Long Short-term Memory

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

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

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

(0) Abstract

RNN에서 긴 간격의 정보를 저장하려면 역전파 시간이 오래 걸리고, 기울기 소실 문제가 발생함.

LSTM 특별한 유닛을 통해 1000개 이상의 타임 스텝에서도 안정적으로 오류를 제어할 수 있음.

또한 기존의 RNN으로 해결할 수 없었던 시간 지연 문제를 해결하였음.

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

(1) Introduction

1 INTRODUCTION

Recurrent networks can in principle use their feedback connections to store representations of recent input events in form of activations ("short-term memory", as opposed to "long-term memory" embodied by slowly changing weights). This is potentially significant for many applications, including speech processing, non-Markovian control, and music composition (e.g., Mozer 1992). The most widely used algorithms for learning *what* to put in short-term memory, however, take too much time or do not work well at all, especially when minimal time lags between inputs and corresponding teacher signals are long. Although theoretically fascinating, existing methods do not provide clear *practical* advantages over, say, backprop in feedforward nets with limited time windows. This paper will review an analysis of the problem and suggest a remedy.

The problem. With conventional "Back-Propagation Through Time" (BPTT, e.g., Williams and Zipser 1992, Werbos 1988) or "Real-Time Recurrent Learning" (RTRL, e.g., Robinson and Fallside 1987), error signals "flowing backwards in time" tend to either (1) blow up or (2) vanish: the temporal evolution of the backpropagated error exponentially depends on the size of the weights (Hochreiter 1991). Case (1) may lead to oscillating weights, while in case (2) learning to bridge long time lags takes a prohibitive amount of time, or does not work at all (see section 3).

The remedy. This paper presents *"Long Short-Term Memory"* (LSTM), a novel recurrent network architecture in conjunction with an appropriate gradient-based learning algorithm. LSTM is designed to overcome these error back-flow problems. It can learn to bridge time intervals in excess of 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag capabilities. This is achieved by an efficient, gradient-based algorithm for an architecture

문제
오차 역전파에서 기울기가 사라짐.

해결책
적절한 그래디언트 기반 학습 알고리즘과 함께 새로운 순환 네트워크 아키텍처인 LSTM 제시

(1) Introduction

Outline of paper. Section 2 will briefly review previous work. Section 3 begins with an outline of the detailed analysis of vanishing errors due to Hochreiter (1991). It will then introduce a naive approach to constant error backprop for didactic purposes, and highlight its problems concerning information storage and retrieval. These problems will lead to the LSTM architecture as described in Section 4. Section 5 will present numerous experiments and comparisons with competing methods. LSTM outperforms them, and also learns to solve complex, artificial tasks no other recurrent net algorithm has solved. Section 6 will discuss LSTM's limitations and advantages. The appendix contains a detailed description of the algorithm (A.1), and explicit error flow formulae (A.2).

섹션 2 : 이전 연구 리뷰

섹션3 : 호흐라이터의 연구를 참고해 오차 소멸 문제 분석

섹션4 : LSTM 구조 소개

섹션5: 다양한 실험을 통한 기존 방법과의 비교

섹션6 : LSTM의 한계와 장점

(3) Constant error backprop

3.1 EXPONENTIALLY DECAYING ERROR

(proof by induction). The sum of the n^{q-1} terms $\prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}$ determines the total error back flow (note that since the summation terms may have different signs, increasing the number of units n does not necessarily increase error flow).

Intuitive explanation of equation (2). If

$$|f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}| > 1.0$$

for all m (as can happen, e.g., with linear f_{l_m}) then the largest product increases exponentially with q. That is, the error blows up, and conflicting error signals arriving at unit v can lead to oscillating weights and unstable learning (for error blow-ups or bifurcations see also Pineda 1988, Baldi and Pineda 1991, Doya 1992). On the other hand, if

$$|f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}| < 1.0$$

for all m, then the largest product *decreases* exponentially with q. That is, the error vanishes, and nothing can be learned in acceptable time.

