Long Short-term Memory

Neural Computation 1997

Sepp Hochreiter, Jiirgen Schmidhuber

Fakultat fiir Informatik

IDSIA

Part 1.

Background

- **Recurrent networks: Short-term memory**
  - It's opposed to "long-term memory" embodied by slowly changing weights
    - Use their feedback connections to store representations of recent input events in form of activations
  - The most widely used algorithms for learning what to put in short-term memory
    - Applications: Speech processing, non-Markovian control, music composition
  - Take too much time or do not work well at all
    - Especially when minimal time lags between inputs & corresponding teacher signals are long
  - Do not provide clear practical advantages over backprop in feedforward nets
    - Feedforward nets has limited time windows
  - To address the issue
    - Review an analysis of the problem and suggest a remedy

Part 1.

Background

- **The problem**
  - Previous Works
    - Back-Propagation Through Time (BPTT, e.g., Williams and Zipser 1992, Werbos 1988)
    - 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
    - Backpropagated error exponentially depends on the size of the weights (Hochreiter 1991)
  - Case (1): Blow up
    - It may lead to oscillating weights
  - Case (2): Vanish
    - Learning to bridge long time lags takes a prohibitive amount of time, or does not work at all

Introduction

- **The remedy: "Long Short-Term Memory" (LSTM)**
  - A recurrent network architecture in conjunction with an appropriate gradient-based learning algorithm
  - Overcome these error back-flow problems
    - An architecture enforcing constant (thus neither exploding nor vanishing) error flow
  - Flow through internal states of special units
    - Provide the gradient computation is truncated at certain architecture-specific points
    - This does not affect long-term error flow though
  - It can learn to bridge time intervals in excess of 1000 steps
    - In case of noisy, incompressible input sequences
    - Without loss of short time lag capabilities

Part 2.

Introduction

- **Outline of paper**
  - Section 2
    - Briefly review previous work
  - Section 3
    - Vanishing errors due to Hochreiter (1991)
    - A naive approach to constant error backprop for didactic purposes
    - Highlight its problems concerning information storage and retrieval
  - Section 4
    - Remedy: LSTM architecture
  - Section 5
    - LSTM outperforms competing methods on numerous experiments
  - Section 6
    - Discuss LSTM's limitations and advantages
  - The appendix
    - A detailed description of the algorithm (A.l)
    - Explicit error flow formulae (A.2)

Previous Work

- **Previous Work**
  - Recurrent nets with time-varying inputs
  - It is opposed to nets with stationary inputs and fixpoint-based gradient calculations, e.g., Almeida 1987, Pineda 1987

Part 3.

Previous Work

- **Gradient-descent variants**
  - Previous works suffer from the same problems as BPTT (Back-Propagation Through Time) and RTRL (Real-Time Recurrent Learning)
- **Time-delays**
  - Time-Delay Neural Networks & Plates method seem practical for short time lags only
  - Plates method updates unit activations based on a weighted sum of old activations

Previous Work

- **Time constants**
  - To deal with long time lags:
  - Use time constants influencing changes of unit activations
  - The time constants need external fine tuning
  - Updates the activation of a recurrent unit by adding the old activation and the (scaled) current net input
  - The net input, however, tends to perturb the stored information, which makes long-term storage impractical

Previous Work

- **Ring's approach**
  - Whenever a unit in his network receives conflicting error signals, he adds a higher order unit influencing appropriate connections
  - Sometimes extremely fast
    - To bridge a time lag involving 100 steps may require the addition of 100
  - Ring’s net does not generalize to unseen lag durations units

Part 3.

Previous Work

- **Bengio et al.'s approaches**
  - Investigate methods such as simulated annealing, multi-grid random search, time-weighted pseudo-Newton optimization, and discrete error propagation
  - "latch" and "2-sequence" problems
    - Very similar to problem 3a with minimal time lag 100 (see Experiment 3)
  - An EM approach for propagating targets
    - With n so-called "state networks", at a given time, their system can be in one of only n different states
    - Solve continuous problems such as the "adding problem" (see Experiment 4)
    - Their system would require an unacceptable number of states

Experiment 3

Percentage of successful trials and number of training sequences until success

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

Experiment 4

Percentage of successful trials and number of training sequences until success

T	minimal lag	# weights	# wrong predictions	Success after
100	50	93	1 out of 2560	74,000
500	250	93	0 out of 2560	209,000
1000	500	93	1 out of 2560	853,000

Bengio, Y. and Frasconi, P. Credit assignment through time: Alternatives to backpropagation. NeurIPS. 1994.

Previous Work

- **Kalman filters (Puskorius and Feldkamp, 1994)**
  - Use Kalman filter techniques to improve recurrent net performance
  - Use “a derivative discount factor imposed to decay exponentially the effects of past dynamic derivatives”
  - There is no reason to believe
    - “Kalman Filter Trained Recurrent Networks” will be useful for very long minimal time lags

Previous Work

- **Second order nets**
  - It is not the first recurrent net method using MUs though
    - LSTM uses multiplicative units (MUs) to protect error flow from unwanted perturbations
    - e.g., Watrous and Kuhn (1992) use Mus in second order nets
  - Some differences to LSTM are:
    (1) Watrous and Kuhn's architecture does not enforce constant error flow
    - It is not designed to solve long time lag problems
  - (2) It has fully connected second-order sigma-pi units
    - While the LSTM architecture's MUs are used only to gate access to constant error flow
  - (3) Watrous and Kuhn's algorithm costs $O(W^2)$ operations per time step
    - Our LSTM only costs $O(W\ )$, where W is the number of weights

Previous Work

- **Simple weight guessing**
  - Avoid long time lag problems of gradient-based approaches
    - Simply randomly initialize all network weights
    - Until the resulting net happens to classify all training sequences correctly
  - Require either many free parameters (e.g., input weights) or high weight precision (e.g., for continuous-valued parameters)
    - Such that guessing becomes completely infeasible

Previous Work

- **Adaptive sequence chunkers**
  - Do have a capability to bridge arbitrary time lags
    - But only if there is local predictability across the subsequences causing the time lags
  - Rapidly solve certain grammar learning tasks involving minimal time lags in excess of 1000 steps
    - Use hierarchical recurrent nets
  - The performance of chunker systems
    - Deteriorates as the noise level increases
    - The input sequences become less compressible
  - LSTM does not suffer from this problem

Constant Error Backprop: Exponentially Decaying Error

- **Conventional BPTT (e.g. Williams and Zipser 1992)**
  - Using mean squared error, k's error signal at time t is denoted by $d_k(t)$

$$\vartheta_k(t) = f'_k(net_k(t))(d_k(t) - y^k(t))$$

  - The activation of a non-input unit i with differentiable activation function f_i

$$y^i(t) = f_i(net_i(t))$$

  - Unit i's current net input, and w_{ij} is the weight on the connection from unit j to I

$$net_i(t) = \sum_j w_{ij} y^j(t-1)$$

  - Some non-output unit j 's backpropagated error signal

$$\vartheta_j(t) = f'_j(net_j(t)) \sum_i w_{ij} \vartheta_i(t+1)$$

Williams, R. J. and Zipser, D. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Back-propagation: Theory, Architectures and Applications (1992)

Constant Error Backprop: Exponentially Decaying Error

- **Conventional BPTT Outline of Hochreiter's analysis**
  - The corresponding contribution to w's total weight update

$$\alpha \vartheta_j(t) y^l(t-1)$$

  - a The learning rate
  - I An arbitrary unit connected to unit j

Hochreiter, J. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fiir Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Miinchen. 1991

Constant Error Backprop

- **Outline of Hochreiter's analysis (1991, page 19-21)**
  - Suppose we have a fully connected net whose non-input unit indices range from 1 to n
  - Let us focus on local error flow from unit u to unit v
  - Later we will see that the analysis immediately extends to global error flow
  - The error occurring at an arbitrary unit u at time step t is propagated "back into time" for q time steps, to an arbitrary unit v

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

$$l_q = v \qquad l_0 = u \qquad \frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}$$

  - Sum of the n^{q-1}: $\prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}$ determine error back flow
    - Note that increasing the number of units n does not necessarily increase error flow, since the summation terms may have different signs

Hochreiter, J. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fiir Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Miinchen. 1991

Constant Error Backprop: Exponentially Decaying Error

- **Outline of Hochreiter's analysis (1991, page 19-21)**
  - 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
    - The error blows up
    - Conflicting error signals arriving at unit v can lead to oscillating weights and unstable learning (for error blow-ups or bifurcations)
  - If $|f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}| < 1.0$ for all m
    - Then the largest product increases exponentially with q
    - The error vanishes
    - Nothing can be learned in acceptable time

Hochreiter, J. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fiir Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Miinchen. 1991

Constant Error Backprop: Exponentially Decaying Error

- **Outline of Hochreiter's analysis (1991, page 19-21)**
  - 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

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

    - Go to zero for $|w_{l_m l_{m-1}}| \to \infty$ and is less than 1.0 for $|w_{l_m l_{m-1}}| < 4.0$
    - E.g., if the absolute maximal weight value w_{max} is smaller than 4.0
    - With conventional logistic sigmoid activation functions, the error flow tends to vanish as long as the weights have absolute values below 4.0, especially in the beginning of the training phase

