Time series prediction

Martin Babka

December 1, 2011

Time series

What to predict?

Time series

- A sequence of values depending on time $\{x(t_0), x(t_1), \dots\}$.
- Output of a process – discrete or continuos (sampled in high enough frequence given by Nyquist theorem).
- Cloudiness, temperature, electricity demand, stock market index, ...

Processing tasks

- Prediction of $x(t)$ given $x(t-1), \dots, x(t-p)$.
- Classification into a few casses, e.g. $x(t)$ will rise, drop or stay.
- Transform series, electricity demand $\rightarrow$ electricity price.

Time series

How to predict

Process's properties w.r.t to time

- Stationary – properties of the time series do not change.
- Non-stationary – properties change with time.

Models for stationary processes

- Hidden Markov Model, Dynamic Bayesian Network, various types of Temporal Neural Networks.
- In general – classifier + short-term memory.

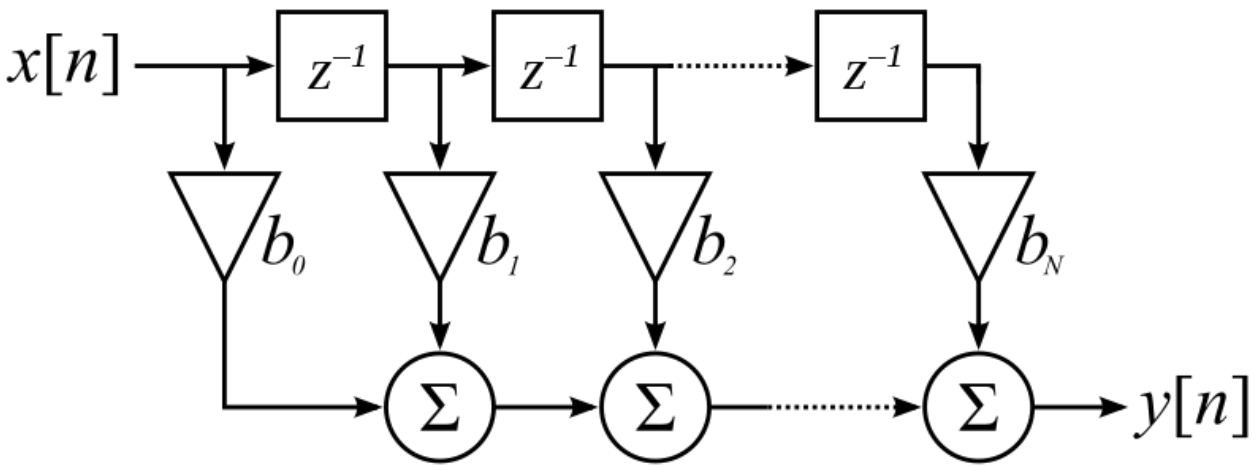
Models for non-stationary processes

- Temporal Neural Networks, Recurrent Neural Networks.
- In general – classifier + short-term memory + ability to adapt.

Memory

Finite Impulse Response

- Memory able to remember a finite number of the most recent values.
- Possible generalization to set up depth/resolution tradeoff.