If f_{l_m} is the logistic sigmoid function, then the maximal value of f'_{l_m} is 0.25. If $y^{l_{m-1}}$ is constant and not equal to zero, then $|f'_{l_m}(net_{l_m})w_{l_m l_{m-1}}|$ takes on maximal values where

$$w_{l_m l_{m-1}} = \frac{1}{y^{l_{m-1}}} \coth(\frac{1}{2} net_{l_m}),$$

계속 반복되면 점점 1보다 커져서 기울기 폭발하거나

1보다 작을 경우엔 기울기 손실

(3) Constant error backprop

3.2 CONSTANT ERROR FLOW : NAIVE APPROACH

3.2 CONSTANT ERROR FLOW: NAIVE APPROACH

A single unit. To avoid vanishing error signals, how can we achieve constant error flow through a single unit j with a single connection to itself? According to the rules above, at time t, j's local error back flow is $\vartheta_j(t) = f'_j(net_j(t))\vartheta_j(t+1)w_{jj}$. To enforce *constant* error flow through j, we require

$$f'_j(net_j(t))w_{jj} = 1.0.$$

Note the similarity to Mozer's fixed time constant system (1992) — a time constant of 1.0 is appropriate for potentially infinite time lags[1].

The constant error carrousel. Integrating the differential equation above, we obtain $f_j(net_j(t)) = \frac{net_j(t)}{w_{jj}}$ for arbitrary $net_j(t)$. This means: f_j has to be linear, and unit j's activation has to remain constant:

$$y_j(t+1) = f_j(net_j(t+1)) = f_j(w_{jj}y^j(t)) = y^j(t).$$

In the experiments, this will be ensured by using the identity function $f_j : f_j(x) = x, \forall x$, and by setting $w_{jj} = 1.0$. We refer to this as the constant error carrousel (CEC). CEC will be LSTM's central feature (see Section 4).

Of course unit j will not only be connected to itself but also to other units. This invokes two obvious, related problems (also inherent in all other gradient-based approaches):

