

Long Short-Term Memory Learns Context Free and Context Sensitive Languages

Felix A. Gers, Jürgen Schmidhuber *

*IDSIA, Galleria 2, 6928 Manno, Switzerland, www.idsia.ch

Abstract

Previous work on learning regular languages from exemplary training sequences showed that Long Short-Term Memory (LSTM) outperforms traditional recurrent neural networks (RNNs). Here we demonstrate LSTM's superior performance on context free language (CFL) benchmarks, and show that it works even better than previous hardwired or highly specialized architectures. To the best of our knowledge, LSTM variants are also the first RNNs to learn a context *sensitive* language (CSL), namely, $a^n b^n c^n$.

1 Introduction

Until recently standard recurrent neural networks RNNs (see survey by Pearlmutter, 1995) have been plagued by a major practical problem: the gradient of the total output error with respect to previous inputs quickly vanishes as the time lags between relevant inputs and errors increase. Hence standard RNNs fail to learn in the presence of time lags exceeding as few as 5 - 10 discrete time steps between relevant input events and target signals.

The recent “*Long Short-Term Memory*” (LSTM) method [3], however, is not affected by this problem. LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through “constant error carousels” (CECs) within special units, without loss of short time lag capabilities. Multiplicative gate units learn to open and close access to the constant error flow. Moreover, LSTM's learning algorithm is more efficient than previous RNN algorithms such as real time recurrent learning (RTRL) and back propagation through time (BPTT): it is local in space and time, with computational complexity $O(1)$ per time step and weight.

Previous work showed that LSTM outperforms traditional RNN algorithms on numerous tasks involving real-valued or discrete inputs and targets [2, 3], including tasks that require to learn the rules of regular languages (RLs) describable by deterministic finite state automata (DFA). Until now, however, it has remained unclear whether LSTM's superiority carries over to tasks involving context free languages (CFLs), such as those discussed in the RNN

literature [7, 9, 8, 5, 6]. Their recognition requires the functional equivalent of a stack. It is conceivable that LSTM has just the right bias for RLs but might fail on CFLs. Here we will focus on the most common CFL benchmarks: $a^n b^n$ and $a^n b^m B^m A^n$. Finally we will apply LSTM to a context *sensitive* language (CSL). The CSLs include the CFLs, which include the RLs. We will focus on the classic example $a^n b^n c^n$, which is a CSL but not a CFL. In general, CSL recognition requires a linear-bounded automaton, a special Turing machine whose tape length is at most linear in the input size. To our knowledge no RNN has been able to learn a CSL.

2 LSTM

We are using LSTM with forget gates and the recently introduced peephole connections [1]. 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 “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*). Peephole connections connect the CEC to the gates. 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 non-linear 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 already fail to learn in the presence of 10 step time lags, despite requiring more

complex update algorithms.

Forward Pass. See Gers and Schmidhuber (2000) for a detailed description of LSTM’s forward pass with forget gates and peephole connections. Essentially, the cell output, y^c , is calculated based on the current cell state s_c and four sources of input: to the cell itself, to the input gate, to the forget gate and input to output gate. All gates have a sigmoid squashing functions with range $[0, 1]$. The state of memory cell $s_c(t)$ is calculated by adding the squashed (the squashing function g is a sigmoid with range $[-1, 1]$), gated input to the state at the previous time step $s_c(t-1)$, which is multiplied by the forget gate activation. The cell output y^c is calculated by multiplying (gating) $s_c(t)$ by the output gate activation.

Gradient-Based Backward Pass. Essentially, LSTM’s backward pass (for details see Hochreiter and Schmidhuber (1997) and Gers et. al. (2000)) is an efficient fusion of slightly modified, truncated BPTT and a customized version of RTRL. We are using iterative gradient descent, minimizing an objective function E , here the usual mean squared error function. Unlike BPTT and RTRL, LSTM’s learning algorithm is local in space and time.

3 Experiments

The network sequentially observes exemplary symbol strings of a given language, presented one input symbol at a time.

Following the traditional approach in the RNN literature we formulate the task as a prediction task. At any given time step the target is to predict the possible next symbols, including the terminal symbol T . When more than one symbol can occur in the next step *all* possible symbols have to be predicted, and none of the others.