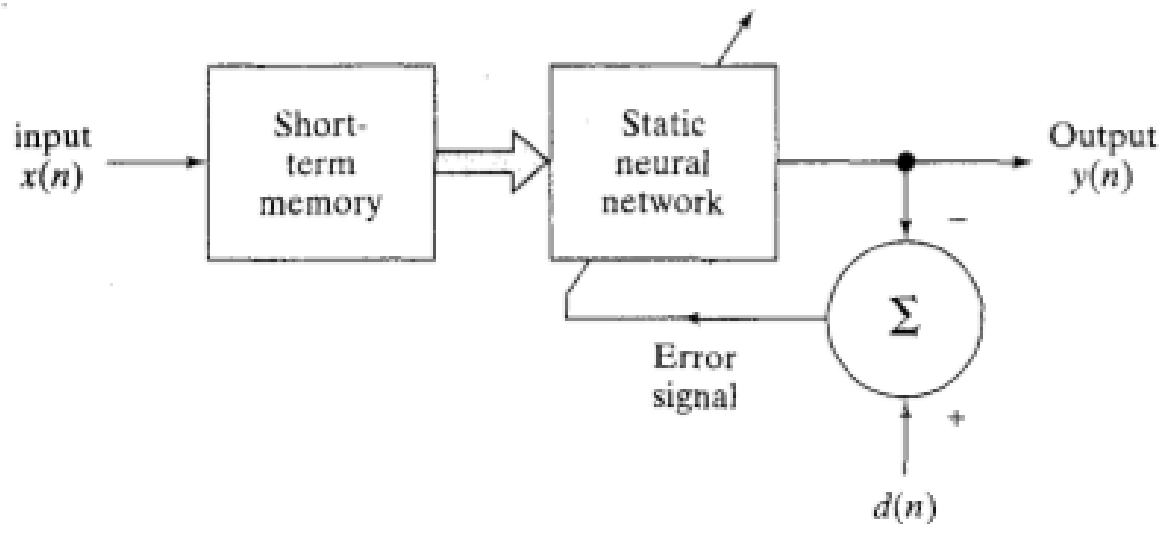
Temporal Neural Networks

Models

- Possible architecture ranges from a perceptron (focused neuronal filter) to a static network (focused time lagged feedforward network).
- Input is a FIR – convolution of the lagged sequence.
- If there are more input variables (= more FIRS) then we obtain a multiple input neuronal filter or a network model called spatio-temporal model.
- Theory says that every myopic shift invariant causal map can be approximated by a memory + static neural network. This means that every non-stationary process can be modelled by spatio-temporal model. [Sandberg 1991]

Models of Temporal Neural Networks

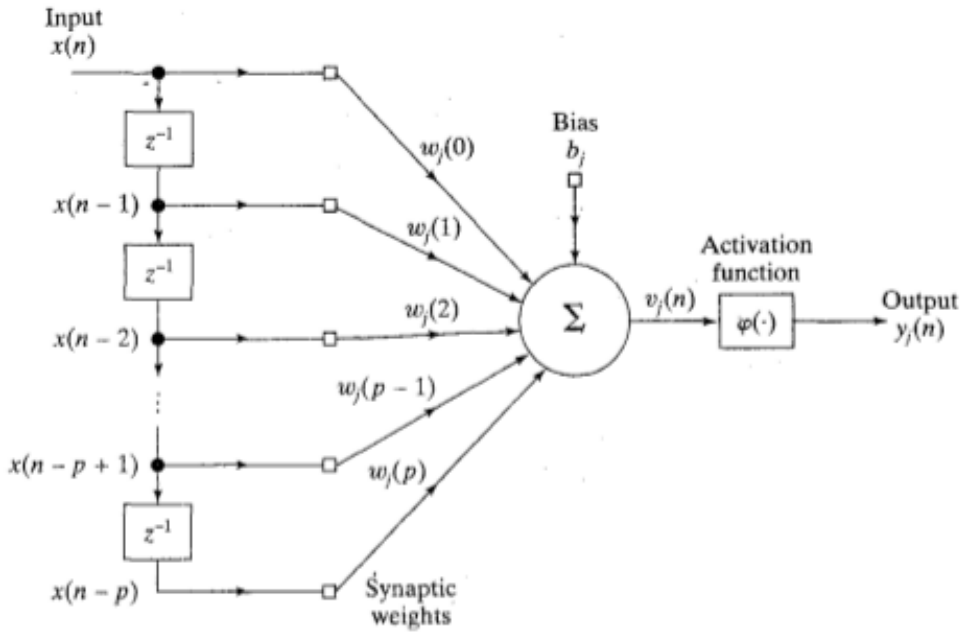
General Temporal Neural Network



Nonlinear filter built on a static neural network.