Hochreiter, J. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fiir Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Miinchen. 1991

Constant Error Backprop: Exponentially Decaying Error

- **Outline of Hochreiter's analysis (1991, page 19-21)**
  - In general the use of larger initial weights will not help though — as seen above
  - For $|w_{l_m l_{m-1}}| \to \infty$
    the relevant derivative goes to zero "faster" than the absolute weight can grow
    - Some weights will have to change their signs by crossing zero
  - Likewise, increasing the learning rate does not help either
    - It will not change the ratio of long-range error flow and short-range error flow
    - BPTT is too sensitive to recent distractions. (A very similar, more recent analysis was presented by Bengio et al. 1994)

Hochreiter, J. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fiir Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Miinchen. 1991

Constant Error Backprop: Exponentially Decaying Error

- **Outline of Hochreiter's analysis (1991, page 19-21)**
  - Global error flow
    - The local error flow analysis above immediately shows that global error flow vanishes, too

$$\sum_{u:\ u \text{ output unit}} \frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)}$$

Hochreiter, J. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fiir Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Miinchen. 1991

Constant Error Backprop: Exponentially Decaying Error

- **Outline of Hochreiter's analysis (1991, page 19-21)**
  - Weak upper bound for scaling factor
    - Slightly extended vanishing error analysis takes n, the number of units, into account
    - For $q > 1$

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

$$\Rightarrow \quad (W_{u^T})^T \; F'(t-1) \prod_{m=2}^{q-1} (WF'(t-m)) \; W_v \; f'_v(net_v(t-q))$$

- The weight matrix W $\quad [W]_{ij} := w_{ij}$
- v's outgoing weight vector W_v $\quad [W_v]_i := [W]_{iv} = w_{iv}$
- u's incoming weight vector W_u $\quad [W_{u^T}]_i := [W]_{ui} = w_{ui}$
- For $m = 1, \ldots, q$, $F'(t-m)$, Diagonal matrix of first order derivatives $[F'(t-m)]_{ij} := 0$
- If $i \neq j$, $[F'(t-m)]_{ij} := f'_i(net_i(t-m))$

Hochreiter, J. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fiir Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Miinchen. 1991

Constant Error Backprop: Exponentially Decaying Error

- **Outline of Hochreiter's analysis (1991, page 19-21)**
  - Using a matrix norm $\| . \|_A$ compatible with vector norm $\| . \|_x$

$$f'_{max} := \max_{m=1,\ldots,q}\{\| F'(t-m) \|_A\}$$

  - For $\max_{i=1,\ldots,n}\{|x_i|\} \leq \| x \|_x$, we get $|x^T y| \leq n \| x \|_x \| y \|_x$

$$|f'_v(net_v(t-q))| \leq \| F'(t-q) \|_A \leq f'_{max}$$

  - Obtain the following inequality

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

  - This inequality results from

$$\| W_v \|_x = \| We_v \|_x \leq \| W \|_A \| e_v \|_x \leq \| W \|_A$$
$$\| W_{u^T} \|_x = \| e_u W \|_x \leq \| W \|_A \| e_u \|_x \leq \| W \|_A$$

Hochreiter, J. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fiir Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Miinchen. 1991

Constant Error Backprop: Exponentially Decaying Error

- **Outline of Hochreiter's analysis (1991, page 19-21)**
  - This inequality results from $\| W_v \|_x = \| We_v \|_x \leq \| W \|_A \| e_v \|_x \leq \| W \|_A$
    $\| W_{u^T} \|_x = \| e_u W \|_x \leq \| W \|_A \| e_u \|_x \leq \| W \|_A$
  - e_k The unit vector whose components are 0 except for the k-th component, which is 1
  - This is a weak, extreme case upper bound
    - It will be reached only if all $\| F'(t-m) \|_A$ take on maximal values
    - If the contributions of all paths across which error flows back from unit u to unit v have the same sign
  - Large $\| W \|_A$ typically result in small values of $\| F'(t-m) \|_A$
  - With norms $\| W \|_A := \max_r \sum_s |w_{rs}| \quad \| x \|_x := \max_r |x_r|$
  - $f'_{max} = 0.25$ for the logistic sigmoid, obtain $|w_{ij}| \leq w_{max} < \frac{4.0}{n} \quad \forall i,j$
  - $\| W \|_A \leq n w_{max} < 4.0$ will result in exponential decay - by setting $\tau := \left(\frac{n w_{max}}{4.0}\right) <$
  - Obtain $\left| \frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} \right| \leq n(\tau)^q$

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

$$\vartheta_j(t) = f'_j(net_j(t))\vartheta_j(t+1)w_{jj}$$

  - 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

Constant Error Flow: Naive Approach

- **The constant error carrousel**
  - Integrating the differential equation above, we obtain $f_j(net_j(t)) = \frac{net_j(t)}{w_{jj}}$
  - 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 $f_j : f_j(x) = x, \forall x$
  - by setting $w_{jj} = 1.0$
  - Refer to this as the constant error carrousel (CEC)
  - CEC will be LSTM's central feature

Constant Error Flow: Naive Approach

- **Input weight conflict**
  - For simplicity, let us focus on a single additional input weight Wji
  - Assumption
    - The total error can be reduced by switching on unit j in response to a certain input, and keeping it active for a long time (until it helps to compute a desired output)
  - Provided i is non zero
    - Since the same incoming weight has to be used for both storing certain inputs and ignoring others
    - Wji will often receive conflicting weight update signals during this time (recall that j is linear)
  - These signals will attempt to make Wji participate in
  - (1) Storing the input (by switching on j)
  - (2) Protecting the input (by preventing j from being switched off by irrelevant later inputs)
  - This conflict makes learning difficult, and calls for a more context-sensitive mechanism for controlling "write operations" through input weights

Constant Error Flow: Naive Approach

- **Output weight conflict**
  - Assume j is switched on and currently stores some previous input
  - For simplicity, let us focus on a single additional outgoing weight Wkj
  - The same Wkj has to be used for both retrieving j's content at certain times and preventing j from disturbing k at other times
  - As long as unit j is non-zero
    - Wkj will attract conflicting weight update signals generated during sequence processing
  - These signals will attempt to make Wkj participate in
  - (1) accessing the information stored in j and—— at different times —
  - (2) protecting unit k from being perturbed by j
    - For instance, with many tasks there are certain "short time lag errors" that can be reduced in early training stages
  - However, at later training stages, j may suddenly start to cause avoidable errors in situations that already seemed under control by attempting to participate in reducing more difficult "long time lag errors"
  - This conflict makes learning difficult, and calls for a more context-sensitive mechanism for controlling "read operations" through output weights

Constant Error Flow: Naive Approach

- **Input and output weight conflicts**
  - These conflicts are not specific for long time lags, but occur for short time lags as well
  - Their effects, however, become particularly pronounced in the long time lag case:
  - As the time lag increases
  - (1) stored information must be protected against perturbation for longer and longer periods, and — especially in advanced stages of learning
  - (2) more and more already correct outputs also require protection against perturbation
  - Due to the problems above the naive approach does not work well except in case of certain simple problems involving local input/output representations and non-repeating input patterns (see Hochreiter 1991 and Silva et al. 1996)
  - The next section shows how to do it right

Long Short-Term Memory: Memory cells and gate units

- **Memory cells and gate units**
  - To construct an architecture that allows for constant error flow through special, self-connected units without the disadvantages of the naive approach
  - Extend the constant error carrousel CEC embodied by the self-connected, linear unit *j* from Section 3.2 by introducing additional features
  - A multiplicative input gate unit
    - Protect the memory contents stored in *j* from perturbation by irrelevant inputs
    - Likewise, a multiplicative output gate unit is introduced which protects other units from perturbation by currently irrelevant memory contents stored in j
  - The resulting, more complex unit is called a memory cell

$$y^{out_j}(t) = f_{out_j}(net_{out_j}(t)); y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$$

Long Short-Term Memory: Memory cells and gate units

- **Memory cells and gate units**
  - The resulting, more complex unit is called a memory cell

$$y^{out_j}(t) = f_{out_j}(net_{out_j}(t)); y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$$

  - The j-th memory cell is denoted C_j
  - Each memory cell is built around a central linear unit with a fixed self-connection (CEC)

  - In addition to net_{cj}, C_j gets input from a multiplicative unit $outj$ (the "output gate"), and from another multiplicative unit inj (the "input gate")
  - in_j's activation at time t is denoted by $y^{in_j}(t)$, $outj$'s by $y^{out_j}(t)$

$$net_{out_j}(t) = \sum_u w_{out_j u} y^u(t-1) \quad net_{in_j}(t) = \sum_u w_{in_j u} y^u(t-1) \quad net_{c_j}(t) = \sum_u w_{c_j u} y^u(t-1)$$

Long Short-Term Memory: Memory cells and gate units

- **Memory cells and gate units**
  - The resulting, more complex unit is called a memory cell

$$y^{out_j}(t) = f_{out_j}(net_{out_j}(t)); y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$$

