New Directions For Recurrent Neural Networks

Alex Graves
RNNs Work!

RNNs — especially LSTM / GRU variants — are now ubiquitous in ML research and routinely used for large-scale commercial tasks, including speech and handwriting recognition, machine translation, text-to-speech and many others.

Increasingly trained **end-to-end**: feed the input sequence in, get the desired output sequence out
RNNs Work!

RNNs — especially LSTM / GRU variants — are now ubiquitous in ML research and routinely used for large-scale commercial tasks, including speech and handwriting recognition, machine translation, text-to-speech and many others.

Increasingly trained end-to-end: feed the input sequence in, get the desired output sequence out

So what can’t they do, and what can we do about it?
Extension 1: External Memory

Problem: **RNN memory is stored in the vector of hidden activations**
- Activation memory is ‘fragile’: tends to be overwritten by new information
- No. of weights and hence computational cost grows with memory size (can’t put a whole book in memory)
- ‘Hard-coded’ memory locations make indirection (and hence variables) hard

Solution: **Give the net access to external memory**
- Less fragile: only some memory is ‘touched’ at each step
- Indirection is possible because memory content is independent of location
- Separates *computation* from *memory*

*Neural Machine Translation by Jointly Learning to Align and Translate*, Bahdanau et. al. (2014)
*Memory Networks*, Weston et. al. (2014)
*Neural Turing Machines*, Graves, Wayne, Danihelka (2014)
Differentiable Neural Computers

*Hybrid computing using a neural network with dynamic external memory,*
Graves, Wayne et. al., *Nature*, 2016
Basic Read/Write Architecture

The **Controller** is a neural network (recurrent or feedforward)

The **Heads** select portions of the memory and **read** or **write** to them

The **Memory** is a real-valued **matrix**
Memory Access

Most networks with external memory (RNNs with attention, Memory Nets, NTM, DNC…) use some form of content-based memory access: find the memory closest (e.g. cosine similarity) to some key vector emitted by the network, return either the memory contents or an associated value vector.

A universal access mechanism (c.f. associative computers)

But maybe not the most convenient for all tasks: e.g. we search real computers using text strings, directory trees, read/write time, user-defined titles or tags… many more mechanisms to be tried.
Dynamic Memory Allocation

- NTM could only ‘allocate’ memory in contiguous blocks, leading to memory management problems
- DNC defines a differentiable free list tracking the usage of each memory location
- Usage is automatically increased after each write and optionally decreased after each read
- The network can then choose to write to the most free location in memory, rather than searching by content
Memory Allocation Test
Memory Resizing Test
Searching By Time

- We wanted DNC to be able to iterate through memories in chronological order.
- To do this it maintains a **temporal link matrix** $L_t$ whose $i,j^{th}$ element is interpreted as the probability that memory location $i$ was **written to** immediately before location $j$.
- When reading from memory, DNC can choose to follow these links instead of searching by content.
- Unlike **location-based** access this facilitates two cognitively important functions:
  - **Sequence chunking** (don’t write at every step)
  - **Recoding** (iteratively reprocess a sequence, chunking each time)
London Underground with DNC

a. Read and Write Weightings
b. Read Mode
c. London Underground Map
d. Read Key
e. Location Content
## bAbI Results

<table>
<thead>
<tr>
<th>Task</th>
<th>bAbI Best Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LSTM (Joint)</td>
</tr>
<tr>
<td>1: 1 supporting fact</td>
<td>24.5</td>
</tr>
<tr>
<td>2: 2 supporting facts</td>
<td>53.2</td>
</tr>
<tr>
<td>3: 3 supporting facts</td>
<td>48.3</td>
</tr>
<tr>
<td>4: 2 argument rels.</td>
<td>0.4</td>
</tr>
<tr>
<td>5: 3 argument rels.</td>
<td>3.5</td>
</tr>
<tr>
<td>6: yes/no questions</td>
<td>11.5</td>
</tr>
<tr>
<td>7: counting</td>
<td>15.0</td>
</tr>
<tr>
<td>8: lists/sets</td>
<td>16.5</td>
</tr>
<tr>
<td>9: simple negation</td>
<td>10.5</td>
</tr>
<tr>
<td>10: indefinite knowl.</td>
<td>22.9</td>
</tr>
<tr>
<td>11: basic coreference</td>
<td>6.1</td>
</tr>
<tr>
<td>12: conjunction</td>
<td>3.8</td>
</tr>
<tr>
<td>13: compound coref.</td>
<td>0.5</td>
</tr>
<tr>
<td>14: time reasoning</td>
<td>55.3</td>
</tr>
<tr>
<td>15: basic deduction</td>
<td>44.7</td>
</tr>
<tr>
<td>16: basic induction</td>
<td>52.6</td>
</tr>
<tr>
<td>17: positional reas.</td>
<td>39.2</td>
</tr>
<tr>
<td>18: size reasoning</td>
<td>4.8</td>
</tr>
<tr>
<td>19: path finding</td>
<td>89.5</td>
</tr>
<tr>
<td>20: agent motiv.</td>
<td>1.3</td>
</tr>
<tr>
<td>Mean Err. (%)</td>
<td>25.2</td>
</tr>
<tr>
<td>Failed (err. &gt; 5%)</td>
<td>15</td>
</tr>
</tbody>
</table>

