Neural Turing Machines

and memory-augmented techniques

Ananth Mahadevan, Christabella Irwanto

Agenda

all about Neural Turing Machines (NTMs)

- Motivation
- How NTMs work
- Experiments
- Discussion

Motivation

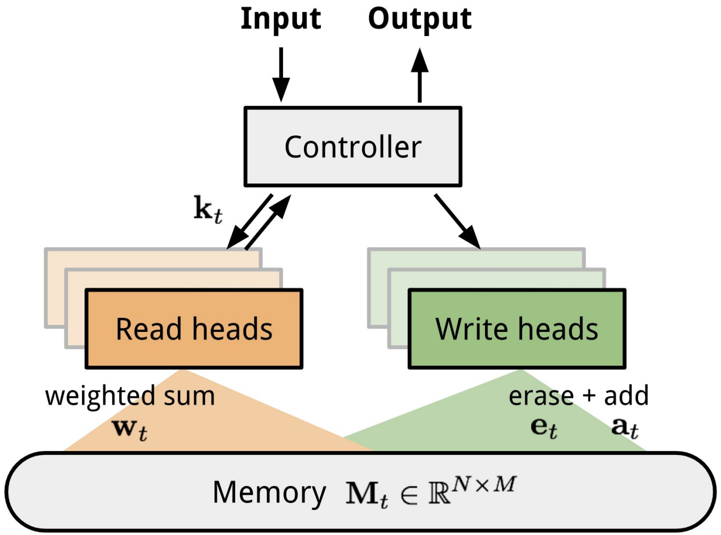
- Neuroscience angle: "working memory" is limited
    - Cognitive system with limited capacity for **temporarily** storing and manipulating information
    - Short term memory cannot manipulate the information
- For a hard mathematical problem/memorizing many numbers, ~~we~~ most of us (except Ananth) would need to **offload some computation to pen and paper**
- Like computer programs
- Learn how to manage memory to solve a problem: sort of "learning to program"
    - NTM learns its own basic algorithms for tasks such as copying, sorting, and associative recall

NTM

- Entire structure is differentiable
- Part of broader trend to "**differentiable programming**"
  - incorporation of **classic data structures** (e.g. RAM, stacks, queues) into **gradient-based learning systems**
  - Increasing memory capacity in NTM is easier than in pure LSTM networks.
- Possibility of combining the best of program induction and deep learning
  - Structured representations – *objects, forces, agents, causality, compositionality* – help explain important facets of human learning and thinking
  - Deep learning systems not shown to work with these representations, but shown surprising effectiveness of gradient descent in large models

Neural Turing Machine (NTM)

- Neural network *controller* (e.g. feed forward or RNN)
  - also outputs "heads" that parameterize reads and writes to/from the...
- Memory bank! (stores processed information)



NTM unfolded

- Controller accepts input x and read vector r, outputs read and write heads

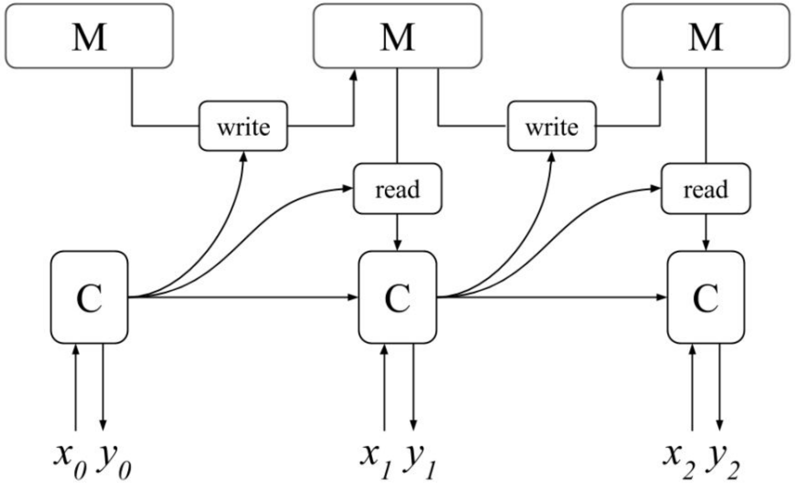


FIGURE 2.13: The neural Turing machine unrolled through time (Olah and Carter, 2016).

Reading

- At time **t**, normalized weight vector $\mathbf{w_t}$ controls how much attention to give to different memory locations

$$\sum_i w_t(i) = 1, \qquad 0 \le w_t(i) \le 1, \forall i.$$

- Read vector $\mathbf{r_t}$ is simply a sum weighted by attention intensity

$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i)$$

- Differentiable with respect to both the weight and the memory bank
- A form of attention implemented as memory addressing

Writing

- Inspired by input and forget gates in LSTM
- Given weight vector $\mathbf{w_t}$ emitted by a write head at time **t**, we
    - **erase** with $\mathbf{e_t}$

$$\tilde{\mathbf{M}}_t(i) \longleftarrow \mathbf{M}_{t-1}(i)\left[\mathbf{1} - w_t(i)\mathbf{e}_t\right]$$

    - **add** with $\mathbf{a_t}$

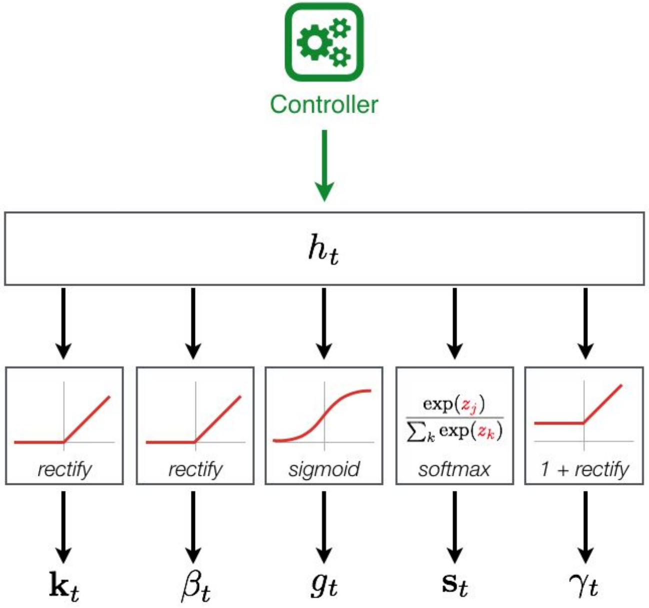
$$\mathbf{M}_t(i) \longleftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\,\mathbf{a}_t$$