Input Gate

$$net_{in_j}(t) = \sum_u w_{in_j u} y^u(t-1)$$

Output Gate

$$net_{out_j}(t) = \sum_u w_{out_j u} y^u(t-1)$$

Memory Cells

$$net_{c_j}(t) = \sum_u w_{c_j u} y^u(t-1)$$

  - All these different types of units: input units, gate units, memory cells
    - Convey useful information about the current state of the net
  - For instance, an input gate (output gate) may use inputs from other memory cells to decide whether to store (access) certain information in its memory cell
  - There even may be recurrent self-connections like $w_{c_j c_j}$
  - It is up to the user to define the network topology

Long Short-Term Memory: Memory cells and gate units

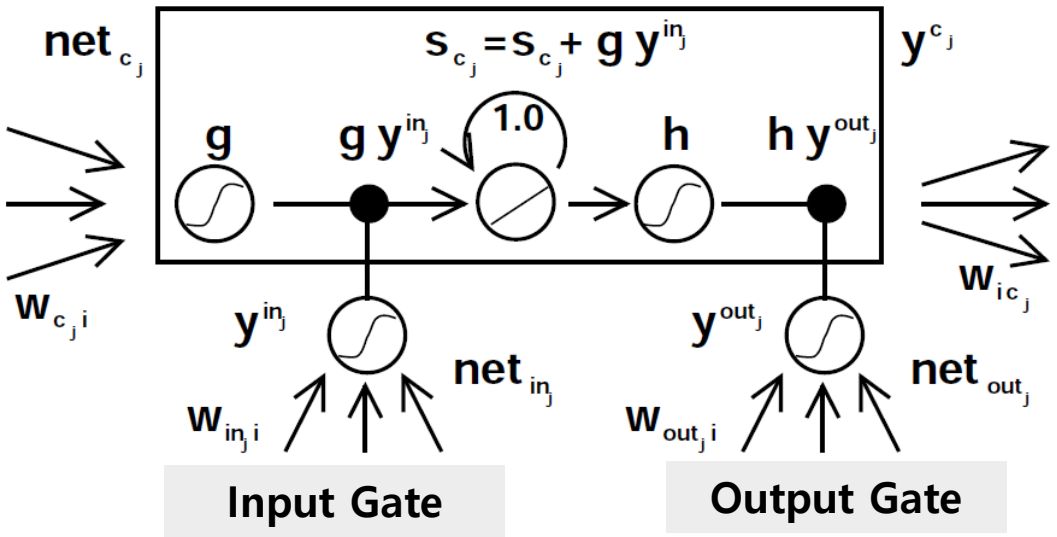
- **Memory cells and gate units**
  - At time t, C_j's output $y^{c_j}(t)$ is computed as $y^{c_j}(t) = y^{out_j}(t)h(s_{c_j}(t))$
  - Where the "internal state" $s_{c_j}(t)$

$$s_{c_j}(0) = 0, s_{c_j}(t) = s_{c_j}(t-1) + y^{in_j}(t)g\left(net_{c_j}(t)\right) \text{ for } t > 0$$

  - Differentiable function g squashes net_{c_j}
  - Differentiable function h scales memory cell outputs computed from the internal state

Memory Cells

$s_{c_j} = s_{c_j} + g\, y^{in_j}$



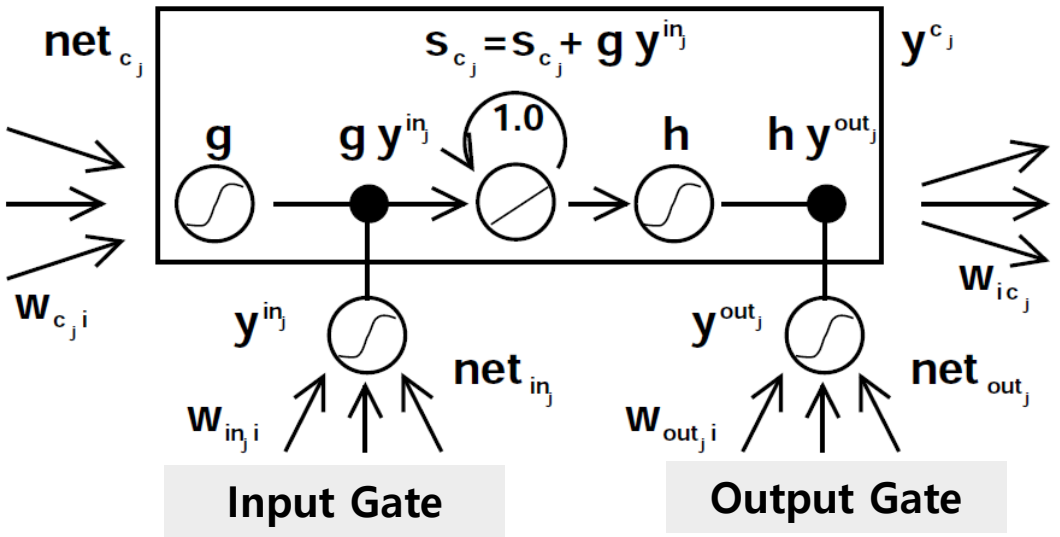
Architecture of memory cell

- The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step
- Build the basis of the "constant error carrousel"
- The gate units open and close access to CEC

Long Short-Term Memory: Memory cells and gate units

- **Why gate units?**
  - Input weight conflicts
    - in_j controls the error flow to memory cell C_j's input connections $w_{c_j i}$
  - C_j's output weight conflicts
    - out_j's controls error flow from unit j's output connections
  - The net can use in_j to decide when to keep or override information in memory cell C_j
  - The net can use out_j to decide when to access memory cell C_j and when to prevent other units from being perturbed by C_j

Memory Cells



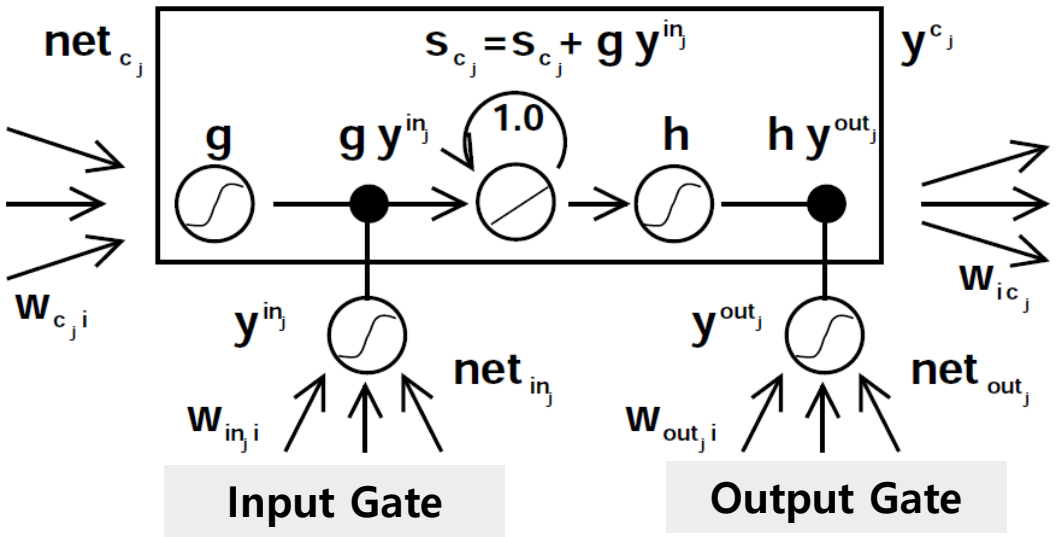
Architecture of memory cell

- The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step
- Build the basis of the "constant error carrousel"
- The gate units open and close access to CEC

Long Short-Term Memory: Memory cells and gate units

- **Why gate units?**
  - Error signals trapped within a memory cell's CEC cannot change
    - But different error signals flowing into the cell (at different times) via its output gate may get superimposed
  - The output gate: Learn which errors to trap in its CEC, by appropriately scaling them
  - The input gate: Learn when to release errors, again by appropriately scaling them
  - Essentially, the multiplicative gate units open and close access to constant error flow through CEC

Memory Cells



Architecture of memory cell

- The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step
- Build the basis of the "constant error carrousel"
- The gate units open and close access to CEC

Long Short-Term Memory: Memory cells and gate units

- **Why gate units?**
  - Distributed output representations typically do require output gates
  - Not always are both gate types necessary, though — one may be sufficient
  - It will be possible to use input gates only (Experiments 2a and 2b)
    - In fact, output gates are not required in case of local output encoding
    - Preventing memory cells from perturbing already learned outputs can be done by simply setting the corresponding weights to zero

Experiment 2a

Percentage of successful trials and number of training sequences until success

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

Experiment 2b

LSTM with very long minimal time lags $q + 1$ and a lot of no ise, p is the number of available distractor symbols

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

Long Short-Term Memory: Memory cells and gate units

- **Why gate units?**
  - Even in this case: output gates can be beneficial
  - They prevent the net's attempts at storing long time lag memories (which are usually hard to learn) from perturbing activations representing easily learnable short time lag memories

Experiment 1

Percentage of successful trials and number of training sequences until success

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

Part 6.