Models of Temporal Neural Networks

Focused networks - perceptron



Focused neuronal filter.

$$\vec{x}(n) = (x(n), x(n-1), \ldots, x(n-p))$$
$$\vec{w} = (w(0), \ldots, w(p))$$
$$y = \varphi(\vec{w}^T \vec{x}(n) + b)$$

Models of Temporal Neural Networks

Focused networks - network

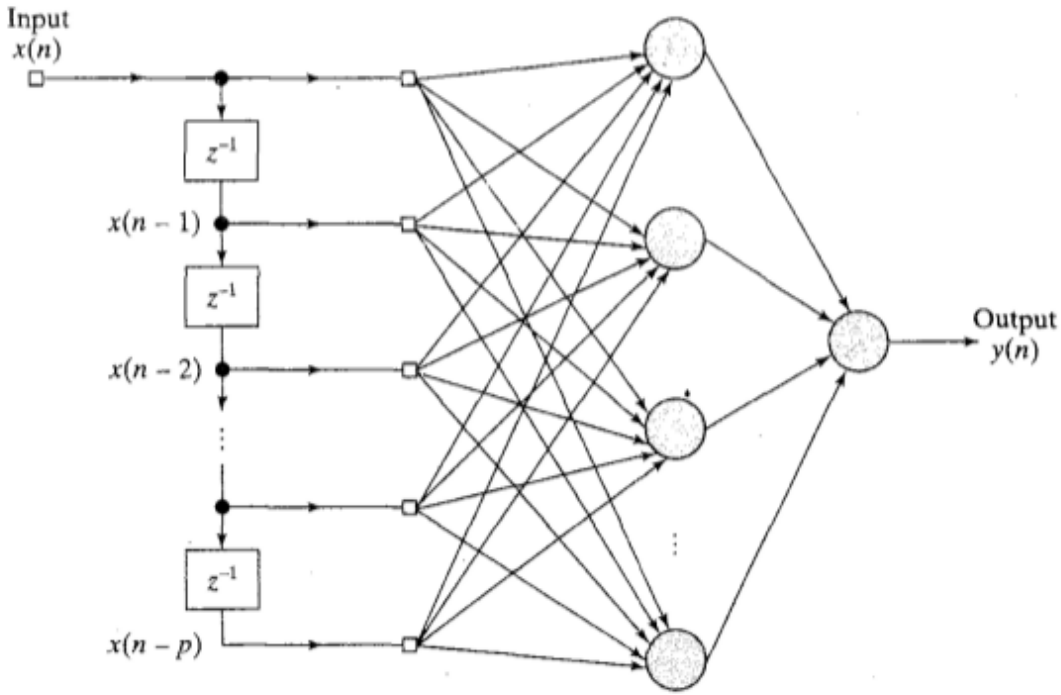
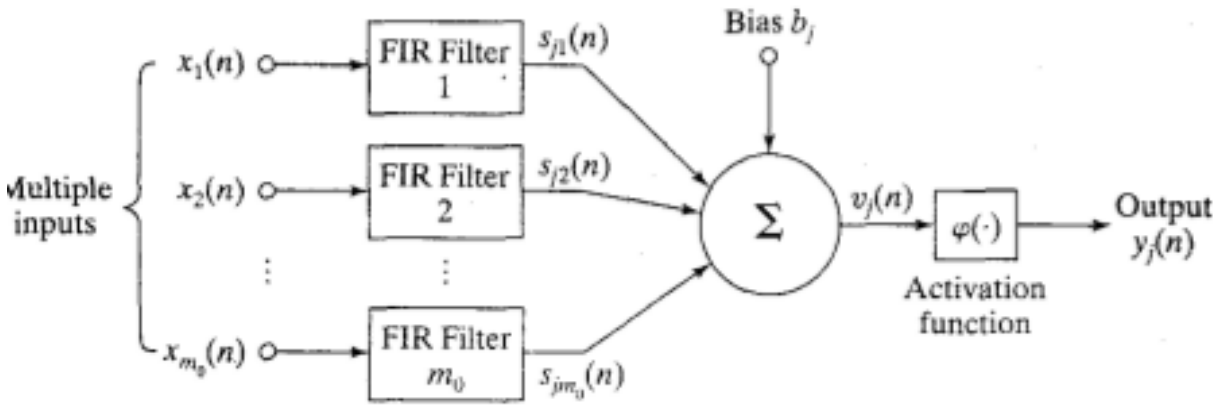