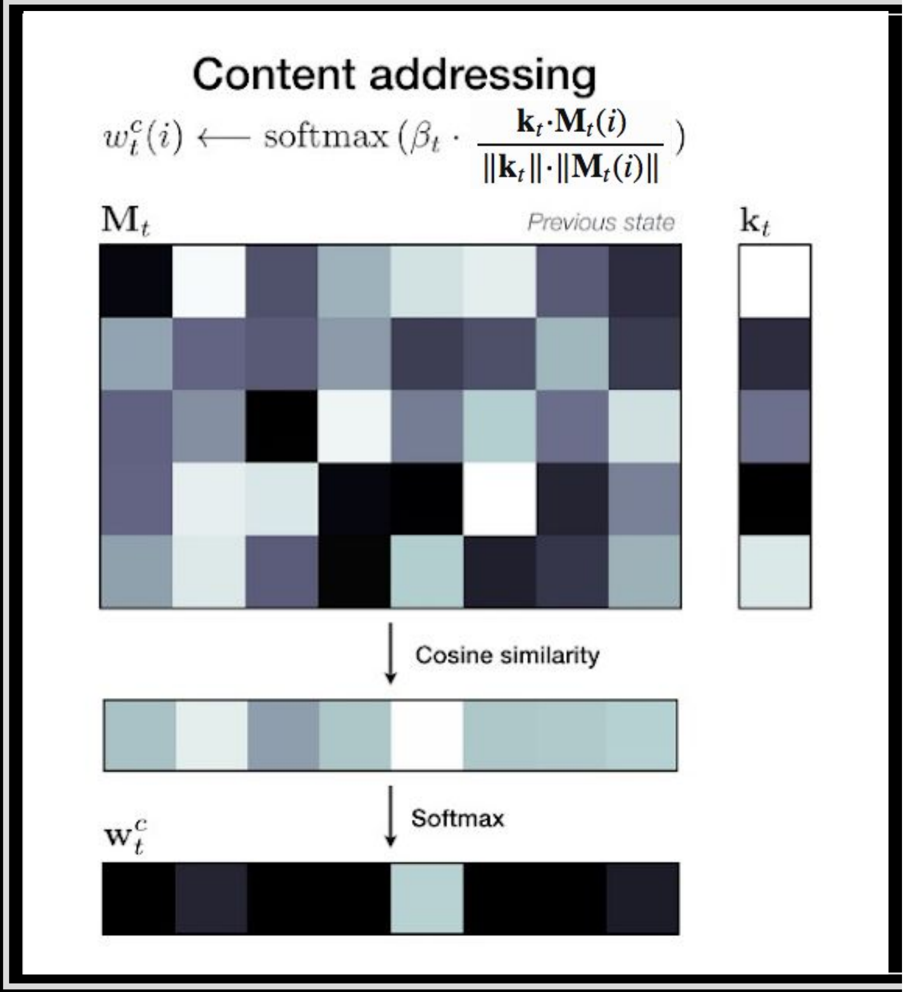
Addressing mechanisms

- How is $\mathbf{w_t}$ produced?
- $\mathbf{w_t}$ is updated through a series of four intermediate smooth operations
    - *content addressing*
    - *interpolation*
    - *convolutional shift*
    - *sharpening*
- Operations depend on parameters from the controller
    - functions of hidden state $\mathbf{h_t}$ emitted by the controller.

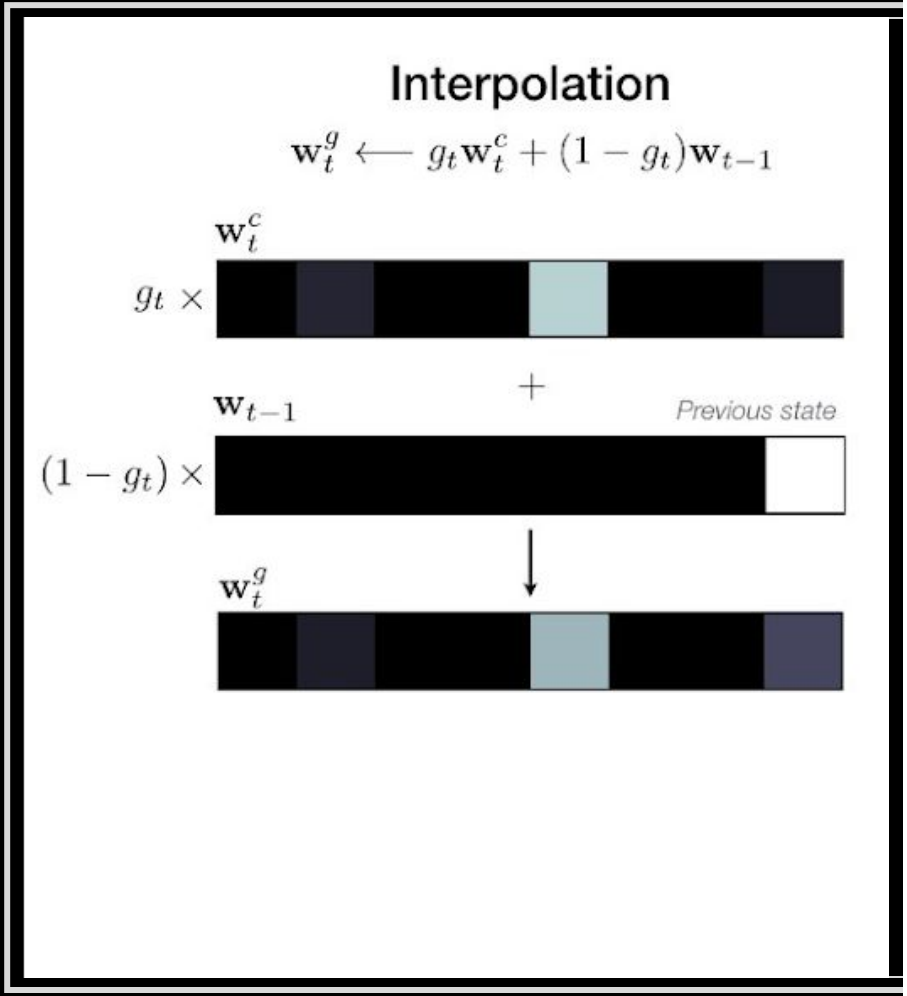
Parameters for weight ($\mathbf{w}_t$) updates