Long Short-Term Memory: Memory cells and gate units

- **Network topology**
  - Use networks with one input layer, one hidden layer, and one output layer
  - The self-connected hidden layer contains memory cells and corresponding gate units
    - For convenience, refer to both memory cells and gate units as being located in the hidden layer
    - Contain "conventional" hidden units providing inputs to gate units and memory cells
  - All units (except for gate units) in all layers have directed connections (serve as inputs) to all units in the layer above (or to all higher layers - Experiments 2a and 2b)

Experiment 2a

Percentage of successful trials and number of

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

Experiment 2b

LSTM with very long minimal time lags $q + 1$ and a lot of

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

Long Short-Term Memory: Memory cells and gate units

- **Memory cell blocks**
  - S memory cells sharing the same input gate and the same output gate form a structure called a "memory cell block of size S"
  - Memory cell blocks facilitate information storage — as with conventional neural nets, it is not so easy to code a distributed input within a single cell
  - Since each memory cell block has as many gate units as a single memory cell (namely two), the block architecture can be even slightly more efficient (see paragraph computational complexity)

Long Short-Term Memory: Memory cells and gate units

- **Learning**
  - Use a variant of "Real-Time Recurrent Learning (RTRL, e.g., Robinson and Fallside 1987)" which properly takes into account the altered, multiplicative dynamics caused by input and output gates
  - To ensure non-decaying error backprop through internal states of memory cells, as with truncated "Back-Propagation Through Time" (BPTT, e.g., Williams and Zipser 1992, Werbos 1988)
  - Errors arriving at "memory cell net inputs"
  - (for cell Cj, this includes net_{c_j} net_{in_j} net_{out_j}) do not get propagated back further in time (although they do serve to change the incoming weights)

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

Long Short-Term Memory: Memory cells and gate units

- **Computational Complexity**
  - As with Mozer's focused recurrent backprop algorithm (Mozer 1989), only the derivatives need to be stored and updated
  - Hence the LSTM algorithm is very efficient, with an excellent update complexity of where W the number of weights (see details in appendix A.I)
  - Hence, LSTM and BPTT for fully recurrent nets have the same update complexity per time step (while RTRL's is much worse)
  - Unlike full BPTT, however, LSTM is local in space and time
  - There is no need to store activation values observed during sequence processing in a stack with potentially unlimited size.

Long Short-Term Memory: Memory cells and gate units

- **Abuse problem and solutions**
  - In the beginning of the learning phase
    - Error reduction may be possible without storing information over time
  - Abuse memory cells (e.g., bias cells)
    - Make their activations constant and use the outgoing connections as adaptive thresholds for other units
  - The potential difficulty
    - Take a long time to release abused memory cells and make them available for further learning
  - Abuse problem
    - It appears if two memory cells store the same (redundant) information
  - Solution (1) Sequential network construction (e.g., Fahlman 1991)
    - A memory cell and the corresponding gate units are added to the network whenever the error stops decreasing (see Experiment 2 in Section 5)
  - Solution (2) Output gate bias (see Experiments 1, 3, 4, 5, 6)
    - Each output gate gets a negative initial bias, to push initial memory cell activations towards zero. Memory cells with more negative bias automatically get "allocated" later

Long Short-Term Memory: Memory cells and gate units

- **Internal state drift and remedies**
  - Its internal state s_j will tend to drift away over time
    - If memory cell c_j's inputs are mostly positive or mostly negative
    - Potentially dangerous: for the $h'(s_j)$ will then adopt very small values, and the gradient will vanish
  - Solution
    - Choose an appropriate function h
    - But $h(x) = x$ for instance, has the disadvantage of unrestricted memory cell output range
    - ➡ At the beginning of learning is to initially bias the input gate in_j towards zero
  - Although there is a tradeoff between the magnitudes of $h'(s_j)$ on the one hand of y^{in_j} & f'_{in_j} on the other

    The potential negative effect of input gate bias is negligible compared to the one of the drifting effect
  - With logistic sigmoid activation functions (see experiments 4 and 5)
    - There appears to be no need for fine-tuning the initial bias, as confirmed by

Experiments

- **Introduction**
  - Problem (1):
    Which tasks are appropriate to demonstrate the quality of a novel long time lag algorithm?

  - Requirement (1):
    Minimal time lags between relevant input signals and corresponding teacher signals must be long for all training sequences

Experiments

- **Introduction**
  - Problem (2):
    In fact, many previous recurrent net algorithms sometimes manage to generalize from very short training sequences to very long test sequences
    - A real long time lag problem does not have any short time lag exemplars in the training set
    - For instance, Elman's training procedure, BPTT, offline RTRL, online RTRL
    - etc., fail miserably on real long time lag problems. See, e.g., Hochreiter (1991) and Mozer (1992)
  - Requirement (2):
    The tasks should be complex enough such that they cannot be solved quickly by simple-minded strategies such as random weight guessing

Experiments

- **Guessing can outperform many long time lag algorithms**
  - Many long time lag tasks used in previous work
    - It can be solved more quickly by simple random weight guessing than by the proposed algorithms
    - For instance, guessing solved a variant of Bengio and Frasconi's "parity problem" (1994) problem much faster than the seven methods tested by Bengio et al (1994)
  - Similarly for some of Miller and Giles's problems (1993)
    - Of course, this does not mean that guessing is a good algorithm
    - It just means that some previously used problems are not extremely appropriate to demonstrate the quality of previously proposed algorithms

Experiments

- **What's common to Experiments 1-6.**
  - All our experiments (except for Experiment 1) involve long minimal time lags
  - There are no short time lag training exemplars facilitating learning
    - Solutions to most of our tasks are sparse in weight space
  - They require either many parameters/inputs or high weight precision, such that random weight guessing becomes infeasible

Experiments

- **Whafs common to Experiments 1-6.**
  - Always use on-line learning (as opposed to batch learning), and logistic sigmoids as activation functions.
    - For Experiments 1 and 2, initial weights are chosen in the range [-0.2,0.2], for the other experiments in [-0.1,0.1]
  - Training sequences are generated randomly according to the various task descriptions. In slight deviation from the notation in Appendix $A1$, each discrete time step of each input sequence involves three processing steps:
  - (1) Use current input to set the input units
  - (2) Compute activations of hidden units (including input gates, output gates, memory cells)
  - (3) Compute output unit activations
  - Except for Experiments 1, 2a, and 2b, sequence elements are randomly generated on-line, and error signals are generated only at sequence ends
  - Net activations are reset after each processed input sequence

Experiments

- **Whafs common to Experiments 1-6.**
  - For comparisons with recurrent nets taught by gradient descent
    - Give results only for "Real-Time Recurrent Learning (RTRL)", except for comparison $2a$, which also includes "Back-Propagation Through Time" (BPTT)
  - Note, however, that untruncated BPTT computes exactly the same gradient as offline RTRL
  - With long time lag problems, offline RTRL (or BPTT) and the online version of RTRL (no activation resets, online weight changes)
    - Lead to almost identical, negative results (as confirmed by additional simulations in Hochreiter 1991; see also Mozer 1992)
  - This is because offline RTRL, online RTRL, and full BPTT all suffer badly from exponential error decay

Experiments

- **Whafs common to Experiments 1-6.**
  - Our LSTM architectures are selected quite arbitrarily
  - If nothing is known about the complexity of a given problem, a more systematic approach would be: start with a very small net consisting of one memory cell
  - If this does not work, try two cells, etc. Alternatively, use sequential network construction (e.g., Fahlman 1991).

Experiments

- **Outline of Experiments 1**
  - Focus on a standard benchmark test for recurrent nets: the embedded Reber grammar
  - Since it allows for training sequences with short time lags, it is not a long time lag problem
  - We include it because (1) it provides a nice example where LSTM's output gates are truly beneficial, and (2) it is a popular benchmark for recurrent nets that has been used by many authors
  - Want to include at least one experiment where conventional BPTT and RTRL do not fail completely (LSTM, however, clearly outperforms them)
  - The embedded Reber grammar's minimal time lags represent a border case in the sense that it is still possible to learn to bridge them with conventional algorithms
  - Only slightly longer minimal time lags would make this almost impossible
  - The more interesting tasks in our paper, however, are those that RTRL, BPTT, etc. cannot solve at all

Experiments

- **Outline of Experiments 2**
  - Focus on noise-free and noisy sequences involving numerous input symbols distracting from the few important ones
  - The most difficult task (Task 2c) involves hundreds of distractor symbols at random positions, and minimal time lags of 1000 steps
  - LSTM solves it, while BPTT and RTRL already fail in case of 10-step minimal time lags (see also, e.g., Hochreiter 1991 and Mozer 1992)
  - For this reason RTRL and BPTT are omitted in the remaining, more complex experiments, all of which involve much longer time lags

Experiments

- **Outline of Experiments 3**
  - Address long time lag problems with noise and signal on the same input line
  - Experiments 3a/3b focus on Bengio et al.'s 1994 “2-sequence problem
  - Because this problem actually can be solved quickly by random weight guessing, we also include a far more difficult 2-sequence problem (3c) which requires to learn real-valued, conditional expectations of noisy targets, given the inputs

- **Outline of Experiments 4 & 5**
  - Involve distributed, continuous-valued input representations and require learning to store precise, real values for very long time periods
  - Relevant input signals can occur at quite different positions in input sequences
  - Again minimal time lags involve hundreds of steps
  - Similar tasks never have been solved by other recurrent net algorithms