FIGURE 13.10 Focused time lagged feedforward network (TLFN); the bias levels have been omitted for convenience of presentation.

Models of Temporal Neural Networks

Spatio-temporal models



Multiple input neuronal filter.

$$\vec{y_j}(n) = \varphi\left(\sum_{i=0}^{m_0}\sum_{l=0}^{p} w_{ji}(l)x_i(n-l) + b_j\right)$$

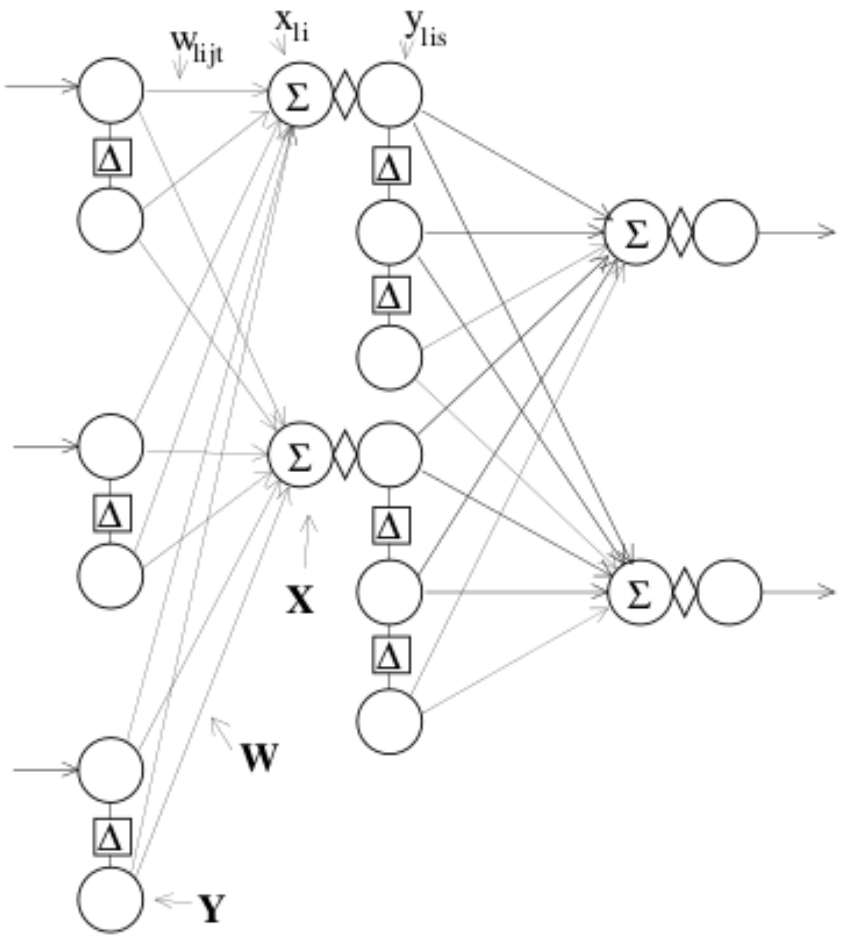
Temporal Neural Networks

Non-stationary processes

Distributed time lagged feedforward network

- The influence of time is distrubuted throughout the network.
- With each neuron we have a FIR that saves its last states – each neuron in the network is a multiple input neuronal filter.
- Learning using Temporal Back-Propagation.
- Model also explained in An Overview of Temporal Backpropagation (1991) by Timothy Edwards.

