Neural Turing machine

Deep Learning Lecture 11

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2017
(Slides credit to Graves et al.)

Table of Contents

Memory matters

- RNN allows the networks to have some short term memory
  - Btw, even with LSTM, memory tends to be "forgotten" after a short period of time
  - For more complicated tasks, like Q&A system, we need to have longer-term memory
  - BTW, we may consider the weights inside the network as long term memory. But they are difficult to be manipulated with
- We will consider neural Turing machine (NTM) today, which process input in sequences, much like an LSTM, but with additional benefits:
  1. The external memory allows the network to learn algorithmic tasks easier
  2. Having larger capacity, without increasing the network's trainable parameters

Memory matters

- RNN allows the networks to have some short term memory
  - Btw, even with LSTM, memory tends to be “forgotten” after a short period of time
  - For more complicated tasks, like Q&A system, we need to have longer-term memory
  - BTW, we may consider the weights inside the network as long term memory. But they are difficult to be manipulated with
- We will consider neural Turing machine (NTM) today, which process input in sequences, much like an LSTM, but with additional benefits:
  1. The external memory allows the network to learn algorithmic tasks easier
  2. Having larger capacity, without increasing the network’s trainable parameters

Memory matters

- RNN allows the networks to have some short term memory
  - Btw, even with LSTM, memory tends to be "forgotten" after a short period of time
  - For more complicated tasks, like Q&A system, we need to have longer-term memory
  - BTW, we may consider the weights inside the network as long term memory. But they are difficult to be manipulated with
- We will consider neural Turing machine (NTM) today, which process input in sequences, much like an LSTM, but with additional benefits:
  1. The external memory allows the network to learn algorithmic tasks easier
  2. Having larger capacity, without increasing the network's trainable parameters

Memory matters

- RNN allows the networks to have some short term memory
  - Btw, even with LSTM, memory tends to be "forgotten" after a short period of time
  - For more complicated tasks, like Q&A system, we need to have longer-term memory
  - BTW, we may consider the weights inside the network as long term memory. But they are difficult to be manipulated with
- We will consider neural Turing machine (NTM) today, which process input in sequences, much like an LSTM, but with additional benefits:
  1. The external memory allows the network to learn algorithmic tasks easier
  2. Having larger capacity, without increasing the network's trainable parameters

Memory matters

- RNN allows the networks to have some short term memory
  - Btw, even with LSTM, memory tends to be “forgotten” after a short period of time
  - For more complicated tasks, like Q&A system, we need to have longer-term memory
  - BTW, we may consider the weights inside the network as long term memory. But they are difficult to be manipulated with
- We will consider neural Turing machine (NTM) today, which process input in sequences, much like an LSTM, but with additional benefits:
  1. The external memory allows the network to learn algorithmic tasks easier
  2. Having larger capacity, without increasing the network’s trainable parameters

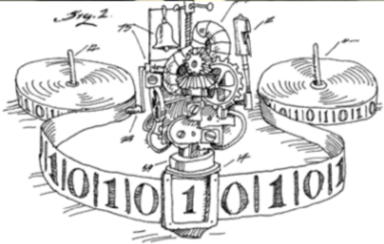
Memory matters

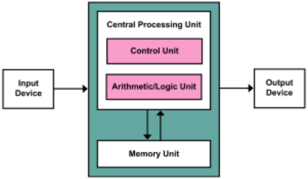
- RNN allows the networks to have some short term memory
  - Btw, even with LSTM, memory tends to be “forgotten” after a short period of time
  - For more complicated tasks, like Q&A system, we need to have longer-term memory
  - BTW, we may consider the weights inside the network as long term memory. But they are difficult to be manipulated with
- We will consider neural Turing machine (NTM) today, which process input in sequences, much like an LSTM, but with additional benefits:
  1. The external memory allows the network to learn algorithmic tasks easier
  2. Having larger capacity, without increasing the network’s trainable parameters

Memory matters

- RNN allows the networks to have some short term memory
  - Btw, even with LSTM, memory tends to be "forgotten" after a short period of time
  - For more complicated tasks, like Q&A system, we need to have longer-term memory
  - BTW, we may consider the weights inside the network as long term memory. But they are difficult to be manipulated with
- We will consider neural Turing machine (NTM) today, which process input in sequences, much like an LSTM, but with additional benefits:
  1. The external memory allows the network to learn algorithmic tasks easier
  2. Having larger capacity, without increasing the network's trainable parameters

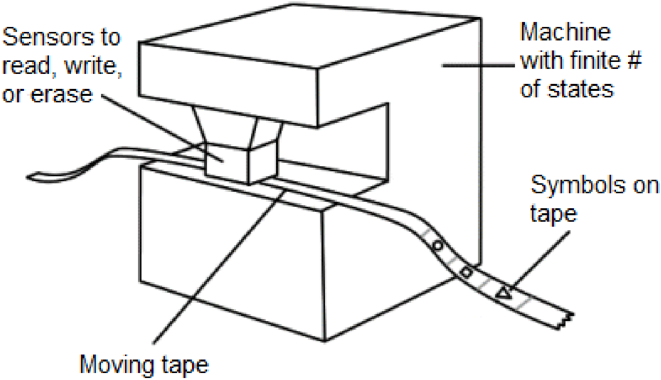
Standing on the shoulders of giants





Turing machine

A Turing machine is a theoretical generalized computer, composed of a tape on which symbols representing instructions are imprinted. The tape can move backwards and forwards in the machine, which can read the intructions and write the result-ant output back onto the tape.



