Learning Beyond Finite Memory in Recurrent Networks Of Spiking Neurons

Peter Tiňo[1] Ashley Mills[1]

School Of Computer Science, University Of Birmingham, Birmingham B15 2TT, UK
P.Tino@cs.bham.ac.uk, msc57ajm@cs.bham.ac.uk

Abstract. We investigate possibilities of inducing temporal structures without fading memory in recurrent networks of spiking neurons strictly operating in the pulse-coding regime. We extend the existing gradient-based algorithm for training feed-forward spiking neuron networks (*SpikeProp* [1]) to recurrent network topologies, so that temporal dependencies in the input stream are taken into account. It is shown that temporal structures with unbounded input memory specified by simple Moore machines (MM) can be induced by recurrent spiking neuron networks (RSNN). The networks are able to discover *pulse-coded* representations of abstract information processing states coding potentially unbounded histories of processed inputs.

1 Introduction

A considerable amount of work has been devoted to studying computations on time series in recurrent neural networks (RNNs). 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, even though RNNs are capable of simulating Turing machines [2], *induction* of non-trivial temporal structures beyond finite memory can be problematic [3]. Finite state machines (FSMs) and automata constitute a simple, yet well established and easy to analyze framework for describing temporal structures that go beyond finite memory relationships. In general, for a finite description of the string mapping realized by an FSM, one needs a notion of an abstract information processing state that can encapsulate histories of processed strings of *arbitrary* finite length. Indeed, FSMs have been a popular benchmark in the recurrent network community and there is a huge amount of literature dealing with empirical and theoretical aspects of learning finite state machines/automata in RNNs (e.g. [4, 5]).

However, the RNNs under consideration have been based on traditional rate-coded artificial neural network models that describe neural activity in terms of *rates* of spikes[1] produced by individual neurons. Several models of spiking neurons, where the input and output information is coded in terms of *exact timings of individual spikes* (*pulse coding*) have been proposed (see e.g. [6]).

[1] identical electrical pulses also known as action potentials

Learning algorithms for acyclic networks of such (biologically more plausible) artificial neurons have been developed and tested [1, 7].

Maass [8] proved that networks of spiking neurons with feedback connections (recurrent spiking neuron networks – RSNNs) can simulate Turing machines. Yet, virtually no systematic work has been reported on *inducing* deeper temporal structures in such networks. There are recent developments along this direction, e.g. [9, 10]. Such studies, however, usually make a leap in the coding strategy, shifting from the emphasis on spike timings in individual neurons (pulse coding) into more space-rate-based population codings. Even though most of experimental research focuses on characterizations of potential information processing states using temporal statistics of rate properties in spike trains (e.g. [11]) there is some experimental evidence that in certain situations the temporal information may be pulse-coded [12].

In this study we are concerned with possibilities of *inducing* deep temporal structures *without fading memory* in recurrent networks of spiking neurons. 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.

2 Recurrent Spiking Neural Network

First, we briefly describe the formal model of spiking neurons, the spike response model [13], employed in this study. 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. The response function can be either excitatory, or inhibitory.

Formally, denote the set of all (presynaptic) neurons emitting spikes to neuron j by Γ_j. Let the last spike time of a presynaptic neuron $i \in \Gamma_j$ be t_i^a. 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:

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

Here, σ_{ij}^k is 1 and -1 if the synapse k between neurons i, j is concerned with transmitting excitatory and inhibitory, respectively. The decay constant τ governs the rate at which neurotransmitter released from the presynaptic membrane reaches the post synaptic membrane. $\mathcal{H}(t)$ is the Heaviside step function which is 1 for $t > 0$, and is otherwise 0, ensuring that the axonal delay d_{ij}^k is enforced. Neuron j fires a spike (and depolarizes) when the accumulated potential $x_j(t)$ reaches a threshold Θ.

