Recurrent Nets that Time and Count

Felix A. Gers   Jürgen Schmidhuber
felix@idsia.ch   juergen@idsia.ch
IDSIA, Corso Elvezia 36, 6900 Lugano, Switzerland, www.idsia.ch

Abstract

The size of the time intervals between events conveys information essential for numerous sequential tasks such as motor control and rhythm detection. While Hidden Markov Models tend to ignore this information, recurrent neural networks (RNNs) can in principle learn to make use of it. We focus on Long Short-Term Memory (LSTM) because it usually outperforms other RNNs. Surprisingly, LSTM augmented by "peephole connections" from its internal cells to its multiplicative gates can learn the fine distinction between sequences of spikes separated by either 50 or 49 discrete time steps, without the help of any short training exemplars. Without external resets or teacher forcing or loss of performance on tasks reported earlier, our LSTM variant also learns to generate very stable sequences of highly nonlinear, precisely timed spikes. This makes LSTM a promising approach for real-world tasks that require to time and count.

1 Introduction

Humans quickly learn to recognize pattern sequences whose defining aspects are embodied by the time intervals between sub-patterns, such as rhythms. And not only drummers are able to generate precisely timed sequences of motor commands. This motivates us to study artificial systems that learn to separate or generate patterns whose information is partially conveyed by the delays between events.

Widely used approaches to sequence processing, such as Hidden Markov Models (HMMs), typically discard such information — in HMM-based speech recognition we do not care for the difference between slow and fast versions of a given spoken word. Recurrent neural networks (RNNs), however, can in principle learn to use it. In fact, RNNs are suited for tasks no other current sequence learning method can solve. Traditional discrete symbolic grammar learning algorithms may faster learn grammatical structure of discrete, noise-free event sequences, but cannot deal with noise or with sequences of real-valued inputs. HMMs are well-suited for noisy inputs but limited to discrete state spaces. Typical RNNs, however, perform gradient descent in a very general space of potentially noise-resistant algorithms using distributed, continuous-valued internal states to map real-valued input sequences to real-valued output sequences (Pearlmutter, 1995).

Certain smoothly oscillating outputs, without long time lags between significant output changes, are learnable by RNNs, either by teacher-forcing (Williams & Zipser, 1989), or by clamping network inputs first to the target and then to the network output (Doya & Yoshizawa, 1989; Tsung & Cottrell, 1995). Here we will study more difficult problems of periodic function approximation (PFA) with long time lags. They require the network to learn to estimate the size of time intervals, and to count discrete time steps, possibly in absence of any short time lags between relevant events, which could facilitate learning. This is extremely difficult for traditional RNNs (Wiles & Elman, 1995; Rodriguez & Wiles, 1998).

Previous experiments with Long Short-Term Memory (LSTM, Hochreiter and Schmidhuber, 1997) already demonstrated LSTM's superiority over traditional RNNs, especially in presence of long time lags (Hochreiter & Schmidhuber, 1997; Gers, Schmidhuber, & Cummins, 1999). But they did not require to time and count. For instance, some of the previous tasks require to predict inputs depending on events that occurred 50 discrete time steps ago but not on anything that happened during the most recent 49 steps. Right before the critical prediction, however, there is a helpful "time marker" input informing the network that the next time step will be crucial — the network does not have to learn to count to 50, it just has to learn to store relevant information for 50 steps and use it once the time marker is observed.

The tasks in the present paper, however, do not involve time markers at all. Instead they require to time and count precisely across long time lags. We will first introduce novel "peephole connections" to

