Recurrent Neural Net Learning and Vanishing Gradient

International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6(2):107–116, 1998

Sepp Hochreiter
Institut für Informatik
Technische Universität München
D-80290 München, Germany
E-Mail: hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Abstract. Recurrent nets are in principle capable to store past inputs to produce the currently desired output. This recurrent net property is used in time series prediction and process control. Practical applications involve temporal dependencies spanning many time steps between relevant inputs and desired outputs. In this case, however, gradient descent learning methods take to much time. The learning time problem appears because the error vanishes as it gets propagated back. The decaying error flow is theoretically analyzed. Then methods trying to overcome vanishing gradient are mentioned. Finally, experiments comparing conventional algorithms and alternative methods are presented. Experiments using advanced methods show that learning long time lags problems can be done in reasonable time.

Keywords: recurrent neural nets, vanishing gradient, long-term dependencies.

1 INTRODUCTION

Recurrent neural nets are capable to extract temporal dependencies. Therefore recurrent nets are used for many applications including temporal delays of relevant signals, e.g., speech processing, non-Markovian control, time series analysis, process control (e.g.[18]), and music composition (e.g.[14]). Recurrent nets must learn which past inputs have to be stored and computed 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, suffer from too much learning time, when minimal time lags between relevant inputs and corresponding teacher signals are extended. For instance, with "backprop through time" (BPTT, e.g. [28]) or "Real-Time Recurrent Learning" (RTRL, e.g., [20]), error signals flowing backwards in time tend to vanish. In this case long-term dependencies are hard to learn because of insufficient weight changes. The next Section 2 theoretically analysis the vanishing gradient. Then Section 3 presents methods trying to overcome the problem of vanishing gradient. In the experimental Section 4 conventional algorithms are compared with some advanced methods on tasks including long time lags. This paper is partly based on [8].

2 DECAYING GRADIENT

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

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

Error signals are set to zero if activations are reset at time τ: $\vartheta_j(\tau) = 0$ ($f'_j(net_j(\tau)) = 0$). The weight update at time τ is $w_{jl}^{new} = w_{jl}^{old} + \alpha \vartheta_j(\tau) y^l(\tau - 1)$, where α is the learning rate, and l is an arbitrary unit connected to unit j.

Vanishing error problem. See also [2] and [7]. Propagating back an error occurring at an unit u at time step t to an unit v for q time steps, scales the error by:

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

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

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^{n} \ldots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}} . \tag{2}$$

Analyzing equation (2). Here the relation between the experimentally observed vanishing gradient and equation (2) should be given. The sum of the n^{q-1} terms $\prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}$ scales the error back flow. These terms 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 $\rho(m, l_m, l_{m-1}) := |f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}| < 1.0$ for all m the largest product in (2) *decreases* exponentially with q, that is, the error flow vanishes. A vanishing error back flow has almost no effect on weight updates. Given constant $y^{l_{m-1}} \neq 0$ $\rho(m, l_m, l_{m-1})$ is maximal where $w_{l_m l_{m-1}} = \frac{1}{y^{l_{m-1}}} \coth(\frac{1}{2} net_{l_m})$. Increasing the absolute weight values $|w_{l_m l_{m-1}}| \to \infty$ lead to $\rho(m, l_m, l_{m-1}) \to 0$. Thus, the vanishing gradient cannot be avoided by increasing the absolute weight values.

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

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

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 is $[Y(t)]_i := y^i(t)$ with net input vector $Net(t) := W\ Y(t-1)$ and weight matrix $[W]_{ij} := w_{ij}$. The activation function vector is $[F(Net(t)]_i := f_i(net_i(t))$, therefore $Y(t) = F(Net(t)) = F(W\ Y(t-1))$. $F'(t)$ is the diagonal matrix of first order derivatives defined as: $[F'(t)]_{ij} := f_i'(net_i(t))$ if $i = j$, and $[F'(t)]_{ij} := 0$ otherwise. W_v is unit v's outgoing weight vector ($[W_v]_i := [W]_{iv} = w_{iv}$), 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 $[\frac{\partial Y(t)}{\partial net_v(t-q)}]_i := \frac{\partial y^i(t)}{\partial net_v(t-q)}$ for $q \geq 0$ and the matrix $\nabla_{Y(t-1)}Y(t)$ is defined as $[\nabla_{Y(t-1)}Y(t)]_{ij} := \frac{\partial y^i(t)}{\partial y^j(t-1)}$.