In a feed-forward spiking neuron network (FFSNN), the first neurons to fire a spike are the input units. Spatial spike patterns across input neurons 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. The output spike times represent the 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[2]. After the simulation interval has expired, the output spike pattern is read-out and interpreted and a new simulation interval is initialized by presenting a new input spike pattern in the input layer. Given a mechanism for temporal encoding and decoding of the input and output information, respectively, Sander, Bohte and Kok have recently formulated a back-propagation-like supervised learning rule for training FFSNN, called *SpikeProp* [1]. Synaptic efficacies on connections to the output unit j are updated as follows:

$$\Delta w_{ij}^k = -\eta \cdot \epsilon_{ij}^k (t_j^a - t_i^a - d_{ij}^k) \cdot \delta^j, \tag{3}$$

where

$$\delta^j = \frac{(t_j^d - t_j^a)}{\sum_{i \in \Gamma_j} \sum_{k=1}^{m} w_{ij}^k \cdot \epsilon_{ij}^k (t_j^a - t_i^a - d_{ij}^k) \cdot (1/(t_j^a - t_i^a - d_{ij}^k) - 1/\tau)} \tag{4}$$

and $\eta > 0$ is the learning rate. The numerator is the difference between the desired t_j^d and actual t_j^a firing times of the output neuron j within the simulation interval.

Synaptic efficacies on connections to the hidden unit i are updated analogously:

$$\Delta w_{hi}^k = -\eta \cdot \epsilon_{hi}^k (t_i^a - t_h^a - d_{hi}^k) \cdot \delta^i, \tag{5}$$

where

$$\delta^i = \frac{\sum_{j \in \Gamma^i} (\sum_{k=1}^{m} w_{ij}^k \cdot \epsilon_{ij}^k (t_j^a - t_i^a - d_{ij}^k) \cdot (1/(t_j^a - t_i^a - d_{ij}^k) - 1/\tau)) \cdot \delta^j}{\sum_{h \in \Gamma_i} \sum_{k=1}^{m} w_{hi}^k \cdot \epsilon_{hi}^k (t_i^a - t_h^a - d_{hi}^k) \cdot (1/(t_i^a - t_h^a - d_{hi}^k) - 1/\tau)} \tag{6}$$

and Γ^i denotes the set of all (postsynaptic) neurons to which neuron i emits spikes. The numerator pulls in contributions from the layer succeeding that for which δ's are being calculated[3].

Obviously, FFSNN cannot properly deal with temporal structures in the input stream that go beyond finite memory. One possible solution is to turn FFSNN into a recurrent spiking neuron network (RSNN) by extending the feed-forward architecture with feedback connections. In analogy with RNN, we select

[2] The period of neuron refractoriness (a neuron is unlikely to fire shortly after producing a spike), is not modeled, and thus to maintain biological plausibility a neuron may only fire once within the simulation interval (see e.g. [1]).

[3] When a neuron does not fire, its contributions are not incorporated into the calculation of δ's for other neurons, neither is a δ calculated for it.

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) and feed back its spiking patterns through the delay synaptic channels to an auxiliary layer at the input level, called the context layer.* The input and context layers now collectively form a new 'extended input layer' of the RSNN. The delay feedback connections temporally translate spike patterns in the recurrent layer by the delay constant Δ,

$$\alpha(t) = t + \Delta. \tag{7}$$

Such temporal translation can be achieved using a FFSNN.

