International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems

THE VANISHING GRADIENT PROBLEM DURING LEARNING RECURRENT NEURAL NETS AND PROBLEM SOLUTIONS*

Sepp Hochreiter
Institut für Informatik, Technische Universität München
München, D-80290, Germany
E-mail: hochreit@informatik.tu-muenchen.de

Received ( )
Revised ( )

Recurrent nets are in principle capable to store past inputs to produce the currently desired output. Because of this property recurrent nets are used in time series prediction and process control. Practical applications involve temporal dependencies spanning many time steps, e.g. between relevant inputs and desired outputs. In this case, however, gradient based learning methods take too much time. The extremely increased learning time arises because the error vanishes as it gets propagated back. In this article the decaying error flow is theoretically analyzed. Then methods trying to overcome vanishing gradients are briefly discussed. Finally, experiments comparing conventional algorithms and alternative methods are presented. With advanced methods long time lag problems can be solved in reasonable time.

Keywords: recurrent neural nets; vanishing gradient; long-term dependencies; Long Short-Term Memory.

1. Introduction

Recurrent neural nets can extract temporal dependencies. Therefore recurrent nets are used for applications including temporal delays of relevant signals, e.g., speech processing, non-Markovian control, time series analysis, process control,[2] and music composition.[3] Recurrent nets must learn which past inputs have to be stored to produce the current desired output. With gradient based learning methods the current error signal has to "flow back in time" over the feedback connections to past inputs for building up an adequate input storage. Conventional backpropagation, however, suffers from a too long learning time, when minimal time lags between relevant inputs and corresponding teacher signals are extended. For instance, with "backprop through time" (BPTT[4]) or "Real-Time Recurrent Learning" (RTRL[5]), error signals flowing backwards in time tend to vanish. Long-term dependencies are hard to learn because of insufficient weight changes. Section 2 theoretically analyzes the vanishing gradient. Section 3 presents methods trying to overcome the problem of vanishing gradients. In Section 4 conventional algorithms are compared with advanced methods on several tasks including long time lags.

*This article is partly based on previous publications.[1]

2. Decaying Gradient

2.1. *Conventional backpropagation through time (BPTT)*

Assume a fully connected recurrent net with units $1, \ldots, n$ is given. The activation of a non-input unit i with activation function f_i and net input $net_i(t) = \sum_j w_{ij} y^j(t-1)$ is $y^i(t) = f_i(net_i(t))$. w_{ij} is the weight on the connection from unit j to i. $d_k(t)$ denotes output unit k's target at current time t. Using mean squared error, k's external (target) error is

$$E_k(t) = \left(d_k(t) - y^k(t)\right) \tag{1}$$

(all non-output units i have zero external error $E_i(t) = 0$). At an arbitrary time $\tau \leq t$ non-input unit j's error signal is the sum of the external error and the backpropagated error signal from the previous time step:

$$\vartheta_j(\tau) = f_j'(net_j(\tau)) \left(E_j(\tau) + \sum_i w_{ij} \vartheta_i(\tau+1) \right). \tag{2}$$

Error signals are set to zero when activations are reset at time τ:
$\vartheta_j(\tau) = 0$ and $f_j'(net_j(\tau)) = 0$. The weight update at time τ is

$$w_{jl}^{new} = w_{jl}^{old} + \alpha \vartheta_j(\tau) y^l(\tau - 1), \tag{3}$$

where α is the learning rate, and l is an arbitrary unit connected to unit j.

