)/2 |
| $X_8$ | (-0.9847, -0.9862, 0.9867, -0.9883) | (-1, -1, 1 , -1)/3 |
| $X_9$ | (-0.9818, -0.9604, 0.9863, -0.9948), | (-1, -1, 1 , -1)/3 |
| $X_{10}$ | (-0.9818, -0.9918, -0.9871, 0.9921) | (-1, -1, -1 , 1)/4 |
| $X_{11}$ | (-0.9697, -0.9098, -0.9947, 0.9843) | (-1, -1, -1 , 1)/4 |

## VI. CONCLUSION

The SRN offers useful functionality among neural network models. Training, however, has often proven a barrier. We have

investigated the SRN for a simple WTA problem for its capability of comparing theoretically predicted versus actual performance. EKF training is shown to be effective for SRN architectures. The issues uncovered in training for this problem extend to other, less tractable problems. This is demonstrated by applying the same technique to an engine classification problem.

## APPENDIX

Local stability analysis is helpful for understanding whether the dynamic system, with a specified weight matrix, possesses and reaches equilibrium points and what are conditions on the weight update rules [15].

Let $y = [y_1, y_2, \ldots, y_N^T]$ represent the output vector of the SRN. At the equilibrium points, the derivative of SRN outputs $y$, typically representing solutions for a given problem, has the property of $\partial y_k/\partial t = 0$. The neuron dynamics of SRN is described by (3). A bipolar sigmoid with slope coefficient $\gamma \in R^+$, i.e., (4), is assumed as the transfer function for all neurons.

By the chain rule, we have $\partial y_k^{(n+1)}/\partial t) = (\partial y_k^{(n+1)}/\partial \mathrm{net}_k)(\partial \mathrm{net}_k/\partial t)$ for $k = 1, 2, \ldots, N$, where $\mathrm{net}_k = \sum_{j=1}^{N} W_{k,j} y_k^{(n)} + B_k$. After simple substitutions, the matrix form of the previous equation can be written as (A1), shown at the bottom of the page. At the equilibrium point, the vector of partial derivatives in (A1) is set to zero, which can be achieved by setting the left-most matrix to the zero matrix of appropriate dimensions. It is noted that if $y_k^{(n+1)} = 1$ or $y_k^{(n+1)} = -1$, then $(1 - y_k^{(n+1)})(1 + y_k^{(n+1)}) = 0$ for all $k$. Clearly, (A1) shows that the corners of a unit hypercube can be selected as the equilibrium points of an SRN. Due to properties of the bipolar sigmoid transfer function, it is impossible to reach the corners of the unit hypercube in finite time, with finite values of weights and excitations. Moreover, only a portion, $N$ of $2^N$, of the vertices, i.e., $Y = [y_1, y_2, \ldots, y_N]^T$ $y_i = 1 y_j = -1$ $\forall j \neq i$ and $i = 1, \ldots, N$, are desired (but not precisely attainable) equilibrium points. Other sets of equilibrium points, within the unit hypercube in the output space of SRN, can be obtained by solving (A1) as follows: Since $\partial y^{(n+1)}/\partial t = \partial y^{(n)}/\partial t = 0$, (A1) can be written as (A2), shown at the bottom of the page, where $I$ is an $N \times N$ identity matrix. If the matrix in the parentheses has an inverse, then the vector of partial derivatives in (A2) is equal to zero. The condition for the existence of the inverse matrix leads to the necessary condition of equilibrium points within the hypercube. Such internal equilibrium points are attainable in architectures with bipolar sigmoids for finite values of weights and excitations.

