LONG SHORT-TERM MEMORY


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

Dougal Sutherland, 9/25/13
Problem with regular RNNs

• The standard learning algorithms for RNNs don’t allow for long time lags

• Problem: error signals going “back in time” in BPTT, RTRL, etc either exponentially blow up or (usually) exponentially vanish

• The toy problems solved in other papers are often solved faster by randomly guessing weights
Exponential decay of error signals

Error from unit $u$, at step $t$, to unit $v$, $q$ steps prior, is scaled by

$$\frac{\partial \theta_v(t - q)}{\partial \theta_u(t)} = \begin{cases} f_v'(net_v(t - 1))w_{uv} & q = 1 \\ f_v'(net_v(t - q)) \sum_{l=1}^{n} \frac{\partial \theta_l(t-q+1)}{\partial \theta_u(t)}w_{lv} & q > 1 \end{cases}$$

Summing over a path with $l_q=v$, $l_0=u$:

$$\frac{\partial \theta_v(t - q)}{\partial \theta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f_{l_m}'(net_{l_m}(t - m))w_{l_m l_{m-1}}$$

$$|f_{l_m}'(net_{l_m}(t - m))w_{l_m l_{m-1}}|$$

if always $> 1$: exponential blow-up in $q$
if always $< 1$: exponential decay in $q$
Exponential decay of error signals

\[ |f'_{l_m}(\text{net}_{l_m}(t - m))w_{l_m l_{m-1}}| \]

• For logistic activation, \( f'(\cdot) \) has maximum \( \frac{1}{4} \).
• For constant activation, term is maximized with
  \[ w_{l_m l_{m-1}} = \frac{1}{y_{l_{m-1}}} \coth\left(\frac{1}{2} \text{net}_{l_m}\right) \]
• If weights have magnitude \(< 4\), we get exponential decay.
• If weights are big, the derivative gets bigger faster and error still vanishes.
• Larger learning rate doesn’t help either.
Global error

• If we sum the exponential decay across all units, we see will still get vanishing global error

• Can also derive a (very) loose upper bound:
  – If max weight < 4/n, get exponential decay.
  – In practice, should happen much more often.
Constant error flow

• How can we avoid exponential decay?
• Need $f'_j(net_j(t))w_{jj} = 1.0$

• So we need a linear activation function, with constant activation over time
  – Here, use the identity function with unit weight.

• Call it the Constant Error Carrousel (CEC)
CEC issues

• Input weight conflict
  – Say we have a single outside input $i$
  – If it helps to turn on unit $j$ and keep it active, $w_{ji}$ will want to both:
    • Store the input (switching on $j$)
    • Protect the input (prevent $j$ from being switched off)
  – Conflict makes learning hard
  – Solution: add an “input gate” to protect the CEC from irrelevant inputs
CEC issues

• Output weight conflict
  – Say we have a single outside output $k$
  – $w_{kj}$ needs to both:
    • Sometimes get the output from the CEC $j$
    • Sometimes prevent $j$ from messing with $k$
  – Conflict makes learning hard
  – Solution: add an “output gate” to control when the stored data is read
LSTM memory cell

Figure 1: Architecture of memory cell $c_j$ (the box) and its gate units $i_n, o_n$. 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.
LSTM

• Later on, added a “forget gate” to cells
• Can connect cells into a memory block
  – Same input/output gates for multiple memory cells
• Here, used one fully-connected hidden layer
  – Consisting of memory blocks only
  – Could use regular hidden units also
• Learn with a variant of RTRL
  – Only propagates errors back in time in memory cells
  – $O(\# \text{ weights})$ update cost
LSTM topology example
Issues

• Sometimes memory cells get abused to work as constant biases
  – One solution: sequential construction. (Train until error stops decreasing; then add a memory cell.)
  – Another: output gate bias. (Initially suppress the outputs to make the abuse state farther away.)
Issues

• Internal state drift
  – If inputs are mostly the same sign, memory contents will tend to drift over time
  – Causes gradient to vanish
  – Simple solution: initial bias on input gates towards zero
**Experiment 1**

