Framewise Phoneme Classification with Bidirectional LSTM Networks

Alex Graves and Jürgen Schmidhuber
IDSIA
Galleria 2
6928 Manno-Lugano
Switzerland
E-mail: alex@idsia.ch, juergen@idsia.ch

***Abstract*— In this paper, we apply bidirectional training to a Long Short Term Memory (LSTM) network for the first time. We also present a modified, full gradient version of the LSTM learning algorithm. On the TIMIT speech database, we measure the framewise phoneme classification ability of bidirectional and unidirectional variants of both LSTM and conventional Recurrent Neural Networks (RNNs). We find that the LSTM architecture outperforms conventional RNNs and that bidirectional networks outperform unidirectional ones.**

I. Introduction

The goal of continuous speech recognition is to provide a mapping from sequences of acoustic frames to sequences of linguistic symbols (phonemes, syllables or words). This mapping is asychronous, since each symbol may occupy several frames, and the symbol boundaries are not generally known in advance. The direct application of neural networks to continuous speech recognition is therefore problematic, since neural nets are designed to learn synchronous mappings between sequences of input-output pairs. However, in the so-called hybrid approach [18], [5] the classification of individual frames of acoustic data into phonemes is used as a first step towards full speech recognition. Assuming that the classifications can be interpreted as posterior probabilities of phoneme occupancy (as they can for the results in this paper — see Section V-B), Bayes' theorem is then applied to convert them to scaled likelihoods of acoustic data given the phoneme class. These likelihoods are then used by Hidden Markov Models to find the most probable sequence of phonemes, and thereby recognise the utterance.

We have focused on the sub-task of framewise phoneme classification because we believe that an improvement there will lead to an improvement in the overall performance of a hybrid recognition system.

The structure of the rest of this paper is as follows: in Section II we discuss bidirectional networks, and answer a possible objection to their use in causal tasks; in Section III we describe the Long Short Term Memory (LSTM) network architecture, our modification to its error gradient calculation, and the possibility of training it with different weight update algorithms; in Section IV we describe the experimental data and how we used it in our experiments; in Section V we give the structure and training parameters of the networks; in Section VI we present and discuss our experimental results, and the conclusion is found in Section VII. In Appendix A we provide the pseudocode for training LSTM networks with a full gradient calculation, and in Appendix B we give an outline of bidirectional training with Recurrent Neural Nets.

II. Bidirectional Recurrent Neural Nets

For many sequence processing tasks, it is useful to analyze the future as well as the past of a given point in the series. However, conventional recurrent neural nets are designed to analyse data in one direction only — the past. A partial solution to this shortcoming is to introduce a delay between inputs and their associated targets, thereby giving the net a few timesteps of future context. But this amounts to little more than the fixed time-windows used for MLPs — exactly what RNNs were designed to replace. A better approach is provided by the bidirectional networks pioneered by Schuster [22] and Baldi [2]. In this model, the input is presented forwards and backwards to two separate recurrent nets, both of which are connected to the same output layer. See Appendix B for an outline of the algorithm. Bidirectional recurrent neural nets (BRNNs) have given significantly improved results in sequence learning tasks, notably protein structure prediction (PSP) [1], [6] and speech processing [21], [9].

A. Bidirectional Networks and Online Causal Tasks

In a purely spatial task like PSP, it is clear that any distinction between input directions should be discarded. But for temporal problems such as speech recognition, relying on knowledge of the future seems at first sight to violate causality — at least if the task is online. How can we base our understanding of we've heard on something that hasn't been said yet? However, human listeners do exactly that. Sounds, words, and even whole sentences that at first mean nothing are found to make sense in the light of future context. What we need to bear in mind is the distinction between tasks that are truly online - requiring an output after every input - and those where outputs are only needed at the end of some input segment. For the first class of problems, BRNNs are useless, since meaningful outputs are only available after the net has run backwards. But the point is that speech recognition, along with most other 'online' causal tasks, is in the second class: an output at the end of every sentence is fine. Therefore, we see no objection to using BRNNs to gain improved performance on speech tasks. On a more practical note, given the relative speed of activating neural nets, the delay incurred by running an already trained net backwards as well as forwards is small.