CEC : LSTM 핵심

(4) Long short-term memory

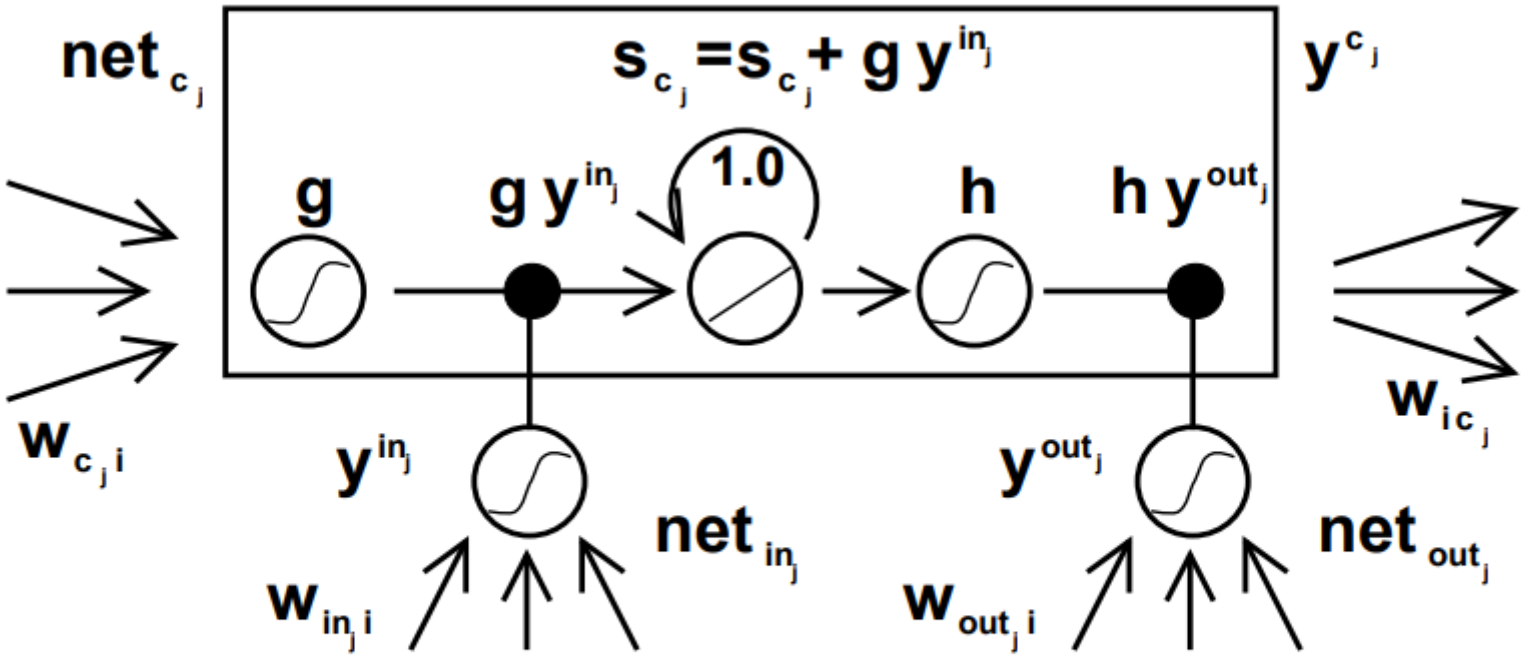
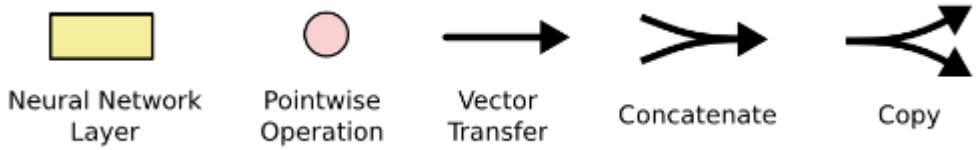
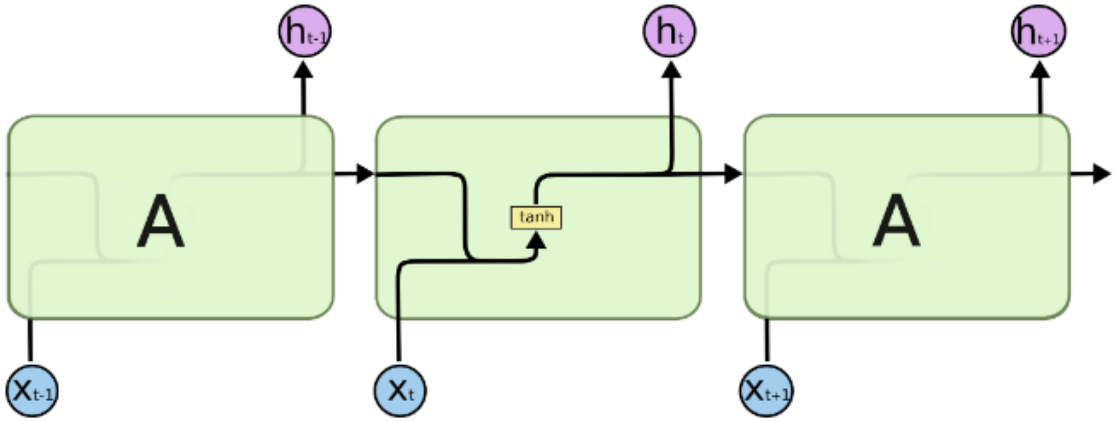


Figure 1: *Architecture of memory cell c_j (the box) and its gate units in_j, out_j. The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step. It builds the basis of the "constant error carrousel" CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.*