The RSNN architecture used in our experiments consists of five layers. The extended input layer (input and context layers, denoted by I and C, respectively) feeds the first auxiliary hidden layer H_1, which in turn feeds the recurrent layer Q. Within each simulation interval, the spike timings of neurons in the input and context layers I and C are stored in the spatial spike train vectors $\mathbf{i}$ and $\mathbf{c}$, respectively. The spatial spike trains of the first hidden and recurrent layers are stored in vectors $\mathbf{h}_1$ and $\mathbf{q}$, respectively. The role of the recurrent layer Q is twofold:

1. The spike train $\mathbf{q}$ codes information about the history of inputs seen so far. This information is passed to the next simulation interval through the delay FFSNN network[4] α, $\mathbf{c} = \alpha(\mathbf{q})$. The delayed spatial spike train $\mathbf{c}$ appears in the context layer. Spike train $(\mathbf{i}, \mathbf{c})$ of the extended input consists of the history-coding spike train $\mathbf{c}$ and a spatial spike train $\mathbf{i}$ coding the external inputs (input symbols).
2. The recurrent layer feeds the second auxiliary hidden layer H_2, which finally feeds the output layer O. The spatial spike trains in the second hidden and output layers are stored in vectors $\mathbf{h}_2$ and $\mathbf{o}$, respectively.

Parameters, such as the length Υ of the simulation interval, feedback delay Δ and spike time encodings of input/output symbols have to be carefully coordinated. The simulation interval for processing of the nth input item starts at

$$t_{start}(n) = (n-1) \cdot \Upsilon.$$

Absolute desired output spike times for the nth input need to be adjusted with the $t_{start}(n)$. The initial context spike pattern, $\mathbf{c}_{start}$, is imposed externally at the beginning of training. The firing times of the recurrent neurons at the end of simulation interval n, $\mathbf{q}(n)$, are translated in time by the delay FFSNN α to give the state (context) inputs $\mathbf{c}(n+1)$ at the start of simulation epoch $(n+1)$. The simulation interval Υ needs to be set so that temporal proximity of $\mathbf{c}(n+1)$ and the input firing times $\mathbf{i}(n+1)$ at the start of simulation epoch $(n+1)$ is achieved.

[4] the delay function $\alpha(t)$ is applied to each component of $\mathbf{q}$

2.1 Training – SpikePropThroughTime

We extended the SpikeProp algorithm [1] for training FFSNN to recurrent models in the spirit of Back Propagation Through Time for rate-based RNN [14], i.e. using the unfolding-in-time methodology. We call this learning algorithm *SpikePropThroughTime*.

Given an input string of length n, n copies of the base RSNN are made, stacked on top of each other, and sequentially simulated, incrementing t_{start} by Υ after each simulation interval. Expanding the base network through time via multiple copies simulates processing of the input stream by the base RSNN.

Adaptation δ's (see equations (4) and (6)) are calculated for each of the network copies. The synaptic efficacies (weights) in the base network are then updated using δ's calculated in each of the copies by adding up, for every weight, the n corresponding weight-update contributions of equations (3) and (5).

In a FFSNN, when calculating the δ's for a hidden layer, the firing times from the preceding and succeeding layers are used. Special attention must be paid when calculating δ's of neurons in the recurrent layer Q. Context spike train $\mathbf{c}(n+1)$ in copy $(n+1)$ is the delayed recurrent spike train $\mathbf{q}(n)$ from the nth copy. The relationship of firing times in $\mathbf{c}(n+1)$ and $\mathbf{h}_1(n+1)$ contains the information that should be incorporated into the calculation of the δ's for recurrent units in copy n. The delay constant Δ is subtracted from the firing times $\mathbf{h}_1(n+1)$ of H_1 and then, when calculating the δ's for recurrent units in copy n, these temporally translated firing times are used as if they were simply another hidden layer succeeding Q in copy n. Denoting by $\Gamma_{2,n}$ the set of neurons in the second auxiliary hidden layer H_2 of the nth copy and by $\Gamma_{1,n+1}$ the set of neurons in the first auxiliary hidden layer H_1 of the copy $(n+1)$, the δ of the ith recurrent unit in the nth copy is calculated as