Assume that we have an equilibrium point inside the hypercube, i.e., $y = [y_1, y_2, \ldots, y_N]^T$ with $y_i > 0$ and $y_j = y_k < 0$ $\forall j, k \neq i, j, k \in \{1, \ldots, N\} - \{i\}$. Without loss of generality, the matrix $M$ in the parentheses in (A2) can be written as

$$M = \left( I - \gamma \begin{bmatrix} \alpha & 0 & \cdots & 0 \\ 0 & \beta & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta \end{bmatrix} \begin{bmatrix} 1 & -p & \cdots & -p \\ -p & 1 & \cdots & -p \\ \vdots & \vdots & \ddots & \vdots \\ -p & -p & \cdots & 1 \end{bmatrix} \right)$$

where $0 < \alpha = (1 - y_i^2), \beta = (1 - y_j^2) < 1$.

The determinant of $M$ can be obtained as shown in the third equation at the bottom of the page, where

$$|A| = [(1 - \gamma\alpha) - \gamma\alpha p] \underbrace{\begin{vmatrix} 1 - \gamma\beta & \gamma\beta p & \cdots & \gamma\beta p \\ \gamma\beta p & 1 - \gamma\beta & \cdots & \gamma\beta p \\ \vdots & \vdots & \ddots & \vdots \\ \gamma\beta p & \gamma\beta p & \cdots & 1 - \gamma\beta \end{vmatrix}}_{N-1}$$

and

$$|B| = \left[ \gamma\alpha p - (1 - \gamma\beta)\frac{\alpha}{\beta} \right] \underbrace{\begin{vmatrix} \gamma\beta p & \gamma\beta p & \cdots & \gamma\beta p \\ \gamma\beta p & 1 - \gamma\beta & \cdots & \gamma\beta p \\ \vdots & \vdots & \ddots & \vdots \\ \gamma\beta p & \gamma\beta p & \cdots & 1 - \gamma\beta \end{vmatrix}}_{N-1}.$$

$$\begin{bmatrix} \frac{\partial y_1^{(n+1)}}{\partial t} \\ \vdots \\ \frac{\partial y_N^{(n+1)}}{\partial t} \end{bmatrix} = \gamma \begin{bmatrix} \left(1 - y_1^{(n+1)}\right)\left(1 + y_1^{(n+1)}\right) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \left(1 - y_N^{(n+1)}\right)\left(1 + y_N^{(n+1)}\right) \end{bmatrix} \begin{bmatrix} W_{11} & \cdots & W_{1N} \\ \vdots & \ddots & \vdots \\ W_{N1} & \cdots & W_{NN} \end{bmatrix} \begin{bmatrix} \frac{\partial y_1^{(n)}}{\partial t} \\ \vdots \\ \frac{\partial y_N^{(n)}}{\partial t} \end{bmatrix}. \tag{A1}$$

$$\left( I - \gamma \begin{bmatrix} \left(1 - y_1^{(n)}\right)\left(1 + y_1^{(n)}\right) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \left(1 - y_N^{(n)}\right)\left(1 + y_N^{(n)}\right) \end{bmatrix} \begin{bmatrix} W_{11} & \cdots & W_{1N} \\ \vdots & \ddots & \vdots \\ W_{N1} & \cdots & W_{NN} \end{bmatrix} \right) \begin{bmatrix} \frac{\partial y_1^{(n)}}{\partial t} \\ \vdots \\ \frac{\partial y_N^{(n)}}{\partial t} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{A2}$$

$$|M| = \begin{vmatrix} (1 - \gamma\alpha) - \gamma\beta p \times \frac{\alpha}{\beta} & \gamma\alpha p - (1 - \gamma\beta) \times \frac{\alpha}{\beta} & \cdots & 0 \\ \gamma\beta p & 1 - \gamma\beta & \cdots & \gamma\beta p \\ \vdots & \vdots & \ddots & \vdots \\ \gamma\beta p & \gamma\beta p & \cdots & 1 - \gamma\beta \end{vmatrix} = |A| - |B|$$

*Proposition 1:*

$$\underbrace{\begin{vmatrix} b & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{N} = (a-b)^{N-1}b$$

$$\forall a, b \in R; \qquad a \neq b; \qquad N \in I$$

*Proof:* The proposition is proved by induction.

When

$$N = 1, \quad \underbrace{\begin{vmatrix} b & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{N} = |b| = (a-b)^{1-1}b.$$

The proposition holds.