Experiments

- **Outline of Experiments 6**
  - Involves tasks of a different complex type that also has not been solved by other recurrent net algorithms
  - Again, relevant input signals can occur at quite different positions in input sequences
  - The experiment shows that LSTM can extract information conveyed by the temporal order of widely separated inputs

Experiment 1: Embedded Reber Grammar

- **Task 1**
  - 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
  - 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
  - (2) it shows nicely how output gates can be beneficial

Figure 3: Transition diagram for the Reber grammar

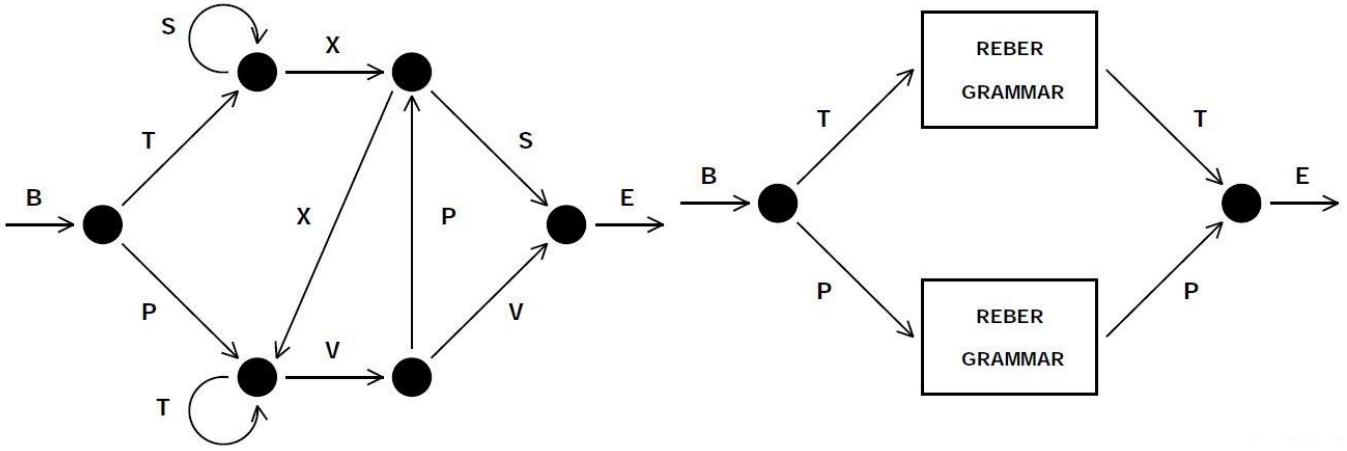


Figure 4: Transition diagram for the embedded Reber grammar. Each box represents a copy of the Reber grammar (see Figure 3)

Experiment 1: Embedded Reber Grammar

- **Task 1**
  - Starting at the leftmost node of the directed graph in Figure 4, symbol strings are generated sequentially (beginning with the empty string)
    - By following edges — and appending the associated symbols to the current string
    - Until the rightmost node is reached. Edges are chosen randomly if there is a choice (probability: 0.5)
  - The net's task is to read strings, one symbol at a time, and to permanently predict the next symbol (error signals occur at every time step)
  - To correctly predict the symbol before last, the net has to remember the second symbol

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

Experiment 1: Embedded Reber Grammar

- **Task 1: Training/Testing**
  - After string presentation, all activations are reinitialized with zeros
  - A trial is considered successful if all string symbols of all sequences in both test set and training set are predicted correctly
    - That is, if the output unit(s) corresponding to the possible next symbol(s) is (are) always the most active ones

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

Experiment 1: Embedded Reber Grammar

- **Task 1: Architectures**
  - (1) memory cells
    - The output layer's only incoming connections originate at memory cells
    - Each memory cell and each gate unit receives incoming connections from all memory cells and gate units (the hidden layer is fully connected — less connectivity may work as well)
    - The input layer has forward connections to all units in the hidden layer
    - The gate units are biased
    - These architecture parameters make it easy to store at least 3 input signals (architectures 3-2 and 4-1 are employed to obtain comparable numbers of weights for both architectures: 264 for 4-1 and 276 for 3-2)
    - Other parameters may be appropriate as well, however
    - All sigmoid functions are logistic with output range [0,1], except for h, whose range is [-1,1], and g, whose range is [-2, 2]
    - All weights are initialized in [-0.2,0.2], except for the output gate biases, which are initialized to -1, -2, and -3, respectively (see abuse problem, solution (2) of Section 4). We tried learning rates of 0.1, 0.2 and 0.5

Experiment 1: Embedded Reber Grammar

- **Task 1: Results**
  - Use 3 different, randomly generated pairs of training and test sets
  - With each such pair we run 10 trials with different initial weights
  - See Table 1 for results (mean of 30 trials)
  - Unlike the other methods, LSTM always learns to solve the task
  - Even when we ignore the unsuccessful trials of the other approaches, LSTM learns much faster

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

Experiment 1: Embedded Reber Grammar

- **Task 1: Importance of output gates**
  - The experiment provides a nice example where the output gate is truly beneficial
  - Learning to store the first T or P should not perturb activations representing the more easily learnable transitions of the original Reber grammar
  - This is the job of the output gates. Without output gates, we did not achieve fast learning

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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2a: noise-free sequences with long time lags**
  - There are p + 1 possible input symbols denoted ap_i,ap = x, ap+i = y. ai is "locally" represented by the p + 1-dimensional vector whose i-th component is 1 (all other components are 0). A net with p + 1 input units and p +1 output units sequentially observes input symbol sequences, one at a time, permanently trying to predict the next symbol — error signals occur at every single time step. To emphasize the "long time lag problem55, we use a training set consisting of only two very similar sequences: (있 , 이,(切 , • •. , ap_i,y) and (況, 이 , 必2, … ,ap-i,x). Each is selected with probability 0.5. To predict the final element, the net has to learn to store a representation of the first element for p time steps

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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2a: Architectures**
  - RTRL: one self-recurrent hidden unit, p-₩-1 non-recurrent output units. Each layer has connections from all layers below. All units use the logistic activation function sigmoid in [0,1]
  - BPTT: same architecture as the one trained by RTRL. CH: both net architectures like RTRL's, but one has an additional output for predicting the hidden unit of the other one (see Schmidhuber 1992b for details)
  - LSTM: like with RTRL, but the hidden unit is replaced by a memory cell and an input gate (no output gate required). 公 is the logistic sigmoid, and h is the identity function h : h{x) = x, Vx. Memory cell and input gate are added once the error has stopped decreasing (see abuse problem: solution (1) in Section 4).

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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2a: Results**
  - As expected, the chunker failed to solve this task (so did BPTT and RTRL, of course). LSTM, however, was always successful
  - On average (mean of 18 trials), success for p = 100 was achieved after 5,680 sequence presentations. This demonstrates that LSTM does not require sequence regularities to work well

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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2b: no local regularities**
  - With the task above, the chunker sometimes learns to correctly predict the final element, but only because of predictable local regularities in the input stream that allow for compressing the sequence. In an additional, more difficult task (involving many more different possible sequences), we remove compressibility by replacing the deterministic subsequence (ai,(12,. • • ,ap_i) by a random subsequence (of length p — 1) over the alphabet ai, fl25 • • • 5 ftp-i- We obtain 2 classes (two sets of sequences) {(있 , a打 , 匹2 , ... , a«p_1 ,y) ₩ 1 < ...,ip-i < P~ 1} and {(x, ail5ai25... ,aip_15x) | 1 < ■ ■ ■ ,^-1 < P~ 1}- Again, every next sequence element has to be predicted. The only totally predictable targets, however, are x and y, which occur at sequence ends. Training exemplars are chosen randomly from the 2 classes. Architectures and parameters are the same as in Experiment 2a. A successful run is one that fulfills the following criterion: after training, during 10,000 successive, randomly chosen input sequences, the maximal absolute error of all output units is below 0.25 at sequence end

Experiment 2: Noise-free And Noisy Sequences

- **Task 2c: very long time lags —— no local regularities**
  - This is the most difficult task in this subsection. To our knowledge no other recurrent net algorithm can solve it. Now there are 모나4 possible input symbols denoted ai, ...,ap_i,ap, ap+i = e,ap+2 = b, ap+s = x, ap+4 = y. ai,ap are also called adistractor symbols". Again, is locally represented by the p+4-dimensional vector whose zth component is 1 (all other components are 0). A net with p + 4 input units and 2 output units sequentially observes input symbol sequences, one at a time. Training sequences are randomly chosen from the union of two very similar subsets of sequences: {(&, y, , a«2,..., diq+k, e, y) | 1 < ...,iq+k < q} and {(&, ,aj2,.. .,aiq+k,e,x) | 1 < f 2 , ... , 나< q}- To produce a training sequence, we (1) randomly generate a sequence prefix of length q + 2, (2) randomly generate a sequence suffix of additional elementsb,e, x, y、) with probability 옮 or, alternatively, an 公 with probability 습. In the latter case, we (3) ıe second element. For a given k, sequences with length q + k + 4. d length is

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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2c: very long time lags - no local regularities**
  - The expected number of occurrences of element ㄸ나 , 1 <《 < p, in a sequence is 丄^으 « The goal is to predict the last symbol, which always occurs after the "trigger symbol" e. Error signals are generated only at sequence ends. To predict the final element, the net has to learn to store a representation of the second element for at least g + 1 time steps (until it sees the trigger symbol e). Success is defined as "prediction error (for final sequence element) of both output units always below 0.2, for 10,000 successive, randomly chosen input 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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2c: Architecture/Learning**
  - The net has p + 4 input units and 2 output units. Weights are initialized in [-0.2,0.2]. To avoid too much learning time variance due to different weight initializations, the hidden layer gets two memory cells (two cell blocks of size 1 — although one would be sufficient). There are no other hidden units. The output layer receives connections only from memory cells. Memory cells and gate units receive connections from input units, memory cells and gate units (i.e., the hidden layer is fully connected). No bias weights are used, h and g are logistic sigmoids with output ranges [—1,1] and [—2, 2], respectively. The learning rate is 0.01.