In general, the BRNNs examined here make the following assumptions about their input data: that it can be divided into finitely long segments, and that each of these is unaffected by the others. For speech corpora like TIMIT, made up of separately recorded utterances, this is clearly the case. For real speech, the worst it can do is neglect contextual effects that extend across segment boundaries — e.g. the ends of sentences or dialogue turns. Moreover, such long term effects are routinely neglected by current speech recognition systems.

III. LSTM

The Long Short Term Memory architecture was first presented in [15] and later extended in [11]. It was motivated by an analysis of error flow in existing RNNs [14], which found that long time lags were inaccessible to existing architectures, because backpropagated error either blows up or decays exponentially.

An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of a differentiable version of the memory chips in a digital computer. Each one contains one or more recurrently connected memory cells and three multiplicative units - the input, output and forget gates - that provide continuous analogues of write, read and reset operations for the cells. More precisely, the input to the cells is multiplied by the activation of the input gate, the output to the net is multiplied by that of the output gate, and the previous cell values are multiplied by the forget gate (see Figure 1). The net can only interact with the cells via the gates.

Recently, we have concentrated on applying LSTM to real world sequence processing problems. In particular, we have studied isolated word recognition [13], [12] and continuous speech recognition [8], [3], with promising results.

A. LSTM Gradient Calculation

The original LSTM training algorithm [11] used an error gradient calculated with a combination of Real Time Recurrent Learning (RTRL)[19] and Back Propagation Through Time (BPTT)[23]. The backpropagation was truncated after one timestep, because it was felt that long time dependencies would be dealt with by the memory blocks, and not by the (vanishing) flow of backpropagated error gradient. Partly to check this supposition, and partly to ease the implementation of Bidirectional LSTM, we calculated the full error gradient for the LSTM architecture. See Appendix A for the revised pseudocode. For both bidirectional and unidirectional nets, we found that gradient descent on the full gradient gave slightly higher performance than the original algorithm. It had the added benefit of making the LSTM architecture directly comparable to other RNNs, since it could now be trained with

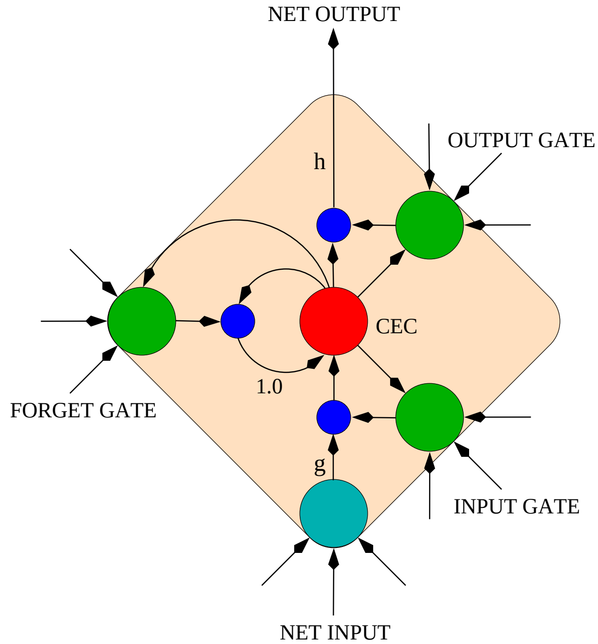

Fig. 1. LSTM memory block with one cell. The internal state of the cell is maintained with a recurrent connection of weight 1.0. The three gates collect activations from inside and outside the block, and control the cell via multiplicative units (small circles). The input and output gates scale the input and output of the cell while the forget gate scales the internal state—for example by resetting it to 0 (making it forget). The cell input and output squashing functions (g and h) are applied at the indicated places.

standard BPTT. Also, since the full gradient can be checked numerically, its implementation was easier to debug.