$$\begin{aligned}\delta^i = &\frac{\sum_{j\in\Gamma_{1,n+1}}(\sum_{k=1}^{m} w_{ij}^k \cdot \epsilon_{ij}^k(t_j^a - \Delta - t_i^a - d_{ij}^k)\cdot(1/(t_j^a - \Delta - t_i^a - d_{ij}^k) - 1/\tau))\cdot\delta^j}{\sum_{h\in\Gamma_i}\sum_{k=1}^{m} w_{hi}^k \cdot \epsilon_{hi}^k(t_i^a - t_h^a - d_{hi}^k)\cdot(1/(t_i^a - t_h^a - d_{hi}^k) - 1/\tau)} \\ &+ \frac{\sum_{j\in\Gamma_{2,n}}(\sum_{k=1}^{m} w_{ij}^k \cdot \epsilon_{ij}^k(t_j^a - t_i^a - d_{ij}^k)\cdot(1/(t_j^a - t_i^a - d_{ij}^k) - 1/\tau))\cdot\delta^j}{\sum_{h\in\Gamma_i}\sum_{k=1}^{m} w_{hi}^k \cdot \epsilon_{hi}^k(t_i^a - t_h^a - d_{hi}^k)\cdot(1/(t_i^a - t_h^a - d_{hi}^k) - 1/\tau)}\end{aligned} \tag{8}$$

3 Learning beyond finite memory in RSNN – inducing Moore machines

One of the simplest computational models that encapsulates the concept of unbounded input memory is the Moore machine [15]. Formally, an (initial) Moore machine (MM) M is a 6-tuple $M = (U, V, S, \beta, \gamma, s_0)$, where 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 and $\gamma : S \to V$ is the output function. Given an input string $u = u_1u_2...u_n$ of symbols from U ($u_i \in U$, $i = 1, 2, ..., n$), the machine M acts as a transducer by responding with the output string $v = M(u) = v_1v_2...v_n$, $v_i \in V$, computed as follows: first

the machine is initialized with 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 $v_i = \gamma(s_i)$.

Given a target Moore machine M, a set of training examples is constructed by explicitly constructing input strings u over U and then determining the corresponding output string $M(u)$ over V (by traversing edges of the graph of M, starting in the initial state, as prescribed by the input string u). The training set $\mathcal{D}$ consists of N couples of input-output strings,

$\mathcal{D} = \{(u^1, M(u^1)), (u^2, M(u^2)), ..., (u^N, M(u^N))\}$.

In this study, we consider binary input and output alphabets $U = V = \{0, 1\}$ and a special end-of-string symbol '2'. We adopt the strategy of *inducing the initial state* in the recurrent network (as opposed to externally imposing it – see [16, 17]). The context layer of the network is initialized with the fixed predefined context spike train $\mathbf{c}(1) = \mathbf{c}_{start}$ only at the beginning of training. From the network's point of view, the training set is a couple $(\tilde{u}, M(\tilde{u})$ of the long concatenated input sequence

$$\tilde{u} = u^1 2 u^2 2 u^3 2 ... 2 u^{N-1} 2 u^N 2$$

and the corresponding output sequence

$$M(\tilde{u}) = M(u^1)\gamma(s_0)M(u^2)\gamma(s_0)M(u^3)\gamma(s_0)...\gamma(s_0)M(u^{N-1})\gamma(s_0)M(u^N)\gamma(s_0).$$

Input symbol '2' is instrumental in inducing the start state by acting as an end-of-string reset symbol initiating transition from every state of M to the initial state s_0.

The external input spike train $\mathbf{i}$ is partitioned into two disjoint sets of firing patterns: input neurons $\mathbf{i}^s$ coding the current input symbol, and reference neurons $\mathbf{i}^r$ which always fire at the same time relative to t_{start} in any simulation interval. Conversion of input symbols into spike trains $\mathbf{i}^s$ is described in table 1. We convert symbols into binary bitstrings and then encode each binary bit of the bitstring as alternating *high* ($6ms$) and *low* ($0ms$) firing times.