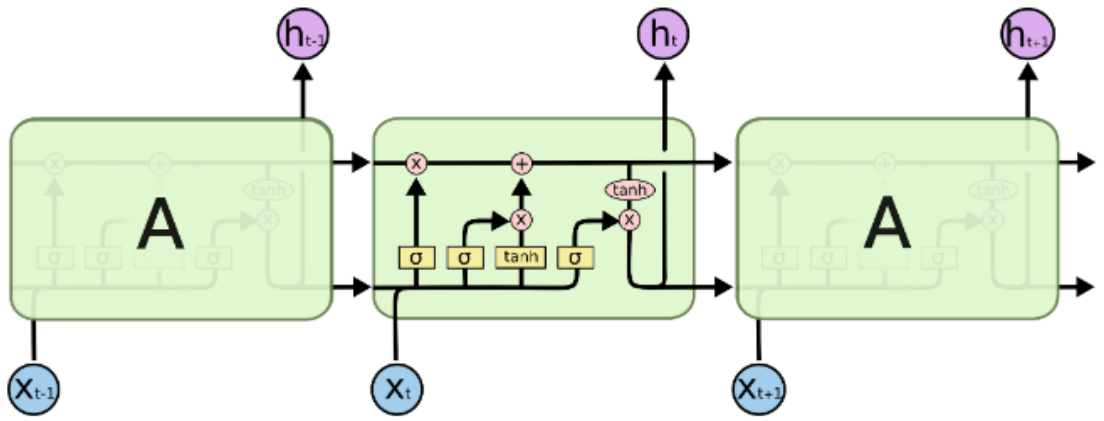
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

(4) Long short-term memory



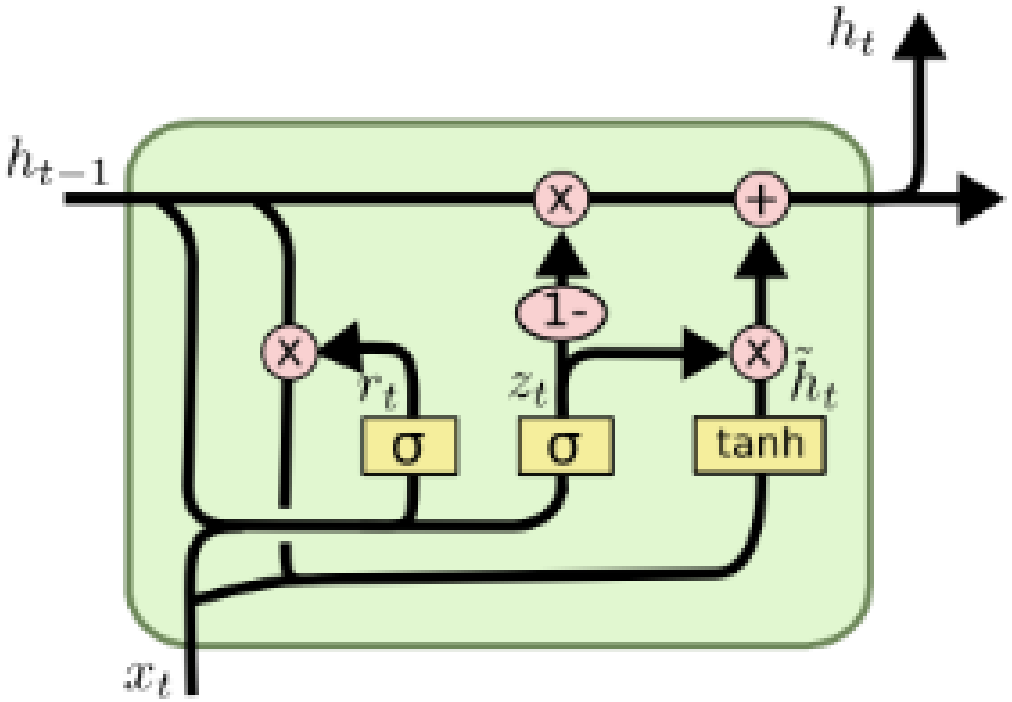


The repeating module in a standard RNN contains a single layer.