From the definitions $\nabla_{Y(t-1)}Y(t) = F'(t)\ W$ 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:

$$\begin{aligned}\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} &= \frac{\partial net_u(t)}{\partial net_v(t-q)} = \nabla_{Y(t-1)} net_u(t)\ \frac{\partial Y(t-1)}{\partial net_v(t-q)} = \\ &\nabla_{Y(t-1)} net_u(t)\ \prod_{m=1}^{q-2} (\nabla_{Y(t-m-1)} Y(t-m))\ \ \frac{\partial Y(t-q+1)}{\partial net_v(t-q)} = \\ &(W_{u^T})^T\ \prod_{m=1}^{q-2} (F'(t-m)W)\ \ F'(t-q+1)\ W_v\ f_v'(net_v(t-q)),\end{aligned} \tag{3}$$

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 $|x^T y| \ \leq \ n\ \| \ x\ \|_x \ \| \ y\ \|_x$. Since $|f_v'(net_v(t-q))| \leq \| \ F'(t-q)\ \|_A \leq f'_{max}$, the following inequality is obtained:

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

This inequality results from $\| \ W_v\ \|_x = \| \ We_v\ \|_x \leq \| \ W\ \|_A \ \| \ e_v\ \|_x \leq \| \ W\ \|_A$ and $\| \ W_{u^T}\ \|_x = \| \ e_u W\ \|_x \leq \| \ W\ \|_A \ \| \ e_u\ \|_x \leq \| \ W\ \|_A$, 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 equation (2)). Large $\| \ W\ \|_A$, however, leads to small values of $\| \ F'(t-m)\ \|_A$. 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 $|w_{ij}| \leq w_{max} < \frac{4.0}{n}\ \ \forall i,j$ one gets $\| \ W\ \|_A \leq n w_{max} < 4.0$. If setting $\mu := \left(\frac{n w_{max}}{4.0}\right) < 1.0$ then we get the exponential decay $\left| \frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} \right| \leq \ n\ (\mu)^q$.

Remember that large $\| \ W\ \|_A$ leads to small $\| \ F'(t-m)\ \|_A$ and, therefore, vanishing gradient is almost ever observed.

3 METHODS FOR LONG TIME LAG LEARNING

Gradient descent based algorithms. The methods [5, 6, 27, 22, 15], and other mentioned in [16] suffer from the vanishing gradient. They are hardly able to learn long-term dependencies. To overcome the vanishing gradient problem there are four types of solutions:

(1) Methods which do not use gradients.

(2) Methods which keep gradients on larger values.

(3) Methods which operate on higher levels.

(4) Methods which use special architectures.

(1) Global search methods do not use a gradient. In [2] methods such as simulated annealing, multi-grid random search were investigated. Random weight guessing was tested in [21]. It was found that global search methods work well on "simple" problems involving long-term dependencies. "Simple" problems are characterized by being solved with nets having few parameters and not needing the computation of precise continuous values.

(2) The gradient can be kept on larger values by time-weighted pseudo-Newton optimization and discrete error propagation [2]. It seems that these methods have problems learning to store precise real-valued information over time.

(3) An EM approach for target propagation was proposed in [1]. This approach uses a discrete number of states and, therefore, will have problems with continuous values.

Kalman filter techniques are used in [18] for recurrent network training. But a derivative discount factor leads to vanishing gradient problems.

If a long-time lag problem contains local regularities then a hierarchical chunker system [23] works well.

In [19] higher order units to bridge long time lags were used. This very fast method is not capable to generalize to temporal dependencies not being trained and the number of additive units increase with the time lags.

(4) Second order nets (using sigma-pi units) are in principle capable to increase the error flow. But vanishing error problems can hardly be avoided. For experiments with these network types see [26] and [13].

