TRTRL: A Localized Resource-Efficient Learning Algorithm for Recurrent Neural Networks

August 8, 2006

Danny Budik, Itamar Elhanany

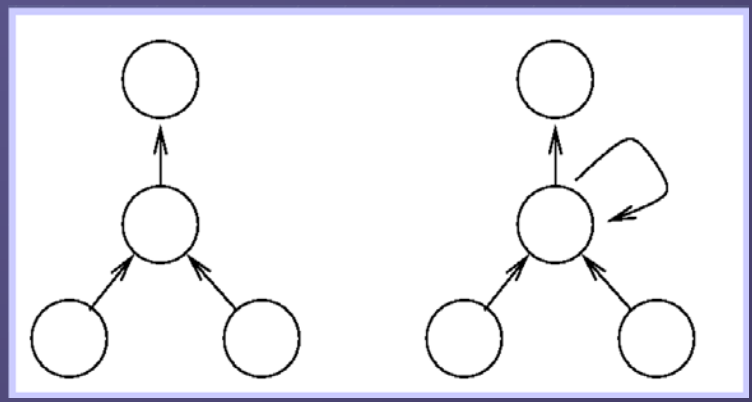
Machine Intelligence Lab
Department of Electrical & Computer Engineering
University of Tennessee

Outline

- Background and Motivation
- Overview of RTRL
- The Truncated-RTRL (TRTRL) algorithm
- Simulations results
- Conclusions and Future

Recurrent Neural Networks (RNNs)

- Feedforward neural networks are stateless or (memory-less) and statically map from input space to output space
- RNNs contain internal feedback loops that allow the network to capture temporal dependencies
- Many tasks require learning of temporal sequences, including
  - Sequential pattern recognition (e.g. speech recognition)
  - Time series prediction



RNNs (cont.)

- Numerous RNN architectures have been proposed
  - Elman Networks – simplistic (Markovian-type) RNNs
  - Back-propagation through time (BPTT)
  - Time-Delay Neural Networks (TDNNs)
- The above are not designed for online (real-time) applications
- Real-Time Recurrent Learning (RTRL)
  - Targets online training of RNNs
  - Calculates the gradient without need for explicit history
  - However, it requires extensive storage and computational resources that limit its scalability

Real Time Recurrent Learning (RTRL)

- Originally described by R.J. Williams in 1989
- Activation function of neuron k is defined as

$$y_k = f_k(s_k(t))$$

- where s_k denotes the weighted sum of all activations leading to neuron k
- The network error at time t is defined by:

$$J(t) = -\frac{1}{2} \sum_{k \in outputs} [y_k(t) - d_k(t)]^2$$

$$= -\frac{1}{2} \sum_{k \in outputs} [e_k(t)]^2$$

- where $d_k(t)$ denotes the desired target value for output neuron k

RTRL: Updating Weights

- The error is minimized along a positive multiple of the performance measure gradient such that

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

$$\Delta w_{ij}(t) = \alpha \frac{\partial J(t)}{\partial w_{ij}(t)}$$

$$= \sum_{k \in outputs} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}(t)}$$

- The partial derivatives of the activation function with respect to the weights are identified as sensitivity elements and are denoted by

$$p_{ij}^k(t+1) = \frac{\partial y_k(t+1)}{\partial w_{ij}(t)}$$

Recursive Calculation of the Sensitivity Elements in RTRL

- The sensitivities of node k with respect to the change in weight w_{ij} can be obtained as follows

$$p_{ij}^{k}(t+1) = f_k'\left(s_k(t)\right)\left[\sum_{l \in N} w_{kl} p_{ij}^{l}(t) + \delta_{ik} z_j(t)\right]$$

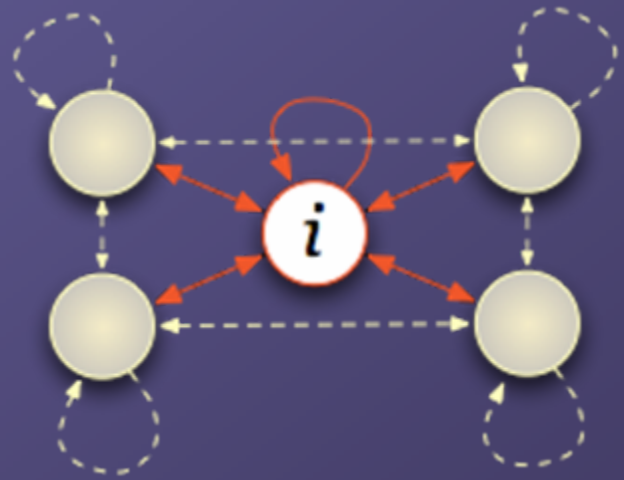
- These key components require each neuron to
  - Perform $O(N^3)$ multiplications → resulting in a total computational complexity of $O(N^4)$
  - Access storage of $O(N^2)$ → yielding a total storage requirement of $O(N^3)$

- Example: For N=100 → 100 million multiplications and over 2 MB of storage!

