Learning Beyond Finite Memory in Recurrent Networks Of Spiking Neurons

Peter Tiňo **Ashley J.S. Mills**
School Of Computer Science
University of Birmingham, UK

Some motivations

- A considerable amount of work has been devoted to studying computations on time series in a variety of connectionist models.

- RNN- feedback delay connections between the neural units

- Feedback connections endow RNNs with a form of 'neural memory' that makes them (theoretically) capable of processing time structures over *arbitrarily long* time spans.

- However, *induction* of nontrivial temporal structures beyond finite memory can be problematic.

- Useful benchmark - FSM. In general, one needs a notion of an abstract information processing state that can encapsulate histories of processed strings of *arbitrary* finite length.

Motivations cont'd

- RNNs have been based on traditional rate coding
- Controversial whether, when describing computations performed by a real biological system, one can abstract from the individual spikes and consider only macroscopic quantities, such as the number of spikes emitted by a single neuron (or a population of neurons) per time interval.
- Spiking neurons - the input and output information is coded in terms of *exact timings of individual spikes*
- Learning algorithms for acyclic spiking neuron networks have been developed.
- No systematic work on induction of deeper temporal structures.

Related work

- Maass (1996) proved that networks of spiking neurons with feedback connections (recurrent spiking neuron networks – RSNNs) can simulate Turing machines. No *induction* studies though.

- Natschläger and Maass (2002) - induction of finite memory machines (of depth 3) in feed-forward spiking neuron networks. A memory mechanism was implemented in a biologically realistic model of dynamic synapses (Maass & Markram, 2002).

- Floreano, Zufferey & Nicoud (2005) evolved controllers containing spiking neuron networks for vision-based mobile robots and adaptive indoor micro-flyers.

- Maass, Natschläger and H. Markram (2002) - liquid state machines with *fixed* recurrent neural circuits.

However ...

In such studies, there is usually a leap in the coding strategy from emphasis on spike timings in individual neurons (pulse coding) into more space-rate-based population codings.

We will strictly adhere to *pulse-coding*, e.g. all the input, *output* and *state* information is coded in terms of spike trains on subsets of neurons.

Natschläger and Ruf (1998):

... this paper is not about biology but about possibilities of computing with spiking neurons which are inspired by biology ... a thorough understanding of such simplified networks is necessary for understanding possible mechanisms in biological systems ...

Formal spiking neuron

Spike response model (Gerstner, 1995)

Spikes emitted by neuron i are propagated to neuron j through several synaptic channels $k = 1, 2, ..., m$, each of which has an associated synaptic efficacy (weight) w_{ij}^k, and an axonal delay d_{ij}^k.

In each synaptic channel k, input spikes get delayed by d_{ij}^k and transformed by a response function ϵ_{ij}^k which models the rate of neurotransmitter diffusion across the synaptic cleft.

Γ_j - the set of all (presynaptic) neurons emitting spikes to neuron j

Formal spiking neuron cont'd

The accumulated potential at time t on soma of unit j is

$$x_j(t) = \sum_{i \in \Gamma_j} \sum_{k=1}^{m} w_{ij}^k \cdot \epsilon_{ij}^k (t - t_i^a - d_{ij}^k), \tag{1}$$

where the response function ϵ_{ij}^k is modeled as:

$$\epsilon_{ij}^k(t) = \pm \cdot (t/\tau) \cdot exp(1 - (t/\tau)) \cdot \mathcal{H}(t). \tag{2}$$

τ - membrane potential decay time constant,
$\mathcal{H}(t)$ - the Heaviside step function

Neuron j fires a spike (and depolarizes) when the accumulated potential $x_j(t)$ reaches a threshold Θ.

Feed-forward spiking neuron network (FFSNN)

The first neurons to fire a spike are the input units (code the information to be processed by the FFSNN).

The spikes propagate to subsequent layers, finally resulting in a pattern of spike times across neurons in the output layer (response of FFSNN to the current input).

The input-to-output propagation of spikes through FFSNN is confined to a simulation interval of length Υ. All neurons can fire at most once within the simulation interval (neuron refractoriness).