- 5 parameters specific to each read/write head
- Each box is a 1-layer neural network with some activation function

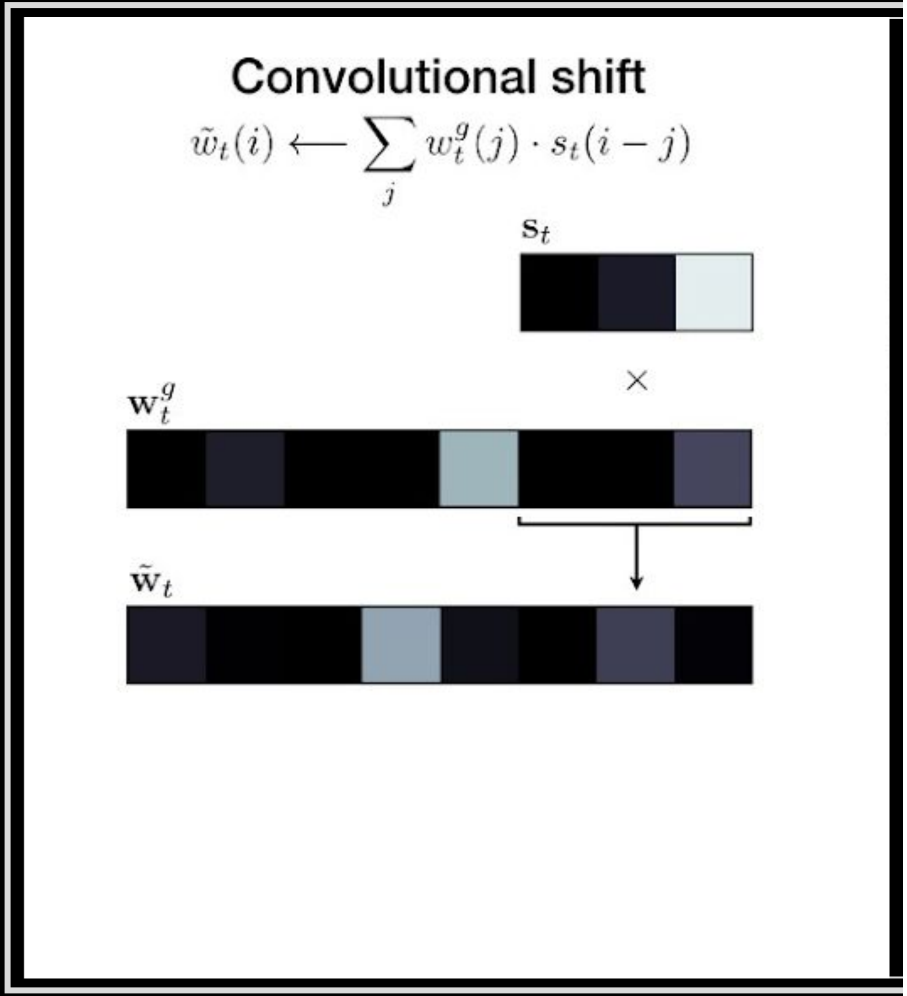




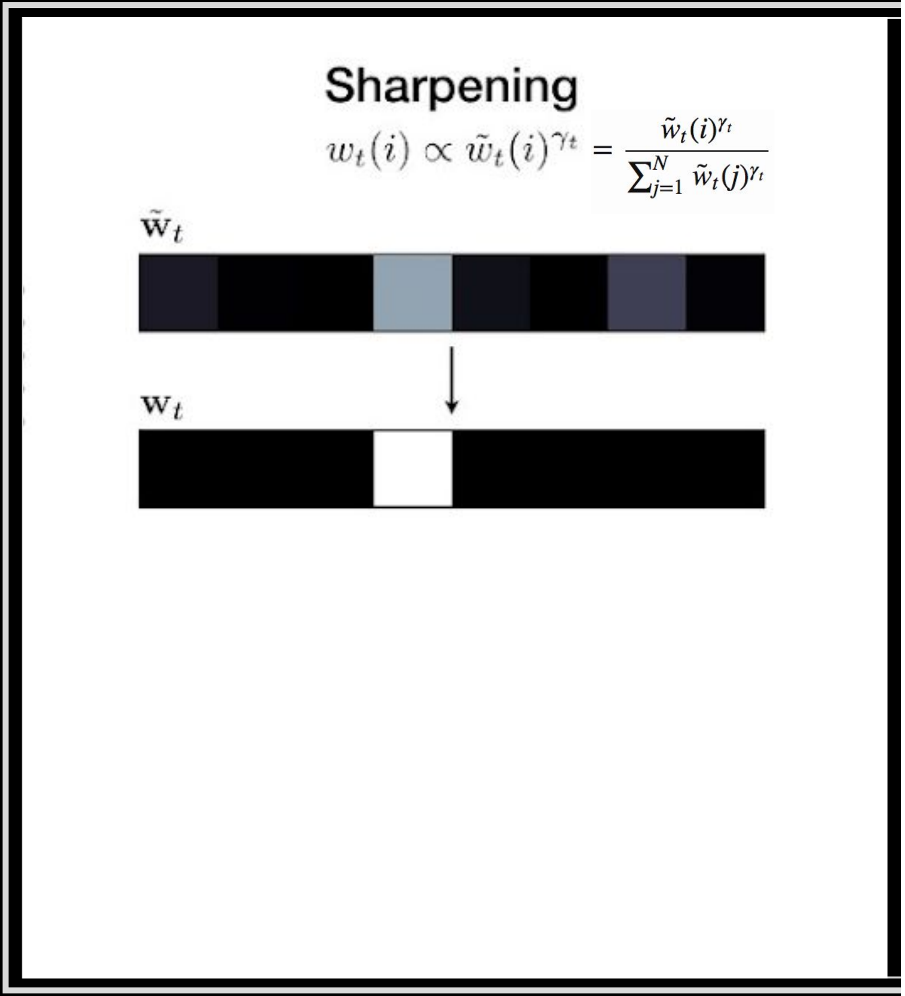
- retrieve specific informations in memory
- compute cosine similarity between key vector $\mathbf{k_t}$ extracted by controller from input and memory
- then normalized by softmax
  - with strength multiplier $\boldsymbol{\beta}_\mathbf{t}$ to amplify or attenuate the focus of the distribution