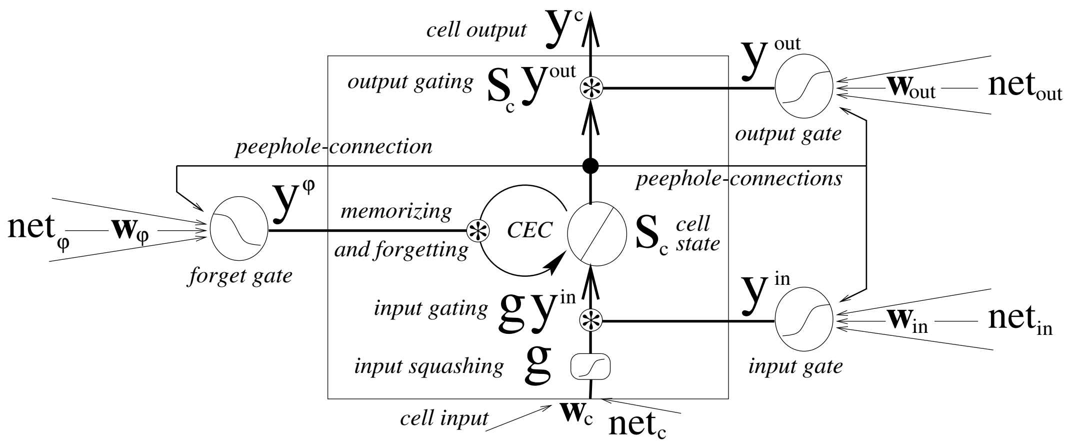

Figure 1: LSTM memory block with one cell, peephole connections connect s_c to the gates.

overcome a weakness in LSTM's connection scheme. Then we show that the resulting LSTM extension can indeed solve such tasks, by learning to translate very rare changes of the target signal into appropriate sequence-processing or sequence-generating algorithms embodied in LSTM's recurrent connections.

2 Extending LSTM with "Peephole Connections"

We are building on LSTM with forget gates (Gers et al., 1999), simply called "LSTM" in what follows.

LSTM. The basic unit of an LSTM network is the *memory block* containing one or more *memory cells* and three adaptive, multiplicative gating units shared by all cells in the block. Each memory cell has at its core a recurrently self-connected linear unit called the "Constant Error Carousel" (CEC) whose activation is called the cell *state*. The CECs enforce *constant* error flow and overcome a fundamental problem plaguing previous RNNs: they prevent error signals from decaying quickly as they "get propagated back in time." The adaptive gates control input and output to the cells (*input and output gate*) and learn to reset the cell's state once its contents are out of date (*forget gate*). All errors are cut off once they leak out of a memory cell or gate, although they do serve to change the incoming weights. The effect is that the CECs are the only part of the system through which errors can flow back forever, while gates etc. learn the nonlinear aspects of sequence processing. This makes LSTM's updates efficient without significantly affecting learning power: LSTM's learning algorithm is local in space and time; its computational complexity per time step and weight is $O(1)$. The CECs permit LSTM to bridge huge time lags (1000 discrete time steps and more) between relevant events, while traditional RNNs with more complex update algorithms already fail to learn in presence of 10 step time lags although they require more expensive updates.

Limitations of LSTM. Each gate receives connections from the input units and the outputs of all cells. But there is no direct connection from the CEC it is supposed to control. All it can observe directly is the cell output, which is close to zero as long as the output gate is closed. The resulting lack of essential information may harm network performance, especially in case of the tasks we are going to study here.

Peephole connections. Our simple remedy is to add weighted "peephole" connections from the CEC to the gates of the same memory block. See Figure 1. The gates learn to shield the CEC from unwanted inputs (forward pass) or unwanted error signals (backward pass). To keep the shield intact, during learning no error signals are propagated back from gates via peephole connections to CEC. Peephole connections are treated almost like regular connections to gates (e.g., from the input) except for update timing. All units (cells, gates) within a layer are updated in arbitrary order, which is fine since the only source of recurrent connections is the cell output y^c. Peephole connections from within the cell (or recurrent connections from gates) require a refinement of LSTM's update scheme.

Updates for peephole LSTM. Each memory cell component should be updated based on the most recent activations or states of connected sources. For each discrete time step we use a 2 phase update scheme (in case of recurrent connections from gates, phase 1 should be subdivided into a,b,c):

1. (a) Input gate activation y^{in}, (b) forget gate activation y^{φ}, (c) cell input and cell state s_c,

2. output gate activation y^{out} and cell output y^c.

So the output gate is updated after cell state s_c, seeing via its peephole connection the actual value of $s_c(t)$ (already affected by forget gate and recent input), and possibly the current input and forget gate activations.

3 Forward Pass

Before specifying the equations for the extended LSTM model, we introduce a minor simplification unrelated to the central idea of this paper. Traditional LSTM memory cells incorporate an input squashing function g *and* an output squashing function (called h in earlier LSTM publications). We remove the latter due to lack of empirical evidence that it is really needed.