Error flow scaling factor. See also earlier contributions to the analysis of the vanishing gradient.[6,7,1] Backpropagating an error occurring at an unit u at time step t to an unit v for q time steps, scales the error by:

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \begin{cases} f_v'(net_v(t-1)) w_{uv} & q = 1 \\ f_v'(net_v(t-q)) \sum_{l=1}^{n} \frac{\partial \vartheta_l(t-q+1)}{\partial \vartheta_u(t)} w_{lv} & q > 1 \end{cases}. \tag{4}$$

With $l_q = v$ and $l_0 = u$, the scaling factor is

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f_{l_m}'(net_{l_m}(t-m)) w_{l_m l_{m-1}}. \tag{5}$$

Analyzing Eq. (5). The relation between the experimentally observed vanishing gradient and Eq. (5) will be explained. The sum of the n^{q-1} terms

$$\prod_{m=1}^{q} f_{l_m}'(net_{l_m}(t-m)) w_{l_m l_{m-1}} \tag{6}$$

scales the error back flow. If

$$\rho(m, l_m, l_{m-1}) := \left| f_{l_m}'(net_{l_m}(t-m)) w_{l_m l_{m-1}} \right| < 1.0 \tag{7}$$

for all m the largest product in Eq. (5) *decreases* exponentially with q, that is, the error flow vanishes. A vanishing error back flow has almost no effect on weight updates. Given constant $y^{l_{m-1}} \neq 0$, $\rho\left(m, l_m, l_{m-1}\right)$ is maximal where

$$w_{l_m l_{m-1}} = \frac{1}{y^{l_{m-1}}} \coth\left(\frac{1}{2} net_{l_m}\right). \tag{8}$$

For increasing absolute weight values $\left|w_{l_m l_{m-1}}\right| \to \infty$, $\rho\left(m, l_m, l_{m-1}\right)$ converges to zero. Thus, the vanishing gradient cannot be avoided by increasing the absolute weight values. Terms as in formula (6) may have different signs. Therefore, increasing the number of units n does not necessarily increase the absolute error flow value. But with more units the expectation of the error back flow's absolute value increases.

If f_{l_m} is the logistic sigmoid function, the maximal value of f'_{l_m} is 0.25. Therefore, $\rho\left(m, l_m, l_{m-1}\right)$ is less than 1.0 for $\left|w_{l_m l_{m-1}}\right| < 4.0$. If $w_{max} < 4.0$ holds for the absolute maximal weight value w_{max} (e.g. initialization) then all $\rho\left(m, l_m, l_{m-1}\right)$ are smaller than 1.0. Hence, with logistic activation functions the error flow tends to vanish especially at the beginning of learning.

Increasing the learning rate does not countermand the effects of vanishing gradients, because it won't change the ratio of long-range error flow and short-range error flow (recent inputs have more influence on the current output).

2.2. *Upper bound for the absolute scaling factor*

Matrix A's element in the i-th column and j-th row is denoted by $[A]_{ij}$. The i-th component of vector x is denoted by $[x]_i$. The activation vector at time t with net input vector $Net\left(t\right) := W\ Y\left(t-1\right)$ and weight matrix $[W]_{ij} := w_{ij}$ is $[Y\left(t\right)]_i := y^i\left(t\right)$ (for simplicity the external input is suppressed).

The activation function vector is $[F\left(Net\left(t\right)\right)]_i := f_i\left(net_i\left(t\right)\right)$, therefore

$$Y\left(t\right) = F\left(Net\left(t\right)\right) = F\left(W\ Y\left(t-1\right)\right). \tag{9}$$

$F'\left(t\right)$ is the diagonal matrix of first order derivatives defined as:
$[F'\left(t\right)]_{ij} := f'_i\left(net_i\left(t\right)\right)$ if $i = j$, and $[F'\left(t\right)]_{ij} := 0$ otherwise. W_v is unit v's outgoing weight vector ($[W_v]_i := [W]_{iv} = w_{iv}$) and W_{u^T} is unit u's incoming weight vector ($[W_{u^T}]_i := [W]_{ui} = w_{ui}$). The vector $\frac{\partial Y(t)}{\partial net_v(t-q)}$ is defined as

$$\left[\frac{\partial Y\left(t\right)}{\partial net_v\left(t-q\right)}\right]_i := \frac{\partial y^i\left(t\right)}{\partial net_v\left(t-q\right)} \tag{10}$$

for $q \geq 0$ and the matrix $\nabla_{Y(t-1)} Y\left(t\right)$ is defined as

$$\left[\nabla_{Y(t-1)} Y\left(t\right)\right]_{ij} := \frac{\partial y^i\left(t\right)}{\partial y^j\left(t-1\right)}. \tag{11}$$