Turing machine

- Turing machine is a powerful model
  - Anything a real computer can compute, a Turing machine can compute it
- A computational model is known to be Turing complete if it can simulate a Turing machine
- RNN is known to be Turing complete (Siegelmann et al. 95)
- But our end goal is not to have neural networks to replace our computers
  - We like to have neural networks to replace our programmers
  - Key idea: turn neural networks into a differentiable neural computer by giving them read-write access to external memory

Turing machine

- Turing machine is a powerful model
  - Anything a real computer can compute, a Turing machine can compute it
- A computational model is known to be Turing complete if it can simulate a Turing machine
- RNN is known to be Turing complete (Siegelmann et al. 95)
- But our end goal is not to have neural networks to replace our computers
  - We like to have neural networks to replace our programmers
  - Key idea: turn neural networks into a differentiable neural computer by giving them read-write access to external memory

Turing machine

- Turing machine is a powerful model
  - Anything a real computer can compute, a Turing machine can compute it
- A computational model is known to be Turing complete if it can simulate a Turing machine
- RNN is known to be Turing complete (Siegelmann et al. 95)
- But our end goal is not to have neural networks to replace our computers
  - We like to have neural networks to replace our programmers
  - Key idea: turn neural networks into a differentiable neural computer by giving them read-write access to external memory

Turing machine

- Turing machine is a powerful model
  - Anything a real computer can compute, a Turing machine can compute it
- A computational model is known to be Turing complete if it can simulate a Turing machine
- RNN is known to be Turing complete (Siegelmann et al. 95)
- But our end goal is not to have neural networks to replace our computers
  - We like to have neural networks to replace our programmers
  - Key idea: turn neural networks into a differentiable neural computer by giving them read-write access to external memory

Neural Turing machine (NTM)

- Can we teach a machine to write program?
  - Sure!
- A learned ”memory copy” algorithm by NTM

```
initialise: move head to start location
while input delimiter not seen do
    receive input vector
    write input to head location
    increment head location by 1
end while
return head to start location
while true do
    read output vector from head location
    emit output
    increment head location by 1
end while
```

Neural Turing machine (NTM)

- Can we teach a machine to write program?
  - Sure!
- A learned "memory copy" algorithm by NTM

initialise: move head to start location
while input delimiter not seen **do**
    receive input vector
    write input to head location
    increment head location by 1
end while
return head to start location
while true **do**
    read output vector from head location
    emit output
    increment head location by 1
end while

Neural Turing machine

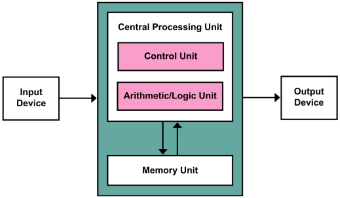
- Question: How can we train computer to write program?
- Answer: Some "random access memory" will help
- Question: To train a networks, the model has to be differentiable. But conventional way of memory addressing is not differentiable
- Answer: Soft-"addressing" (attention)

Neural Turing machine

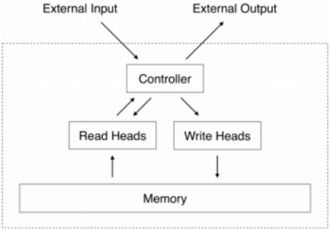
- Question: How can we train computer to write program?
- Answer: Some "random access memory" will help
- Question: To train a networks, the model has to be differentiable. But conventional way of memory addressing is not differentiable
- Answer: Soft-"addressing" (attention)

Neural Turing machine

- Question: How can we train computer to write program?
- Answer: Some “random access memory” will help
- Question: To train a networks, the model has to be differentiable. But conventional way of memory addressing is not differentiable
- Answer: Soft-“addressing” (attention)



$\Rightarrow$



Reading from memory

- Consider $\mathbf{M}_t$, a $M \times N$ matrix, as a memory block just like RAM in conventional computer system
- Unlike our PCs, we don't read from a particular location
- We read to all locations at the same time

$$\mathbf{r}_t \leftarrow [w_t(1), w_t(2), \cdots, w_t(N)] \begin{bmatrix} \mathbf{M}_t(1) \\ \mathbf{M}_t(2) \\ \cdots \\ \mathbf{M}_t(N) \end{bmatrix},$$

where $\sum_i w_t(i) = 1$

- Note that this addressing model is differentiable and hence is trainable

Reading from memory

- Consider $\mathbf{M}_t$, a $M \times N$ matrix, as a memory block just like RAM in conventional computer system
- Unlike our PCs, we don't read from a particular location
- We read to all locations at the same time

$$\mathbf{r}_t \leftarrow [w_t(1), w_t(2), \cdots, w_t(N)] \begin{bmatrix} \mathbf{M}_t(1) \\ \mathbf{M}_t(2) \\ \cdots \\ \mathbf{M}_t(N) \end{bmatrix},$$

where $\sum_i w_t(i) = 1$

- Note that this addressing model is differentiable and hence is trainable

Reading from memory

- Consider $\mathbf{M}_t$, a $M \times N$ matrix, as a memory block just like RAM in conventional computer system
- Unlike our PCs, we don't read from a particular location
- We read to all locations at the same time

$$\mathbf{r}_t \leftarrow [w_t(1), w_t(2), \cdots, w_t(N)] \begin{bmatrix} \mathbf{M}_t(1) \\ \mathbf{M}_t(2) \\ \cdots \\ \mathbf{M}_t(N) \end{bmatrix},$$

where $\sum_i w_t(i) = 1$

- Note that this addressing model is differentiable and hence is trainable

Writing to memory

Writing to memory is split into two separate steps:

Erase

$$\tilde{\mathbf{M}}_t(i) \leftarrow \mathbf{M}_{t-1}(i) \odot [\mathbf{1} - w_t(i)\mathbf{e}_t],$$

where $\odot$ is element-wise multiplication

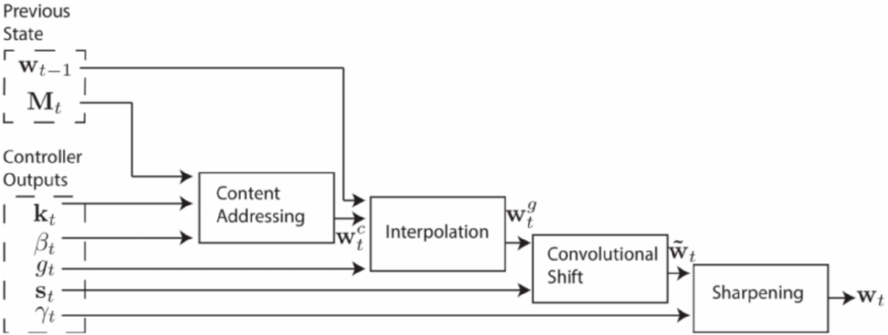
Add

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

- $\mathbf{e}_t$: Erase vector
- $\mathbf{a}_t$: Add vector

Addressing mechanisms

How to pick and update addressing weight $\mathbf{w}_t$?



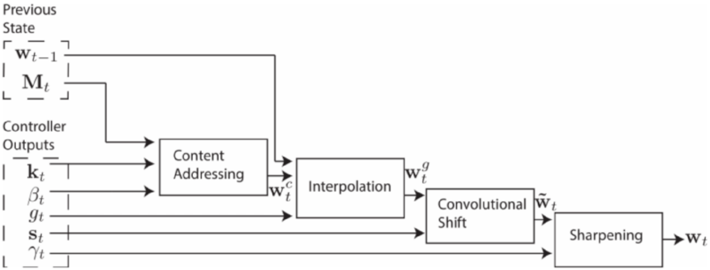
Content-based addressing

Pick address locations that matches with an input key $\mathbf{k}_t$

$$w_t^c(i) \leftarrow \frac{\exp\left(\beta_t K[\mathbf{k}_t, \mathbf{M}_t(i)]\right)}{\sum_j \exp\left(\beta_t K[\mathbf{k}_t, \mathbf{M}_t(j)]\right)},$$

where

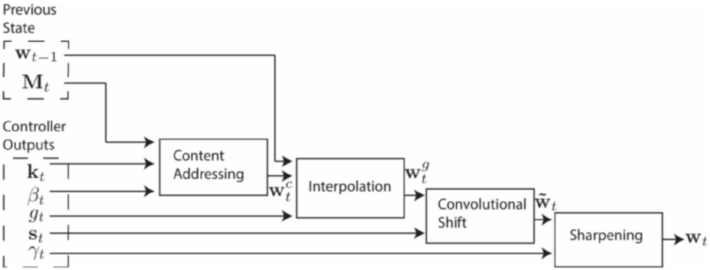
- $K[\mathbf{u}, \mathbf{v}] = \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\cdot\|\mathbf{v}\|}$ is a similarity measure (cosine similarity)
- $\mathbf{k}_t$ is a length-M key vector
- β_t is a key strength parameter



Interpolation gate (*addressing inertia*)

$$\mathbf{w}_t^g \leftarrow g_t \mathbf{w}_t^c + (1 - g_t)\mathbf{w}_{t-1},$$

where g_t is called the *interpolation gate* in the original paper

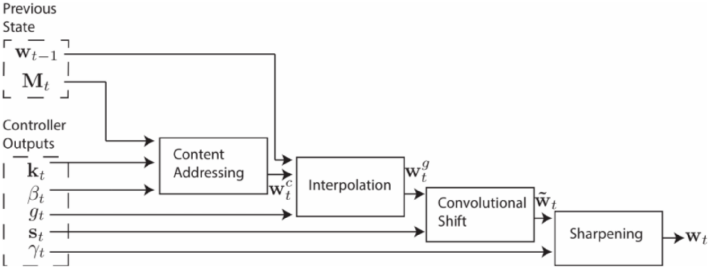


Circular convolution (*spreading*)