- interpolation gate scalar $\mathbf{g_t}$ blends newly-generated content-based weight vector $\mathbf{w^c}_\mathbf{t}$ with weight vector from previous time step $\mathbf{w_{t-1}}$



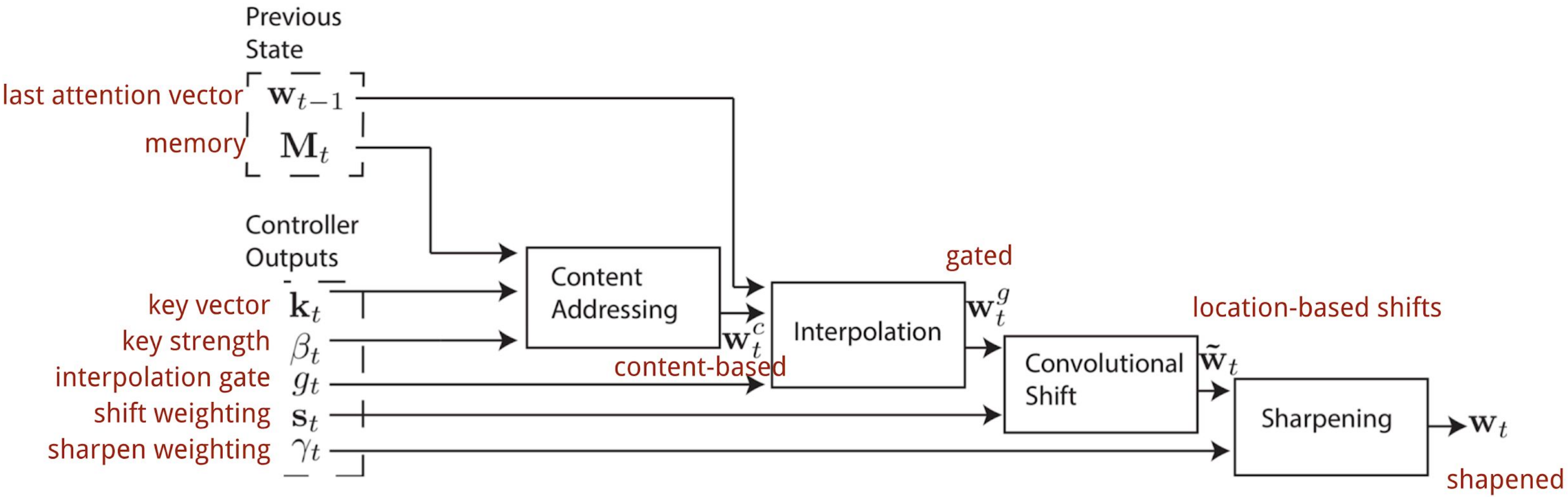
- location-based addressing done by 1-d convolution of $\mathbf{w^g_t}$ with kernel $\mathbf{s_t(.)}$, a function of the position offset $\mathbf{i - j}$



- shifted attention vector is sharpened with a sharpening scalar $\gamma_t \geq 1$

Flow diagram of addressing mechanisms

- Complete process of generating the attention vector $\mathbf{w_t}$



Controller Network Architecture

Feed forward vs recurrent:

- LSTM has its own internal memory complementary to **M**
- Feed forward has better transparency

Analogy:

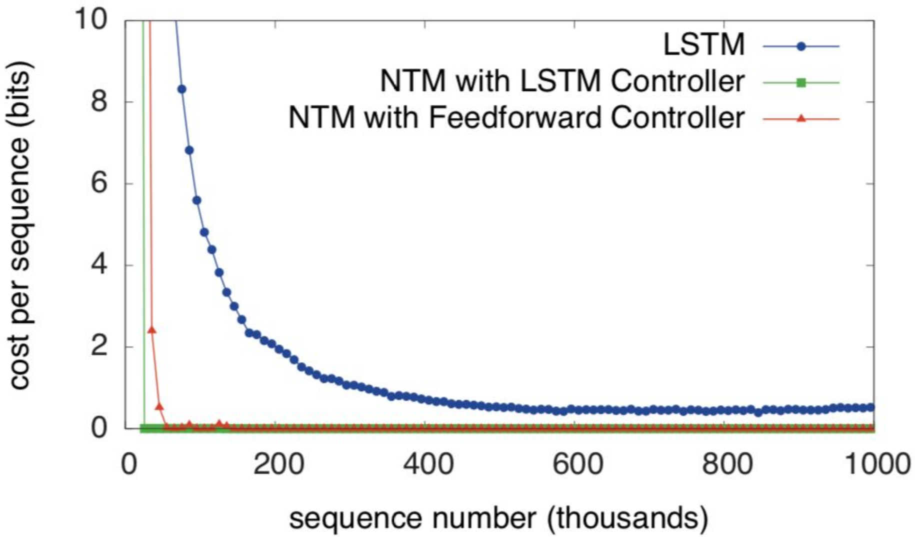
– Controller ⇔ CPU

– Memory Matrix ⇔ RAM

– Hidden Activations of LSTM ⇔ Registers in Processor

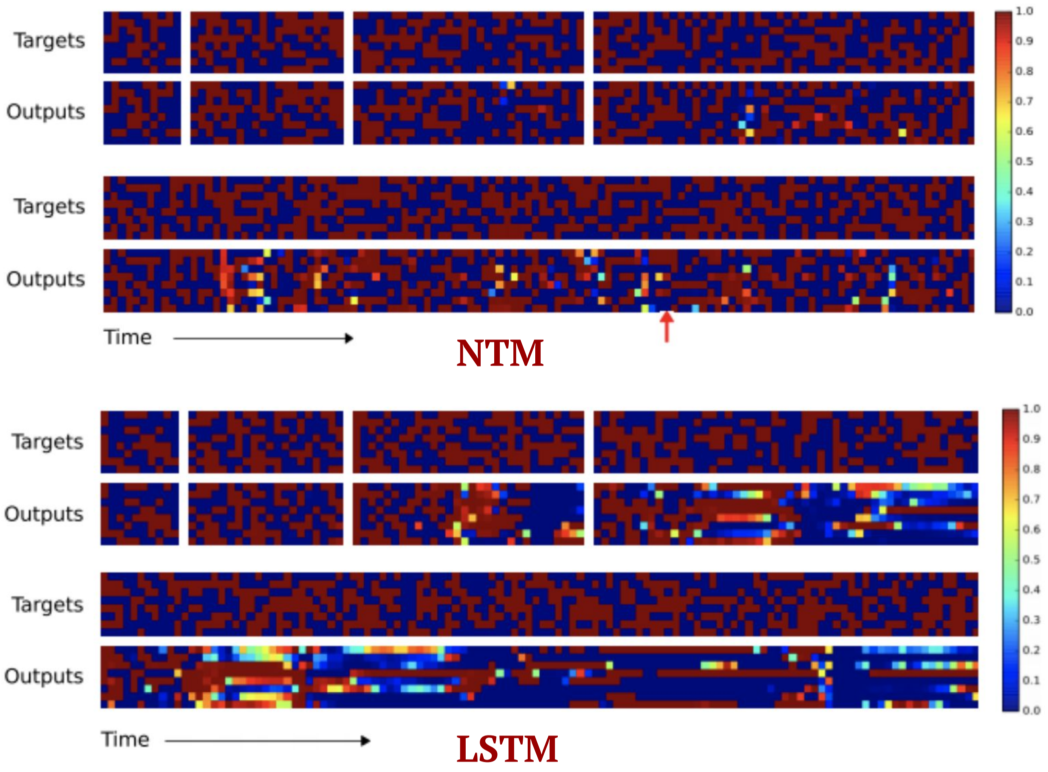
Experiments: Copy

- RNNs have struggled to remember information over long periods
- Given *n* random eight bit vectors followed by a delimiter flag, repeat it
- "Cost per sequence": number of bits incorrectly recalled over a sequence



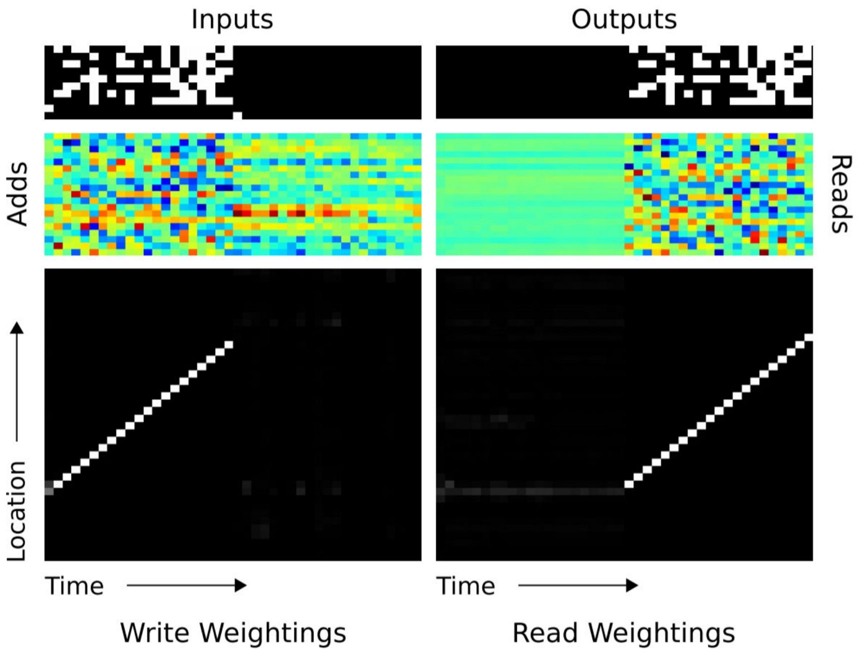
Experiments: Copy

- **Training** sequences of lengths 1 to 20
- **Test** with lengths 10, 20, 30, 50, 120
- [blue, red] -> [0, 1]
- NTM far fewer errors on longer sequences; scales better



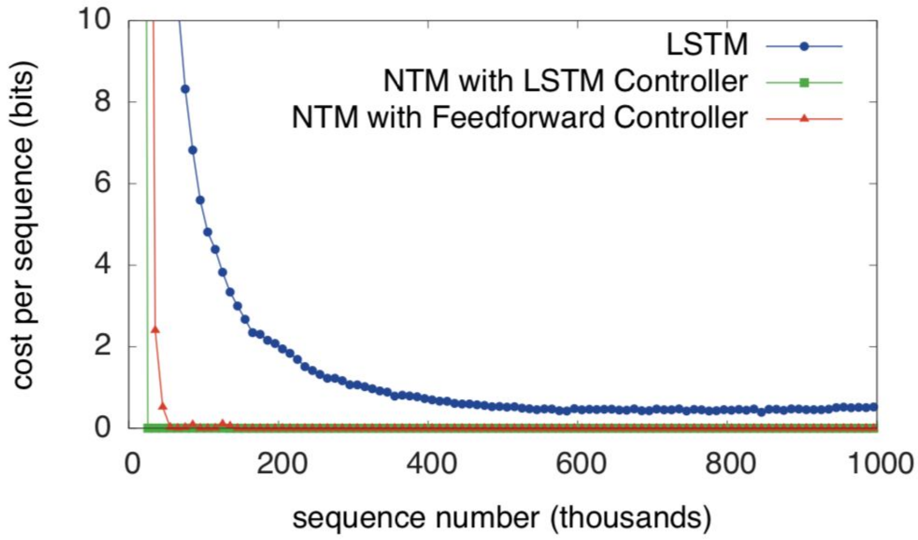
Experiments: Copy

- Visualize how the NTM reads from and writes to **M**



Experiments: Repeat Copy

- Like a **nested for loop:** repeatedly **copy** sequence some ***x* number of times**
- Training input: random-length sequence of 8-bit binary vectors + scalar *x* from 1-10