Bohte, Kok and La Poutré (2002) - a back-propagation-like supervised learning rule for training FFSNN called *SpikeProp*.

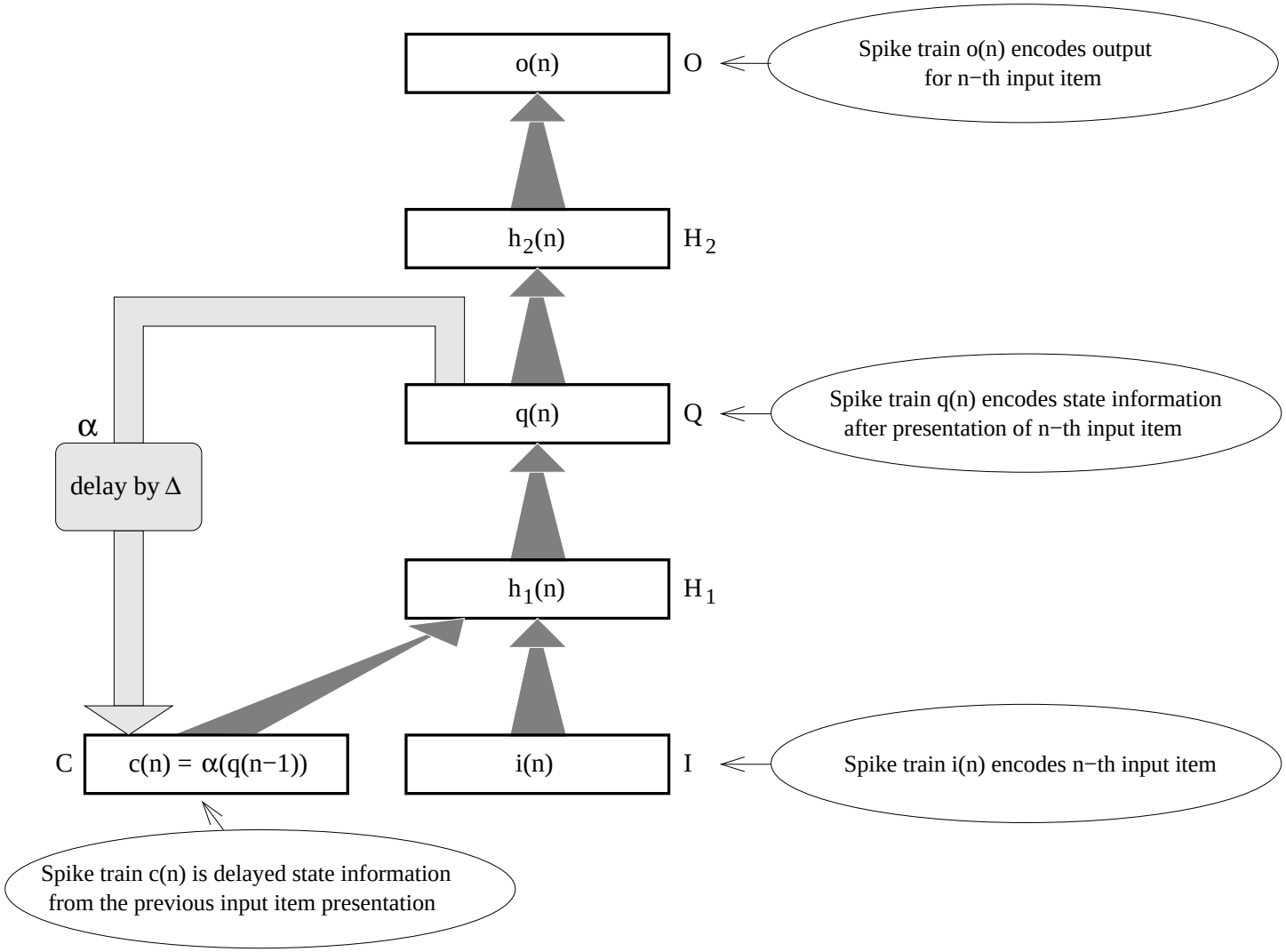
Temporal dependencies?