Truncated Real Time Recurrent Learning (TRTRL)

Motivation:
To obtain a scalable version of the RTRL algorithm while minimizing performance degradation

How?
Limit the sensitivities of each neuron to its ingress (incoming) and egress (outgoing) links



Performing Sensitivity Calculations in TRTL

- For all nodes that are not in the output set, the egress sensitivity values for node j are calculated by imposing $j=k$ in the original RTRL sensitivity equation, such that

$$p_{ij}^{j}(t+1) = f_j'\left(s_j\left(t\right)\right)\left[w_{ji}p_{ij}^{i}(t) + \delta_{ij}y_j(t)\right]$$

- Similarly, the ingress sensitivity values for node j are given by

$$p_{ij}^{i}(t+1) = f_i'\left(s_i\left(t\right)\right)\left[w_{ij}p_{ij}^{j}(t) + z_j(t)\right]$$

- For output neurons, a nonzero sensitivity element must exist in order to update the weights

$$p_{ij}^{o}(t+1) = f_o'\left(s_o\left(t\right)\right)\left[w_{oi}p_{ij}^{i}(t) + w_{oj}p_{ij}^{j}(t) + \delta_{io}z_j(t)\right]$$

Resource Requirements of TRTRL

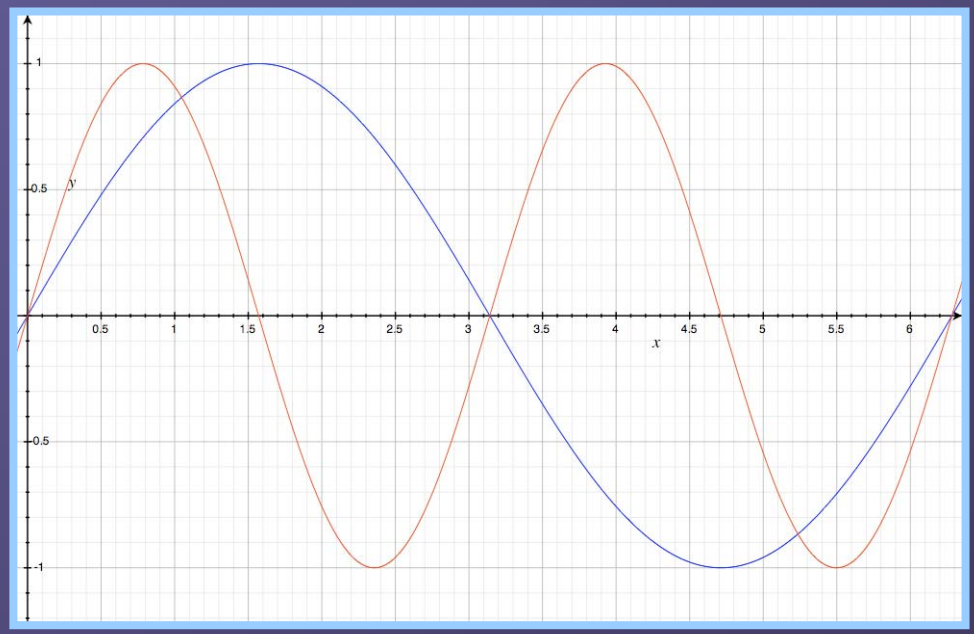
- The network structure remains the same with TRTRL, only the calculation of sensitivities is reduced

- Significant reduction in resource requirements …
  - Computational load for each neuron drops to from $O(N^3)$ to $O(2KN)$, where K denotes the number of output neurons
    - Total computational complexity is now $O(2KN^2)$
  - Storage requirements drop from $O(N^3)$ to $O(N^2)$

- Example revisited: For N=100, 10 outputs → 100k multiplications and only 20kB of storage!