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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2c: Results**
  - 20 trials were made for all tested pairs (p, q). Table 3 lists the mean of the number of training sequences required by LSTM to achieve success (BPTT and RTRL have no chance of solving non-trivial tasks with minimal time lags of 1000 steps).

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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2c: Scaling**
  - Table 3 shows that if we let the number of input symbols (and weights) increase in proportion to the time lag, learning time increases very slowly. This is a another remarkable property of LSTM not shared by any other method we are aware of. Indeed, RTRL and BPTT are far from scaling reasonably — instead, they appear to scale exponentially, and appear quite useless when the time lags exceed as few as 10 steps.

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

Experiment 2: Noise-free And Noisy Sequences

- **Task 2c: Distractor influence**
  - In Table 3, the column headed by gives the expected frequency of distractor symbols. Increasing this frequency decreases learning speed, an effect due to weight oscillations caused by frequently observed input symbols

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3a**
  - This experiment serves to illustrate that LSTM does not encounter fundamental problems if noise and signal are mixed on the same input line. We initially focus on Bengio et al.'s simple 1994 "2-sequence problem55; in Experiment 3c we will then pose a more challenging 2-sequence problem
  - ("2-sequence problem^^). The task is to observe and then classify input sequences. There are two classes, each occurring with probability 0.5. There is only one input line. Only the first N real-valued sequence elements convey relevant information about the class. Sequence elements at positions t > N are generated by a Gaussian with mean zero and variance 0.2. Case N = 1: the first sequence element is 1.0 for class 1, and -1.0 for class 2. Case TV = 3: the first three elements are 1.0 for class 1 and -1.0 for class 2. The target at the sequence end is 1.0 for class 1 and 0.0 for class 2. Correct classification is defined as "absolute output error at sequence end below 0.2". Given a constant T, the sequence length is randomly selected between T and T + T/10 (a difference to Bengio et al.'s problem is that they also permit shorter sequences of length T/2).

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3a: Guessing**
  - Bengio et al. (1994) and Bengio and Eras coni (1994) tested 7 different methods on the 2-sequence problem. We discovered, however, that random weight guessing easily outperforms them all, because the problem is so simple5. See Schmidhuber and Hochreiter (1996) and Hochreiter and Schmidhuber (1996, 1997) for additional results in this vein
- **Task 3a: LSTM architecture.**
  - Use a 3-layer net with 1 input unit, 1 output unit, and 3 cell blocks of size 1. The output layer receives connections only from memory cells. Memory cells and gate units receive inputs from input units, memory cells and gate units, and have bias weights. Gate units and output unit are logistic sigmoid in [0,1], h in [—1,1], and g in [—2,2].

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3a: Training/Testing**
  - All weights (except the bias weights to gate units) are randomly initialized in the range [—0.1,0.1]. The first input gate bias is initialized with —1.0, the second with —3.0, and the third with —5.0. The first output gate bias is initialized with —2.0, the second with —4.0 and the third with —6.0. The precise initialization values hardly matter though, as confirmed by additional experiments. The learning rate is 1.0. All activations are reset to zero at the beginning of a new sequence
  - Stop training (and judge the task as being solved) according to the following criteria: STI: none of 256 sequences from a randomly chosen test set is misclassified. ST2: STI is satisfied, and mean absolute test set error is below 0.01. In case of ST2, an additional test set consisting of 2560 randomly chosen sequences is used to determine the fraction of misclassified sequences

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3a: Results**
  - All weights (except the bias weights to gate units) are randomly initialized in the range [—0.1,0.1]. The first input gate bias is initialized with —1.0, the second with —3.0, and the third with —5.0. The first output gate bias is initialized with —2.0, the second with —4.0 and the third with —6.0. The precise initialization values hardly matter though, as confirmed by additional experiments. The learning rate is 1.0. All activations are reset to zero at the beginning of a new sequence
  - Stop training (and judge the task as being solved) according to the following criteria: STI: none of 256 sequences from a randomly chosen test set is misclassified. ST2: STI is satisfied, and mean absolute test set error is below 0.01. In case of ST2, an additional test set consisting of 2560 randomly chosen sequences is used to determine the fraction of misclassified sequences

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3b**
  - Architecture, parameters, etc. like in Task 3a, but now with Gaussian noise (mean 0 and variance 0.2) added to the information-conveying elements (t <= TV). We stop training (and judge the task as being solved) according to the following, slightly redefined criteria: STI: less than 6 out of 256 sequences from a randomly chosen test set are misclassified. ST2: STI is satisfied, and mean absolute test set error is below 0.04. In case of ST2, an additional test set consisting of 2560 randomly chosen sequences is used to determine the fraction of misclassified sequences

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3b: Results**
  - The results represent means of 10 trials with different weight initializations. LSTM easily solves the problem.

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3c**
  - Architecture, parameters, etc. like in Task 3a, but with a few essential changes that make the task non-trivial: the targets are 0.2 and 0.8 for class 1 and class 2, respectively, and there is Gaussian noise on the targets (mean 0 and variance 0.1; st.dev. 0.32). To minimize mean squared error, the system has to learn the conditional expectations of the targets given the inputs. Misclassification is defined as "absolute difference between output and noise-free target (0.2 for class 1 and 0.8 for class 2) > 0.1. " The network output is considered acceptable if the mean absolute difference between noise-free target and output is below 0.015. Since this requires high weight precision, Task 3c (unlike 3a and 3b) cannot be solved quickly by random guessing

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3c: Training/Testing**
  - The learning rate is 0.1. We stop training according to the following criterion: none of 256 sequences from a randomly chosen test set is misclassified, and mean absolute difference between noise free target and output is below 0.015. An additional test set consisting of 2560 randomly chosen sequences is used to determine the fraction of misclassified sequences

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3c: Results**
  - The results represent means of 10 trials with different weight initializations. Despite the noisy targets, LSTM still can solve the problem by learning the expected target values.

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

Experiment 3: Noise And Signal on Same Channel

- **Task 3c: Results**
  - The results represent means of 10 trials with different weight initializations. Despite the noisy targets, LSTM still can solve the problem by learning the expected target values.

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

Experiment 4: Adding Problem

- **Task 4**
  - The difficult task in this section is of a type that has never been solved by other recurrent net algorithms. It shows that LSTM can solve long time lag problems involving distributed, continuousvalued representations
  - Each element of each input sequence is a pair of components. The first component is a real value randomly chosen from the interval [—1,1]; the second is either 1.0, 0.0, or -1.0, and is used as a marker: at the end of each sequence, the task is to output the sum of the first components of those pairs that are marked by second components equal to 1.0. Sequences have random lengths between the minimal sequence length T and 71 + 옳. In a given sequence exactly two pairs are marked as follows: we first randomly select and mark one of the first ten pairs (whose first component we call Xi). Then we randomly select and mark one of the first j — 1 still unmarked pairs (whose first component we call X2). The second components of all remaining pairs are zero

T	minimal lag	# weights	# wrong predictions	Success after
100	50	93	1 out of 2560	74,000
500	250	93	0 out of 2560	209,000
1000	500	93	1 out of 2560	853,000

ıts are -1. (In the rare case
.± to zero.) An error signal is
l:?2 (the sum Xr +X2 scaled
he absolute error at the
sequence end is below 0.04.

Experiment 4: Adding Problem

- **Task 4: Architecture**
  - Use a 3-layer net with 2 input units, 1 output unit, and 2 cell blocks of size 2. The output layer receives connections only from memory cells. Memory cells and gate units receive inputs from memory cells and gate units (i.e., the hidden layer is fully connected — less connectivity may work as well)
  - The input layer has forward connections to all units in the hidden layer. All non-input units have bias weights. These architecture parameters make it easy to store at least 2 input signals (a cell block size of 1 works well, too). All activation functions are logistic with output range [0,1], except for h, whose range is [—1,1], and g, whose range is [—2,2].

T	minimal lag	# weights	# wrong predictions	Success after
100	50	93	1 out of 2560	74,000
500	250	93	0 out of 2560	209,000
1000	500	93	1 out of 2560	853,000

Experiment 4: Adding Problem

- **Task 4: State drift versus initial bias**
  - Note that the task requires storing the precise values of real numbers for long durations — the system must learn to protect memory cell contents against even minor internal state drift (see Section 4). To study the significance of the drift problem, we make the task even more difficult by biasing all non-input units, thus artificially inducing internal state drift. All weights (including the bias weights) are randomly initialized in the range [—0.1,0.1]. Following Section 4's remedy for state drifts, the first input gate bias is initialized with —3.0, the second with —6.0 (though the precise values hardly matter, as confirmed by additional experiments).