With Time-Delay Neural Networks (TDNN, e.g. [11]) old net activations are fed back into the net using fixed delay lines. These delay lines can be viewed as "jump ahead" connections between copies in a time-unfolded network. In best case the length of the delay line is the ratio of the error flowing steps in a conventional net and the error flowing steps using a TDNN. In TDNN the error decrease is slowed down because the error uses "shortcuts" as it gets propagated back. TDNN have to deal with a trade-off: increasing the delay line length increases the error flow but the net has more parameters/units. For a

special case of TDNN (called NARX networks) see [12]. A weighted sum of old activations instead of a fixed delay line was used in [17]. A more complex version of a TDNN was proposed in [4]. The error flow can be controlled by designing special units.

In [14] time constants determine the scaling factor of the error if it gets propagated back for one time step at a single unit. But only with external time constant fine tuning extended time gaps can be processed. In [25] a single unit is updated by adding the old activation and the scaled current net input. But the stored value is sensible to perturbations by later irrelevant net inputs.

"Long Short Term Memory" (LSTM) [8, 10, 9] uses a special architecture to enforces constant error flow through special units. Unlike in [25] perturbations by current irrelevant signals are prevented by multiplicative units.

4 EXPERIMENTS

4.1 EXPERIMENT 1: EMBEDDED REBER GRAMMAR

Task. The "embedded Reber grammar" was used by various previous authors, e.g., [24], [3] and [6]. This task do 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 gradient descent methods.

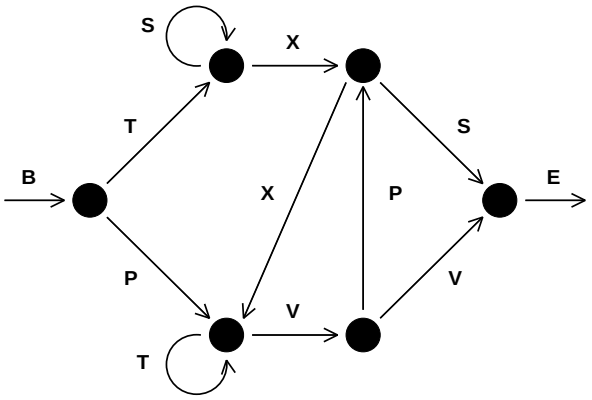

Figure 1: *Reber grammar.*

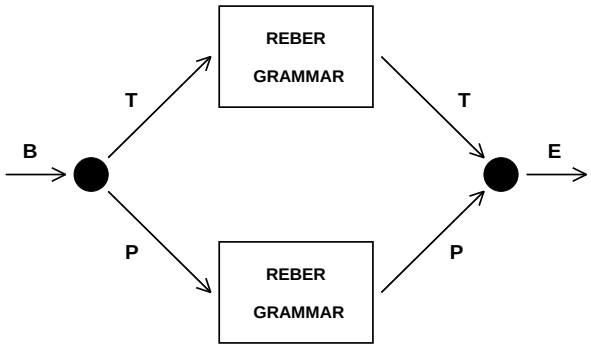

Figure 2: *Embedded Reber grammar.*

Being at the leftmost node (with an empty string) in Figure 2 a string is produced by following the directed edges and adding the corresponding symbols to the current string until being in the rightmost node. Alternative edges are chosen randomly (probability: 0.5). The net sequentially processes the string getting as input the actual symbol and having to predict the next symbol. To know the last but one string symbol the net have to store the second symbol.

The task was tried to be solved by RTRL, Elman nets (ELM), Fahlman's "Recurrent Cascade-Correlation" (RCC), and LSTM. Experimental details can be found in the ref-

Table 1: *EXPERIMENT 1 — Embedded Reber grammar. Percentage of successful trials and learning time for successful trials for RTRL (results taken from [24]), Elman nets (results taken from [3]), RCC (results taken from [6]) and LSTM (results taken from [8]).*

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	3 blocks, size 2	276	0.5	100	8,440

erences listed in table 1. Table 1 gives the results. Only *LSTM always learned the task* and correct solution are learned faster then with gradient based methods.

4.2 EXPERIMENT 2: LONG TIME LAGS

The limitations of gradient descent based methods can be seen on this simple task involving long minimal time lags. But advanced methods are able to learn the task with minimal time lags of 100.