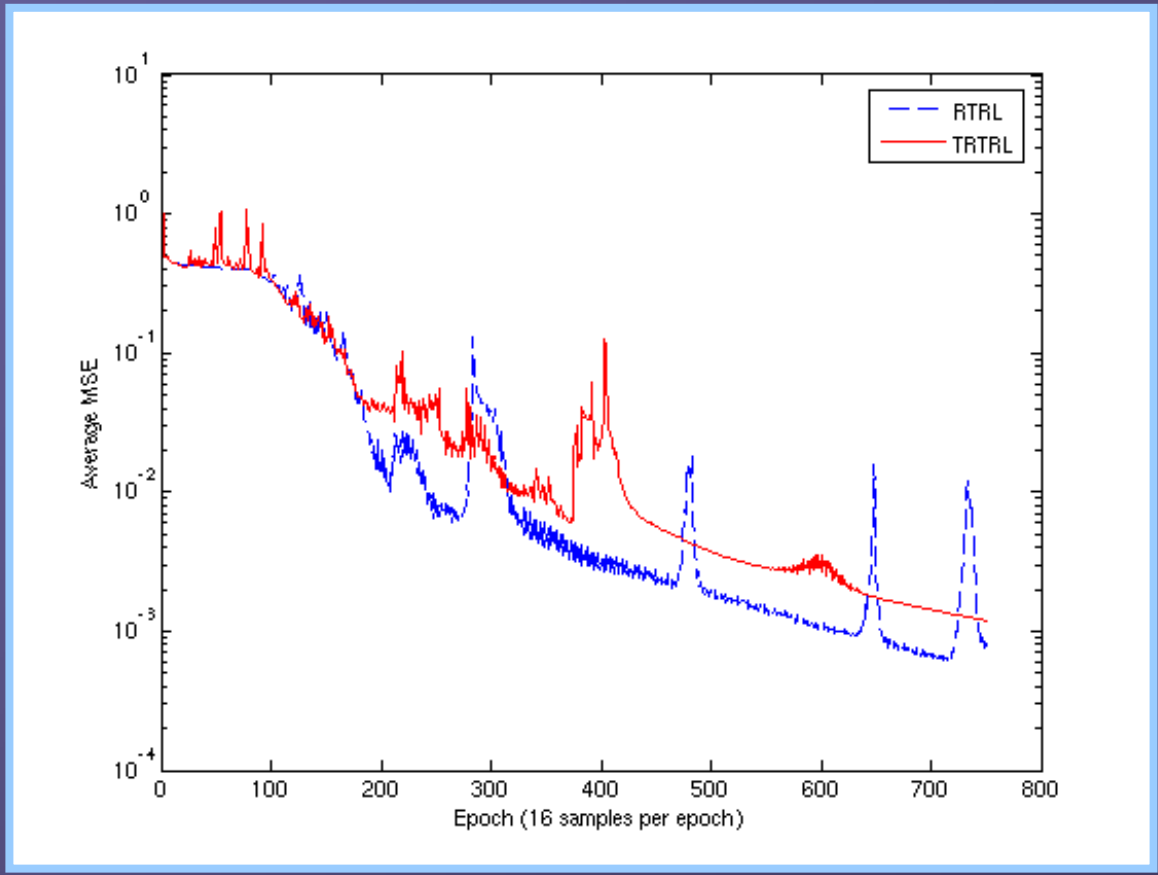
Evaluating TRTRL Performance

- Testing criteria
  - Tests must have nonlinear functions which exploit the characteristics of RNNs
  - Temporal dependency: two identical inputs could have different outputs depending on the time step