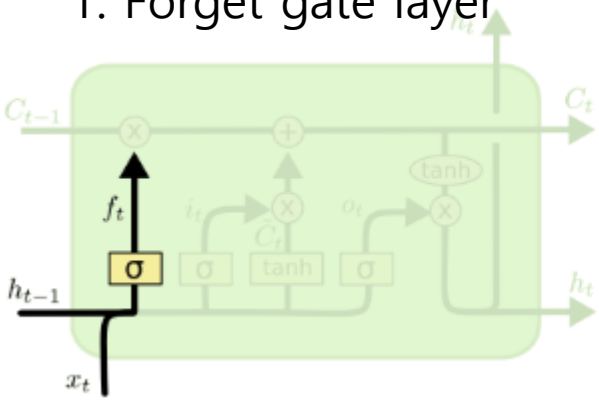
The repeating module in an LSTM contains four interacting layers.

(4) Long short-term memory



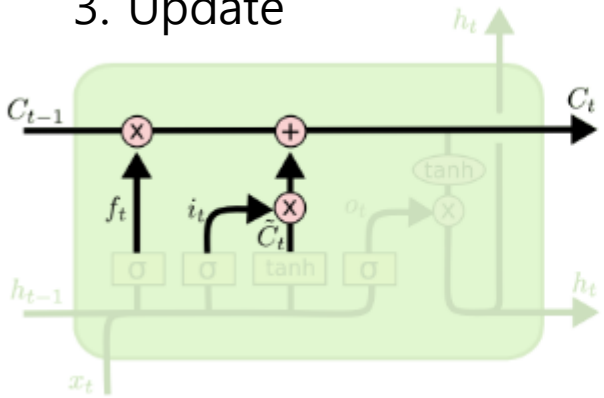
(4) Long short-term memory

1. Forget gate layer



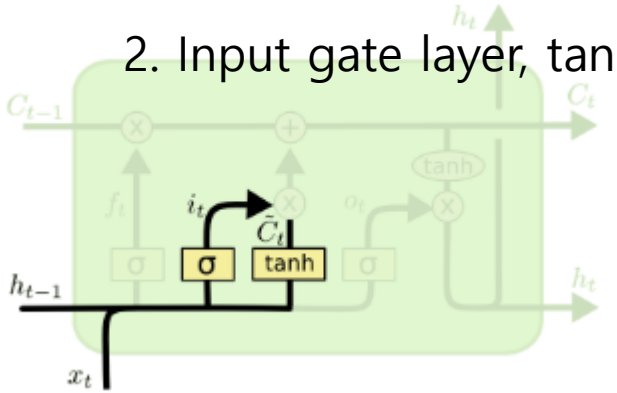
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

3. Update



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

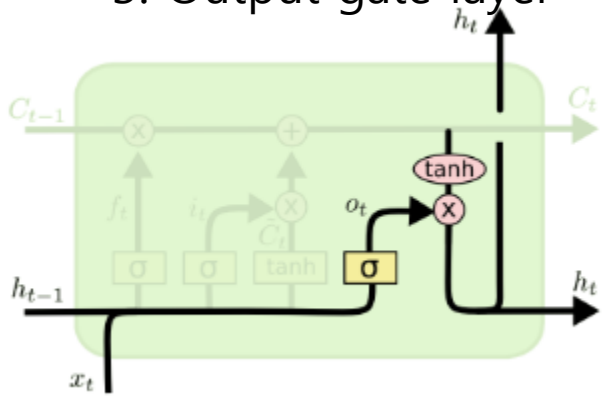
2. Input gate layer, tanh layer



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

3. Output gate layer



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

(5) Experiments

5.1 Experiment 1 : Embedded reber grammar

5.1 EXPERIMENT 1: EMBEDDED REBER GRAMMAR

Task. Our first task is to learn the "embedded Reber grammar", e.g. Smith and Zipser (1989), Cleeremans et al. (1989), and Fahlman (1991). Since it allows for training sequences with short time lags (of as few as 9 steps), it is *not* a long time lag problem. We include it for two reasons: (1) it is a popular recurrent net benchmark used by many authors — we wanted to have at least one experiment where RTRL and BPTT do not fail completely, and (2) it shows nicely how output gates can be beneficial.