T	minimal lag	# weights	# wrong predictions	Success after
100	50	93	1 out of 2560	74,000
500	250	93	0 out of 2560	209,000
1000	500	93	1 out of 2560	853,000

Experiment 4: Adding Problem

- **Task 4: Training/Testing**
  - The learning rate is 0.5. Training is stopped once the average training error is below 0.01, and the 2000 most recent sequences were processed correctly. Results. With a test set consisting of 2560 randomly chosen sequences, the average test set error was always below 0.01, and there were never more than 3 incorrectly processed sequences. Table 7 shows details
  - The experiment demonstrates: (1) LSTM is able to work well with distributed representations. (2) LSTM is able to learn to perform calculations involving continuous values. (3) Since the system manages to store continuous values without deterioration for minimal delays of 종 time steps, there is no significant, harmful internal state drift.

T	minimal lag	# weights	# wrong predictions	Success after
100	50	93	1 out of 2560	74,000
500	250	93	0 out of 2560	209,000
1000	500	93	1 out of 2560	853,000

Experiment 5: Multiplication Problem

- **Task 5**
  - One may argue that LSTM is a bit biased towards tasks such as the Adding Problem from the previous subsection. Solutions to the Adding Problem may exploit the CEC5s built-in integration capabilities. Although this CEC property may be viewed as a feature rather than a disadvantage (integration seems to be a natural subtask of many tasks occurring in the real world), the question arises whether LSTM can also solve tasks with inherently non-integrative solutions. To test this, we change the problem by requiring the final target to equal the product (instead of the sum) of earlier marked inputs

T	minimal lag	# weights	n_{seq}	# wrong predictions	MSE	Success after
100	50	93	140	139 out of 2560	0.0223	482,000
100	50	93	13	14 out of 2560	0.0139	1,273,000

Experiment 5: Multiplication Problem

- **Task 5: Architecture**
  - Like in Section 5.4. All weights (including the bias weights) are randomly initialized in the range [—0.1,0.1]

- **Task 5: Training/Testing**
  - The learning rate is 0.1. We test performance twice: as soon as less than nseq of the 2000 most recent training sequences lead to absolute errors exceeding 0.04, where nseq = 140, and nseq = 13. Why these values? nseq = 140 is sufficient to learn storage of the relevant inputs. It is not enough though to fine-tune the precise final outputs. nseq = 13, however, leads to quite satisfactory results.

T	minimal lag	# weights	n_{seq}	# wrong predictions	MSE	Success after
100	50	93	140	139 out of 2560	0.0223	482,000
100	50	93	13	14 out of 2560	0.0139	1,273,000

Experiment 5: Multiplication Problem

- **Task 5: Results**
  - For nseq = 140 (nseq = 13) with a test set consisting of 2560 randomly chosen sequences, the average test set error was always below 0.026 (0.013), and there were never more than 170 (15) incorrectly processed sequences. Table 8 shows details. (A net with additional standard hidden units or with a hidden layer above the memory cells may learn the fine-tuning part more quickly.) The experiment demonstrates: LSTM can solve tasks involving both continuous-valued representations and non-integrative information processing.

T	minimal lag	# weights	n_{seq}	# wrong predictions	MSE	Success after
100	50	93	140	139 out of 2560	0.0223	482,000
100	50	93	13	14 out of 2560	0.0139	1,273,000

Experiment 6: Temporal Order

- **Task 6**
  - In this subsection, LSTM solves other difficult (but artificial) tasks that have never been solved by previous recurrent net algorithms. The experiment shows that LSTM is able to extract information conveyed by the temporal order of widely separated inputs.

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

Experiment 6: Temporal Order

- **Task 6a: two relevant, widely separated symbols**
  - The goal is to classify sequences. Elements and targets are represented locally (input vectors with only one non-zero bit). The sequence starts with an E, ends with a B (the "trigger symbol55) and otherwise consists of randomly chosen symbols from the set {a, b, c, d} except for two elements at positions t± and that are either X or Y. The sequence length is randomly chosen between 100 and 110, t± is randomly chosen between 10 and 20, and 知 is randomly chosen between 50 and 60. There are 4 sequence classes Q,R, S, U which depend on the temporal order of X and Y. The rules are: X,X

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

Experiment 6: Temporal Order

- **Task 6b: three relevant, widely separated symbols**
  - Again, the goal is to classify sequences. Elements/targets are represented locally. The sequence starts with an E, ends with a B (the "trigger symbol55), and otherwise consists of randomly chosen symbols from the set {a, b, c, d} except for three elements at positions ii, and ts that are either X or Y. The sequence length is randomly chosen between 100 and 110, ti is randomly chosen between 10 and 20, fa is randomly chosen between 33 and 43, and ts is randomly chosen between 66 and 76. There are 8 sequence classes Q, R, S, U, V, A, B, C which depend on the temporal order of the Xs and Ys. The rules are: X, X, X Q; X, X,Y R- X,Y,X S; X,Y,Y U₩ Y,X,X-^
  - There are as many output units as there are classes. Each class is locally represented by a binary target vector with one non-zero component. With both tasks, error signals occur only at the end of a sequence. The sequence is classified correctly if the final absolute error of all output units is below 0.3

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

Experiment 6: Temporal Order

- **Task 6: Architecture**
  - We use a 3-layer net with 8 input units, 2 (3) cell blocks of size 2 and 4 (8) output units for Task 6a (6b). Again all non-input units have bias weights, and the output layer receives connections from memory cells only. Memory cells and gate units receive inputs from input units, memory cells and gate units (i.e., the hidden layer is fully connected — less connectivity may work as well). The architecture parameters for Task 6a (6b) make it easy to store at least 2 (3) input signals. All activation functions are logistic with output range [0,1], except for h, whose range is [—1,1], and g, whose range is [—2, 2].

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

Experiment 6: Temporal Order

- **Task 6: Training/Testing**
  - The learning rate is 0.5 (0.1) for Experiment 6a (6b). Training is stopped once the average training error falls below 0.1 and the 2000 most recent sequences were classified correctly. All weights are initialized in the range [—0.1,0.1]. The first input gate bias is initialized with —2.0, the second with —4.0, and (for Experiment 6b) the third with —6.0 (again, we confirmed by additional experiments that the precise values hardly matter).

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

Experiment 6: Temporal Order

- **Task 6: Results**
  - With a test set consisting of 2560 randomly chosen sequences, the average test set error was always below 0.1, and there were never more than 3 incorrectly classified sequences. Table 9 shows details
  - The experiment shows that LSTM is able to extract information conveyed by the temporal order of widely separated inputs. In Task 6a, for instance, the delays between first and second relevant input and between second relevant input and sequence end are at least 30 time steps.

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

Experiment 6: Temporal Order

- **Task 6: Typical solutions**
  - In Experiment 6a, how does LSTM distinguish between temporal orders (X, y) and (y, X)? One of many possible solutions is to store the first X or V in cell block 1, and the second X/Y in cell block 2. Before the first X/Y occurs, block 1 can see that it is still empty by means of its recurrent connections. After the first X/Y, block 1 can close its input gate. Once block 1 is filled and closed, this fact will become visible to block 2 (recall that all gate units and all memory cells receive connections from all non-output units). Typical solutions, however, require only one memory cell block. The block stores the first X or Y; once the second X/Y occurs, it changes its state depending on the first stored symbol. Solution type 1 exploits the connection between memory cell output and input gate unit — the following events cause different input gate activations: aX occurs in conjunction with a 五lied block^^; UX occurs in conjunction with an empty block". Solution type 2 is based on a strong positive connection between memory cell output and memory cell input. The previous occurrence of X (y) is represented by a positive (negative) internal state. Once the input gate opens for the second time, so does the output gate, and the memory cell output is fed back to its own input. This causes (X, Y) to be represented by a positive internal state, because X contributes to the new internal state twice (via current internal state and cell output feedback).

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

state.

Discussion

- **Limitations of LSTM**
  - The particularly efficient truncated backprop version of the LSTM algorithm will not easily solve problems similar to “strongly delayed XOR problems55, 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

Discussion

- **Limitations of LSTM**
  - 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 32 in the fully connected case. Note, however, that our experiments use quite comparable weight numbers for the architectures of LSTM and competing approaches
  - Generally speaking, due to its constant error flow through CECs within memory cells, LSTM runs into problems similar to those of feedforward nets seeing the entire input string at once. For instance, there are tasks that can be quickly solved by random weight guessing but not by the truncated LSTM algorithm with small weight initializations, such as the 500-step parity problem (see introduction to Section 5). Here, LSTM5s problems are similar to the ones of a feedforward net with 500 inputs, trying to solve 500-bit parity. Indeed LSTM typically behaves much like a feedforward net trained by backprop that sees the entire input. But that's also precisely why it so clearly outperforms previous approaches on many non-trivial tasks with significant search spaces.