FFSNN cannot properly deal with temporal structures in the input stream that go beyond finite memory.

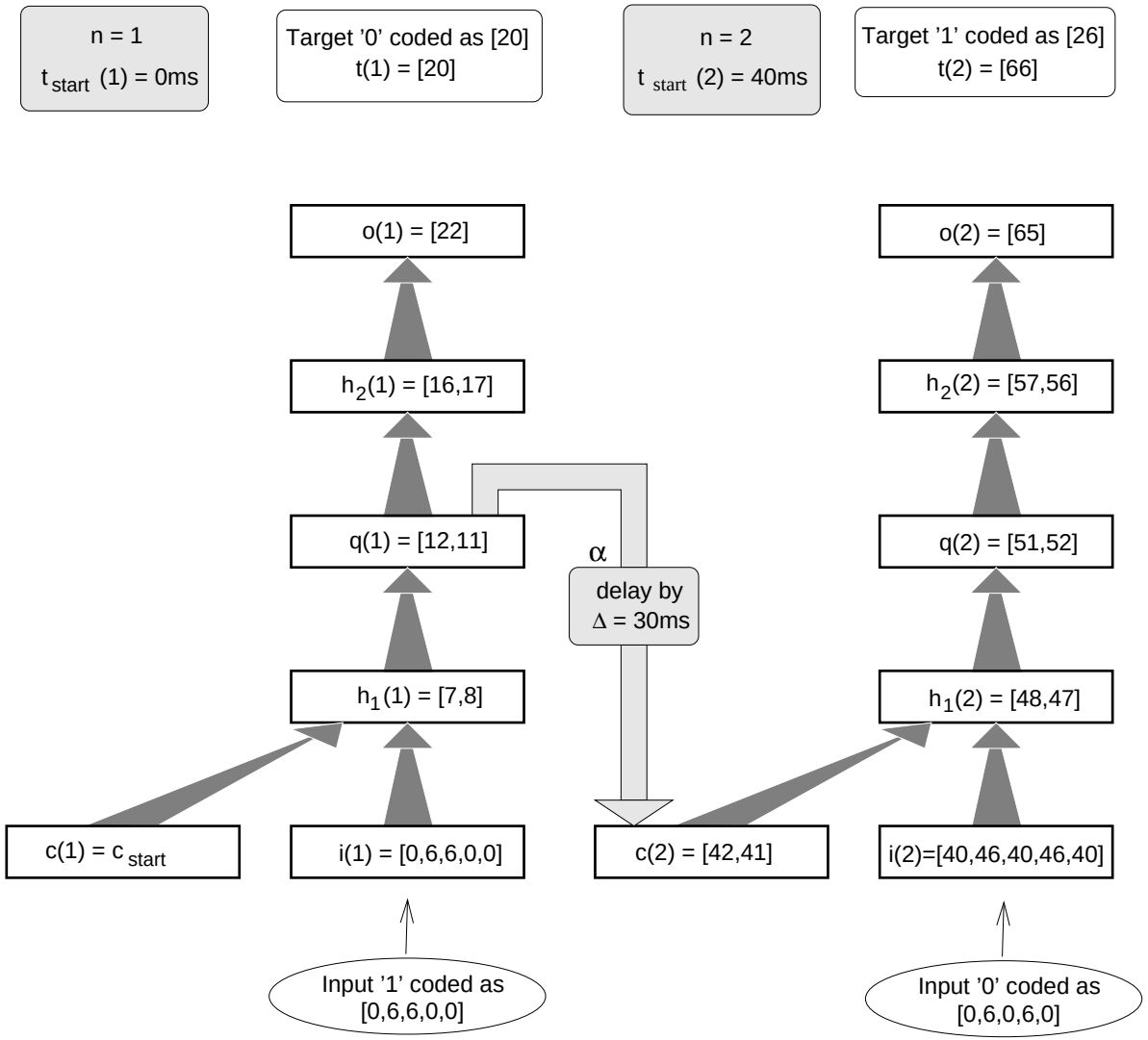
Turn FFSNN into a recurrent spiking neuron network (RSNN) by extending the feedforward architecture with feedback connections.