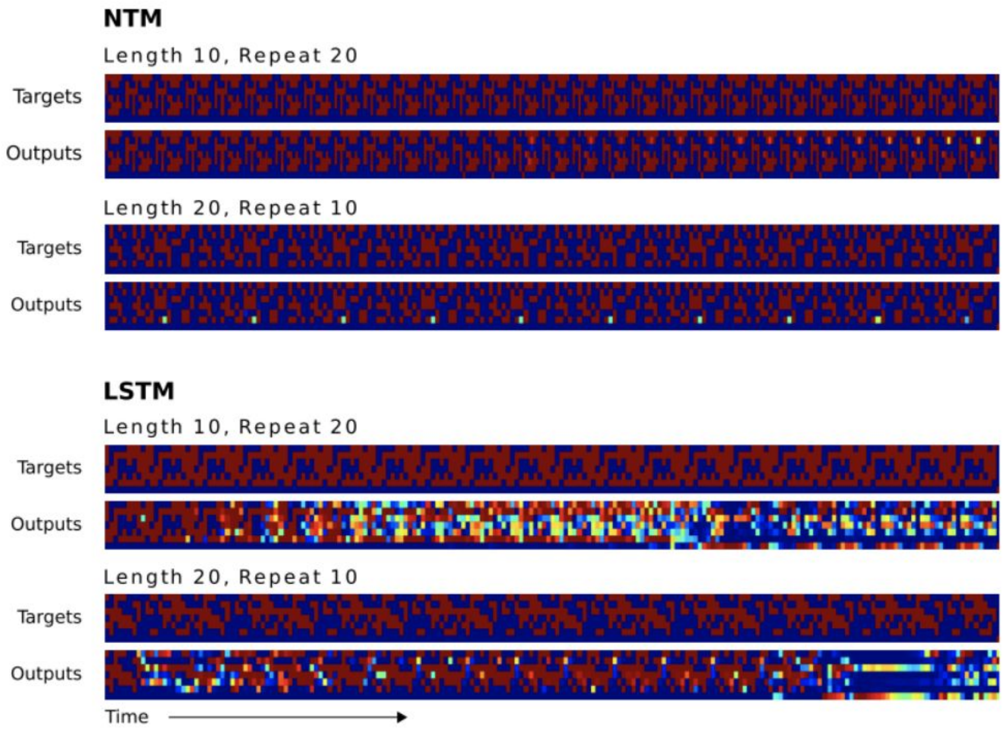
Experiments: Repeat Copy

Increased # repeats:

- NTM much better, but still keeps falsely predicting end of sequence (emits delimiter flag after every repetition beyond the 11th)

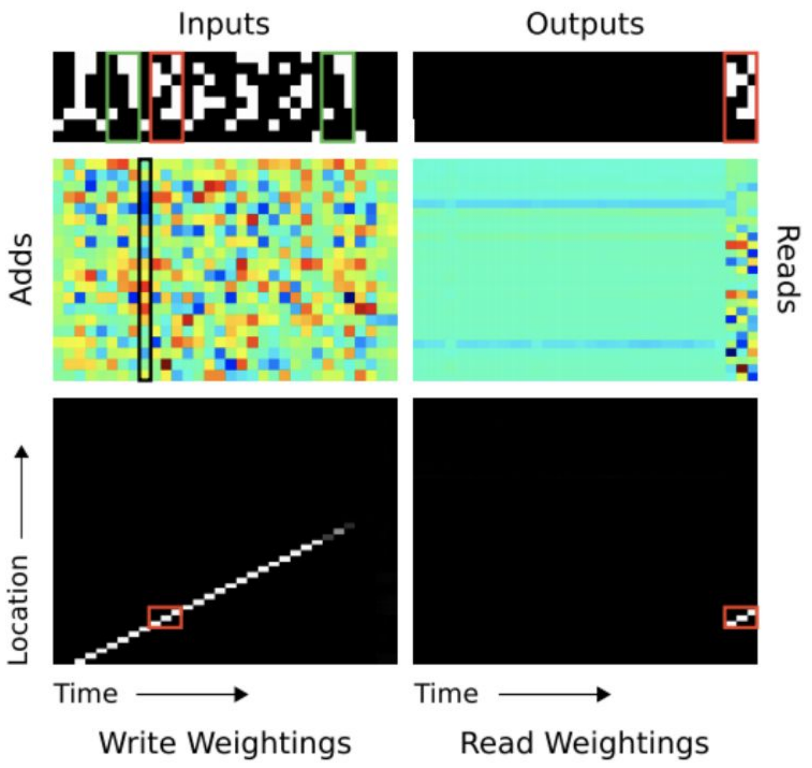
Increased length:

- NTM generalizes much better



Experiment: Associative recall

- Can NTMs can learn "indirection" i.e. one data item pointing to another?
- **input:**
  - list of items
  - query of one of the items (green)
- **output:**
  - next item in the list (red)
- at delimiter flag, NTM forms compressed representation (black box in “Adds”) of each item



Experiment: Associative recall

- **Feed forward**-controller NTM **outperforms LSTM**-controller NTM
  - suggests that NTM's memory is a superior data storage system than LSTM's internal state

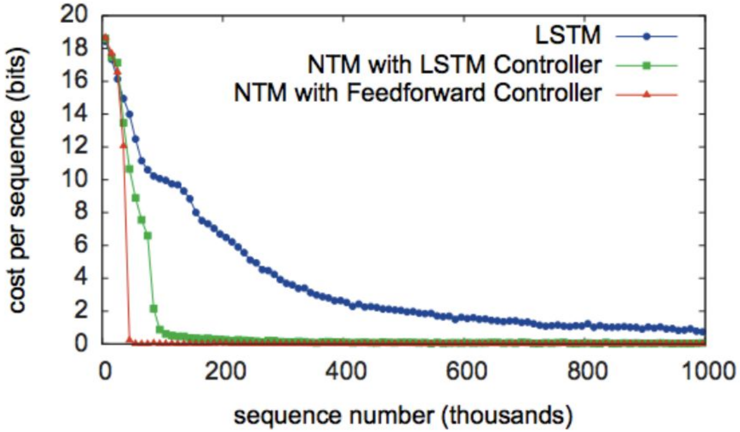


Figure 10: Associative Recall Learning Curves for NTM and LSTM.