When $N = k$, we assume that

$$\underbrace{\begin{vmatrix} b & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{k} = (a-b)^{k-1}b$$

holds.

When $N = k+1$

$$\underbrace{\begin{vmatrix} b & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{k+1} = \begin{vmatrix} 0 & b-a & 0 & \cdots & 0 \\ b & a & b & \cdots & b \\ b & b & a & \cdots & b \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b & b & b & \cdots & a \end{vmatrix}$$

$$= (a-b)\underbrace{\begin{vmatrix} b & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{k}$$

$$= (a-b)(a-b)^{k-1}b = (a-b)^{(k+1)-1}b.$$

The proposition holds, and it is now proved. □

*Proposition 2:*

$$\underbrace{\begin{vmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{N} = (a-b)^{N-1}(a+(N-1)b)$$

$$\forall a, b \in R; \qquad a \neq b; \qquad N \in I.$$

*Proof:* The proposition is proved by induction.

When $N = 1$

$$\underbrace{\begin{vmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{N} = |a| = (a-b)^{1-1}(a+(1-1)b) = a.$$

The proposition holds.

When $N = k$, we assume that

$$\underbrace{\begin{vmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{k} = (a-b)^{k-1}(a+(k-1)b)$$

holds.

When $N = k+1$

$$\underbrace{\begin{vmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{k+1} = \begin{vmatrix} a-b & b-a & 0 & \cdots & 0 \\ b & a & b & \cdots & b \\ b & b & a & \cdots & b \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b & b & b & \cdots & a \end{vmatrix}$$

$$= (a-b)\underbrace{\begin{vmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{k} + (a-b)\underbrace{\begin{vmatrix} b & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{k}.$$

From the Proposition 1 and the induction assumption, we know that

$$\underbrace{\begin{vmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{vmatrix}}_{k+1} = (a-b)(a-b)^{k-1}(a+(k-1)b)$$

$$+ (a-b)(a-b)^{k-1}b$$

$$= (a-b)^{(k+1)-1}(a+((k+1)-1)b).$$

The proposition holds, and it is now proved. □

This means that $|A|$, after some simple substitutions and calculation, is

$$|A| = [(1-\gamma\alpha) - \gamma\alpha p](1-\gamma\beta-\gamma\beta p)^{N-2} [(1-\gamma\beta) + (N-2)\gamma\beta p]$$

thus $|B|$ can be obtained by Proposition 1

$$|B| = \left[\gamma\alpha p - (1-\gamma\beta)\frac{\alpha}{\beta}\right](1-\gamma\beta-\gamma\beta p)^{N-2}\gamma\beta p.$$

$$p \neq \frac{(\gamma\alpha - 1)(N-2) \pm \sqrt{(N-2)^2(1-\gamma\alpha)^2 + 4(N-1)(1-\gamma\alpha)(1-\gamma\beta)\alpha/\beta}}{-2(N-1)\gamma\alpha}. \quad \text{(A4)}$$

$$\gamma_1 = \frac{[N^2\beta + 2(N-1)(\alpha-\beta)] - 2\sqrt{N^2\beta(N-1)(\alpha-\beta) + (N-1)^2(\alpha-\beta)}}{N^2\alpha\beta^2}$$
$$\gamma_2 = \frac{[N^2\beta + 2(N-1)(\alpha-\beta)] + 2\sqrt{N^2\beta(N-1)(\alpha-\beta) + (N-1)^2(\alpha-\beta)}}{N^2\alpha\beta^2}.$$

Therefore

$$|M| = (1-\gamma\beta-\gamma\beta p)^{N-2}[(1-\gamma\alpha)(1-\gamma\beta) + (1-\gamma\alpha)(N-2)\gamma\beta p - (N-1)\gamma^2\alpha\beta p^2].$$