Perturb to diversify the target addresses

$$\tilde{w}_t(i) \leftarrow \sum_{j=0}^{N-1} w_t^g(j) s_t(i-j),$$

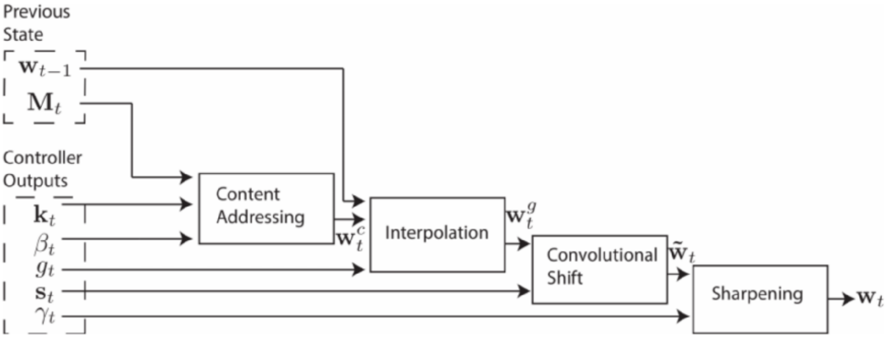
where $s_t(\cdot)$ is also called *shift weighting*. E.g., $s_t(\Delta) = \begin{cases} 0.1 & \text{if } \Delta = -1 \\ 0.8 & \text{if } \Delta = 0 \\ 0.1 & \text{if } \Delta = 1 \\ 0 & \text{otherwise} \end{cases}$



Sharpening

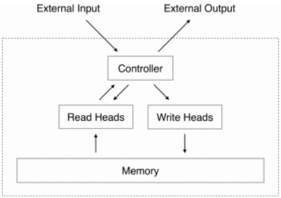
$$w_t(i) \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}},$$

where $\gamma_t > 1$ and this operation counteracts the blurring effect of the last step



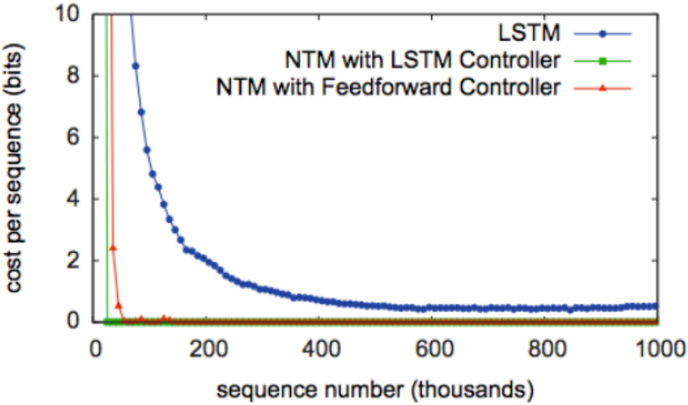
Experiments

- Test NTM's ability to learn simple algorithms like copying and sorting
- Demonstrate that solutions generalize well beyond the range of training
- Tested with three architectures
  - NTM with feed forward controller
  - NTM with LSTM controller
  - Standard LSTM network



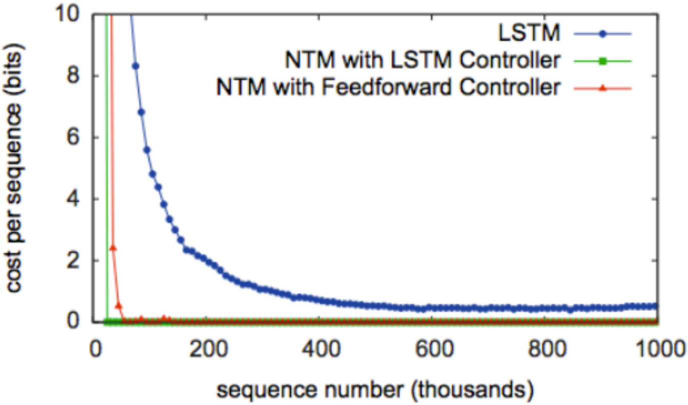
Experiment 1: Memory block copy

- Tests whether NTM can store and retrieve data
  - Trained to copy sequences of **8** bit vectors
  - The input sequence is followed by a delimiter
- Sequences vary between **1**−**20** vectors
  - Trained to copy up to **20** consecutive vectors



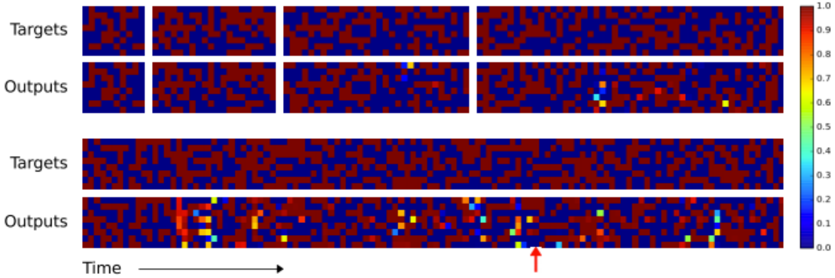
Experiment 1: Memory block copy

- Tests whether NTM can store and retrieve data
  - Trained to copy sequences of **8** bit vectors
  - The input sequence is followed by a delimiter
- Sequences vary between **1**–**20** vectors
  - Trained to copy up to **20** consecutive vectors



Experiment 1: Memory block copy

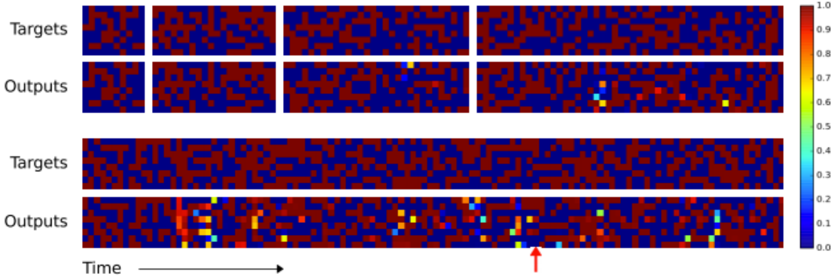
NTM



- Text vector lengths: 10, 20, 30, and 50
- Generalized well
- A "synchronization" (duplication) error at the red arrow. But overall subjectively similar to the targets