B. LSTM Training Algorithm

The effectiveness of LSTM networks comes from their novel architecture, and not from the way they are trained: indeed, almost all LSTM experiments so far (including this one) have used one of the simplest training algorithms — gradient descent with momentum. However, there is no reason that alternative methods developed for training RNNs could not equally be applied to LSTM.

In the past, training with decoupled Kalman filters [16], has given improved results for several tasks. We are currently experimenting with a range of weight training algorithms, including Stochastic Meta-Descent [20], RPROP [17] and Robinson's algorithm [18].

IV. Experimental Data

Our experiments were carried out on the TIMIT database [10] of prompted utterances, collected by Texas Instruments. The utterances were chosen to be phonetically rich, and the speakers represent a wide variety of American dialects. The audio data is divided into sentences, each of which is accompanied by a complete phonetic transcript.

We preprocessed the audio data into 12 Mel-Frequency Cepstrum Coefficients (MFCC's) from 26 filter-bank channels. We also extracted the log-energy and the first order derivatives of it and the other coefficients, giving a vector of 26 coefficients per frame in total. The frame size was 5 ms and the input window was 10 ms.

A. *Reduced Phoneme Set*

Of the 62 distinct phonemes given in the TIMIT lexicon, we created a reduced set of 43 by making the following identifications:

- The closures 'bcl','dcl','gcl','pcl','tck','kcl' and 'tcl' were identified either with the appropriate stop following them ('b','d' or 'jh','g','p','t','k' and 'ch', respectively), or with the glottal stop 'q', if they were followed by silence or a vowel.
- The silence markers 'pau','sil' and 'h#' were considered equivalent.
- The flaps 'dx' and 'nx' were identified with 'd' and 'n' respectively.
- The nasals 'em', 'eng' and 'en' were identified with 'm', 'ng' and 'n'.
- The semivowel/glides 'el' and 'hv' were identified with 'l' and 'hh'.
- The vowels 'ux', 'ax-h' and 'ix' were identified with 'uw', 'ax' and 'ih'.

These identifications were based on a desire to minimise the set of phonetic outputs that would allow us to ultimately identify the words, rather than an effort to improve our results.

B. *Training and Testing Sets*

The TIMIT database comes partitioned into training and test sets, containing 4620 and 1680 utterances respectively. We used 462 of the training set utterances as a validation set and trained on the rest. When the experiments were finished we restored the nets to the weights that gave the lowest error on the validation set. We then measured their performance on the test set. We did not measure test set error during the experiments. We quote results for the training and test sets separately.

V. Experimental Setup

A. *Topology*

We found that large networks, of around 200,000 weights, gave good performance. However, it is possible that smaller nets would have generalised better to the test set. With the aim of keeping the number of parameters roughly constant, we chose the following topologies:

- A unidirectional net with a hidden LSTM layer containing 205 memory blocks, with one cell each.
- A bidirectional net with two hidden LSTM layers (one forwards and one backwards) each containing 140 one cell memory blocks.
- A unidirectional net with a hidden layer containing 410 sigmoidal units.
- A bidirectional net with two hidden layers (one forwards and one backwards) containing 280 sigmoidal units each.

All nets contained an input layer of size 26 (an input for each MFCC coefficient), and an output layer of size 43. The input layers were fully connected to the hidden layers and the hidden layers fully connected to themselves and the output layers. All LSTM blocks had the following activation functions: logistic sigmoids in the range $[-2, 2]$ for the input and output squashing functions of the cell (g and h in Figure 1), and in the range $[0, 1]$ for the gates. The non-LSTM nets had logistic sigmoid activations in the range $[0, 1]$ in the hidden layers.

B. *Output Layers*

For the output layers, we used the cross entropy objective function and the softmax activation function, as is standard for 1 of K classification [4]. The softmax function ensures that the network outputs are all between zero and one, and that they sum to one on every timestep. This allows them to be interpreted as the posterior probabilities of the phonemes at a given frame, given all the inputs up to the current one (with unidirectional nets) or all the inputs in the whole sequence (with bidirectional nets).