Distributed time lagged feedforward network



Temporal Back-Propagation

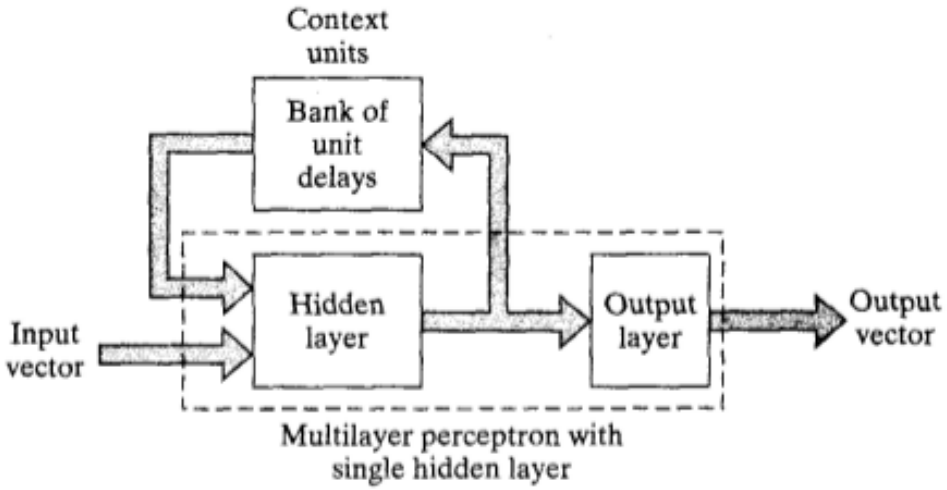
- Motivated by unfolding the network into a static one and following the known derivation of Back-Propagation. Formulas for $\Delta\vec{w}_{ij}$ are the same except in this case we have vectors.
- The more layers the network contains the more data need to be stored when learning because of causality constraints.
- When adapting the weights are changed in advance. The formulas work with older values – compensated by mild learning rates.
- More computationally demanding than simple Back-Propagation.
- When modeling stationary processes it is not necessary to use this model.
- Possible uses with chaotic systems in Wan 1994 – prediction of pulsation of NH_3 laser.

Recurrent Neural Networks

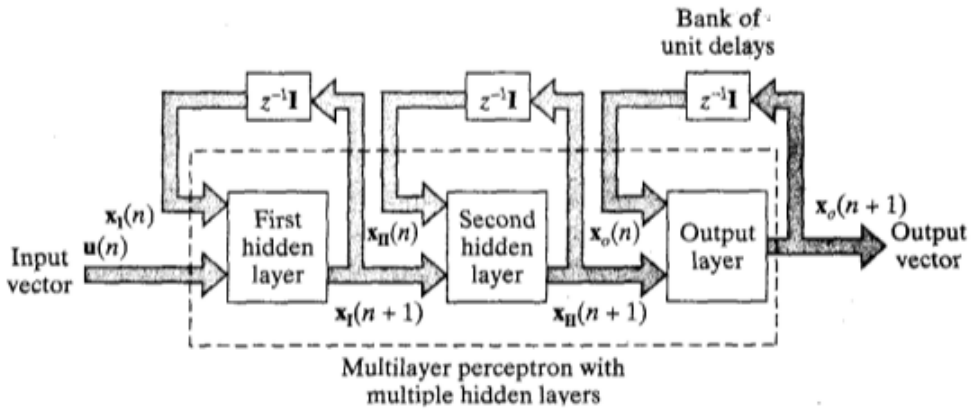
Models

- Memory is created also by the network itself. The network is no longer a DAG, it contains *feedback loops*.
- Network keeps its *state*.
- Multilayer perceptrons with one hidden layer with recurrencies (Elman type RNN).
- Perceptrons with multiple hidden layers.
- Second order models (best suited for modeling FSAs).
- Fully connected RNNs are referred to as NARX (Nonlinear Autoregressive with Exogeneous inputs).