The cell output, y^c, is calculated based on the current cell state s_c and three sources of input: net_c is input to the cell itself while net_{in}, net_{φ} and net_{out} are inputs to the input, forget and output gates. We consider discrete time steps $t = 0, 1, 2, \ldots$. A single step involves the update of all units (forward pass) and the computation of error signals for all weights (backward pass). Throughout this paper j indexes memory blocks; v indexes memory cells in block j (with S_j cells), such that c_j^v denotes the v-th cell of the j-th memory block; w_{lm} is the weight on the connection from unit m to unit l. Index m ranges over all source units, as specified by the network topology and m_j additionally includes the cells of the j-th memory block: $m_j = \{m, c_j^v\}$ with $v = 1, \ldots, S_j$. For the gates, f_l , $l \in \{\varphi, in, c_j^v\}$ is a logistic sigmoid (with range $[0, 1]$).
Step 1a,1b. The input gate activation y^{in} and the forget gate activation y^{φ} are computed as follows:

$$net_{l_j}(t) = \sum_{m_j} w_{l_j m_j} \; y_l^{m_j} \;\; ; \;\; y^{l_j}(t) = f_{l_j}(net_{l_j}(t)) \quad \text{with} \;\; l \in \{\varphi, in\} \;\; . \tag{1}$$

The peephole connections for the input gate and the forget gate are incorporated in equation 1 by including the CECs of memory block j as source units: $y_{in}^{m_j} = y_{\varphi}^{m_j} := \{y^m(t-1), \; s_{c_j^v}(t-1)\}$ with $v = 1, \ldots, S_j$.
Step 1c. The state of memory cell $s_c(t)$ is calculated as follows:

$$net_{c_j^v}(t) = \sum_m w_{c_j^v m} \; y^m(t-1) \;\; , \;\; s_{c_j^v}(t) = y^{\varphi_j}(t) \; s_{c_j^v}(t-1) + y^{in_j}(t) \;\; g(net_{c_j^v}(t)) \;\; , \;\; s_{c_j^v}(0) = 0 \;\; . \tag{2}$$

Step 2. The output gate activation y^{out} is computed as follows:

$$net_{out_j}(t) \;\; = \;\; \sum_{m_j} w_{out_j m_j} \; y_{out}^{m_j} \;\; , \;\; y^{out_j}(t) = f_{out_j}(net_{out_j}(t)) \;\; . \tag{3}$$

To incorporate the peephole connections for the output gate in equation 3, $y_{out}^{m_j}$ takes into account the CECs of memory block j with the cell states $s_c(t)$, actualized in step 1: $y_{out}^{m_j} = \{y^m(t\text{–}1), s_{c_j^v}(t)\}$ with $v = 1, \ldots, S_j$. The cell output y^c is computed as follows: $y^{c_j^v}(t) = y^{out_j}(t) \; s_{c_j^v}(t)$.

4 Gradient-Based Backward Pass

The revised update scheme (see Section 2) for memory blocks allows for treating peephole connections like regular connections. This provokes only minor changes in the backward pass (Gers et al., 1999), we will not re-derive it here. Like LSTM but unlike BPTT and RTRL, peephole LSTM is local in space and time. The increase in complexity due to peephole connections is small: 3 weights per cell.

5 Experiments

We study LSTM's performance on PFA tasks that require to time and count and are unsolvable by traditional RNNS. We compare to peephole LSTM, analyze the solutions, or explain why none was found.
Spike timing (ST) (Section 5.2). This PFA task requires to reproduce spike trains with constant time delays. Unlike previously studied PFA tasks (Williams & Zipser, 1989; Doya & Yoshizawa, 1989; Tsung

& Cottrell, 1995) it is highly nonlinear and involves long time lags between significant output changes. Conventional RNNs cannot learn those. Previous work also did not focus on stability issues, but we demand that the approximation is stable for as many as 1000 successive periods. We systematically investigate the effect of minimal time lag on task difficulty. Then we study a variant of this task which requires the spike time delays to reflect the magnitude of an input signal that may change after every spike (IST task).