Table 1. Encoding of inputs 0, 1 and 2 in the input spike train $\mathbf{i}^s$.

input	Bit1		Bit2	
0	0	6	0	6
1	0	6	6	0
2	6	0	0	6

The network is trained using *SpikePropThroughTime* (section 2.1) to minimize the squared error between the desired output spike trains derived from $M(\tilde{u})$ when the RSNN is driven by the input $\tilde{u}$. The RSNN is unfolded and *SpikePropThroughTime* is applied for each training pattern $(u^i, M(u^i))$, $i = 1, 2, ..., N$.

The firing threshold is almost always 50, and the weights are initialized to random values from the interval[5] $(0, 10)$. We use a dynamic learning rate strategy that detects oscillatory behavior and plateaus within the error space. The action to take upon detecting oscillation or plateau, is respectively to decrease the learning rate by multiplying by an 'oscillation-counter-coefficient' (< 1), or increase the learning rate by multiplying by a 'plateau-counter-coefficient' (> 1) (see e.g. [18]).

The network had one output neuron 5 neurons in each of the layers I, C, H_1, Q and H_2 (1 inhibitory neuron and 4 excitatory neurons). Each connection between neurons had $m = 16$ synaptic channels, realizing axonal delays between $1ms$ and $16ms$. with delays $d_{ij}^k = k$, $k = 1, 2, ..., m$. 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. We used *SpikePropThroughTime* to train RSNN. The training was error-monitored and training was stopped the network had perfectly learned the target (zero thresholded output error). The maximum number of training epochs (sweeps through the training set) was 10000.

First, we experimented with 'cyclic' automata C_p of period $p \geq 2$: $U = \{0\}$; $V = \{0, 1\}$; $S = \{1, 2, ..., p\}$; $s_0 = 1$; for $1 \leq i < p$, $\beta(i, 0) = i+1$ and $\beta(p, 0) = 1$; $\gamma(1) = 0$ and for $1 < i \leq p$, $\gamma(i) = 1$. The RSNN perfectly learned machines C_p, $2 \leq p \leq 5$. The training set had to be incrementally constructed by iteratively training with one presentation of the cycle, then two presentations etc. We examined state information in RSNN coded as *spike trains* in the recurrent layer Q. The abstract information processing states extracted by the network during the training manifested themselves as groupings of normalized spike trains[6] $(\mathbf{q}(n) - t_{start}(n))$. We were able to use the emerging clusters of normalized spike trains to extract abstract knowledge induced in the network in the form of MM, but a detailed account of this issue is beyond the scope of this paper.

Second, we trained RSNN on a two-state machine M_2: $U = V = \{0, 1\}$; $S = \{1, 2\}$; $s_0 = 1$; $\beta(1, 0) = 1$, $\beta(2, 0) = 2$, $\beta(1, 1) = 2$ and $\beta(2, 1) = 1$; $\gamma(1) = 0$ and $\gamma(2) = 1$. Repeated presentation of only 5 carefully selected training patterns of length 4 were sufficient for a perfect induction of this machine. Again, we observed that the two abstract information processing states of M_2 were induced in the network as two clusters of normalized spike trains in the recurrent layer Q.

We stress, that given that the network can only observe the inputs, the above MMs would require an *unbounded input memory buffer*. So no mechanism with vanishing (input) memory can implement string mappings represented by such MMs. After training, the respective RSNN emulated the operation of these

[5] The weights must be initialized so that the neurons in subsequent layers are sufficiently excited by those in their previous layer that they fire, otherwise the network would be unusable. There is no equivalent in traditional rate-based neural networks to the non-firing of a neuron in this sense. The setting of initial weights and firing thresholds used in this study follows that of [7].

[6] The spike times $\mathbf{q}(n)$ entering the quantization phase are made relative to the start time $t_{start}(n)$ of the simulation interval

MM perfectly and apparently indefinitely – the networks had zero test (output-thresholded) error over test sets having length of the order 10^4.