RNN - models



Simple Recurrent network (SRN).



Recurrent multilayer perceptron.

State-space model

SRN - 1 layer

Let $\vec{x}$ be a state (neurons from hidden layer), $\vec{u}$ be the presented input and $\vec{y}$ be the output vector.

$$\vec{x}(n+1) = f(\vec{x}, \vec{u})$$
$$\vec{y}(n) = C\vec{x}(n)$$

Function f is usually computed as (matrices W_a and W_b denote weights between state neurons and states or inputs).

$$\vec{x}(n+1) = \varphi(W_a\vec{x}(n) + W_b\vec{u}(n))$$

Recurrent Neural Networks

Remarks

Computational power

- Every finite state automaton can be simulated by a RNN.
- Every Turing machine can be simulated by a RNN.
- Each RNN can be simulated by NARX with one hidden layer and a prescribed activation function (modification of the sigmoid). Such a NARX is almost SRN.
- McCulloch and Pitts, *Minsky*, Kleene.

Learning

- The algorithm are for learning of SRNs with one hidden layer.
- The learning can be done during processing so that the learning process never stops.

Recurrent Neural Networks

Learning

Epochwise learning

- A predicted sequence is presented to the RNN.
- Overall epoch error is computed and the weights are adapted.
- Each epoch we start in a different state different from the finish state of the last epoch. Possibly start from an initial state.
- Back-Propagation through time.

Continuous tranining

- Use when there are no reset states and/or on-line learning is required.
- RNN is learning and processing at the same time.
- Real-time recurrent algorithm.

Back-Propagation Through Time

- Based on the unfolding the network in epoch time $[n_0, n_1]$.
- Overall epoch error $\frac{1}{2}\sum_{n=n_0}^{n_1}\sum_k (d_k(n) - y_k(n))^2$ is used to adapt the weights.

Adaptation

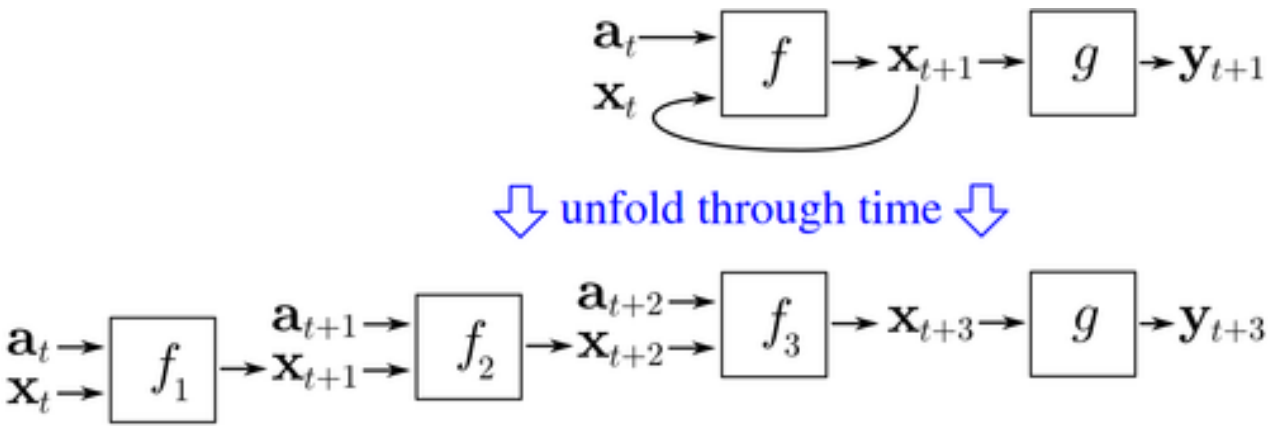
$$\Delta w_{ji}(n_1) = -\eta \sum_{n=n_0+1}^{n_1} \delta_j x_i(n-1)$$

$$\delta_j(n) = \varphi'(v_j(n)) e_j(n) \text{ for } n = n_1$$

$$\delta_j(n) = \varphi'(v_j(n)) \left(e_j(n) + \sum_k w_{jk}\delta_k(n+1) \right) \text{ for } n_0 < n < n_1$$