Counting spike time delays (CSD) (Section 5.3). The tasks above already require the network to learn the size of time intervals by implicitly counting time steps. Here we study more explicit counting tasks. Can LSTM learn the difference between almost identical pattern sequences that differ by very similar time delays (either n or $n+1$) between pattern changes? Can LSTM also do this in presence of never-ending, continual input streams? How does the difficulty of this problem depend on n ?

The continual CSD task is obtained by exchanging inputs and targets in the IST task above: the network has to learn to count time delays between sharp spikes (non-continual version: NCSD task).

5.1 Network Topology and Experimental Parameters

We found that comparatively small LSTM nets already can solve the tasks above. One single input unit (used only for tasks where there is input indeed) is fully connected to the hidden layer consisting of a single memory block with 1 cell and 3 gates. The cell output is connected to the cell input, to all gates, and to a single output unit. All gates, the cell itself and the output unit are biased. The bias weights to input gate, forget gate and output gate are initialized with 0.0, -2.0 and $+2.0$, respectively. Precise bias initialization is not critical though—other values work just as well. All other weights are initialized randomly in the range $[-0.1, 0.1]$. In absence of the 3 peephole connections there are 14 adjustable weights (9 "unit-to-unit" connections and 5 bias connections). The cell's input squashing function g is the identity function. The squashing function of the output unit is a logic sigmoid with range $[0, 1]$.

Our networks process continual streams of (here 1-dimensional) input and target vectors. Only at stream beginnings the network is reset. It always must learn to predict the targets $t_k(t)$, producing a stream of output values (predictions) $y_k(t)$. A prediction is considered correct if the absolute output error is below 0.49. Streams are stopped as soon as the network makes an incorrect prediction or after a given maximal number of successive periods: 100 for training streams, 1000 for test streams (with F predictions per period, where F is the task's frequency parameter).

Learning and testing alternate: after each training stream, we freeze the weights and output a test stream. Our performance measure is the achieved test stream size: 1000 periods correspond to a "perfect" solution. Training is stopped once a task is learned or after a maximal number of 10^6 training streams. Weight changes are made after each target presentation. At the beginning of each training stream, the learning rate α is initialized with 10^{-5} and then adjusted by applying the momentum algorithm (Plaut, Nowlan, & Hinton, 1986) with momentum parameter 0.999 for the ST and IST tasks, 0.99 for the NCSD task and 0.9999 for the CSD task.

For tasks IST and CSD, the stochastic input streams are generated online. A perfect solution correctly processes 10 test streams, to make sure the network provides stable performance independent of the stream beginning, which we found to be critical. All results are averages over 10 independently trained networks.

5.2 Spike Timing

The target is a spike train, represented by a series of 0s and 1s, where a 1 indicates a spike. Spikes occur at times $T(n)$ with $n = 0, 1, \ldots$: $T(0) := F + I(0)$, $T(n+1) := T(n) + F + I(n)$, where F is the minimal time delay between spikes and I the integer network input (only for the IST task), randomly reset at times $T(n)$.

Results. The ST and IST task could *not* be learned by networks without peephole connections (only results with peephole LSTM are reported). The results for different spike time delays F (see Table 1) suggest that the necessary training time depends exponentially on the time delay of the task. A qualitative explanation is: larger time delays necessitate finer tuning of the weights, which demands more training. The network output during test runs with network solutions for the ST task with $F = 10$ is shown in the top row of Figure 2. Peephole LSTM also solves the IST task with $F = 10$ and a network input I in $[0, 1]$ or $[0, 1, 2]$ (Table 2).

Analysis. Figure 2 shows test runs with network solutions for the ST task. The output gates only opens at the onset of a spike and re-closes after the spike. Hence, during a spike, the output of the cell is equal to its state (see middle row of Figure 2). The opening of the output gates is triggered by the cell state s_c. It

Tables: Column "T": Task. Column "F": Minimal spike time delay. Column "$I \in$": Set of input symbols. Column "Sol.": Percentage of perfect solutions. Column "Train. [10^3]": Average number of training streams.

T	F	$I \in$	Peephole LSTM Sol.	Peephole LSTM Train. [10^3]
ST	10	-	100	41 ± 4
	20	-	100	67 ± 8
	30	-	80	845 ± 82
	40	-	100	1152 ± 101
	50	-	100	2538 ± 343
IST	10	$[0,1]$	50	1647 ± 46
	10	$[0,1,2]$	30	954 ± 393

Table 1: Results of ST and IST tasks.