Segue to code

Experiment: Dynamic n-grams

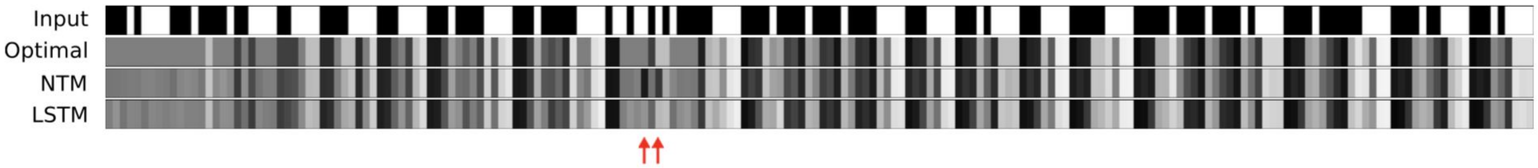
- Whether NTMs could learn posterior predictive distributions
- N-grams (sequences of N items), which given previous items in the sequence, determine some probability distribution over the next item in the sequence
- Optimal estimator:

$$P(B = 1|N_1, N_0, \mathbf{c}) = \frac{N_1 + \frac{1}{2}}{N_1 + N_0 + 1}$$

- **B** is the next bit, **c** is the previous 5-bit context, and **N0** and **N1** are the number of zeros and ones observed after **c**
- NTM achieves small, but significant performance advantage over LSTM, but never quite reaches optimum cost

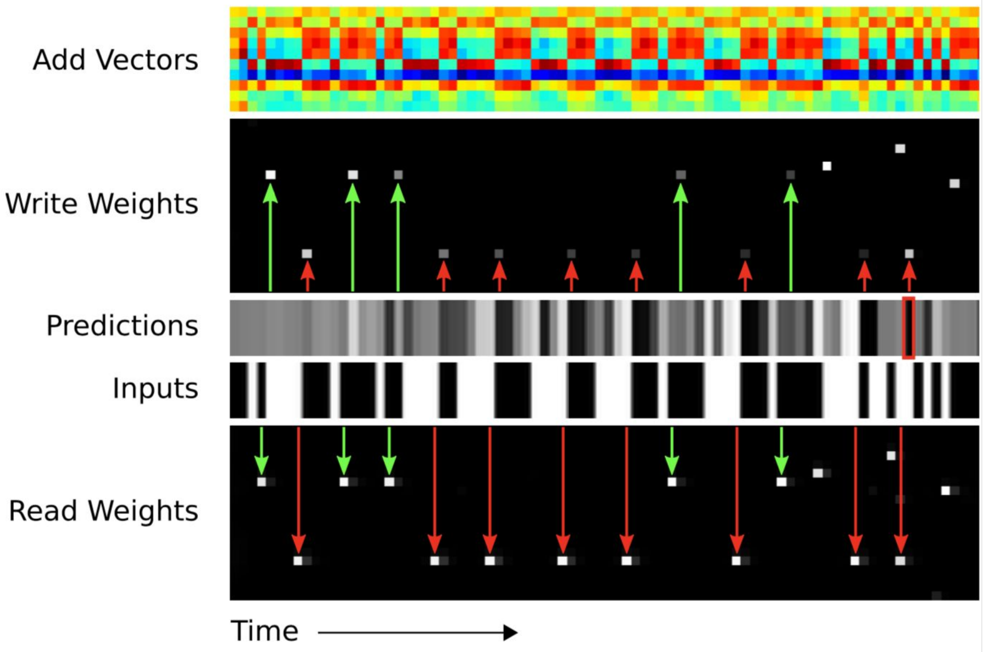
Experiment: Dynamic n-grams

- Top row: test sequence
- Rows below: predictive distributions by optimal estimator, NTM, and LSTM
- NTM predictions mostly indistinguishable from optimal (although some clear mistakes e.g. red arrows)
- LSTM good but appears to diverge further as sequence progresses
  - speculate that LSTM "forgets" observations at start



Experiment: Dynamic n-grams

- red and green arrows where same context is observed (“00010” for green, “01111” for red)
- same location read then written to
- Network uses writes to keep count of the fraction of ones and zeros following each context in the sequence so far
- Add vectors clearly anti-correlated at 0s and 1s, suggesting a distributed “counter.”