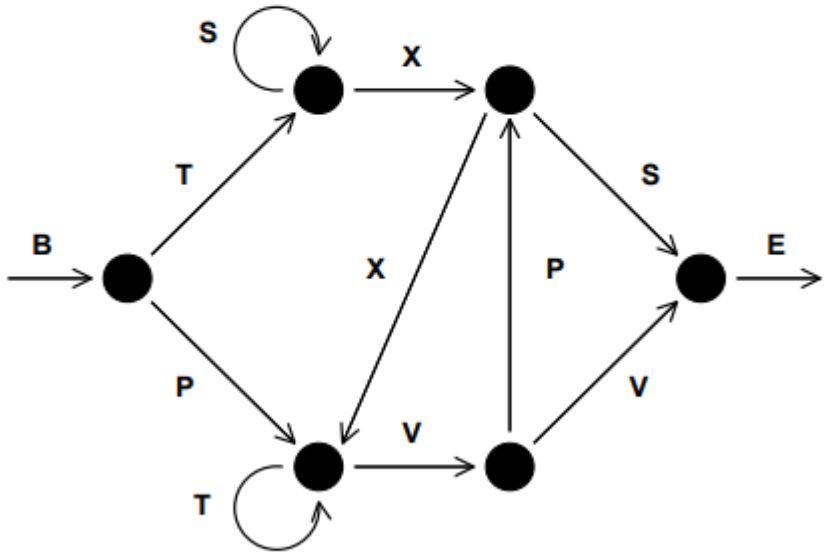


Figure 3: *Transition diagram for the Reber grammar.*

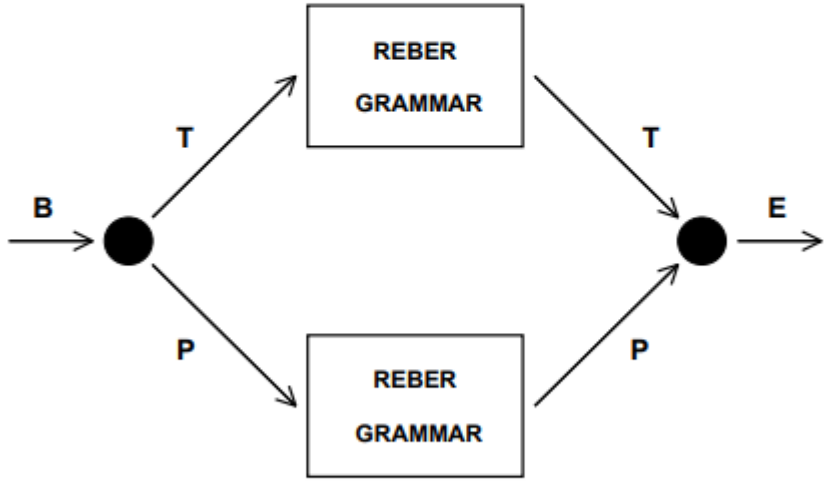


Figure 4: *Transition diagram for the embedded Reber grammar. Each box represents a copy of the Reber grammar (see Figure 3).*

(5) Experiments

5.1 Experiment 1 : Embedded reber grammar

5.1 EXPERIMENT 1: EMBEDDED REBER GRAMMAR

method	hidden units	# weights	learning rate	% of success	success after
RTRL	3	≈ 170	0.05	"some fraction"	173,000
RTRL	12	≈ 494	0.1	"some fraction"	25,000
ELM	15	≈ 435		0	>200,000
RCC	7-9	≈ 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

Table 1: *EXPERIMENT 1: Embedded Reber grammar: percentage of successful trials and number of sequence presentations until success for RTRL (results taken from Smith and Zipser 1989), "Elman net trained by Elman's procedure" (results taken from Cleeremans et al. 1989), "Recurrent Cascade-Correlation" (results taken from Fahlman 1991) and our new approach (LSTM). Weight numbers in the first 4 rows are estimates — the corresponding papers do not provide all the technical details. Only LSTM almost always learns to solve the task (only two failures out of 150 trials). Even when we ignore the unsuccessful trials of the other approaches, LSTM learns much faster (the number of required training examples in the bottom row varies between 3,800 and 24,100).*