We experimented with more complicated forms of MMs, but did not succeed to train them fully. However, the machines were at least partially induced, and the nature of such partial induction could be understood by the cluster analysis of the normalized recurrent spike trains, but, again, this issue is beyond the scope of this paper.

4 Discussion

We were able to train RSNN to mimic target MMs requiring unbounded input memory on only a relatively simple set of MMs. Spiking neurons used in this paper produce a spike only when the accumulated potential $x_j(t)$ reaches a threshold Θ. This leads to discontinuities in the error-surface. Gradient-based methods for training feed-forward networks of spiking neurons alleviate this problem by resorting to simplifying assumptions on spike patterns within a single simulation interval [1]. The situation is much more complicated in the case of RSNN. A small weight perturbation can prevent a recurrent neuron from firing in the shorter time scale of a simulation interval. That in turn can have serious consequences for further long-time-scale processing. Especially so, if such a change in short-term behavior appears at the beginning of presentation of a long input string.

The error surface becomes erratic, as evidenced in Figure 1. We took a RSNN trained to perfectly mimic the cycle-four machine C_4. We studied the influence of perturbing weights $\mathbf{w}_*$ in recurrent part of the RSNN (e.g. between layers I, C, H_1 and Q) on the test error calculated on long test string of length 1000. For each weight perturbation extent ρ, we randomly sampled 100 weight vectors $\mathbf{w}$ from the hyperball of radius ρ centred at $\mathbf{w}_*$. Shown are the mean and standard deviation values of the absolute output error per symbol for $0 < \rho \leq 3$. Clearly, small perturbations of weights lead to large abrupt changes in the test error.

Obviously, gradient-based methods, like our *SpikePropThroughTime* have problems in locating good minima on such error surfaces. We tried

- fast evolutionary strategies (FES) with (recommended) configuration (30,200)-ES[7], employing the Cauchy mutation function [19],
- (extended) Kalman filtering in the parameter space [20], and
- a recent powerful evolutionary method for optimisation on real-valued domains [21]

to find RSNN parameters, but without much success. The abrupt and erratic nature of the error surface makes it hard, even for evolutionary techniques, to locate a good minimum.

We tried RSNNs with varying numbers of neurons in the hidden and recurrent layers. In general, the increased representational capacity of RSNNs with more

[7] in each generation 30 parents generate 200 offspring through recombination and mutation

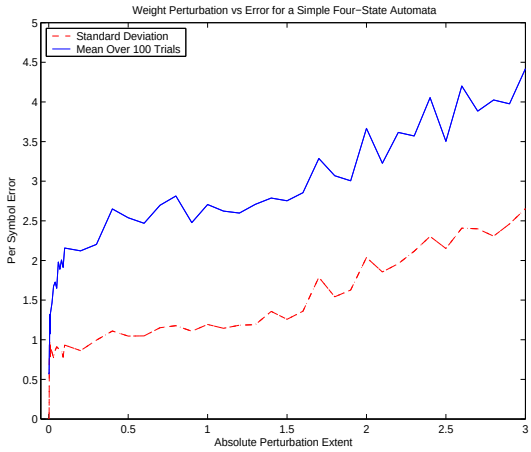

Fig. 1. Maximum radius of weight perturbation vs test error of RSNN trained to mimic the cycle-four machine C_4. For each setting of weight perturbation extent ρ, we randomly sampled 100 weight vectors $\mathbf{w}$ from the hyperball of radius ρ centred at induced RSNN weights $\mathbf{w}_*$. Shown are the mean (solid line) and standard deviation (dashed line) values of the absolute output error per symbol.

neural units could not be utilized because of the problems with finding good weight settings due the erratic nature of the error surface.