From the definitions

$$\nabla_{Y(t-1)} Y\left(t\right) = F'\left(t\right)\ W \tag{12}$$

is obtained. Again, the scaling factor of an error flowing back from an unit u (at time t) for q time steps to an unit v is computed:

$$\frac{\partial \vartheta_v\,(t-q)}{\partial \vartheta_u\,(t)} = \frac{\partial net_u\,(t)}{\partial net_v\,(t-q)} = \nabla_{Y(t-1)} net_u\,(t)\;\; \frac{\partial Y\,(t-1)}{\partial net_v\,(t-q)} =$$

$$\nabla_{Y(t-1)} net_u\,(t)\;\prod_{m=1}^{q-2}\left(\nabla_{Y(t-m-1)} Y\,(t-m)\right)\;\; \frac{\partial Y\,(t-q+1)}{\partial net_v\,(t-q)} =$$

$$\left(W_{u^T}\right)^T\;\prod_{m=1}^{q-2}\left(F'\,(t-m)\,W\right)\;\; F'\,(t-q+1)\;\; W_v\;\; f_v'\left(net_v\,(t-q)\right), \tag{13}$$

where T is the transposition operator.

Using a matrix norm $\|.\|_A$ compatible with vector norm $\|.\|_x$, f'_{max} is defined as $f'_{max} := \max_{m=1,\ldots,q}\{\|F'\,(t-m)\|_A\}$. For $\max_{i=1,\ldots,n}\{|x_i|\} \leq \|x\|_x$ one gets $\left|x^T y\right| \leq n\;\|x\|_x\;\|y\|_x$. Since $|f_v'\,(net_v\,(t-q))| \leq \|F'\,(t-q)\|_A \leq f'_{max}$, the following inequality is obtained:

$$\left|\frac{\partial \vartheta_v\,(t-q)}{\partial \vartheta_u\,(t)}\right| \leq n\;\;\left(f'_{max}\right)^q\;\|W_v\|_x\;\|W_{u^T}\|_x\;\|W\|_A^{q-2} \;\leq\; n\;\;\left(f'_{max}\;\|W\|_A\right)^q. \tag{14}$$

This inequality results from

$$\|W_v\|_x \;=\; \|We_v\|_x \;\leq\; \|W\|_A\;\|e_v\|_x \;\leq\; \|W\|_A \tag{15}$$

and

$$\|W_{u^T}\|_x \;=\; \left\|W^T e_u\right\|_x \;\leq\; \|W\|_A\;\|e_u\|_x \;\leq\; \|W\|_A\,, \tag{16}$$

where e_k is the unit vector of zeros and only the k-th component is 1.

This best case upper bound will only be reached if all $\|F'\,(t-m)\|_A$ are maximal, and contributions from all error flow paths have equal sign (see the product terms in Eq. (5)). A large $\|W\|_A$, however, leads to small values of $\|F'\,(t-m)\|_A$ because most sigmoid units are saturated and the derivatives are small (also confirmed by experiments). Taking the norms $\|W\|_A := \max_r \sum_s |w_{rs}|$ and $\|x\|_x := \max_r |x_r|$, $f'_{max} = 0.25$ holds for the logistic sigmoid. For

$$|w_{ij}| \leq w_{max} < \frac{4.0}{n}\;\;\forall i,j \tag{17}$$

one gets $\|W\|_A \leq n w_{max} < 4.0$. With the value $\mu := \left(\frac{n w_{max}}{4.0}\right) < 1.0$ we get the exponential decay

$$\left|\frac{\partial \vartheta_v\,(t-q)}{\partial \vartheta_u\,(t)}\right| \;\leq\; n\,(\mu)^q\,. \tag{18}$$

2.3. *Information theoretic consideration*

From another point of view the vanishing gradient corresponds to a vanishing information in the internal states of a recurrent net. Let G be the function mapping the previous internal states $Y\,(t-1)$ to the actual internal states

$$G\,(Y\,(t-1)) := Y\,(t) = F\,(Net\,(t)) = F\,(W\;Y\,(t-1))\,. \tag{19}$$

Considering $Y(t)$ and $Y(t-1)$ as random variables the mutual information between these random variables is

$$H(Y(t)) - H(Y(t) \mid Y(t-1)), \tag{20}$$

where H denotes the entropy of a random variable and $(Y(t) \mid Y(t-1))$ means the conditional random variable $Y(t)$ given $Y(t-1)$.