Experiment 1: Memory block copy

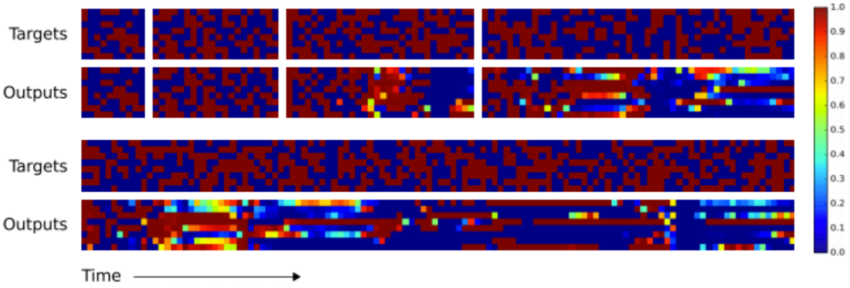
NTM



- Text vector lengths: 10, 20, 30, and 50
- Generalized well
- A "synchronization" (duplication) error at the red arrow. But overall subjectively similar to the targets

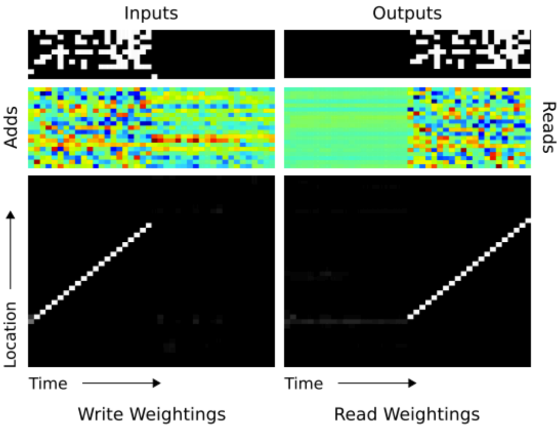
Experiment 1: Memory block copy

LSTM



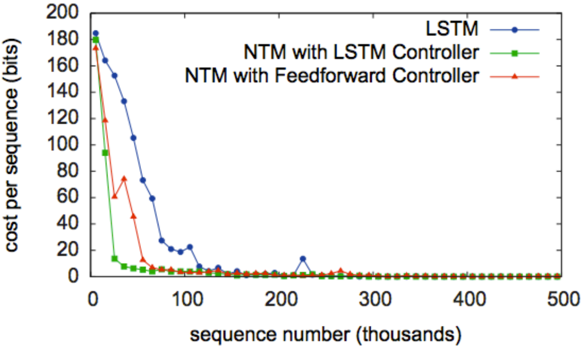
- Fail to generalize

Experiment 1: Memory block copy



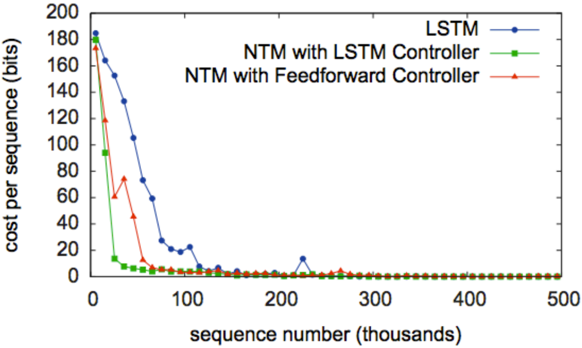
Experiment 2: Repeat memory copy

- Tests whether NTM can learn simple nested function
- Extend copy by repeatedly copying input specified number of times
- Training is a random length sequence of **8** bit binary inputs plus a scalar value for # of copies (both randomly chosen from $\mathbf{1}-\mathbf{10}$)

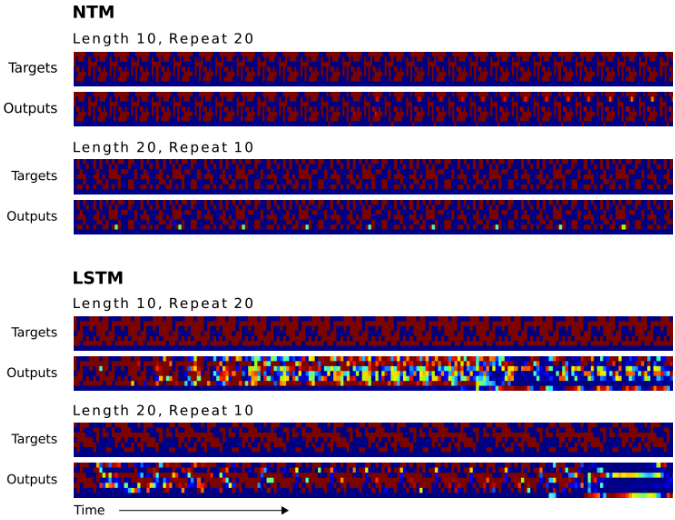


Experiment 2: Repeat memory copy

- Tests whether NTM can learn simple nested function
- Extend copy by repeatedly copying input specified number of times
- Training is a random length sequence of **8** bit binary inputs plus a scalar value for # of copies (both randomly chosen from **1**−**10**)