Clearly, $|M|$ is not equal to zero if

$$(1-\gamma\beta-\gamma\beta p)^{N-2} \neq 0, \quad \text{i.e.} \quad p \neq (1-\gamma\beta)/\gamma\beta \quad \text{(A3)}$$

and the part in brackets, which is quadratic in $p$, is not equal to zero, i.e., as shown in (A4), at the top of the page. Thus, a weight matrix (5), for which the conditions (A3) and (A4) are satisfied, will guarantee the convergence of the WTA network to the equilibrium $y = [y_1, \ldots, y_N]^T$. Furthermore, if the expression inside the square root in (A4) is less than zero, then the part in brackets of $|M|$ has no zero solution for a real value of $p$, i.e.,

$$(N-2)^2(1-\gamma\alpha)^2\beta + 4(N-1)(1-\gamma\alpha)(1-\gamma\beta)\alpha < 0$$

that is

$$N^2\alpha^2\beta^2\gamma^2 - [2N^2\alpha\beta + 4(N-1)(\alpha^2-\alpha\beta)]\gamma + [N^2 + 4(N-1)(\alpha-\beta)] < 0 \quad \text{(A5)}$$

i.e.,

$$\gamma_1 < \gamma < \gamma_2, \quad \gamma_1, \gamma_2 > 0, \qquad \text{if } \alpha > \beta \quad \text{(A6)}$$

where the second equation shown at the top of the page holds.

An empirical observation based on extensive simulations performed on WTA problems indicates that the SRN tends to converge to a fixed point, starting with random initial values for the input vectors. Furthermore, there are proofs that provide sufficient conditions for the SRN with the shared weights and sigmoid transfer function to have a unique equilibrium point for all nonzero inputs [17]. However, our problem is sufficiently different to violate conditions of the existing global stability analysis techniques.

## REFERENCES


[1] R. Lippmann, "An introduction to computing with neural nets," *IEEE Acoust. Speech Signal Process. Mag.*, vol. 4, no. 2, pp. 4–22, Apr. 1987.

[2] S. Kaski and T. Kohonen, "Winner-take-all network for physiological models of competitive learning," *Neural Netw.*, vol. 7, pp. 973–984, 1994.

[3] J. P. F. Sum and P. K. S. Tam, "Note on the MAXNET dynamic," *Neural Comput.*, vol. 8, no. 3, pp. 491–499, 1996.

[4] J. P. F. Sum, C. S. Leung, P. K. S. Tam, G. H. Young, W. K. Kan, and L. W. Chan, "Analysis for a class of winner-take-all model," *IEEE Trans. Neural Netw.*, vol. 10, no. 1, pp. 64–71, Jan. 1999.

[5] J. Choi and B. J. Sheu, "A high precision VLSI winner-take-all circuit for self organizing neural networks," *IEEE J. Solid-State Circuits*, vol. 28, no. 5, pp. 576–594, May 1993.

[6] S. Smedley, J. Taylor, and M. Wilby, "A scalable high-speed current winner take all network for VLSI neural applications," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 42, no. 5, pp. 289–291, May 1995.

[7] L. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Trans. Circuits Syst.*, vol. 35, no. 10, pp. 1257–1272, Oct. 1988.

[8] L. Andrew, "Improving the robustness of winner-take-all cellular neural network," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 43, no. 4, pp. 329–334, Apr. 1996.

[9] X. Z. Pang and P. J. Werbos, "Neural network design for J function approximation in dynamic programming," *Math. Model. Sci. Comput. (Principia Scientia J.)*, vol. 5, no. 2/3, 1996.

[10] D. C. Wunsch II, "The cellular simultaneous recurrent network adaptive critic design for the generalized maze problem has a simple closed-form solution," in *Proc. Int. Joint Conf. Neural Netw.*, Como, Italy, Jul. 2000, vol. 3, pp. 79–82.

[11] P. J. Werbos, "Backpropagation through time: What it does and how to do it," in *Proc. IEEE*, Oct. 1990, vol. 78, no. 10, pp. 1550–1560.

