Neural Turing Machines

Alex Graves, Greg Wayne & Ivo Danihelka

Aviral Kumar

Indian Institute of Technology Bombay

Table of contents

Introduction

RNNs:

- Learn, perform complex data transformations
- *Turing-Complete* $\implies$ simulate arbitrary programs
- Not so simple in practice

LSTM:

- Addresses vanishing gradient and exploding gradient problems
- Can learn over longer durations; selectively forget information

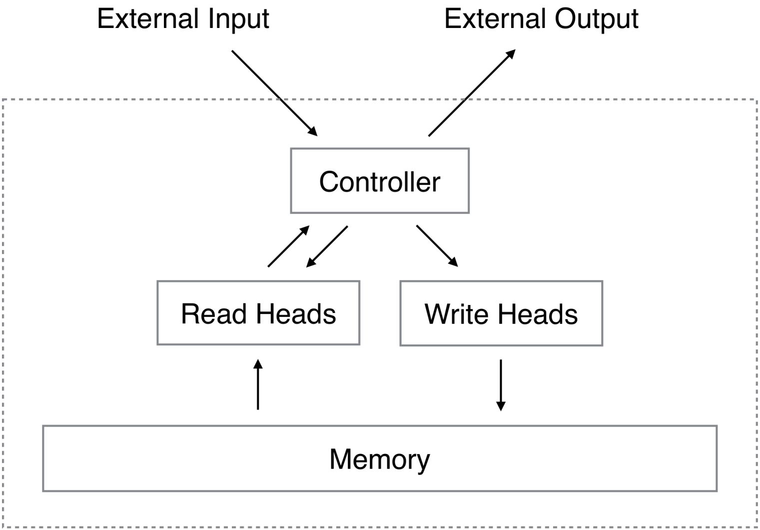
Idea

- Extend the capabilities of Neural Networks by coupling them to external memory sources.
- Add a memory and perform rule-based manipulation
- Learn Programs using inputs and outputs
- Analogous to TM

$$TM = Finite\ State\ Automaton\ +\ Memory\ Tape$$

$$NTM = Neural\ Network\ +\ Memory$$

NTM Architecture



Every component is differentiable $\Longrightarrow$ gradient descent

Read-Write Architecture

- **Blurry** reads and writes
- **Attentional "focus"** mechanism
- Memory location in focus determined by outputs from heads
- $Memory = N \times M$ matrix; $N = \#(\text{memory locations})$; $M = size(Mem_i)$
- Outputs define normalised weighting over rows in a memory matrix
- Ability to focus sharply and weakly both

Addressing Mechanisms

Two types: **Content-based** & **Location-based**

Content-based Addressing

Focus attention on locations with similar current values and values emitted by controller

Location-based Addressing

Focus attention on specific locations in the memory matrix

Content-based Addressing

M length key vector $\mathbf{k_t}$ (produced by the head) compared to $\mathbf{M_t(i)}$ by similarity measure $S[\cdot,\cdot]$
Positive key strength β_t to amplify or attenuate a focus
Output: Normalised weighting w_t^c

$$w_t^c(i) \leftarrow \frac{exp(\beta_t S[\mathbf{k_t}, \mathbf{M_t(i)}])}{\sum_j exp(\beta_t S[\mathbf{k_t}, \mathbf{M_t(j)}])}$$

In the paper, S is cosine similarity

Location-based addressing

Two issues to handle : random-access jumps, iterations
Notion of **shift** of a weighting (used to shift focus across locations)

1. Interpolation

Scalar interpolation gate $g_t \in (0,1)$: Scalar to blend between the weightings $\mathbf{w_t}$ and $\mathbf{w_{(t-1)}}$
Gated weighting: $\mathbf{w_t^g} \leftarrow g_t \mathbf{w_t^c} + (1 - g_t)\mathbf{w_{t-1}^c}$

2. Shift Weighting ($\mathbf{s_t}$)

Defines a normalised distribution over integer shifts
Examples: Softmax, discrete values

Rotation applied to $\mathbf{w_t^g}$ by s_t is given b the circular convolution:

$$\widetilde{w}_t(i) \leftarrow \left(\sum_{j=0}^{N-1} w_t^g(i) s_t(i-j) \right) \mod N$$

Sharpening : $w_t(i) \leftarrow \frac{\widetilde{w}_t(i)^{\gamma_t}}{\sum_j \widetilde{w}_t(j)^{\gamma_t}}; \gamma_t \geq 1$

Combined Addressing Modes

Three modes of addressing:

1. Purely Content-based

2. Content-based + Shifting

Allows focus on the top of a specific block and then move to a particular position in the block (Arrays, structures)

3. Rotation without using content

Iterate through a sequence of addresses by advancing same distance at each time step. (Iterators, Loops)

Reading

Weightings are normalised: $\sum_i w_t(i) = 1, 0 \leq w_t(i) \leq 1, \forall i$
M-length read vector $\mathbf{r_t}$ is defined as convex-combination of row vectors $\mathbf{M_t(i)}$ in memory:

$$\mathbf{r_t} \leftarrow \sum_i w_t(i)\mathbf{M_t(i)}$$

Writing

Input and Forget Gates LSTM $\implies$ Write $=$ *erase* $+$ *add*

Erase Operation

Read Head emits $\mathbf{e_t}, \forall i, \mathbf{e_t(i)} \in (0, 1)$. Reset memory as:

$$\widetilde{\mathbf{M}}_t(i) \leftarrow \mathbf{M}_{t-1}(i)[\mathbf{1} - w_t(i)\mathbf{e_t}]$$