T	F	$I \in$	LSTM Sol.	LSTM Train. [10^3]	Peephole LSTM Sol.	Peephole LSTM Train. [10^3]
NCSD	10	-	100	321 ± 28	100	250 ± 28
	20	-	100	1464 ± 195	100	1526 ± 206
	30	-	100	35043 ± 4401	80	25770 ± 4183
	40	-	0	75066 ± 9116	70	51373 ± 5509
	50	-	0	–	10	64970
CSD	10	$[0,1]$	10	177	20	549 ± 235
	10	$[0,1,2]$	20	585 ± 275	60	196 ± 53

Table 2: Results of NCSD and CSD tasks.

starts to opens when the input from the peephole connection outweighs the bias. The opening self-reinforces via the connection from the cell output, which produces the high nonlinearity necessary for the spike. In the IST task network input is translated into a scaling factor for the growth of s_c.

5.3 Counting Spike Time Delays

Network input and target of the IST task are reversed. For the non-continual version of the task (NCSD) a stream consists of a single period. A perfect solution correctly processes all possible input test streams.
Results. Table 2 reports the results for the NCSD task with $I \in [0,1]$ for different time delays F. The results suggest that the difficulty of the task depends exponentially on the time delay F, as for the ST task (see Section 5.2). Peephole LSTM outperforms LSTM. Also the continual CSD task for $F=10$ with $I \in [0,1]$ or $I \in [0,1,2]$, was solved with and without peephole connections (Table 2).
Analysis. LSTM learned to count in two principled ways. The first is to slightly increase the cell contents s_c at each time step, so that the elapsed time can be read off the value of s_c. This kind of solution is shown on the right hand side of Figure 3 (the state reset performed by the forget gate is only essential for continual online prediction across many periods). The second counting method is to establish internal oscillators (oscillating cell states) and derive the elapsed time from their phases. Both kinds of solutions could be learned with and without peephole connections, as it is never necessary here to close the output gate.

6 Conclusion

Previous work demonstrated LSTM's superiority over other RNNs in presence of significant time lags between relevant events but did not require the network to extract relevant information conveyed by the size of the time lags. Here we show that LSTM can solve such tasks, too, by learning to time and count and act in a highly nonlinear fashion, provided we extend LSTM cells by peephole connections that allow them to inspect their current internal states. It is remarkable that LSTM can learn precise and stable timing algorithms without teacher forcing and despite the very uninformative target signals which change very rarely. This makes LSTM a promising approach for numerous real-world tasks that require to learn to measure the size of time intervals. In particular, we intend to apply LSTM to music and rhythm classification.
Acknowledgment. This work was supported by SNF grant 2100-49'144.96 "LSTM."

Reference

Doya, K., & Yoshizawa, S. (1989). Adaptive neural oscillator using continuous-time backpropagation learning. *Neural Networks*, *2*(5), 375–385.

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM. In *Proc. ICANN'99, Int. Conf. on Artificial Neural Networks*, Vol. 2, pp. 850–855 Edinburgh, Scotland. IEE, London. Extended version submitted to Neural Computation.

Test run with network solutions: Top: Target values t_k and network output y_k. Middle: Cell state s_c and cell output y_c. Bottom: Activations of the gates (input gate y_{in}, forget gate y_φ and output gate y_{out}).