For the entropy of $Y(t)$

$$H(Y(t)) \leq H(Y(t-1)) + E(\log|\det(J(Y(t-1)))|) \tag{21}$$

holds,[8] where E is the expectation of a random variable and J is the Jacobian of G. The stability of the recurrent net requires

$$|\det(J(Y(t-1)))| \leq 1. \tag{22}$$

This requirement makes the recurrent net resistant to noise (small changes of the input do not drive the net to very different states).[9,6] The left-hand side of Eq. (22) is usually below 1 such that the information in the internal states vanishes over time. The problem of vanishing information gets worse with increasing length of the time interval over which the information has to be stored. To avoid vanishing information

$$\det(J(Y(t-1))) = 1 \tag{23}$$

must be enforced (volume-conserving mappings[8]). A volume-conserving mapping restricted to only one internal state is the main idea of "Long Short Term Memory" (LSTM[10,11,1]) mentioned in the next section.

3. Methods for Long Time Lag Learning

Gradient descent based algorithms. Many widely used methods [12,13,14,15,16,17] suffer from a vanishing gradient. They have considerable difficulties learning long-term dependencies. To overcome the vanishing gradient problem there are four types of solutions:

(i) Methods which do not use gradients.
(ii) Methods which enforce higher gradients.
(iii) Methods which operate on higher levels.
(iv) Methods which use special architectures.

(i) Global search methods do not use gradient information. Methods such as simulated annealing, multi-grid random search,[6] and random weight guessing[18] were investigated. It was found that global search methods work well on "simple" problems involving long-term dependencies. "Simple" problems are characterized by solutions with nets containing few parameters and the absence of precise computation (these solutions correspond to "flat minima"[19]).

(ii) Larger values of the gradient can be enforced by time-weighted pseudo-Newton optimization and discrete error propagation.[6] It seems that these methods have problems learning to store precise real-valued information over time.

(iii) An EM approach for target propagation has previously been proposed.[20] This approach uses a discrete number of states and, therefore, will have problems with continuous values.

Kalman filter techniques are used for recurrent network training.[2] But a derivative discount factor leads to vanishing gradient problems.

If a long-time lag problem contains local regularities, a hierarchical chunker system works well.[21]

(iv) Second order nets (using sigma-pi units) are in principle capable to increase the error flow, but vanishing error problems can hardly be avoided.[22,23]

With a "Time-Delay Neural Network" (TDNN[24]), net activations from previous time steps are fed back into the net using fixed delay lines. In TDNNs the error decrease is slowed down because the error uses "shortcuts" as it gets propagated back. TDNNs have to deal with a trade-off: increasing the length of delay line increases the error flow but the net has more parameters/units. Special cases of TDNNs are NARX networks,[25] and the weighted sum of old activations instead of a fixed delay line.[26] A more complex version of a TDNN, called the "Gamma Memory", was proposed,[27] but its performance on problems involving long-term dependencies does not appear to be better than the performance of TDNNs.

In some architectures time constants determine the scaling factor of the error if it gets propagated back for one time step at a single unit.[3] But extended time gaps cannot be processed because an appropriate time constant fine tuning is almost impossible.

Updating a single unit by adding the old activation and the scaled current net input avoids the vanishing gradient.[28] But the stored value is sensible to perturbations by later irrelevant net inputs. "Long Short Term Memory" (LSTM[10,11,1]) uses a special architecture to enforce constant error flow through special units (including a volume-conserving mapping). Unlike in the method avoiding vanishing gradients[28] mentioned above, perturbations by current irrelevant signals are prevented by multiplicative units.