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 which gives a $p+1$ dimensional input vector. Strings are processed sequentially and the net have to predict the next string symbol. For predicting the last symbol the net has to remember the first symbol. Therefore this task involves a minimal time lag of p.

RTRL [20], BPTT, the neural sequence chunker (CH) [23], and LSTM are applied to the task. Experimental details can be found in [8]. Table 2 gives the results. Gradient based methods (RTRL, BPTT) get into trouble when the minimal time lag exceeds 10 steps. CH and LSTM are able to solve the task with long time lags.

Note: The sequences have local regularities required by the neural sequence chunker but not by LSTM. LSTM performs well at sequences without local regularities (see [8]).

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 mentioned. But most approaches have 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 reasonable time. With conventional methods two advanced

Table 2: *EXPERIMENT 2 — Long time lags. Success percentage and learning time until success. With 100 time step delays, only CH and LSTM are successful.*

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

methods (the neural sequence chunker and long short term memory) were compared and it was seen that they performed well on long time lag problems.

6 ACKNOWLEDGMENTS

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

References

[1] Y. Bengio and P. Frasconi. Credit assignment through time: Alternatives to backpropagation. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 75–82. San Mateo, CA: Morgan Kaufmann, 1994.

[2] 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.

[3] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland. Finite-state automata and simple recurrent networks. *Neural Computation*, 1:372–381, 1989.

[4] B. de Vries and J. C. Principe. A theory for neural networks with time delays. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 162–168. San Mateo, CA: Morgan Kaufmann, 1991.

[5] J. L. Elman. Finding structure in time. Technical Report CRL 8801, Center for Research in Language, University of California, San Diego, 1988.

[6] S. E. Fahlman. The recurrent cascade-correlation learning algorithm. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 190–196. San Mateo, CA: Morgan Kaufmann, 1991.

[7] J. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991. See www7.informatik.tu-muenchen.de/~hochreit.

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. Technical Report FKI-207-95, Fakultät für Informatik, Technische Universität München, 1995. Revised 1996 (see www.idsia.ch/~juergen, www7.informatik.tu-muenchen.de/~hochreit).

[9] S. Hochreiter and J. Schmidhuber. Bridging long time lags by weight guessing and "Long Short Term Memory". In F. L. Silva, J. C. Principe, and L. B. Almeida, editors, *Spatiotemporal models in biological and artificial systems*, pages 65–72. IOS Press, Amsterdam, Netherlands, 1996. Serie: Frontiers in Artificial Intelligence and Applications, Volume 37.

[10] S. Hochreiter and J. Schmidhuber. LSTM can solve hard long time lag problems. In *Advances in Neural Information Processing Systems 9*. MIT Press, Cambridge MA, 1997. To be presented at NIPS 96.

[11] K. Lang, A. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–43, 1990.

[12] T. Lin, B. G. Horne, P. Tino, and C. L. Giles. Learning long-term dependencies is not as difficult with NARX recurrent neural networks. Technical Report UMIACS-TR-95-78 and CS-TR-3500, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, 1995.

[13] 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.

[14] M. C. Mozer. Induction of multiscale temporal structure. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 275–282. San Mateo, CA: Morgan Kaufmann, 1992.

[15] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

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

[17] T. A. Plate. Holographic recurrent networks. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 34–41. San Mateo, CA: Morgan Kaufmann, 1993.

[18] 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.

[19] M. B. Ring. Learning sequential tasks by incrementally adding higher orders. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 115–122. Morgan Kaufmann, 1993.

[20] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

[21] J. Schmidhuber and S. Hochreiter. Guessing can outperform many long time lag algorithms. Technical Report IDSIA-19-96, IDSIA, 1996.

[22] J. H. 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.

[23] J. H. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[24] 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.

[25] G. Sun, H. Chen, and Y. Lee. Time warping invariant neural networks. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 180–187. San Mateo, CA: Morgan Kaufmann, 1993.

[26] R. L. Watrous and G. M. Kuhn. Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4:406–414, 1992.

[27] R. J. Williams. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989.

[28] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. E. Rumelhart, editors, *Back-propagation: Theory, Architectures and Applications*. Hillsdale, NJ: Erlbaum, 1992.