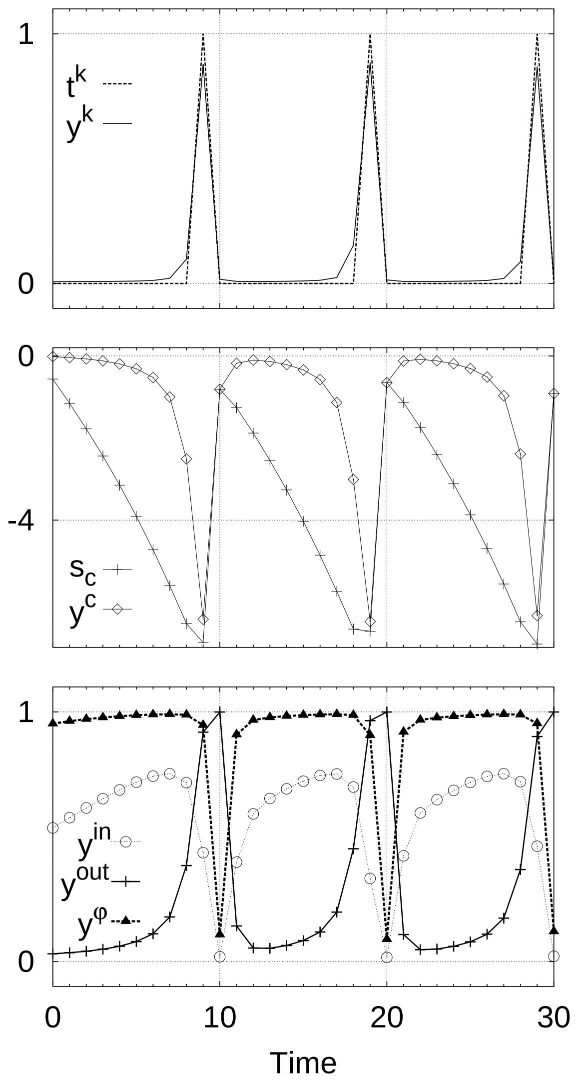

Figure 2: Peephole LSTM, ST task: $F = 10$.

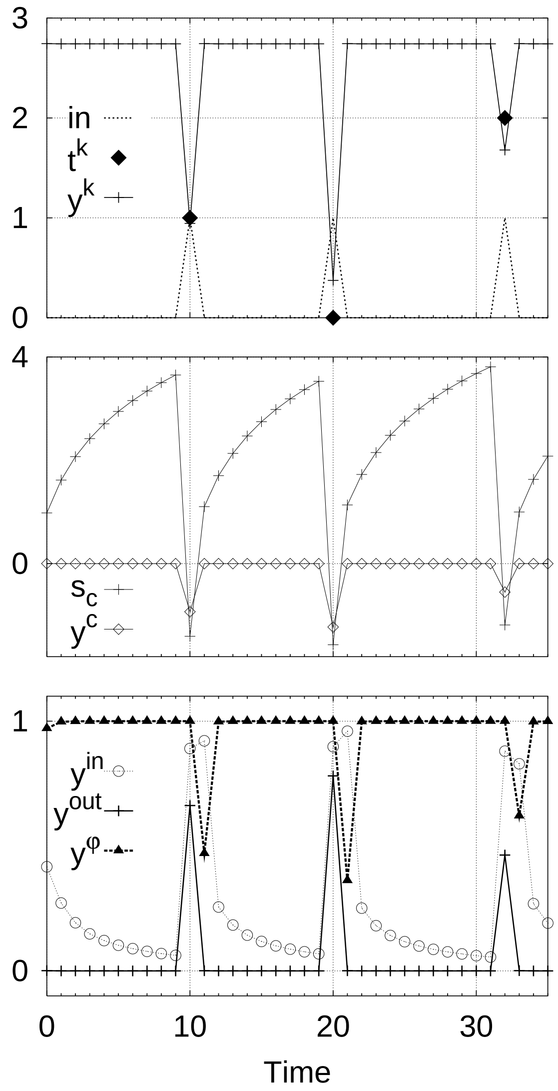

Figure 3: Peephole LSTM, CSD task: F=10 , $I \in [0, 1, 2]$.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Pearlmutter, B. A. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, *6*(5), 1212–1228.

Plaut, D. C., Nowlan, S. J., & Hinton, G. E. (1986). Experiments on learning back propagation. Tech. rep. CMU–CS–86–126, Carnegie–Mellon University, Pittsburgh, PA.

Rodriguez, P., & Wiles, J. (1998). Recurrent neural networks can learn to implement symbol-sensitive counting. In *Advances in Neural Information Processing Systems*, Vol. 10, pp. 87–93. The MIT Press.

Tsung, F.-S., & Cottrell, G. W. (1995). Phase-space learning. In *Advances in Neural Information Processing Systems*, Vol. 7, pp. 481–488. The MIT Press.

Wiles, J., & Elman, J. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In *In Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, pp. pages 482 – 487 Cambridge, MA. MIT Press.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent net works. *Neural Computation*, *1*(2), 270–280.