4. Experiments

A more detailed presentation of the experiments can be found in an extended article.[1]

4.1. *Experiment 1: embedded Reber grammar*

Task. The "embedded Reber grammar" was often used as a benchmark problem for recurrent nets.[29,30,13] This task does not include long time lags and, therefore, can be learned by conventional methods. The experiment serves to show that even on short time lag problems alternative methods outperform conventional gradient

descent methods. Being at the leftmost node (with an empty string) in Fig. 2

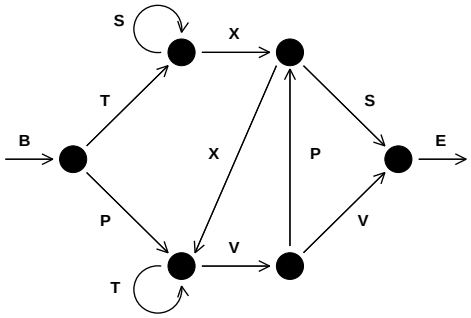

Fig. 1. Reber grammar.

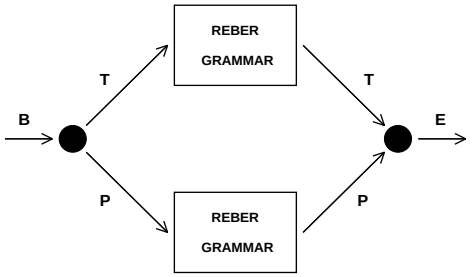

Fig. 2. Embedded Reber grammar. Boxes represent the Reber grammar in Fig. 1.

a string is produced by following the directed edges and adding the corresponding symbols to the current string until arriving in the rightmost node. Alternative edges are chosen randomly (probability: 0.5). The net sequentially processes the string obtaining as input the actual symbol and having to predict the next symbol. In order to predict the last but one string symbol ("T" or "P") the net has to store the second symbol ("T" or "P").

RTRL, Elman nets (ELM), Fahlman's "Recurrent Cascade-Correlation" (RCC), and LSTM are applied to this task. Experimental details can be found in the references listed in Table 1, which gives the results. Only *LSTM almost always learned the task*. Also it learned faster then the competitors.

Table 1. Experiment 1 — embedded Reber grammar. Percentage of successful trials and learning time for successful trials for RTRL, Elman nets, RCC and LSTM. Results taken from other articles.[29,30,13,1]

method	hidden units	# weights	learning rate	% of success	success after
RTRL	3	$\approx$ 170	0.05	"some fraction"	173,000
RTRL	12	$\approx$ 494	0.1	"some fraction"	25,000
ELM	15	$\approx$ 435		0	>200,000
RCC	7-9	$\approx$ 119-198		50	182,000
LSTM	4 blocks, size 1	264	0.1	100	39,740
LSTM	3 blocks, size 2	276	0.1	100	21,730
LSTM	3 blocks, size 2	276	0.2	97	14,060
LSTM	4 blocks, size 1	264	0.5	97	9,500
LSTM	3 blocks, size 2	276	0.5	100	8,440

4.2. *Experiment 2: long time lags*

The limitations of gradient descent based methods can be seen on the simple task 2a involving long minimal time lags. But advanced methods can learn the difficult task 2b even with minimal time lags of 1000.

Task 2a — long time lags with regularities. Two sequences are use for training: $(y, a_1, a_2, \ldots, a_{p-1}, y)$ and $(x, a_1, a_2, \ldots, a_{p-1}, x)$. The symbols are coded locally by a $(p+1)$-dimensional input vector. Strings are processed sequentially and the net

has to predict the next string symbol in each step. For predicting the last symbol the net has to remember the first symbol. Therefore this task involves a minimal time lag of p. *Note:* The sequences have local regularities required by the neural sequence chunker but not by LSTM.

RTRL [5], BPTT, the neural sequence chunker (CH[21]), and LSTM are compared. Table 2 gives the results. Gradient based methods (RTRL, BPTT) get into trouble when the minimal time lag exceeds 10 steps.