[12] B. Pearlmutter, "Gradient calculation for dynamic recurrent neural networks—A survey," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1212–1228, Sep. 1995.

[13] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. New York: Wiley, 1994.

[14] L. A. Feldkamp and G. V. Puskorius, "A signal processing framework based on dynamic neural networks with application to problems in adaptation, filtering and classification," *Proc. IEEE*, vol. 86, no. 11, pp. 2259–2277, Nov. 1998.

[15] G. Serpen and Y. Xu, "Stability of simultaneous recurrent neural network dynamics for static optimization," in *Proc. Int. Joint Conf. Neural Netw.*, Honolulu, HI, May 2002, vol. 3, pp. 2023–2028.

[16] J. J. Hopfield and D. W. Tank, "Computing with neural circuits: A model," *Science*, vol. 233, pp. 625–633, Aug. 1986.

[17] F. Hunt and D. Pearson, "A condition for a unique equilibrium point in a recurrent neural network," in *Proc. Int. Conf. Neural Netw.*, Jun. 1996, vol. 3, pp. 1308–1311.

[18] A. N. Michel, K. Wang, D. Liu, and H. Ye, "Qualitative limitations incurred in implementations of recurrent neural networks," *IEEE Control Syst. Mag.*, vol. 15, no. 3, pp. 52–65, Jun. 1995.

[19] X. Cai and D. C. Wunsch II, "Counterexample of a claim pertaining to the synthesis of a recurrent neural network," in *Proc. Int. Joint Conf. Neural Netw.*, Honolulu, HI, May 2002, vol. 3, pp. 2029–2032.

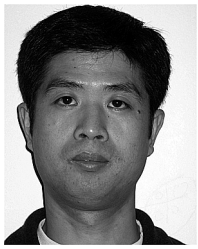
[20] L. Feldkamp, D. Prokhorov, C. Eagen, and F. Yuan, , J. Suykens and J. Vandewalle, Eds., "Enhanced multi-stream kalman filter training for recurrent networks," in *Nonlinear Modeling: Advanced Black-Box Techniques*. Norwell, MA: Kluwer, 1998, pp. 29–53.

[21] L. Feldkamp, T. Feldkamp, and D. Prokhorov, "Neural network training with the nprKF," in *Proc. Int. Joint Conf. Neural Netw.*, Washington, DC, Jul. 2001, pp. 109–114.

[22] D. White and D. Sofge, Eds., *Handbook of Intelligent Control: Neural, Adaptive and Fuzzy Approaches*. New York: Van Nostrand, 1992.

[23] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 279–297, Mar. 1994.

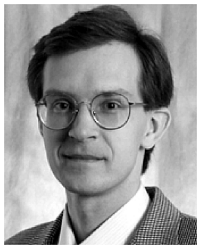
[24] A. Alessandri, M. Cuneo, S. Pagnan, and M. Sanguineti, "On the convergence EKF-based parameters optimization for neural networks," in *Proc. 42nd Conf. Decision Control*, 2003, pp. 5825–5830.
[25] J. de Jess Rubio and W. Yu, "Dead-zone Kalman filter algorithm for recurrent neural networks," in *Proc. 43rd Conf. Decision Control*, 2005, pp. 2562–2567.
[26] X. Liu, K. L. Teo, and B. Xu, "Exponential stability of impulsive high-order Hopfield-type neural networks with time-varying delays," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1329–1339, Sep. 2005.
[27] H. Jiang, L. Zhang, and Z. Teng, "Existence and global exponential stability of almost periodic solution for cellular neural networks with variable coefficients and time-varying delays," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1340–1351, Sep. 2005.
[28] Y. He, M. Wu, and J. She, "An improved global asymptotic stability criterion for delayed cellular neural networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 1, pp. 250–252, Jan. 2006.
[29] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.



**Xindi Cai** (S'00–M'06) received the B.S. degree in information engineering from Northwestern Polytechnical University, Xi'an, Shaanxi, P. R. China, in 1996 and the M.S. degree in computer engineering and the Ph.D. degree in electrical engineering from the University of Missouri—Rolla, Rolla, in 2002 and 2006, respectively.