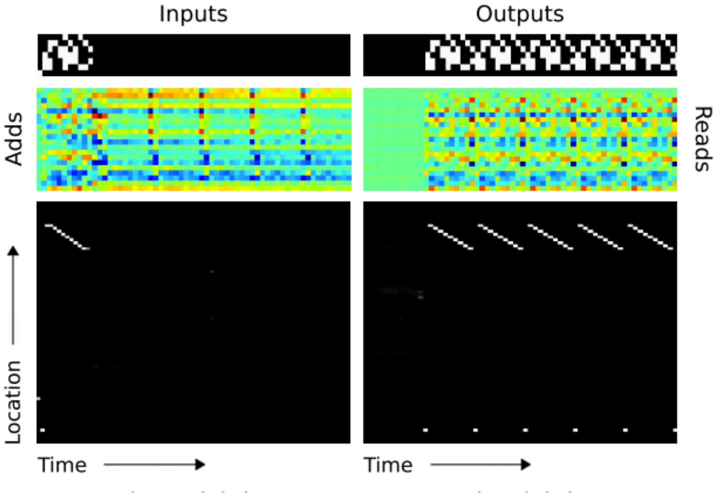


Experiment 2: Repeat memory copy



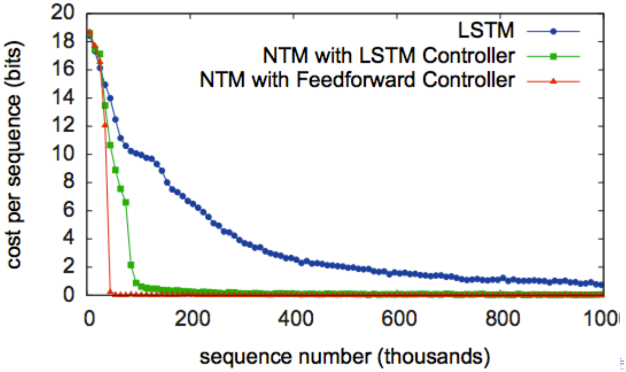
LSTM fails to generalize

Experiment 2: Repeat memory copy



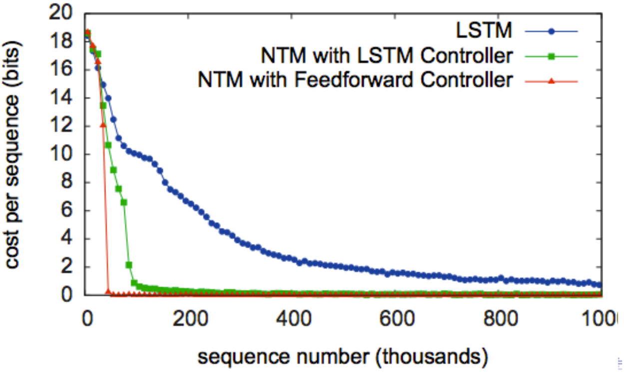
Experiment 3: Associative recall

- Tests NTM's ability to associate data references
- Training input is list of items, followed by a query item
- Output is subsequent item in list
- Each item is a **three sequence 6-bit** binary vector
- Each 'episode' has between two and **six** items

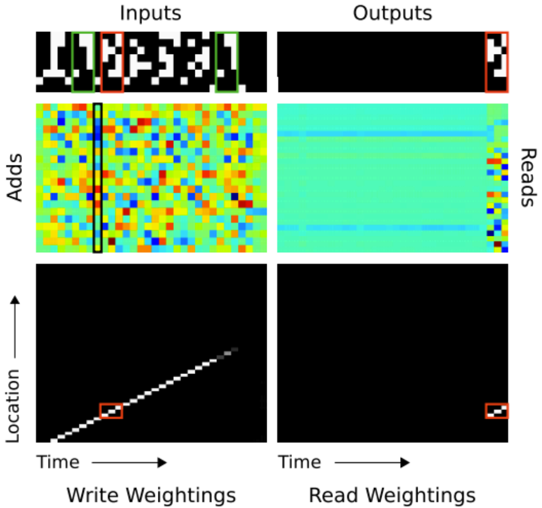


Experiment 3: Associative recall

- Tests NTM's ability to associate data references
- Training input is list of items, followed by a query item
- Output is subsequent item in list
- Each item is a **three sequence 6-bit** binary vector
- Each 'episode' has between two and **six** items



Experiment 3: Associative recall



Experiment 3: Associative recall

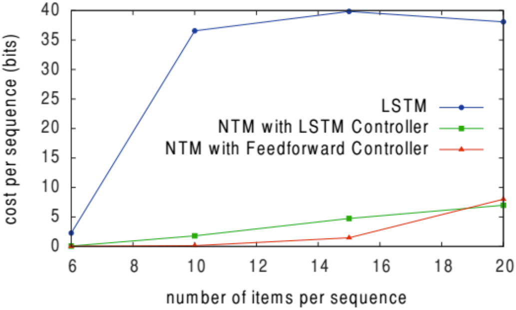


Figure 11: Generalisation Performance on Associative Recall for Longer Item Sequences. The NTM with either a feedforward or LSTM controller generalises to much longer sequences of items than the LSTM alone. In particular, the NTM with a feedforward controller is nearly perfect for item sequences of twice the length of sequences in its training set.