Add Operation

Write Head emits $\mathbf{a_t}$ (add vector). Reset memory as:

$$\mathbf{M}_t(i) \leftarrow \widetilde{\mathbf{M}}_t(i) + w_t(i)\mathbf{a_t}$$

- Both erase and add differentiable
- Fine gain control over the memory, as erase and add vectors independent

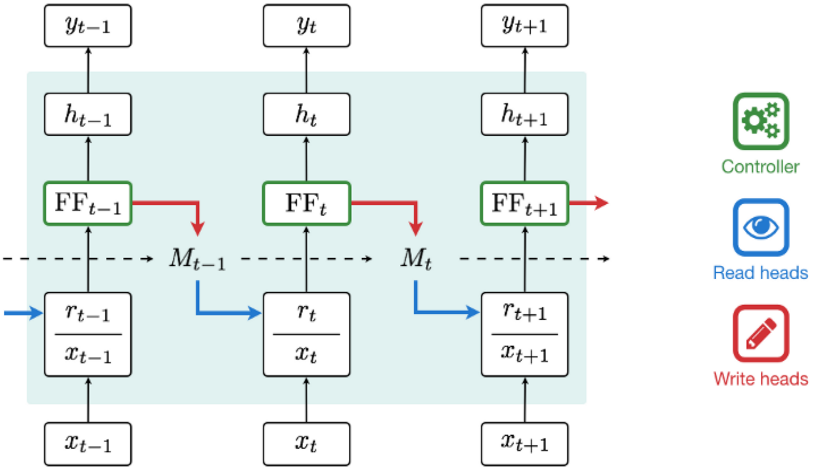
Controller Architecture

- Controller : A neural network : **LSTM-RNN** or **Feed Forward**
- LSTM-RNN's internal memory analogous to RAM (can remember past sequences)
- Feed Forward may require multiple read-write heads (to remember)

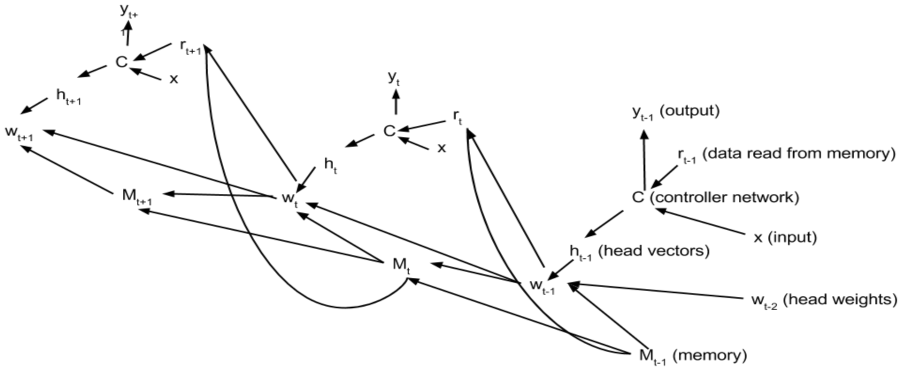
Training

- Fully differentiable from beginning to the end
- Trained using Gradient Descent analogous to LSTM
- Used **RMSProp** (a variant of Stochastic Gradient Descent)to train
- Gradients clipped to the range (-10, 10)
- LSTM NTM had 3 hidden stacked layers

Schematic Diagram - 1



Schematic Diagram - 2



Copy

Test storage and recall of a sequence of information. Trained with sequences of 8 bit randomly chosen vectors, with $L(Seq) \in (1, 20)$.
Generalise the algorithm to longer sequences; not observed in LSTM Networks.
Algorithm:

1. **while** input delimiter not seen **do**
2. receive input vector
3. write input to head location
4. increment head location by 1
5. **end while**
6. return head to start location
7. **while** true **do**
8. read output vector from head location
9. emit output
10. increment head location by 1
11. **end while**

Priority Sort

Inputs: Seq of random binary vectors + a scalar priority for each
Target: Seq contains the binary vectors sorted according to priorities

- Hash table kind of algorithm learned
- Priority determined locations of write
- Output in order of increasing locations

Observations:

- NTM-LSTM and NSTM-FF both outperform LSTM on this task
- Best results using 8 read-write heads, probably due to better sorting with unary vector operations

Associative Recall

Inputs: A sequence of vectors separated by delimiters
Outputs: Given an input item, output the next item in the previously presented sequence
Observations:

- NTM-LSTM and NTM-FF both learn in 10^5 episodes, LSTM takes a million
- NTM-FF learns faster than NTM-LSTM
- External memory of more advantage than internal

Algorithm:
1. When item delimiter is presented, write compressed representation of the previous three time slices of the item.
2. Calculate same representation of query item, use content based lookup to retrieve the item, and then shift by 1 unit to next item.

Comments

- A huge benefit over RNNs and LSTMs is that it can learn generic algorithms, not restricted to properties of inputs/outputs presented.
- The paper could have explained more rigorously (mathematically) the training procedure of NTM and why NTM outperforms LSTM in every task.
- Reinforcement Learning, evolutionary learning could be applied as a future work
- The paper seems to be a bit biased towards proving NTM's performance, but the justification doesn't look enough.
- Alex Graves points out in one of his NIPS tutorials that this framework can now learn shortest path algorithm in a graph.