Every input sequence begins with the start symbol S . The empty string, consisting of ST only, is considered part of each language. A string is accepted when all predictions have been correct. Otherwise it is rejected.

This prediction task is equivalent to a classification task with the two classes “accept” and “reject”, because the system will make prediction errors for all strings outside the language. A system has learned a given language up to string size n once it is able to correctly predict all strings with size $\leq n$.

Symbols are encoded locally in d -dimensional vectors, where d is equal to the number of symbols of the given language plus one for either the start symbol in the input or the terminal symbol in the output (d input units, d output units, each standing for one of the symbols). +1 signifies that a symbol is set and -1 that it is not set; the decision boundary for the network output is 0.0.

CFL $a^n b^n$ [9, 8, 7, 5]. Here the strings in the input sequences are of the form $a^n b^n$, input and output vectors are 3-dimensional. Before the first occurrence of b either a or b , or a or T at sequence beginnings, are possible in the next step. Thus, e.g., for $n =$

4: Input: S a a a a b b b b
Target: a/T a/b a/b a/b a/b b b b T

CFL $a^n b^m B^m A^n$ [6]. The second half of a string from this palindrome or mirror language is completely predictable from the first half. This task involves an intermediate time lag of length $2m$. Input and output vectors are 5-dimensional. Before the first occurrence of B two symbols are possible in the next step. Thus, e.g., for $n = 2, m = 2$:

Input: S a a b b B B A A
Target: a/T a/b a/b b/B b/B B A A T

CSL $a^n b^n c^n$. Input and output vectors are 4-dimensional. Before the first occurrence of b two symbols are possible in the next step. Thus, e.g., for $n = 3$:

Input: S a a a b b b c c c
Target: a/T a/b a/b a/b b b c c c T

3.1 Training and Testing

Learning and testing alternate: after 1000 training sequences we freeze the weights and run a test. Training and test sets incorporate all legal strings up to a given length: $2n$ for $a^n b^n$, $3n$ for $a^n b^n c^n$ and $2(n + m)$ for $a^n b^m B^m A^n$. Only positive exemplars are presented. Training is stopped once all training sequences have been accepted, or after at most 10^7 training sequences. The *generalization set* is the largest accepted test set.

Weight changes are made after each sequence. We apply the momentum algorithm with learning rate α is 10^{-5} and momentum parameter 0.99. All results are averages over 10 independently trained networks with different weight initializations (the same for each experiment).

CFL $a^n b^n$. We study training sets with $n \in \{1, \dots, N\}$. We test all sets with $n \in \{1, \dots, M\}$ and $M \in \{N, \dots, 1000\}$ (sequences of length ≤ 2000).

CFL $a^n b^m B^m A^n$. We use two training sets: a) The same set as used by Rodriguez and Wiles (1999) : $n \in \{1, \dots, 11\}$, $m \in \{1, \dots, 11\}$ with $n + m \leq 12$ (sequences of length ≤ 24). b) The set given by $n \in \{1, \dots, 11\}$, $m \in \{1, \dots, 11\}$ (sequences of length ≤ 48). We test all sets with $n \in \{1, \dots, M\}$, $m \in \{1, \dots, M\}$ and $M \in \{11, \dots, 50\}$ (sequences of length ≤ 200).

CSL $a^n b^n c^n$. We study training sets with the same parameters as for the CFL $a^n b^n$. We test all sets with $n \in \{1, \dots, M\}$ and $M \in \{N, \dots, 500\}$ (sequences of length ≤ 1500).

3.2 Topology and Experimental Parameters

The input units are fully connected to a hidden layer consisting of memory blocks with 1 cell each. The cell outputs are fully connected to the cell inputs, to all gates, and to the output units, which also have direct “shortcut” connections from the input units. 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 -1.0 , $+2.0$ and -2.0 , respectively (these are standard values, which we use for all our experiments; precise initialization is not critical here). All other weights are initialized randomly in the range $[-0.1, 0.1]$. The cell’s input squashing function g is the identity function. The squashing function of the output units is a sigmoid function with the range $[-2, 2]$.

CFL $a^n b^n$. We use one memory block (with one cell). Without the 3 peephole connections there are 37 adjustable weights (28 unit-to-unit and 7 bias connections).