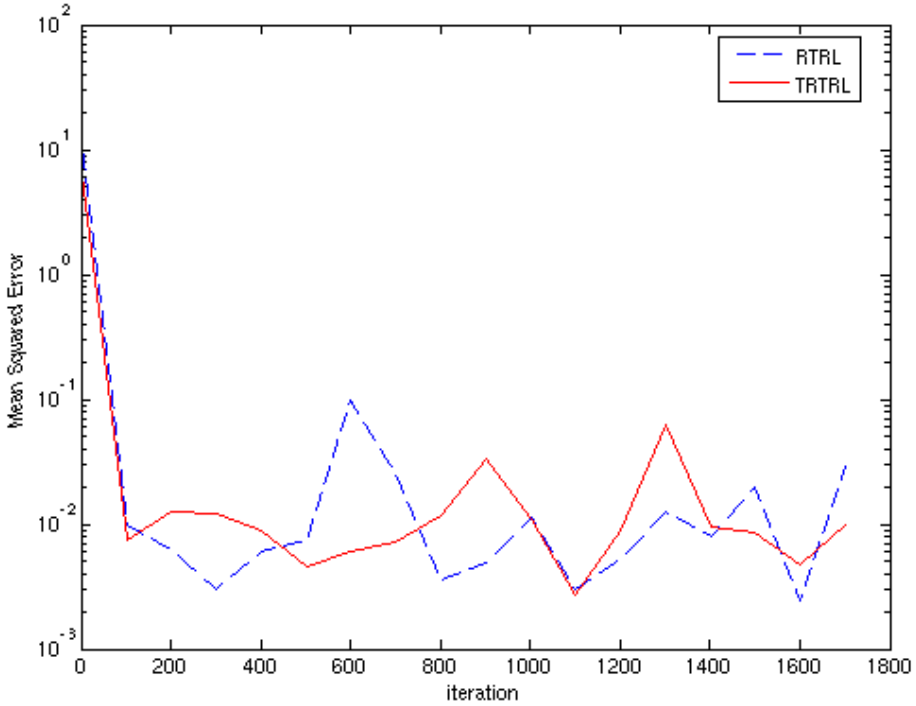
Test case #1: Frequency Doubler

- Input: $sin(x)$, target output $sin(2x)$
- Both networks had 15 neurons



Test case #2: Mackey-Glass Chaotic Series Predictor

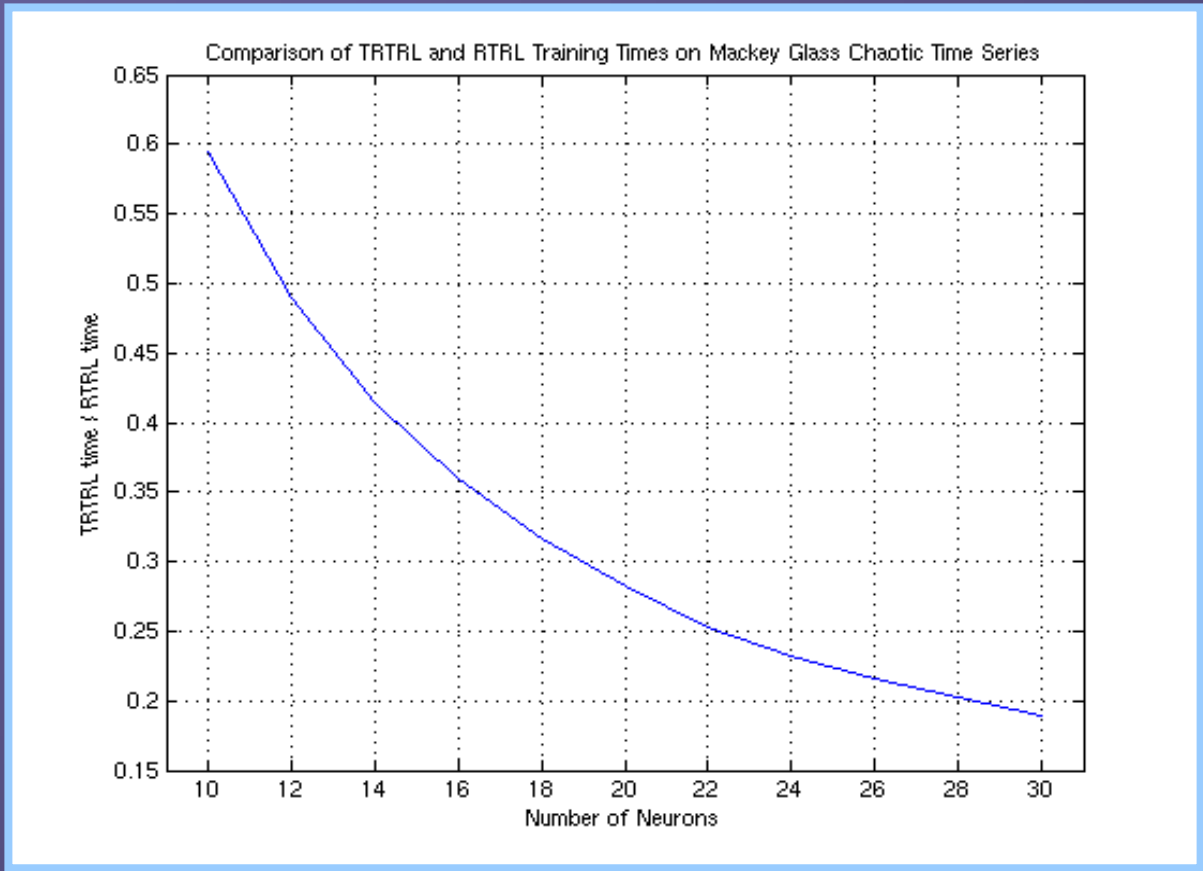
- Series is based on time-delayed differential equations
- Task is much more difficult than the frequency doubler



- Both networks had 25 neurons, predicting output with 30 time step dependencies
- Similar performance was observed on both learning algorithms

RTRL vs. TRTRL Training Time

- As expected, speedup gain for increases with network size
- Non-optimal code written in MATLAB®



Conclusions and Future Work

- High-performance, resource-efficient learning algorithm for training RNNs has been introduced
  - Computational complexity reduced from $O(N^4)$ to $O(N^2)$
  - Storage complexity reduced from $O(N^3)$ to $O(N^2)$
- Minor performance degradation observed

- Future work
  - Algorithm is now more localized lending itself to hardware realization
  - Future work will focus on FPGA implementation of TRTRL