Table 2. Task 2a — long time lags with regularities. Success percentage and learning time until success. Results taken from another article.[1]

Method	Delay p	Learning rate	# weights	% Successful trials	Success after
RTRL	4	1.0	36	78	1,043,000
RTRL	4	4.0	36	56	892,000
RTRL	4	10.0	36	22	254,000
RTRL	10	1.0-10.0	144	0	$>$ 5,000,000
RTRL	100	1.0-10.0	10404	0	$>$ 5,000,000
BPTT	100	1.0-10.0	10404	0	$>$ 5,000,000
CH	100	1.0	10506	33	32,400
LSTM	100	1.0	10504	100	5,040

Task 2b — very long time lags without regularities. The goal is to predict the last symbol of a sequence. There are $p+4$ possible input symbols denoted $a_1, ..., a_{p-1}, a_p, a_{p+1} = e, a_{p+2} = b, a_{p+3} = x, a_{p+4} = y$. Again, a_i is locally represented by a $(p+4)$-dimensional vector. Training sequences are randomly chosen from two very similar sets of sequences:
$\{(b, y, a_{i_1}, a_{i_2}, \ldots, a_{i_{q+k}}, e, y) \mid 1 \leq i_1, i_2, \ldots, i_{q+k} \leq q\}$ and
$\{(b, x, a_{i_1}, a_{i_2}, \ldots, a_{i_{q+k}}, e, x) \mid 1 \leq i_1, i_2, \ldots, i_{q+k} \leq q\}$. The minimal sequence length is $q+4$; k is chosen randomly with probability $\frac{1}{10}(\frac{9}{10})^k$. To solve the task the net has to learn to store a representation of the second element for at least $q+1$ time steps. To our knowledge no other recurrent net algorithm can solve it.

Table 3 lists the mean number of training sequences required by LSTM to be successful. It can be seen in Table 3 that letting the number of input symbols (and weights) increase in proportion to the time lag, learning time increases very slowly.

Table 3. Task 2b — very long time lags without regularities. The rightmost column lists the number of training sequences required by LSTM. Results taken from another article.[1]

q (time lag -1)	# weights	Success after
50	364	30,000
100	664	31,000
200	1264	33,000
500	3064	38,000
1,000	6064	49,000

5. Conclusion

The error flow for gradient based recurrent learning methods was theoretically analyzed. This analysis showed that learning to bridge long time lags can be difficult. Advanced methods to overcome the vanishing gradient problem were men-

tioned, but most of these approaches have serious disadvantages (e.g. practicable only for discrete problems). The experiments confirmed that conventional learning algorithms for recurrent nets cannot learn long time lag problems in a reasonable time. Advanced methods (like LSTM) performed well on long time lag problems involving time lags of 1000 steps.

Acknowledgements

I want to thank Jürgen Schmidhuber who helped a lot to write this paper. This work was supported by *DFG grant SCHM 942/3-1* from "Deutsche Forschungsgemeinschaft".