CFL $a^n b^m B^m A^n$. We use two blocks (with one cell each), resulting 114 adjustable weights (91 unit-to-unit and 13 bias connections).

CSL $a^n b^n c^n$. We use the same topology as for the $a^n b^m B^m A^n$ language, but with 4 input and output units instead of 5, giving 84 adjustable weights (72 unit-to-unit and 12 bias connections).

3.3 Previous results

CFL $a^n b^n$. Published results on the $a^n b^n$ language are summarized in Table 1. RNNs trained with plain BPTT tend to learn to just reproduce the input [9, 8, 5]. Sun et al. (1993) used a highly specialized architecture, the “neural pushdown automaton”, which also did not generalize well.

CFL $a^n b^m B^m A^n$. Rodriguez and Wiles (1998) used BPTT-RNNs with 5 hidden nodes. After training with $n + m \leq 12$ (sequences of length ≤ 24), the best network generalized to sequences up to length 36 ($n = 9, m = 9$). But their networks did not learn the complete training set, and generalization was restricted to few strings with $n, m \in \{1, \dots, 9\}$.

CSL $a^n b^n c^n$. To our knowledge no previous RNN ever learned a CSL.

3.4 LSTM Results

CFL $a^n b^n$. 100% solved for all training sets (lhs of Table 2). Small training sets ($n \in \{1, \dots, 10\}$) were already sufficient for perfect generalization up to the tested maximum: $n \in \{1, \dots, 1000\}$. Note that long sequences of this kind require very stable, finely tuned control of the network’s internal counters.

This performance is much better than in previous approaches, where the largest set was learned by the specially designed neural push-down automa-

ton [7] $n \in \{1, \dots, 160\}$. The latter, however, required training sequences of the same length as the test sequences. From the training set with $n \in \{1, \dots, 10\}$ LSTM generalized to $n \in \{1, \dots, 1000\}$, whereas the best previous result (see Table 1) generalized only to $n \in \{1, \dots, 18\}$ (even with a slightly larger training set: $n \in \{1, \dots, 11\}$).

In contrast to Tonkes and Wiles (1997), we did not observe our networks forgetting solutions as training progresses. So unlike all previous approaches, LSTM reliably finds solutions that generalize well.

CFL $a^n b^m B^m A^n$. Training set a): 100% solved; after $29 \cdot 10^3$ training sequences the best network of 10 generalized to at least $n, m \in \{1, \dots, 22\}$ (all strings until a length of 88 symbols processed correctly); the average generalization set was the one with $n, m \in \{1, \dots, 16\}$ (all strings until a length of 64 symbols processed correctly), learned after $25 \cdot 10^3$ training sequences on average.

Training set b): 100% solved; after $26 \cdot 10^3$ training sequences the best network generalized to at least $n, m \in \{1, \dots, 23\}$ (all strings until a length of 92 symbols processed correctly). The average generalization set was the one with $n, m \in \{1, \dots, 17\}$ (all strings until a length of 68 symbols processed correctly), learned after $82 \cdot 10^3$ training sequences on average.

Unlike the previous approach of Rodriguez and Wiles (1998), LSTM easily learns the complete training set and reliably finds solutions that generalize well.

CSL $a^n b^n c^n$. LSTM learns 4 of the 5 training sets in 10 out of 10 trials (only 9 out of 10 for the training set with $n \in \{1, \dots, 40\}$) and generalizes well (rhs Table 2). Small training sets ($n \in \{1, \dots, 40\}$) were already sufficient for perfect generalization up to the tested maximum: $n \in \{1, \dots, 500\}$, that is, sequences of length up to 1500.

3.5 Analysis

How do the solutions discovered by LSTM work?

CFL $a^n b^n$. The cell state s_c increases while a symbols are fed into the network, then decreases (with the same step size) while b symbols are fed in. At sequence beginnings (when the first a symbols are observed), however, the step size is smaller due to the closed input gate, which is triggered by s_c itself. This results in “overshooting” the initial value of s_c at the end of a sequence and leads to the prediction of the sequence termination.