Select a hidden layer in FFSNN as the layer responsible for coding (through spike patterns) important information about the history of inputs seen so far (recurrent layer). Feed back its spiking patterns through the delay synaptic channels to an auxiliary layer at the input level, called the context layer.

Recurrent spiking neuron network (RSNN)



Unfold RSNN in time



Training RSNN - SpikePropThroughTime

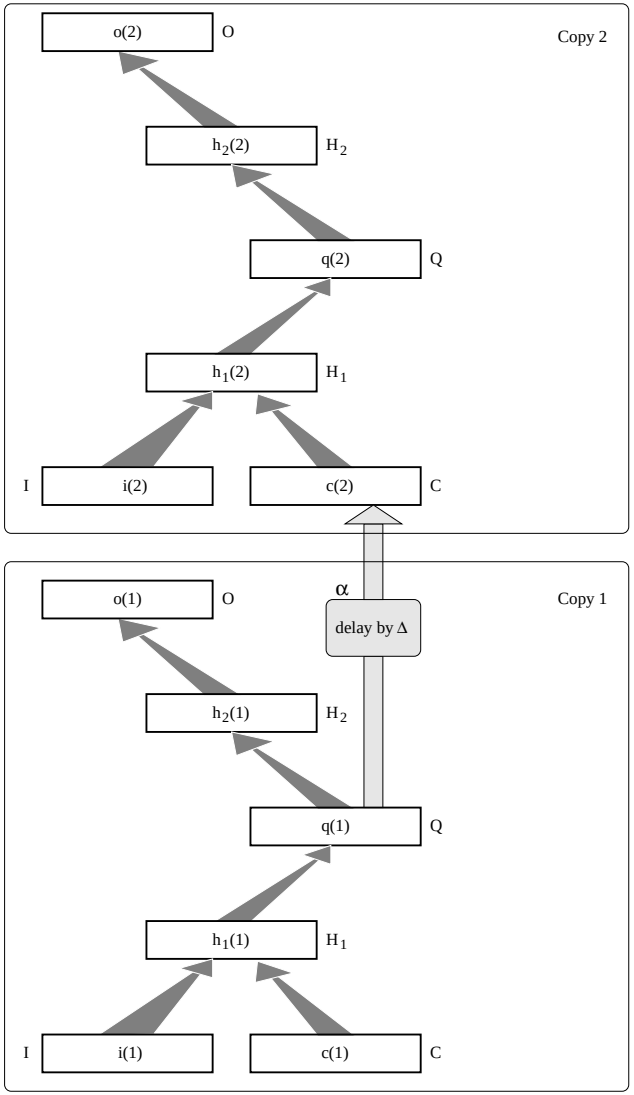
Given an input string of length n, n copies of the base RSNN are stacked on top of each other.

Firing times in the first copy are relative to 0. For copies $n > 1$, the external inputs and desired outputs are made relative to $t_{start}(n) = (n-1) \cdot \Upsilon$.

Adaptation proportions are calculated for weights in each of the network copies. The weights in the base network are then updated by adding up, for every weight, the n corresponding weight-updates.

Special attention must be paid when calculating weight adaptations for neurons in the recurrent layer Q.

SpikePropThroughTime



Encoding the input

Input alphabets of one, or two symbols.

Moreover, a special end-of-string symbol '2' initiating transitions to the initial FSM state.

The input layer I had five neurons. The input symbols '0', '1' and '2' are encoded in the five input units through spike patterns $\mathbf{i}_0 = [0, 6, 0, 6, 0]$, $\mathbf{i}_1 = [0, 6, 6, 0, 0]$ and $\mathbf{i}_2 = [6, 0, 0, 6, 0]$, respectively (firing times are in ms).

The last input neuron acts like a reference neuron always firing at the beginning of any simulation interval.

Encoding the output

Binary output alphabet $V = \{0, 1\}$.

The output layer O consisted of a single neuron.

Spike patterns (in ms) in the output neuron for output symbols '0' and '1' are $\mathbf{o}_0 = [20]$ and $\mathbf{o}_1 = [26]$, respectively.