Experiment: Priority sort

Whether the NTM can sort data—an important elementary algorithm

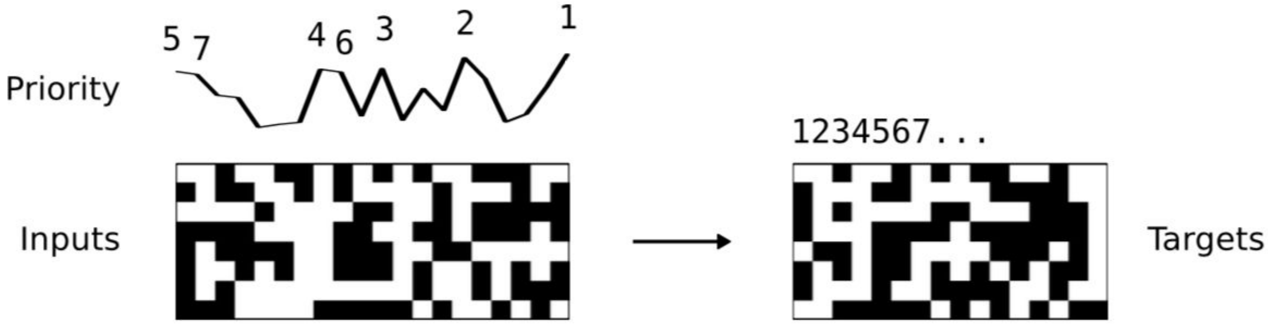
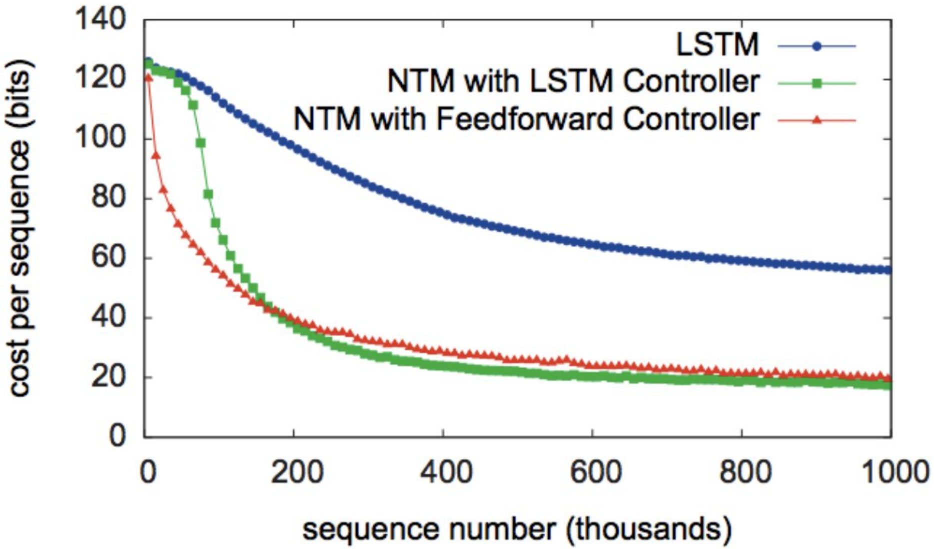


Figure 16: Example Input and Target Sequence for the Priority Sort Task. The input sequence contains random binary vectors and random scalar priorities. The target sequence is a subset of the input vectors sorted by the priorities.

Experiment: Priority sort

Again, NTMs > LSTM



Implementational Checkpoint

- Took 4 years to find a stable implementation
- Paper just on implementing NTMs

arXiv.org > cs > arXiv:1807.08518

Search...

Help | Adv

Computer Science > Machine Learning

Implementing Neural Turing Machines

Mark Collier, Joeran Beel

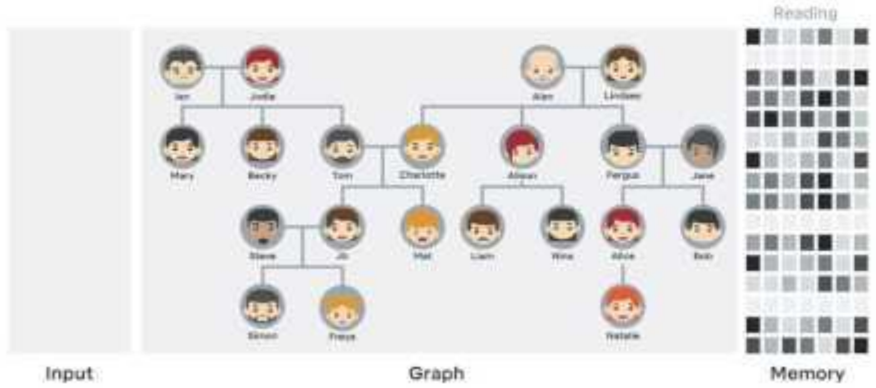
(Submitted on 23 Jul 2018 (v1), last revised 26 Jul 2018 (this version, v3))

Neural Turing Machines (NTMs) are an instance of Memory Augmented Neural Networks, a new class of recurrent neural networks which decouple computation from memory by introducing an external memory unit. NTMs have demonstrated superior performance over Long Short-Term Memory Cells in several sequence learning tasks. A number of open source implementations of NTMs exist but are unstable during training and/or fail to replicate the reported performance of NTMs. This paper presents the details of our successful implementation of a NTM. Our implementation learns to solve three sequential learning tasks from the original NTM paper. We find that the choice of memory contents initialization scheme is crucial in successfully implementing a NTM. Networks with memory contents initialized to small constant values converge on average 2 times faster than the next best memory contents initialization scheme.

Differentiable Neural Computer (DNC)

- Successor of NTMs; more generalizable network structure
- Wide range of tasks including natural language understanding, basic inference, planning, etc.
    - E.g. Solving block puzzles
    - E.g. Finding paths between nodes in a graph after memorizing the graph
- Has ability to **free** and **allocate** memory, i.e. to reuse memory locations
- Using **temporal links**, can search by time of writes

DNC



References

- https://medium.com/snips-ai/ntm-lasagne-a-library-for-neural-turing-machines-in-lasagne-2cdce6837315
- https://github.com/MarkPKCollier/NeuralTuringMachine
- https://github.com/loudinthecloud/pytorch-ntm
- https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
- http://www.robots.ox.ac.uk/~tvg/publications/talks/NeuralTuringMachines.pdf
- https://rylanschaeffer.github.io/content/research/neural_turing_machine/main.html
- Lake, B., Ullman, T., Tenenbaum, J., & Gershman, S. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences, 40*, E253. doi:10.1017/S0140525X16001837
- Graves, Alex; Wayne, Greg; Danihelka, Ivo (2014). "Neural Turing Machines". arXiv:1410.5401
- Graves, Alex; Wayne, Greg; Reynolds, Malcolm; Harley, Tim; Danihelka, Ivo; Grabska-Barwińska, Agnieszka; Colmenarejo, Sergio Gómez; Grefenstette, Edward; Ramalho, Tiago (2016-10-12). "Hybrid computing using a neural network with dynamic external memory". *Nature*. **538** (7626): 471–476. Bibcode:2016Natur.538..471G. doi:10.1038/nature20101. ISSN 1476-4687. PMID 27732574.