다른 방법에 비해 빠르고 정확한 결과를 보여줌

(5) Experiments

5.2 Experiment 2 : Noise-free and noisy sequences

5.2 EXPERIMENT 2: NOISE-FREE AND NOISY SEQUENCES

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

Table 2: *Task 2a: Percentage of successful trials and number of training sequences until success, for "Real-Time Recurrent Learning" (RTRL), "Back-Propagation Through Time" (BPTT), neural sequence chunking (CH), and the new method (LSTM). Table entries refer to means of 18 trials. With 100 time step delays, only CH and LSTM achieve successful trials. Even when we ignore the unsuccessful trials of the other approaches, LSTM learns much faster.*

LSTM이 학습도 빠르고 정확한 결과를 보여줌

(5) Experiments

5.2 Experiment 2 : Noise-free and noisy sequences

5.2 EXPERIMENT 2: NOISE-FREE AND NOISY SEQUENCES

q (time lag -1)	p (# random inputs)	$\frac{q}{p}$	# weights	Success after
50	50	1	364	30,000
100	100	1	664	31,000
200	200	1	1264	33,000
500	500	1	3064	38,000
1,000	1,000	1	6064	49,000
1,000	500	2	3064	49,000
1,000	200	5	1264	75,000
1,000	100	10	664	135,000
1,000	50	20	364	203,000

Table 3: *Task 2c: LSTM with very long minimal time lags $q + 1$ and a lot of noise. p is the number of available distractor symbols ($p + 4$ is the number of input units). $\frac{q}{p}$ is the expected number of occurrences of a given distractor symbol in a sequence. The rightmost column lists the number of training sequences required by LSTM (BPTT, RTRL and the other competitors have no chance of solving this task). If we let the number of distractor symbols (and weights) increase in proportion to the time lag, learning time increases very slowly. The lower block illustrates the expected slow-down due to increased frequency of distractor symbols.*

LSTM으로만 실험

(5) Experiments

5.3 Experiment 3 : Noise and signal on same channel

T	N	stop: ST1	stop: ST2	# weights	ST2: fraction misclassified
100	3	27,380	39,850	102	0.000195
100	1	58,370	64,330	102	0.000117
1000	3	446,850	452,460	102	0.000078

Table 4: *Task 3a: Bengio et al.'s 2-sequence problem. T is minimal sequence length. N is the number of information-conveying elements at sequence begin. The column headed by ST1 (ST2) gives the number of sequence presentations required to achieve stopping criterion ST1 (ST2). The rightmost column lists the fraction of misclassified post-training sequences (with absolute error > 0.2) from a test set consisting of 2560 sequences (tested after ST2 was achieved). All values are means of 10 trials. We discovered, however, that this problem is so simple that random weight guessing solves it faster than LSTM and any other method for which there are published results.*

문제가 너무 간단해서 random weight guessing으로 더 빨리 해결됨.

(5) Experiments

5.3 Experiment 3 : Noise and signal on same channel

T	N	stop: ST1	stop: ST2	# weights	ST2: fraction misclassified
100	3	41,740	43,250	102	0.00828
100	1	74,950	78,430	102	0.01500
1000	1	481,060	485,080	102	0.01207

Table 5: *Task 3b: modified 2-sequence problem. Same as in Table 4, but now the information-conveying elements are also perturbed by noise.*

노이즈를 추가했지만, 노이즈에 의해 불안한 결과를 보임

T	N	stop	# weights	fraction misclassified	av. difference to mean
100	3	269,650	102	0.00558	0.014
100	1	565,640	102	0.00441	0.012

Table 6: *Task 3c: modified, more challenging 2-sequence problem. Same as in Table 4, but with noisy real-valued targets. The system has to learn the conditional expectations of the targets given the inputs. The rightmost column provides the average difference between network output and expected target. Unlike 3a and 3b, this task cannot be solved quickly by random weight guessing.*

Random weight guessing으로 해결되지 않지만 LSTM으로 해결됨.