Moore machines

One of the simplest computational models that encapsulates the concept of unbounded input memory

Initial Moore machine (MM) M is a 6-tuple $M = (U, V, S, \beta, \gamma, s_0)$
U and V are finite input and output alphabets, respectively,
S is a finite set of states, $s_0 \in S$ is the initial state,
$\beta : S \times U \to S$ is the state transition function
$\gamma : S \to V$ is the output function.

Given an input string $u = u_1 u_2 ... u_n$ over U the machine M responds with the output string $v = M(u) = v_1 v_2 ... v_n$ over V: Start in the initial state s_0, then for all $i = 1, 2, ..., n$, the new state is recursively determined, $s_i = \beta(s_{i-1}, u_i)$, and the machine emits the output symbol $v_i = \gamma(s_i)$.

Experimental setup

5 neurons in layers I, C, H_1, Q and H_2.

Within each of those layers, one neuron was inhibitory, all the other ones were excitatory.

Each connection between neurons had $m = 16$ synaptic channels, with delays $d_{ij}^k = k$, $k = 1, 2, ..., m$, realizing axonal delays between $1ms$ and $16ms$.

The decay constant τ in response functions ϵ_{ij} was set to $\tau = 3$.

The length Υ of the simulation interval was set to 40ms.

The delay Δ was 30ms.

Cyclic machines

'cyclic' machine C_p of period $p \geq 2$: $U = \{0\}$; $V = \{0, 1\}$; $S = \{0, 1, 2, ..., p-1\}$; $s_0 = 0$; for $0 \leq i < p-1$, $\beta(i, 0) = i + 1$ and $\beta(p-1, 0) = 0$; $\gamma(0) = 0$ and for $0 < i \leq p-1$, $\gamma(i) = 1$.

The network can only observe inputs. These MMs require an *unbounded input memory buffer*.

The RSNN perfectly learned machines C_p, $2 \leq p < 5$ (no deviations from expected behavior were observed over test sets having length of the order 10^4).

The training set had to be incrementally constructed by iteratively training with one presentation of the cycle, then two presentations etc.

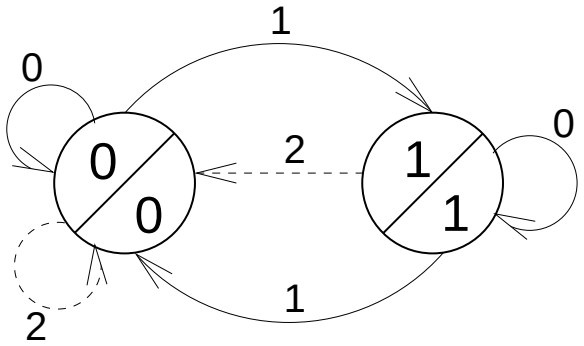
Extract machines from trained RSNN

In analogy with previous work in the domain of rate-based RNN, we cluster the spike trains in the recurrent layer of RSNN into a finite number of 'similar' recurrent normalized spike trains representing abstract information processing states induced by RSNN.

Using the clusters we can 'extract' MM from RSNN. Extracted MM are minimized into the canonical form.

Using the successful networks, we extracted unambiguously all the machines C_p of period $1 \leq p < 5$. The number of clusters in k-means clustering was set to 10.

Two-state machine M_2

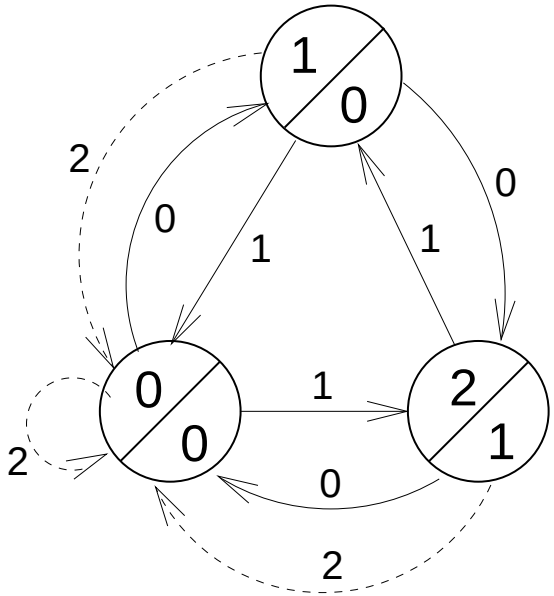


RSNN perfectly learned the machine.