Discussion

- **Limitations of LSTM**
  - LSTM does not have any problems with the notion of “recency” that go beyond those of other approaches. All gradient-based approaches, however, suffer from practical inability to precisely count discrete time steps. If it makes a difference whether a certain signal occurred 99 or 100 steps ago, then an additional counting mechanism seems necessary. Easier tasks, however, such as one that only requires to make a difference between, say, 3 and 11 steps, do not pose any problems to LSTM. For instance, by generating an appropriate negative connection between memory cell output and input, LSTM can give more weight to recent inputs and learn decays where necessary

Discussion

- **Advantages of LSTM**
  - The constant error backpropagation within memory cells results in LSTM5s 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 0(1). This is excellent in comparison to other approaches such as RTRL. Unlike full BPTT, however, LSTM is local in both space and time.

Conclusion

- **Task 6: Typical solutions**
  - Each memory celFs 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 celFs 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 LSTM5s 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.

Part 4.

Constant Error Backprop

$$h : h(x) = x, \forall x \quad (p = 4), \ \tfrac{7}{9} \ p = 10$$

$$\frac{\partial s_{c_j}}{\partial w_{il}} \quad [x]_i \quad [A]_{ij} \quad net_j(t) \quad a_1, ..., a_{p-1}, a_p = x, a_{p+1} = y \quad 100 \quad (a_1, a_2, \ldots, a_{p-1})$$

$$(y, a_1, a_2, \ldots, a_{p-1}, y)$$

$$a_1, a_2, \ldots, a_{p-1}$$

$$p+1 \quad net_{c_j} \ s_{c_j} \quad (x, a_1, a_2, \ldots, a_{p-1}, x) \quad y^{c_j}(t)$$

$$[-0.2, 0.2] \quad [-0.1, 0.1]$$

$$a_1, ..., a_{p-1}, a_p, a_{p+1} = e, a_{p+2} = b, a_{p+3} = x, a_{p+4} = y$$

$$\{(x, a_{i_1}, a_{i_2}, \ldots, a_{i_{p-1}}, x) \mid 1 \leq i_1, i_2, \ldots, i_{p-1} \leq p-1\}$$

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

Part 4.

Constant Error Backprop

$(\neq b, e, x, y)$ $q+2$ $q+k+4$

$a_1, ..., a_{p-1}, a_p, a_{p+1} = e, a_{p+2} = b, a_{p+3} = x, a_{p+4} = y$ $a_1, ..., a_p$

$4 + \sum_{k=0}^{\infty} \frac{1}{10}(\frac{9}{10})^k (q+k) = q+14$

$\{(b, y, a_{i_1}, a_{i_2}, \ldots, a_{i_{q+k}}, e, y) \mid 1 \leq i_1, i_2, \ldots, i_{q+k} \leq q\}$

$a_i, 1 \leq i \leq p$

T/10

$\{(b, x, a_{i_1}, a_{i_2}, \ldots, a_{i_{q+k}}, e, x) \mid 1 \leq i_1, i_2, \ldots, i_{q+k} \leq q\}$

$\frac{q+10}{p} \approx \frac{q}{p}$ T/2

$[-1, 1]$ $[-2, 2]$ $q+1$ $p+4$ $N=1$ $t > N$ $N=3$

T T +

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

Constant Error Backprop

$T + \frac{T}{10}$ $\frac{T}{2} - 1$ $X_1 + X_2$ $0.5 + \frac{X_1 + X_2}{4.0}$

T	minimal lag	# weights	# wrong predictions	Success after
100	50	93	1 out of 2560	74,000
500	250	93	0 out of 2560	209,000
1000	500	93	1 out of 2560	853,000

$(error > 0.04)$ $T = 1000$ $n_{seq} = 140$ $n_{seq} = 13$

$X_1 \times X_2$

$X, X \rightarrow Q;$ $X, Y \rightarrow R;$ $Y, X \rightarrow S;$ $Y, Y \rightarrow U$

Q, R, S, U Q, R, S, U, V, A, B, C

$X, X, X \rightarrow Q;$ $X, X, Y \rightarrow R;$ $X, Y, X \rightarrow S;$ $X, Y, Y \rightarrow U;$ $Y, X, X \rightarrow V;$ $Y, X, Y \rightarrow A;$ $Y, Y, X \rightarrow B;$ $Y, Y, Y \rightarrow C$

Part 4.

Constant Error Backprop

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Task	p	lag	b	s	in	out	w	c	ogb	igb	bias	h	g	α
1-1	9	9	4	1	7	7	264	F	-1,-2,-3,-4	r	ga	h1	g2	0.1
1-2	9	9	3	2	7	7	276	F	-1,-2,-3	r	ga	h1	g2	0.1
to be continued on next page														

continued from previous page

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Task	p	lag	b	s	in	out	w	c	ogb	igb	bias	h	g	α
1-3	9	9	3	2	7	7	276	F	-1,-2,-3	r	ga	h1	g2	0.2
1-4	9	9	4	1	7	7	264	F	-1,-2,-3,-4	r	ga	h1	g2	0.5
1-5	9	9	3	2	7	7	276	F	-1,-2,-3	r	ga	h1	g2	0.5
2a	100	100	1	1	101	101	10504	B	no og	none	none	id	g1	1.0
2b	100	100	1	1	101	101	10504	B	no og	none	none	id	g1	1.0
2c-1	50	50	2	1	54	2	364	F	none	none	none	h1	g2	0.01
2c-2	100	100	2	1	104	2	664	F	none	none	none	h1	g2	0.01
2c-3	200	200	2	1	204	2	1264	F	none	none	none	h1	g2	0.01
2c-4	500	500	2	1	504	2	3064	F	none	none	none	h1	g2	0.01
2c-5	1000	1000	2	1	1004	2	6064	F	none	none	none	h1	g2	0.01
2c-6	1000	1000	2	1	504	2	3064	F	none	none	none	h1	g2	0.01
2c-7	1000	1000	2	1	204	2	1264	F	none	none	none	h1	g2	0.01
2c-8	1000	1000	2	1	104	2	664	F	none	none	none	h1	g2	0.01
2c-9	1000	1000	2	1	54	2	364	F	none	none	none	h1	g2	0.01
3a	100	100	3	1	1	1	102	F	-2,-4,-6	-1,-3,-5	b1	h1	g2	1.0
3b	100	100	3	1	1	1	102	F	-2,-4,-6	-1,-3,-5	b1	h1	g2	1.0
3c	100	100	3	1	1	1	102	F	-2,-4,-6	-1,-3,-5	b1	h1	g2	0.1
4-1	100	50	2	2	2	1	93	F	r	-3,-6	all	h1	g2	0.5
4-2	500	250	2	2	2	1	93	F	r	-3,-6	all	h1	g2	0.5
4-3	1000	500	2	2	2	1	93	F	r	-3,-6	all	h1	g2	0.5
5	100	50	2	2	2	1	93	F	r	r	all	h1	g2	0.1
6a	100	40	2	2	8	4	156	F	r	-2,-4	all	h1	g2	0.5
6b	100	24	3	2	8	8	308	F	r	-2,-4,-6	all	h1	g2	0.1

1	2	3	4	5	6
Task	select	interval	test set size	stopping criterion	success
1	t1	$[-0.2, 0.2]$	256	training & test correctly pred.	see text
2a	t1	$[-0.2, 0.2]$	no test set	after 5 million exemplars	ABS(0.25)
2b	t2	$[-0.2, 0.2]$	10000	after 5 million exemplars	ABS(0.25)
2c	t2	$[-0.2, 0.2]$	10000	after 5 million exemplars	ABS(0.2)
3a	t3	$[-0.1, 0.1]$	2560	ST1 and ST2 (see text)	ABS(0.2)
3b	t3	$[-0.1, 0.1]$	2560	ST1 and ST2 (see text)	ABS(0.2)
3c	t3	$[-0.1, 0.1]$	2560	ST1 and ST2 (see text)	see text
4	t3	$[-0.1, 0.1]$	2560	ST3(0.01)	ABS(0.04)
5	t3	$[-0.1, 0.1]$	2560	see text	ABS(0.04)
6a	t3	$[-0.1, 0.1]$	2560	ST3(0.1)	ABS(0.3)
6b	t3	$[-0.1, 0.1]$	2560	ST3(0.1)	ABS(0.3)

Constant Error Backprop

- **Forward Pass**

$$f(x) = \frac{1}{1+\exp(-x)} \quad h(x) = \frac{2}{1+\exp(-x)} - 1 \quad net_i(t) = \sum_u w_{iu} y^u(t-1)$$

$$y^i(t) = f_i(net_i(t)) \quad net_{in_j}(t) = \sum_u w_{in_j u} y^u(t-1) \quad net_{out_j}(t) = \sum_u w_{out_j u} y^u(t-1)$$

$$y^{out_j}(t) = f_{out_j}(net_{out_j}(t)) \qquad net_{c_j^v} \quad s_{c_j^v} \quad y^{c_j^v}$$

$$net_{c_j^v}(t) = \sum_u w_{c_j^v u} y^u(t-1)$$

$$s_{c_j^v}(t) = s_{c_j^v}(t-1) + y^{in_j}(t) g\left(net_{c_j^v}(t)\right)$$

$$y^{c_j^v}(t) = y^{out_j}(t) h(s_{c_j^v}(t)) \qquad \text{“}\approx_{tr}\text{”}$$

$$net_k(t) = \sum_{u:\ u \text{ not a gate}} w_{ku} y^u(t-1) \qquad y^k(t) = f_k(net_k(t))$$