(6) Discussion

Limitations of LSTM.

- The particularly efficient truncated backprop version of the LSTM algorithm will not easily solve problems similar to "strongly delayed XOR problems", where the goal is to compute the XOR of two widely separated inputs that previously occurred somewhere in a noisy sequence. The reason is that storing only one of the inputs will not help to reduce the expected error — the task is non-decomposable in the sense that it is impossible to incrementally reduce the error by first solving an easier subgoal.

  In theory, this limitation can be circumvented by using the full gradient (perhaps with additional conventional hidden units receiving input from the memory cells). But we do not recommend computing the full gradient for the following reasons: (1) It increases computational complexity. (2) Constant error flow through CECs can be shown only for truncated LSTM. (3) We actually did conduct a few experiments with non-truncated LSTM. There was no significant difference to truncated LSTM, exactly because outside the CECs error flow tends to vanish quickly. For the same reason full BPTT does not outperform truncated BPTT.

- Each memory cell block needs two additional units (input and output gate). In comparison to standard recurrent nets, however, this does not increase the number of weights by more than a factor of 9: each conventional hidden unit is replaced by at most 3 units in the LSTM architecture, increasing the number of weights by a factor of 3^2 in the fully connected case. Note, however, that our experiments use quite comparable weight numbers for the architectures of LSTM and competing approaches.

(6) Discussion

Advantages of LSTM.

- The constant error backpropagation within memory cells results in LSTM's ability to bridge very long time lags in case of problems similar to those discussed above.
- For long time lag problems such as those discussed in this paper, LSTM can handle noise, distributed representations, and continuous values. In contrast to finite state automata or hidden Markov models LSTM does not require an *a priori* choice of a finite number of states. In principle it can deal with unlimited state numbers.
- For problems discussed in this paper LSTM generalizes well — even if the positions of widely separated, relevant inputs in the input sequence do not matter. Unlike previous approaches, ours quickly learns to distinguish between two or more widely separated occurrences of a particular element in an input sequence, without depending on appropriate short time lag training exemplars.
- There appears to be no need for parameter fine tuning. LSTM works well over a broad range of parameters such as learning rate, input gate bias and output gate bias. For instance, to some readers the learning rates used in our experiments may seem large. However, a large learning rate pushes the output gates towards zero, thus automatically countermanding its own negative effects.
- The LSTM algorithm's update complexity per weight and time step is essentially that of BPTT, namely $O(1)$. This is excellent in comparison to other approaches such as RTRL. Unlike full BPTT, however, LSTM is *local in both space and time*.

(7) Conclusion

5.1 Experiment 3 : Noise and signal on same channel

7 CONCLUSION

Each memory cell's internal architecture guarantees constant error flow within its constant error carrousel CEC, provided that truncated backprop cuts off error flow trying to leak out of memory cells. This represents the basis for bridging very long time lags. Two gate units learn to open and close access to error flow within each memory cell's CEC. The multiplicative input gate affords protection of the CEC from perturbation by irrelevant inputs. Likewise, the multiplicative output gate protects other units from perturbation by currently irrelevant memory contents.

Future work. To find out about LSTM's practical limitations we intend to apply it to real world data. Application areas will include (1) time series prediction, (2) music composition, and (3) speech processing. It will also be interesting to augment sequence chunkers (Schmidhuber 1992b, 1993) by LSTM to combine the advantages of both.

8 ACKNOWLEDGMENTS

Thanks to Mike Mozer, Wilfried Brauer, Nic Schraudolph, and several anonymous referees for valuable comments and suggestions that helped to improve a previous version of this paper (Hochreiter and Schmidhuber 1995). This work was supported by *DFG grant SCHM 942/3-1* from "Deutsche Forschungsgemeinschaft".