No mechanism with vanishing input memory can implement string mappings defined by this Moore machine.

Using the successful networks, we extracted unambiguously the machine M_2 (the number of clusters in k-means clustering was 10).

Partial induction

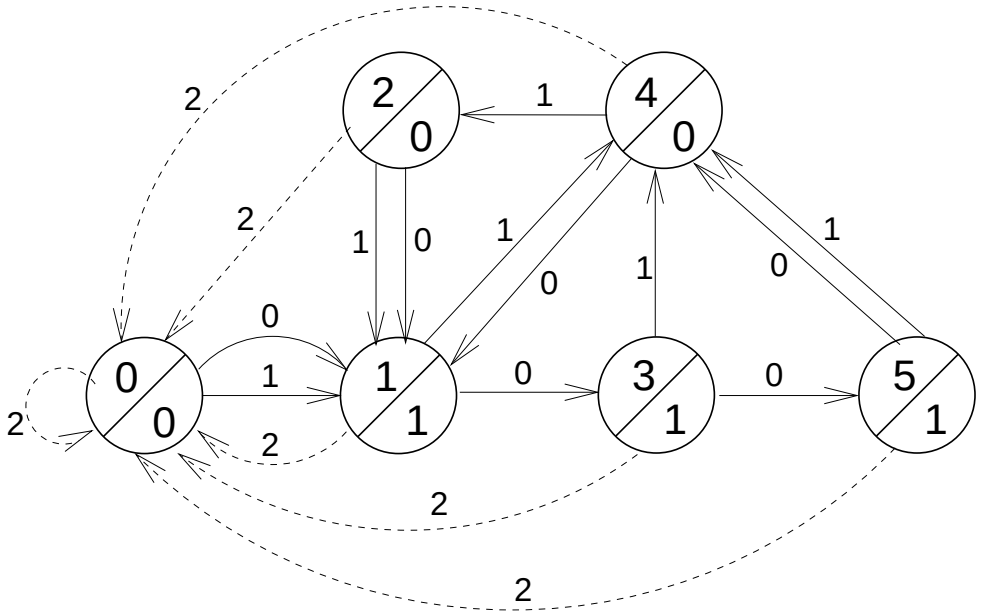


Machine M_3 - two main fixed-input cycles in opposite directions.

Training lead to an error rate of $\approx 0.3ms$ over test strings of length 10000.

Lessons can be learnt by studying extracted machines $\tilde{M}_3$.

Extracted machine



The cycle on input '1' in M_3 has been successfully induced, but the cycle on input '0' has not (cycle of length 4).

The oscillation between states 4 and 1 on strings $\{01\}^+$ in $\tilde{M}_3$ corresponds with the oscillation between states 1 and 2 in M_3.

Discussion

- We were able to train RSNN to mimic target MMs requiring unbounded input memory on only a relatively simple set of MMs.

- Compared with traditional rate-based RNN, two major problems:
  - There are two timescales the network operates on: **(i)** shorter timescale of spike trains coding the input/output/state information within a single simulation interval; and **(ii)** longer timescale of sequences of simulation intervals, each representing a single input-to-output processing step.
  - Discontinuities in the error-surface caused by the spike producing mechanism.

Discussion cont'd

All gradient-based methods will have problems in locating good minima on such error surfaces.

We varied the numbers of neurons in hidden/recurrent layers and tried (without much success)

- fast evolutionary strategies, (30,200)-ES, Cauchy mutation function (Yao & Liu, 1997; Yao, 1999),
- (extended) Kalman filtering in the parameter space (Puskorius & Feldkamp, 2002), and
- evolutionary method of Rowe & Hidovic (2004) for optimization on real-valued domains.

The abrupt and erratic nature of the error surface makes it hard, even for evolutionary techniques, to locate a good minimum.