We note that the finite memory machines of [9] induced in feed-forward spiking neuron networks with dynamic synapses [22] were quite simple (of depth 3). The input memory depth is limited by the feed-forward nature of such networks. As soon as one tries to increase processing capabilities of spiking networks by introducing feedback connections, while insisting on pulse-coding, the induction process becomes complicated. Theoretically, it is perfectly possible to emulate any MM in RSNN. However, weight changes in RSNN lead to complex bifurcation mechanisms, making it hard to induce more complex MM through a guided search in the weight space. It is plausible that in biological systems, long-term dependencies are represented using rate-based codings and/or a Liquid State Machine mechanism [23] with a complex, but non-adaptable, recurrent pulse-coded part.

References

[1] Bohte, S., Kok, J., Poutré, H.L.: Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing **48** (2002) 17–37

[2] Siegelmann, H., Sontag, E.: On the computational power of neural nets. Journal of Computer and System Sciences **50** (1995) 132–150

[3] Bengio, Y., Frasconi, P., Simard, P.: The problem of learning long-term dependencies in recurrent networks. In: Proceedings of the 1993 IEEE International Conference on Neural Networks. Volume 3. (1993) 1183–1188

[4] Giles, C., Miller, C., Chen, D., Chen, H., Sun, G., Lee, Y.: Learning and extracting finite state automata with second–order recurrent neural networks. Neural Computation **4** (1992) 393–405
[5] Casey, M.: The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. Neural Computation **8** (1996) 1135–1178
[6] Gerstner, W.: Spiking neurons. In Maass, W., Bishop, C., eds.: Pulsed Coupled Neural Networks. MIT Press, Cambridge (1999) 3–54
[7] Moore, S.: Back propagation in spiking neural networks. Master's thesis, The University of Bath (2002)
[8] Maass, W.: Lower bounds for the computational power of networks of spiking neurons. Neural Computation **8** (1996) 1–40
[9] Natschläger, T., Maass, W.: Spiking neurons and the induction of finite state machines. Theoretical Computer Science: Special Issue on Natural Computing **287** (2002) 251–265
[10] Floreano, D., Zufferey, J., Nicoud, J.: From wheels to wings with evolutionary spiking neurons. Artificial Life **11** (2005) 121–138
[11] Martignon, L., Deco, G., Laskey, K.B., Diamond, M., Freiwald, W., Vaadia, E.: Neural coding: Higher-order temporal patterns in the neurostatistics of cell assemblies. Neural Computation **12** (2000) 2621–2653
[12] Nadas, A.: Replay and time compression of recurring spike sequences in the hippocampus. The Journal of Neuroscience, **19** (1999) 9497–9507
[13] Gerstner, W.: Time structure of activity in neural network models. Phys. Rev. E **51** (1995) 738–758
[14] Werbos, P.: Generalization of backpropagation with applications to a recurrent gas market model. Neural Networks **1** (1989) 339–356
[15] Hopcroft, J., Ullman, J.: Introduction to automata theory, languages, and computation. Addison–Wesley, Reading, MA (1979)
[16] Forcada, M., Carrasco, R.: Learning the initial state of a second-order recurrent neural network during regular-language inference. Neural Computation **7** (1995) 923–930
[17] Tiňo, P., Šajda, J.: Learning and extracting initial mealy machines with a modular neural network model. Neural Computation **7** (1995) 822–844
[18] Lawrence, S., Giles, C., Fong, S.: Natural language grammatical inference with recurrent neural networks. IEEE Transactions on Knowledge and Data Engineering **12** (2000) 126–140
[19] Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87** (1999) 1423–1447
[20] Puskorius, G., Feldkamp, L.: Recurrent network training with the decoupled extended Kalman filter. In: Proceedings of the 1992 SPIE Conference on the Science of Artificial Neural Networks, Orlando, Florida. (1992)
[21] Rowe, J., Hidovic, D.: An evolution strategy using a continuous version of the gray-code neighbourhood distribution. In et al., K.D., ed.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Lecture Notes in Computer Science, Springer-Verlag (2004) 725–736
[22] Maass, W., Markram, H.: Synapses as dynamic memory buffers. Neural Networks **15** (2002) 155–161
[23] Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation **14** (2002) 2531–2560