In 2006, he joined R&D Department, American Power Conversion Corporation, O'Fallon, MO, where currently, he is a Senior Control Engineer. He also worked as a System Testing Engineer with Lucent Technologies from 1996 to 1999. He is the author of several journal papers and invited book chapters. His research interests include adaptive control, machine learning, neural networks, time series analysis, evolutionary computation, particle swarm optimization, and game of Go.

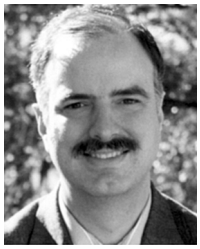
Dr. Cai is a Member of the IEEE Computational Intelligence Society and a Member of the International Neural Network Society (INNS). He has been serving as review committee member for International Joint Conference on Neural Networks (IJCNN) from 2002 to 2007, and reviewer for multiple journals. He is the recipient of two prizes of worldwide competitions for time-series analysis (IJCNN 2001 and 2004), and several Student Travel Grants [IJCNN 2002 and 2005 and International Conference on Cognitive and Neural Systems (ICCNS) 2004].



**Danil V. Prokhorov** (SM'02) received the M.S. degree with Honors from the State Academy of Aerospace Engineering (formerly LIAP), St. Petersburg, Russia, in 1992, and the Ph.D. degree in electrical engineering from Texas Technical University, Lubbock, in 1997.

He was with Ford Research Laboratory, Dearborn, MI, from 1997 until 2005. He had been engaged in application-driven studies of neural networks and their training algorithms. He is currently a Research Manager at Toyota Technical Center, Ann Arbor, MI. He has authored 80 technical publications including several patents. His research interest is in machine learning algorithms and their applications to decision making under uncertainty.

Dr. Prokhorov was awarded the International Neural Network Society (INNS) Young Investigator in 1999. He was the IJCNN 2005 General Chair and IJCNN 2001 Program Chair. He has been a reviewer for numerous journals and conferences, a program committee member of many conferences, and a panel expert for the National Science Foundation (NSF) every year since 1995.



**Donald C. Wunsch II** (M'94–SM'97–F'05) received the B.S. degree in applied mathematics from the University of New Mexico, Albuquerque, in 1984, the M.S. degree in applied mathematics, and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in 1987 and in 1991, respectively, the Executive MBA from Washington University in St. Louis, St. Louis, MO, in 2006, and he also completed a Humanities Honors Program at Seattle University, Seattle, WA, in 1981.

He is the Mary K. Finley Missouri Distinguished Professor of Electrical and Computer Engineering at the University of Missouri—Rolla, Rolla, where he has been since July 1999. He has courtesy appointments in computer science, systems engineering, and business administration. His prior positions were Associate Professor and Director of the Applied Computational Intelligence Laboratory at Texas Technical University, Senior Principal Scientist at Boeing, Consultant for Rockwell International, and Technician for International Laser Systems. He has over 250 publications in his research field of computational intelligence, and has attracted over $5 million in research funding. He has produced seven Ph.D.s in electrical engineering, four in computer engineering, and one in computer science. His research interests are in neural networks, and their applications in: reinforcement learning, approximate dynamic programming, the game of Go, financial engineering, graph theory, risk assessment, representation of knowledge and uncertainty, collective robotics, computer security, critical infrastructure protection, biomedical applications of computational intelligence, telecommunications, and smart sensor networks.

Dr. Wunsch II is a recipient of the Halliburton Award for Excellence in Teaching and Research and the National Science Foundation CAREER Award. He served as a Board member of the International Neural Networks Society (INNS), the University of Missouri Bioinformatics Consortium, the Idaho EPSCOR Project Advisory Board, and the IEEE Neural Networks Council. He also served as Technical Program Co-Chair for International Joint Conference on Neural Networks (IJCNN 2002), General Chair for IJCNN 2003, and the President of the INNS.