Experiment 4: Dynamic N-Grams

- Test whether NTM could rapidly adapt to new predictive distributions
- Trained on 6-gram binary pattern on **200** sequences
- See if an NTM can learn the optimal estimator (Murphy 2012)

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

Experiment 4: Dynamic N-Grams

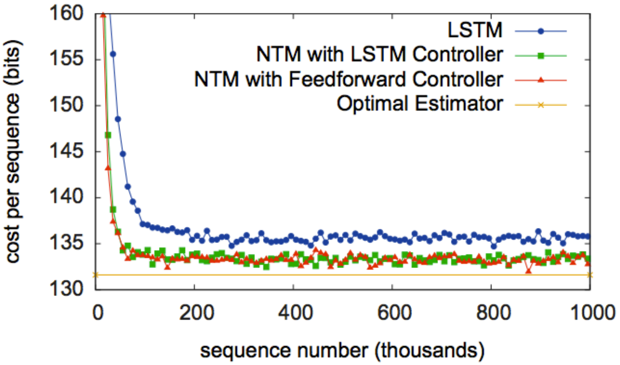
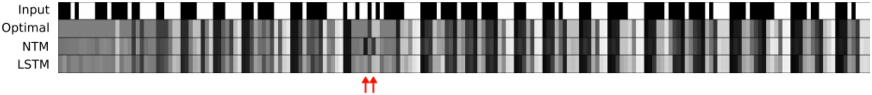


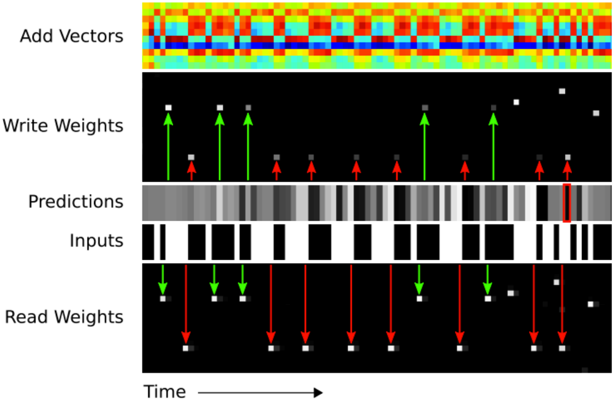
Figure 13: Dynamic N-Gram Learning Curves.

Experiment 4: Dynamic N-Grams



Error occur at the locations indicated by the red arrows

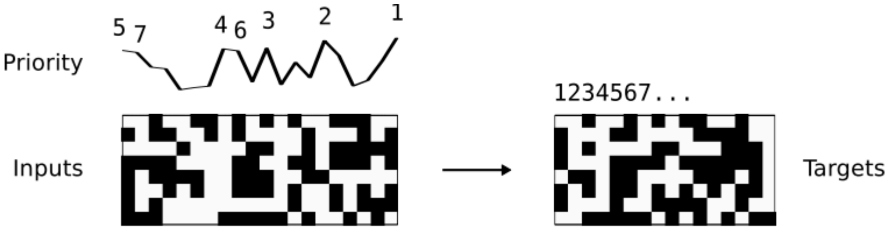
Experiment 4: Dynamic N-Grams



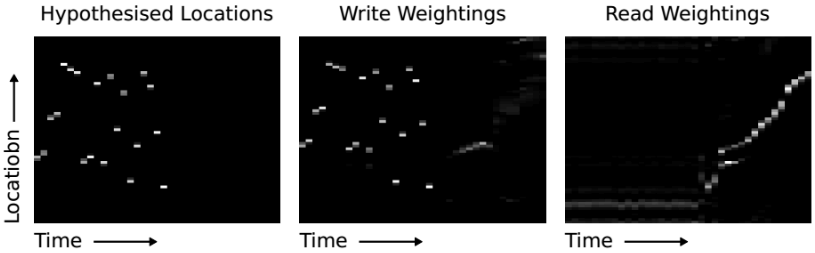
Green and red arrows correspond to places where controller trying to access locations for contexts 00010 and 01111

Experiment 5: Priority sort

- Tests whether NTM can sort data
- Input is sequence of **20** random binary vectors, each with a scalar rating drawn from $[-1, 1]$
- Target sequence is **16** highest priority vectors