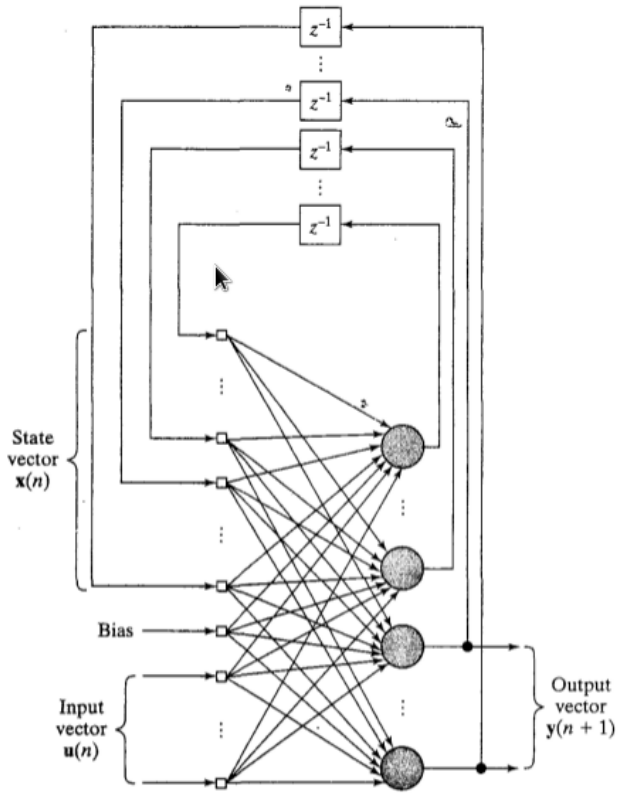
Truncated Back-Propagation Through Time

- Based on the same unfolding of the network as in the previous case.
- The time for which we account is bounded by a constant h.
- We distribute only the current error and thus we get a real-time algorithm.



Real-time recurrent algorithm

- Continuous training - each adaptation step must be fast.
- Approximation of the steepest descent.
- Does not respect overall epoch error. The weights are adapted just according to the last error as in non-batch Back-Propagation.



Fully connected recurrent network for formulation of the RTRL algorithm.

Real-time recurrent algorithm

Adapting weights

$$
\begin{aligned}
U_j(n) &= \text{zero matrix up to the } j\text{-th row containing } (\vec{x}(n), \vec{u}(n)) \\
\vec{\Lambda}_j(0) &= 0 \\
\phi(n) &= \text{diagonal matrix consisting of } \varphi'((\vec{w}_a, \vec{w}_b)^T(\vec{x}(n), \vec{u}(n))) \\
x(0) &= 0 \\
\vec{\Lambda}_j(n+1) &= \Phi(n)(W_a(n)\vec{\Lambda}_j(n) + Uj(n)) \\
\vec{e}(n) &= \vec{d}(n) - C\vec{x}(n) \\
\Delta\vec{w}_j(n) &= \eta C\vec{\Lambda}_j(n)\vec{e}(n)
\end{aligned}
$$

Improvements/Problems

- *Teacher forcing* – replace output of a neuron with the desired response. Leads to a faster training and does only corrections that keep already learned weights intact.
- Present samples in *lexigraphic order* and the shortest samples first.
- *Size* of the samples should be *incrementally increased*.
- Synaptic weights are adjusted only if the correction is greater than some prescribed criterion.
- Use of *weight decay* is recommended – prevents large weights.
- *Vanishing gradient* – RNN either has a long-term memory or is not robust to the presence of noise. Use extended *Kalman filters* for more efficient use of information.

Sources

- Simon Haykin: Neural Networks - A Comprehensive Foundation
- Timothy Edwards: An Overview of Temporal Backpropagation