References

1. S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, 9(8):1735–1780 (1997).
2. G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks", *IEEE Transactions on Neural Networks*, 5(2):279–297 (1994).
3. M. C. Mozer, "Induction of multiscale temporal structure", in *Advances in Neural Information Processing Systems 4*, ed. J. E. Moody *et al.*, (Morgan Kaufmann, San Mateo, 1992), pages 275–282.
4. R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity", in *Back-propagation: Theory, Architectures and Applications*, ed. Y. Chauvin and D. E. Rumelhart (Hillsdale, Erlbaum, New York, 1992).
5. A. J. Robinson and F. Fallside, "The utility driven dynamic error propagation network", Technical Report CUED/F-INFENG/TR.1, Cambridge Univ. Engineering Department, 1987.
6. Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, 5(2):157–166 (1994).
7. S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen", Diploma Thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Univ. München, 1991. See www7.informatik.tu-muenchen.de/~hochreit.
8. G. Deco and W. Brauer, "Nonlinear higher-order statistical decorrelation by volume-conserving neural architectures", *Neural Networks*, 8(4):525–535 (1995).
9. F. J. Pineda, "Dynamics and architecture for neural computation", *Journal of Complexity*, 4:216–245 (1988).
10. S. Hochreiter and J. Schmidhuber, "LSTM can solve hard long time lag problems", in *Advances in Neural Information Processing Systems 9*, ed. M. C. Mozer *et al.* (MIT Press, Cambridge MA, 1997), pages 473–479.
11. S. Hochreiter and J. Schmidhuber, "Bridging long time lags by weight guessing and Long Short-Term Memory", in *Spatiotemporal models in biological and artificial systems*, ed. F. L. Silva *et al.* (IOS Press, Amsterdam, Netherlands, 1996), pages 65–72.
12. J. L. Elman, "Finding structure in time", Technical Report CRL 8801, Center for Research in Language, Univ. of California, San Diego, 1988.
13. S. E. Fahlman, "The recurrent cascade-correlation learning algorithm", in *Advances in Neural Information Processing Systems 3*, ed. R. P. Lippmann *et al.* (Morgan Kaufmann, San Mateo, 1991), pages 190–196.

14. R. J. Williams, "Complexity of exact gradient computation algorithms for recurrent neural networks", Technical Report NU-CCS-89-27, Boston: Northeastern Univ., College of Computer Science, 1989.
15. J. Schmidhuber, "A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks", *Neural Computation*, 4(2):243–248 (1992).
16. B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks", *Neural Computation*, 1(2):263–269 (1989).
17. B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey", *IEEE Transactions on Neural Networks*, 6(5):1212–1228 (1995).
18. J. Schmidhuber and S. Hochreiter, "Guessing can outperform many long time lag algorithms", Technical Report IDSIA-19-96, IDSIA, 1996.
19. S. Hochreiter and J. Schmidhuber, "Flat minima", *Neural Computation*, 9(1):1–42 (1997).
20. Y. Bengio and P. Frasconi, "Credit assignment through time: Alternatives to backpropagation", in *Advances in Neural Information Processing Systems 6*, ed. J. D. Cowan *et al.* (Morgan Kaufmann, San Mateo, 1994), pages 75–82.
21. J. Schmidhuber, "Learning complex, extended sequences using the principle of history compression", *Neural Computation*, 4(2):234–242 (1992).
22. R. L. Watrous and G. M. Kuhn, "Induction of finite-state languages using second-order recurrent networks", *Neural Computation*, 4:406–414 (1992).
23. C. B. Miller and C. L. Giles, "Experimental comparison of the effect of order in recurrent neural networks", *International journal of Pattern Recognition and Artificial Intelligence*, 7(4):849–872 (1993).
24. K. Lang, A. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition", *Neural Networks*, 3:23–43 (1990).
25. T. Lin, B. G. Horne, P. Tino, and C. L. Giles, "Learning long-term dependencies in NARX recurrent neural networks", *IEEE Transactions on Neural Networks*, 7(6):1329–1338 (1996).
26. T. A. Plate, "Holographic recurrent networks", in *Advances in Neural Information Processing Systems 5*, ed. J. D. Cowan *et al.* (Morgan Kaufmann, San Mateo, 1993), pages 34–41.
27. B. de Vries and J. C. Principe, "A theory for neural networks with time delays", in *Advances in Neural Information Processing Systems 3*, ed. R. P. Lippmann *et al.* (Morgan Kaufmann, San Mateo, 1991), pages 162–168.
28. G. Sun, H. Chen, and Y. Lee, "Time warping invariant neural networks", in *Advances in Neural Information Processing Systems 5*, ed. J. D. Cowan *et al.* (Morgan Kaufmann, San Mateo, 1993), pages 180–187.
29. A. W. Smith and D. Zipser, "Learning sequential structures with the real-time recurrent learning algorithm", *International Journal of Neural Systems*, 1(2):125–131 (1989).
30. A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite-state automata and simple recurrent networks", *Neural Computation*, 1:372–381 (1989).