*Ask me anything: dynamic memory networks for natural language processing, Kumar et. al. (2015)*

*End-to-end memory networks, Sukhbaatar et. al. (2015)*
**Sparse Memory Access**

<table>
<thead>
<tr>
<th></th>
<th>Dense $\mathcal{O}(n)$</th>
<th>Sparse $\mathcal{O}(\log n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-based addressing</td>
<td>$\mathcal{O}(n^2)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>Temporal addressing</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>Read</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>Erase</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>Add</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
</tbody>
</table>

Using a KNN

By restricting reads and writes to 8 (say) locations per step.

*Scaling Memory-Augmented Neural Networks with Sparse Reads and Writes, Rae, Hunt et. al. (2016)*
Sparse DNC Efficiency

![Graph showing Wall Time vs. Number of memory slots (N) for SDNC and DNC. The X-axis is logarithmic. The Wall Time for SDNC is 4.42ms for N = 10^3 and 2.55s for N = 10^4. The Wall Time for DNC is higher for both values of N.]

![Graph showing Memory vs. Number of memory slots (N) for SDNC and DNC. The Y-axis is logarithmic. The Memory for SDNC is 23MiB for N = 10^3 and 5.5GiB for N = 10^4. The Memory for DNC is higher for both values of N.]
Problem: The number of steps of computation an RNN gets before emitting an output is determined by the length of the input sequence, not the difficulty of the task.

- Do any three positive integers \(a, b, c\) satisfy \(a^n + b^n = c^n\) for any integer \(n\) greater than two?

Solution: Train the network to learn how long to ‘think’ before it ‘acts’

- separate \textit{computation time} from \textit{data time}
RNN Computation Graph
A time penalty acts to reduce the total number of ‘ponder’ steps

Adaptive Computation Time With Recurrent Neural Networks, Graves (2016)
Addition with ACT

Input seq.  Target seq.
Addition
Results

![Graph showing sequence error rate over iterations with different time penalties. The graph demonstrates a decrease in error rate as iterations increase, with distinct lines for different time penalties, ranging from 0.0001 to 0.1. The line labeled 'Without ACT' is shown in black.](image-url)
Machine Translation

Dataset: WMT14 test set, English to French

(SMT): 37.0 BLEU

Baseline AttLSTM: 3.4 PPL, 37.5 BLEU

AttLSTM + ACT: 3.1 PPL, 38.3 BLEU

Vinyals, Jozefowicz - unpublished (yet)
Pondering Wikipedia (character level)
ACT for Feedforward Nets

ImageNet high ponder cost examples
Extension 3: Beyond BPTT

**Problem:** Most RNNs are trained with Backpropagation Through Time (BPTT)

- Memory cost increases with sequence length
- Weight update frequency decreases
- The better RNNs get, the longer the sequences we train them on

**Solutions:**

1. Truncated backprop (misses long range interactions)
2. RTRL (too expensive)
3. Approximate/local RTRL (promising)
4. **Synthetic Gradients (drastic)**

*Training recurrent net-works online without backtracking.* Ollivier et. al. (2015)

*Long Short-Term Memory.* Hochreiter and Schmidhuber (1997)
**Decoupled Neural Interfaces**

Consider a regular feed-forward network

![Diagram of a feed-forward network with layers and loss](image)

We can create a **model of error gradients** using local information.

The result is Layer 1 can now update **before the execution of Layer 2**.

*Decoupled Neural Interfaces using Synthetic Gradients.*

Jaderberg et. al. (2016)
**Decoupled Neural Interfaces**

The **synthetic gradient model** is trained to predict target gradients.

The target gradients could themselves be bootstrapped from other downstream synthetic gradient models.

Analogous to return prediction bootstrapping in RL: ‘Learn a guess from a guess’
Truncated BPTT
RNN learns to predict the gradients returned by its future self.
Recurrent Models

DNI extends the time over which a truncated BPTT model can learn.

+ Convergence speed
+ Data efficiency
MULTI NETWORK

Two RNNs. Tick at different clock speeds. Must communicate to solve task.
Overall Architecture