Several alternative objective functions have been studied for this task [7]. One modification in particular has been shown to have a positive effect on full speech recognition (though not necessarily on framewise classification). This is to weight the error according to the duration of the current phoneme, which ensures that short phonemes are as significant to the training as longer ones.

C. *Network Training*

All nets were trained with gradient descent (error gradient calculated with BPTT), using a learning rate of 10^{-5} and a momentum of 0.9. At the end of each utterance, weight updates were carried out and network activations were reset to 0. For the unidirectional nets a delay of 4 timesteps was introduced between the target and the current input — i.e. the net always tried to predict the phoneme it had seen 4 timesteps ago.

VI. Results

TABLE I

Framewise Phoneme Classification on the TIMIT Database

System	Training Set	Test Set
Bidirectional LSTM	78.1%	73.2%
Unidirectional LSTM	75.2%	70.1%
Bidirectional RNN	66.6%	65.3%
Unidirectional RNN	63.0%	61.9%

The results in Table I show that LSTM outperforms conventional RNNs on framewise phoneme recognition and that bidirectional training outperforms unidirectional. Since the weight updating method for all nets is identical (gradient descent with momentum), and run with identical parameters, the improvements can only be due to architectural advantages — e.g. the ability of LSTM to bridge long time lags, and that of bidirectional training to process reverse time dependencies.

As can be seen from the greater difference between their training and test set scores, The LSTM nets were more prone to overfitting than conventional RNNs. Indded, in an experiment with an LSTM net half the size of those described here, we achieved a framewise score of 86.4% on the training set (although the score on the test set never got above 70%). We are currently investigating methods for improved generalisation.

VII. Conclusions and Future Work

We have presented a bidirectional LSTM network for the first time. We have also calculated the full error gradient for LSTM weights. Combining these methods, we have achieved a high framewise recognition score on the TIMIT database, and demonstrated the architectural advantage of bidirectional training over unidirectional and of LSTM over conventional RNNs. We conclude both that LSTM is well suited to framewise phoneme classification, and that bidirectional networks are able to capture time dependencies in speech that elude unidirectional nets — even if a target delay is added.

In the future we intend to experiment with alternative LSTM learning algorithms and output error functions, and with methods for improved generalisation. We also intend to implement a hybrid full speech recognition system, combining LSTM with Hidden Markov Models.

Appendix A: Pseudocode for Full Gradient LSTM

The following pseudocode details the forward pass, backward pass, and weight updates of an extended LSTM layer in a multi-layer net. The error gradient is calculated with sequencewise BPTT (i.e. truncated BPTT with weight updates after every input sequence). As is standard with BPTT, the network is unfolded over time, so that connections arriving at layers are viewed as coming from the previous timestep. We have tried to make it clear which equations are LSTM specific, and which are part of the standard BPTT algorithm. Note that for the LSTM specific equations, the order of execution is important.

Notation

The input sequence over which the training takes place is labelled S and it runs from time τ_0 to τ_1. $x_k(\tau)$ refers to the network input to unit k at time τ, and $y_k(\tau)$ to its activation. Unless stated otherwise, all network inputs, activations and partial derivatives are evaluated at time τ — e.g. $y_c = y_c(\tau)$. $E(\tau)$ refers to the (scalar) output error of the net at time τ. The training target for output unit k at time τ is denoted $t_k(\tau)$, and the resulting output error is $e_k(\tau)$. For the output layers used here, $e_k(\tau) = y_k(\tau) - t_k(\tau)$. The error backpropagated to unit k at time τ is denoted $\epsilon_k(\tau)$. N is the set of all units in the network, including input and bias units, that can be connected to other units. Note that this includes LSTM cell outputs, but not LSTM gates or internal states (whose activations are only visible within their own memory blocks). W_{ij} is the weight *from* unit j *to* unit i.

The LSTM equations are given for a single memory block only. The generalisation to multiple blocks is trivial: simply repeat the calculations for each block, in any order. Within each block, we use the suffixes ι, ϕ and ω to refer to the input gate, forget gate and output gate respectively. The suffix c refers to an element in the set of cells C. s_c refer to the state value of cell c — i.e. its value after the input and forget gates have been applied. f is the squashing function of the gates, and g and h are respectively the cell input and output squashing functions (see Figure 1).