CFL $a^n b^m B^m A^n$. The network learned to establish and control two counters, the two symbol pairs (a, A) and (b, B) are treated separately by two different cells, c_2 and c_1 , respectively. Cell c_2 tracks the difference between the number of observed a and A symbols. It opens only at the end of a string, where

Table 1. Previous results for the CFL $a^n b^n$, showing (from left to right) the number of hidden units or state units, the values of n used during training, the number of training sequences, the number of found solutions/trials and the largest accepted test set.

Reference	Hidden Units	Train. Set $[n]$	Train. Str. $[10^3]$	Sol./Tri.	Best Test $[n]$
[7] ¹	5	1,.., 160	13.5	1/1	1,.., 160
[9]	2	1,.., 11	2000	4/20	1,.., 18
[8]	2	1,.., 10	10	13/100	1,.., 12
[5] ²	2	1,.., 11	267	8/50	1,.., 16

Table 2. Results for the CFL $a^n b^n$ (lhs) and for the CSL $a^n b^n c^n$ (rhs), showing (from left to right) the values for n used during training, the average number of training sequences until best generalization was achieved (average over all networks given in parenthesis), the percentage of correct solutions and the best generalization (average over all networks given in parenthesis).

Train. Set $[n]$	CFL $a^n b^n$		CSL $a^n b^n c^n$	
	Train. Str. $[10^3]$	Generalization Set $[n]$	Train. Str. $[10^3]$	Generalization Set $[n]$
1,.., 10	22 (19)	1,.., 1000 (1,.., 118)	54 (62)	1,.., 52 (1,.., 28)
1,.., 20	18 (19)	1,.., 587 (1,.., 148)	28 (43)	1,.., 160 (1,.., 66)
1,.., 30	16 (19)	1,.., 1000 (1,.., 408)	37 (43)	1,.., 228 (1,.., 91)
1,.., 40	25 (28)	1,.., 1000 (1,.., 628)	51 (48)	1,.., 500 (1,.., 120)
1,.., 50	42 (40)	1,.., 767 (1,.., 430)	60 (94)	1,.., 500 (1,.., 409)

it predicts the final T . Cell c_1 treats the embedded $b^m B^m$ substring in a similar way. While values are stored and manipulated within a cell, the output gate remains closed. This prevents the cell from disturbing the rest of the network and also protects its CEC against incoming errors.

CSL $a^n b^n c^n$. The network solutions use a combination of two counters, instantiated separately in the two memory blocks. Here the second cell counts up, given an a input symbol. It counts down, given a b . The second memory block does the same for b , c , and a , respectively.

4 Conclusion

We found that Long Short-Term Memory (LSTM) clearly outperforms previous RNNs not only on regular language benchmarks (according to previous research) but also on context free language (CFL) benchmarks. It learns faster and generalizes better. LSTM also is the first RNN to learn a context sensitive language.

Acknowledgment. This work was supported by SNF grant 2100-49'144.96 "Long Short-Term Memory."

References

[1] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proc. IJCNN'2000, Int. Joint*

¹Sun's training set was augmented stepwise by sequences misclassified during testing, and in the final accepted set n was in $\{1,..,20\}$ except for 20 random sequences up to length $n=160$ (the exact generalization performance was unclear).

²Applying brute force search to the weights of the best network of Rodriguez et al. (1999) further improves performance to acceptance up to $n=28$.

Conf. on Neural Networks, Como, Italy, 2000.

[2] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

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

[5] P. Rodriguez, J. Wiles, and J. Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999.

[6] Paul Rodriguez and Janet Wiles. Recurrent neural networks can learn to implement symbol-sensitive counting. In *Advances in Neural Information Processing Systems*, volume 10, pages 87–93. The MIT Press, 1998.

[7] G. Z. Sun, C. Lee Giles, H. H. Chen, and Y. C. Lee. The neural network pushdown automaton: Model, stack and learning simulations. Technical Report CS-TR-3118, University of Maryland, College Park, August 1993.

[8] B. Tonkes and J. Wiles. Learning a context-free task with a recurrent neural network: An analysis of stability. In *Proceedings of the Fourth Biennial Conference of the Australasian Cognitive Science Society*, 1997.

[9] J. Wiles and J. Elman. 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*, pages pages 482 – 487, Cambridge, MA, 1995". MIT Press.