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

<table>
<thead>
<tr>
<th>method</th>
<th>hidden units</th>
<th># weights</th>
<th>learning rate</th>
<th>% of success</th>
<th>success after</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTRL</td>
<td>3</td>
<td>≈ 170</td>
<td>0.05</td>
<td>“some fraction”</td>
<td>173,000</td>
</tr>
<tr>
<td>RTRL</td>
<td>12</td>
<td>≈ 494</td>
<td>0.1</td>
<td>“some fraction”</td>
<td>25,000</td>
</tr>
<tr>
<td>ELM</td>
<td>15</td>
<td>≈ 435</td>
<td>0.1</td>
<td>0</td>
<td>&gt;200,000</td>
</tr>
<tr>
<td>RCC</td>
<td>7-9</td>
<td>≈ 119-198</td>
<td>50</td>
<td>100</td>
<td>182,000</td>
</tr>
<tr>
<td>LSTM</td>
<td>4 blocks, size 1</td>
<td>264</td>
<td>0.1</td>
<td>100</td>
<td>39,740</td>
</tr>
<tr>
<td>LSTM</td>
<td>3 blocks, size 2</td>
<td>276</td>
<td>0.1</td>
<td>100</td>
<td>21,730</td>
</tr>
<tr>
<td>LSTM</td>
<td>3 blocks, size 2</td>
<td>276</td>
<td>0.2</td>
<td>97</td>
<td>14,060</td>
</tr>
<tr>
<td>LSTM</td>
<td>4 blocks, size 1</td>
<td>264</td>
<td>0.5</td>
<td>97</td>
<td>9,500</td>
</tr>
<tr>
<td>LSTM</td>
<td>3 blocks, size 2</td>
<td>276</td>
<td>0.5</td>
<td>100</td>
<td>8,440</td>
</tr>
</tbody>
</table>

Figure 4: Transition diagram for the embedded Reber grammar.
Experiment 2a

- One-hot coding of $p+1$ symbols
- Task: predict the next symbol
- Train: $(p+1, 1, 2, ..., p-1, p+1), (p, 1, 2, ..., p-1, p)$
  - 5 million examples, equal probability
- Test on same set
- RTRL usually works with $p=4$, never with $p=10$
- Even with $p=100$, LSTM always works
- Hierarchical chunker also works
Experiment 2b

• Replace \((1, 2, \ldots, p-1)\) with random sequence
  – \(\{x, *, *, \ldots, *, x\}\) vs \(\{y, *, *, \ldots, *, y\}\)

• LSTM still works, chunker breaks

Experiment 2c

• ???????????
Experiment 3a

• So far, noise has been on separate channel from signal

• Now mix it:
  – \{1, N(0, .2), N(0, .2), ..., 1\} vs \{-1, N(0, .2), ..., 0\}
  – \{1, 1, 1, N(0, .2), ..., 1\} vs \{-1, -1, -1, N(0, .2), ..., 0\}

• LSTM works okay, but random guessing is better
Experiment 3b

• Same as before, but add Gaussian noise to the initial informative elements
• LSTM still works (as does random guessing)

Experiment 3c

• Replace informative elements with .2, .8
• Add Gaussian noise to targets
• LSTM works, random guessing doesn’t
Experiment 4

• Each sequence element is a pair of inputs: a number [-1, 1] and one of {-1, 0, 1}
  – Mark two elements with a 1 second entry
  – First, last elements get a -1 second entry
  – Rest are 0
  – Last element of sequence is sum of the two marked elements (scaled to [0, 1])
• LSTM always learns it with 2 memory cells
Experiment 5

• Same as experiment 4, but multiplies instead of adds
• LSTM always learns it with 2 memory cells
Experiment 6

• 6a: one-hot encoding of sequence classes
  – E, noise, X1, noise, X2, noise, B, output
    • X1, X2 each have two possible values
  – Noise is random length; separation ~40
  – Output is different symbol for each X1, X2 pair

• 6b: same thing but X1, X2, X3 (8 classes)

• LSTM almost always learns both sequences
  – 2 or 3 cell blocks, of size 2
Problems

• Truncated backprop doesn’t work for e.g. delayed XOR
• Adds gate units (not a big deal)
• Hard to do exact counting
  – Except Gers (2001) figured it out: can make sharp spikes every $n$ steps
• Basically acts like feedforward net that sees the whole sequence at once
Good things

- Basically acts like feedforward net that sees the whole sequence at once
- Handles long time lags
- Handles noise, continuous representations, discrete representations
- Works well without much parameter tuning (on these toy problems)
- Fast learning algorithm
Cool results since initial paper

• Combined with Kalman filters, can learn $A^n B^n C^n$ for $n$ up to 22 million (Perez 2002)
• Learn blues form (Eck 2002)
• Metalearning: learns to learn quadratic functions v. quickly (Hochreiter 2001)
• Use in reinforcement learning (Bakker 2002)
• Bidirectional extension (Graves 2005)
• Best results in handwriting recognition as of 2009