Forward Pass

- Reset all activations to 0.
- Running forwards from time τ_0 to time τ_1, feed in the inputs and update the activations. Store all hidden layer and output activations at every timestep.
- For each LSTM block, the activations are updated as follows:

Input Gates:

$$x_\iota = \sum_{j \in N} w_{\iota j} y_j(\tau - 1) + \sum_{c \in C} w_{\iota c} s_c(\tau - 1)$$
$$y_\iota = f(x_\iota)$$

Forget Gates:

$$x_\phi = \sum_{j \in N} w_{\phi j} y_j(\tau - 1) + \sum_{c \in C} w_{\phi c} s_c(\tau - 1)$$
$$y_\phi = f(x_\phi)$$

Cells:

$$\forall c \in C, \; x_c = \sum_{j \in N} w_{cj} y_j(\tau - 1)$$
$$s_c = y_\phi s_c(\tau - 1) + y_\iota g(x_c)$$

Output Gates:

$$x_\omega = \sum_{j \in N} w_{\omega j} y_j(\tau - 1) + \sum_{c \in C} w_{\omega c} s_c(\tau)$$
$$y_\omega = f(x_\omega)$$

Cell Outputs:

$$\forall c \in C, \; y_c = y_\omega h(s_c)$$

Backward Pass

- Reset all partial derivatives to 0.
- Starting at time τ_1, propagate the output errors backwards through the unfolded net, using the standard BPTT equations:

$$define \quad \delta_k(\tau) = \frac{\partial E(\tau)}{\partial x_k}$$
$$\epsilon_j(\tau_1) = e_j(\tau_1)$$
$$\epsilon_j(\tau - 1) = e_j(\tau - 1) + \sum_{i \in N} w_{ij} \delta_i(\tau)$$

- For each LSTM block the δ values are calculated as follows:

Cell Outputs:

$$\forall c \in C, \; \epsilon_c = \sum_{j \in N} w_{jc} \delta_j(\tau + 1)$$

Output Gates:

$$\delta_\omega = f'(x_\omega) \sum_{c \in C} \epsilon_c h(s_c)$$

States:

$$\frac{\partial E}{\partial s_c}(\tau) = \epsilon_c y_\omega h'(y_c) + \frac{\partial E}{\partial s_c}(\tau+1) y_\phi(\tau+1)$$
$$+\delta_\iota(\tau+1)w_{\iota c} + \delta_\phi(\tau+1)w_{\phi c} + \delta_\omega w_{\omega c}$$

Cells:

$$\forall c \in C,\ \delta_c = y_\iota g'(x_c) \frac{\partial E}{\partial s_c}$$

Forget Gates:

$$\delta_\phi = f'(x_\phi) \sum_{c \in C} \frac{\partial E}{\partial s_c} y_c(\tau - 1)$$

Input Gates:

$$\delta_\iota = f'(x_\iota) \sum_{c \in C} \frac{\partial E}{\partial s_c} g(x_c)$$

- Using the standard BPTT equation, accumulate the δ's to get the partial derivatives of the cumulative sequence error:

$$define \quad E_{total}(S) = \sum_{\tau=\tau_0}^{\tau_1} E(\tau)$$

$$define \quad \bigtriangledown_{ij}(S) = \frac{\partial E_{total}(S)}{\partial w_{ij}}$$

$$\bigtriangledown_{ij}(S) = \sum_{\tau=\tau_0+1}^{\tau_1} \delta_i(\tau) y_j(\tau - 1)$$

Update Weights

- After the presentation of sequence S, with learning rate α and momentum m, update all weights with the standard equation for gradient descent with momentum:

$$\Delta w_{ij}(S) = \alpha \bigtriangledown_{ij}(S) + m \Delta w_{ij}(p-1)$$

Appendix B: Algorithm Outline for Bidirectional Recurrent Neural Networks

From [21] we give the following method for training bidirectional recurrent nets with BPTT. As above, training takes place over an input sequence running from time τ_0 to τ_1. All network activations are set to 0 at τ_0 and τ_1:

Forward Pass Feed all input data for the sequence into the BRNN and determine all predicted outputs.