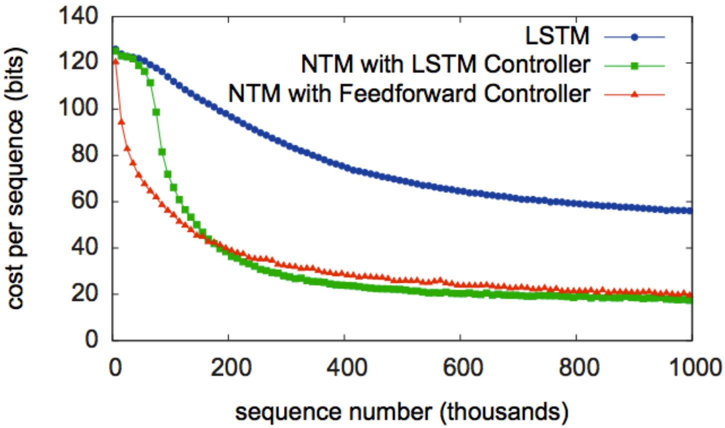


Experiment 5: Priority sort



NTM seems to use priority to determine the relative location of each write

Experiment 5: Priority sort



Experiment parameters

- RMSProp algorithm
- Momentum 0.9
- All LSTM's had three stacked hidden layers

Experiment parameters

Task	#Heads	Controller Size	Memory Size	Learning Rate	#Parameters
Copy	1	100	128×20	10^{-4}	$17,162$
Repeat Copy	1	100	128×20	10^{-4}	$16,712$
Associative	4	256	128×20	10^{-4}	$146,845$
N-Grams	1	100	128×20	3×10^{-5}	$14,656$
Priority Sort	8	512	128×20	3×10^{-5}	$508,305$

Table 1: NTM with Feedforward Controller Experimental Settings

Experiment parameters

Task	#Heads	Controller Size	Memory Size	Learning Rate	#Parameters
Copy	1	100	128×20	10^{-4}	$67,561$
Repeat Copy	1	100	128×20	10^{-4}	$66,111$
Associative	1	100	128×20	10^{-4}	$70,330$
N-Grams	1	100	128×20	3×10^{-5}	$61,749$
Priority Sort	5	2×100	128×20	3×10^{-5}	$269,038$

Table 2: NTM with LSTM Controller Experimental Settings

Experiment parameters

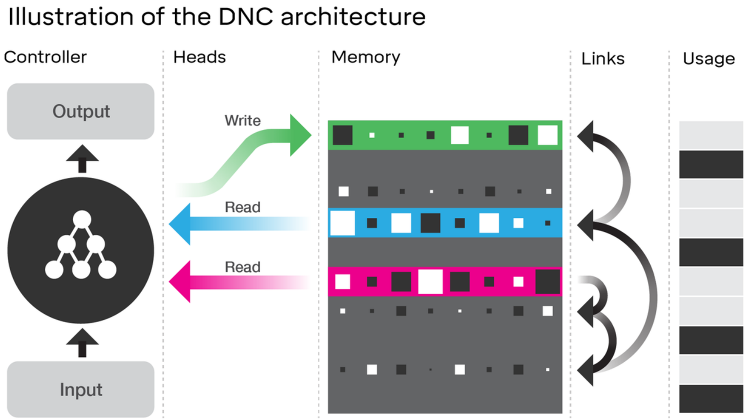
Task	Network Size	Learning Rate	#Parameters
Copy	3×256	3×10^{-5}	$1,352,969$
Repeat Copy	3×512	3×10^{-5}	$5,312,007$
Associative	3×256	10^{-4}	$1,344,518$
N-Grams	3×128	10^{-4}	$331,905$
Priority Sort	3×128	3×10^{-5}	$384,424$

Table 3: LSTM Network Experimental Settings

Follow-up work

- Named “Differentiable neural computer”
- Published in Graves, Alex, et al. ”Hybrid computing using a neural network with dynamic external memory.” Nature (2016)
- Check out this deepmind blog post as well

DNC architecture



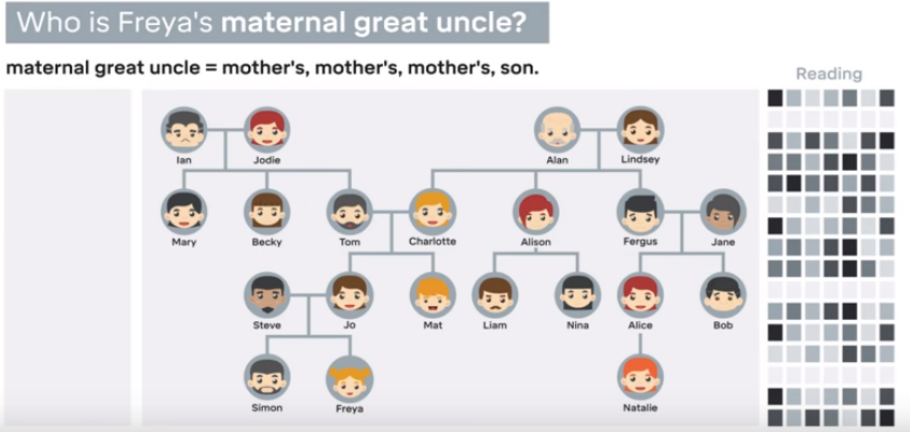
The neural network controller receives external inputs and, based on these, interacts with the memory using read and write operations known as “heads”. To help the controller navigate the memory, DNC stores “temporal links” to keep track of the order things were written in, and records the current “usage” level of each memory location

Memory addressing

Addressing is based on three different distinct forms of attention mechanisms

1. Content lookup: matching key with memory just as in neural Turing machine
2. Temporal link: provide mechanism for head to iterate through the memories in the order they were written (or written)
3. Memory "usage monitoring": DNC keep a "free list" tracking the usage of memory allocation

Family tree puzzle



Conclusions

- NTM is a neural net architecture with external memory that is differentiable end-to-end
- Experiments demonstrate that NTMs are capable of learning simple algorithms and are capable of generalizing beyond training regime
- DNC improves NTM mainly on the memory management part
  - Able to free up unused memory
  - Avoid overwrite useful memory