- Do forward pass just for forward states (from time τ_0 to τ_1) and backward states (from time τ_1 to τ_0).
- Do forward pass for output layer.

Backward Pass Calculate the error function derivative for the sequence used in the forward pass.

- Do backward pass for output neurons.
- Do backward pass just for forward states (from time τ_1 to τ_0) and backward states (from time τ_0 to τ_1).

Update Weights

Acknowledgments

The authors would like to thank Nicole Beringer for her expert advice on linguistics and speech recognition. This work was supported by the SNF under grant number 200020-100249.

References

[1] P. Baldi, S. Brunak, P. Frasconi, G. Pollastri, and G. Soda. Bidirectional dynamics for protein secondary structure prediction. *Lecture Notes in Computer Science*, 1828:80–104, 2001.

[2] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri. Exploiting the past and the future in protein secondary structure prediction. *BIOINF: Bioinformatics*, 15, 1999.

[3] N. Beringer. Human language acquisition methods in a machine learning task. In *Proceedings of the 8th International Conference on Spoken Language Processing*, pages 2233–2236, 2004.

[4] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[5] H.A. Bourlard and N. Morgan. *Connnectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, 1994.

[6] J. Chen and N. S. Chaudhari. Capturing long-term dependencies for protein secondary structure prediction. In Fuliang Yin, Jun Wang, and Chengan Guo, editors, *Advances in Neural Networks - ISNN 2004, International Symposiumon Neural Networks, Part II*, volume 3174 of *Lecture Notes in Computer Science*, pages 494–500, Dalian, China, 2004. Springer.

[7] R. Chen and L. Jamieson. Experiments on the implementation of recurrent neural networks for speech phone recognition. In *Proceedings of the Thirtieth Annual Asilomar Conference on Signals, Systems and Computers*, pages 779–782, 1996.

[8] D. Eck, A. Graves, and J. Schmidhuber. A new approach to continuous speech recognition using LSTM recurrent neural networks. Technical Report IDSIA-14-03, IDSIA, www.idsia.ch/techrep.html, May 2003.

[9] T. Fukada, M. Schuster, and Y. Sagisaka. Phoneme boundary estimation using bidirectional recurrent neural networks and its applications. *Systems and Computers in Japan*, 30(4):20–30, 1999.

[10] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, , and N. L. Dahlgren. Darpa timit acoustic phonetic continuous speech corpus cdrom, 1993.

[11] F. Gers, N. Schraudolph, and J. Schmidhuber. Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, 3:115–143, 2002.

[12] A. Graves, N. Beringer, and J. Schmidhuber. A comparison between spiking and differentiable recurrent neural networks on spoken digit recognition. In *The 23rd IASTED International Conference on modelling, identification, and control*, Grindelwald, 2004.

[13] A. Graves, D. Eck, N. Beringer, and J. Schmidhuber. Biologically plausible speech recognition with lstm neural nets. In *First International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, Lausanne, 2004.

[14] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

[15] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[16] J. A. Pérez-Ortiz, F. Gers, Douglas Eck, and Juergen Schmidhuber. Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. *Neural Networks*, 2002. In press.

[17] M. Riedmiller and H. Braun. Rprop – a fast adaptive learning algorithm. In *Proceedings of the 1992 International Symposium on Computer and Information Sciences*, pages 279–285, Antalya, Turkey, 1992.

[18] A. J. Robinson. An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5(2):298–305, March 1994.

[19] 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.

[20] Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

[21] M. Schuster. *On supervised learning from sequential data with applications for speech recognition*. PhD thesis, Nara Institute of Science and Technolog, Kyoto, Japan, 1999.

[22] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681, November 1997.

[23] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent connectionist networks. In Y. Chauvin and D. E. Rumelhart, editors, *Backpropagation: Theory, Architectures, and Applications*. Erlbaum, Hillsdale, NJ, 1990.