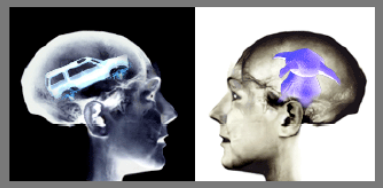


UCSD

Temporal Dynamics of Learning Center

CSE 253
Neural Networks: The Neural Turing Machine

Gary Cottrell

Week 8 Lecture 1

But first...some clicker questions

1. The main idea behind LSTM units is:
   A. To use a weird acronym so we'll all get confused – Long Short Term Memory??
   B. To create a mechanism so error can get propagated very far back in time – the "Constant Error Carousel"
   C. To allow a network to store values for a long period of time, until needed
   D. To use "gates" to control memory
   E. B, C, and D.

But first...some clicker questions

1. The main idea behind LSTM units is:
   A. To use a weird acronym so we'll all get confused – Long Short Term Memory??
   B. To create a mechanism so error can get propagated very far back in time – the "Constant Error Carousel"
   C. To allow a network to store values for a long period of time, until needed
   D. To use "gates" to control memory
   E. **B, C, and D.**

But first...some clicker questions

1. Which of the following is true about BPTT?
   A. The weight update at a certain time step depends only on the most recent time step
   B. Weight matrices at all the distinct unrolled time steps are updated independently using their respective gradients
   C. The weight update at a certain time step depends on multiple time steps within a given time horizon
   D. For a particular weight matrix, the gradients at all time steps are added and there is one common weight update performed
   E. Both (c) and (d)

But first...some clicker questions

1. Which of the following is true about BPTT?
   A. The weight update at a certain time step depends only on the most recent time step
   B. Weight matrices at all the distinct unrolled time steps are updated independently using their respective gradients
   C. The weight update at a certain time step depends on multiple time steps within a given time horizon
   D. For a particular weight matrix, the gradients at all time steps are added and there is one common weight update performed
   E. **Both (c) and (d)**

But first...some clicker questions

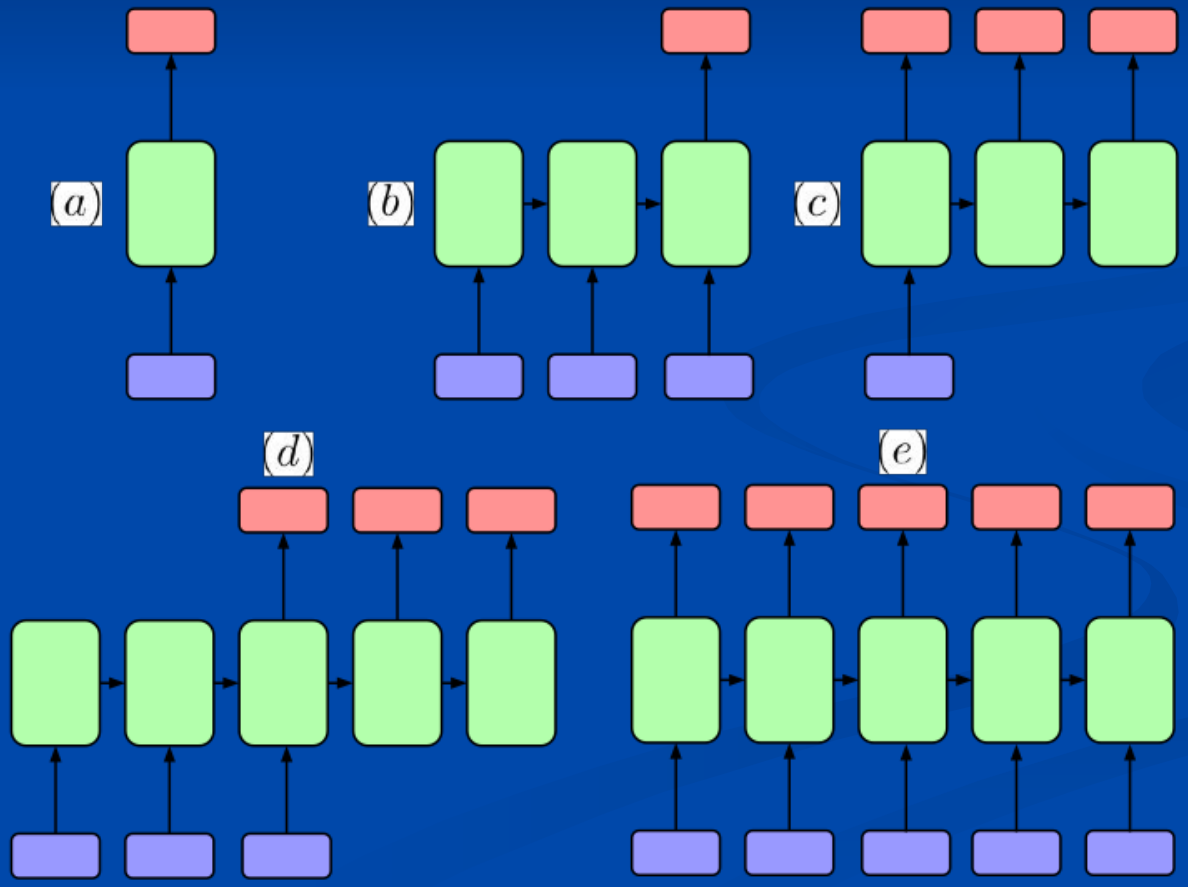
2. Recurrent networks can be used to
   A. Caption images
   B. Translate from English to French
   C. Understand speech
   D. A & B
   E. A, B, and C

But first...some clicker questions

2. Recurrent networks can be used to
   A. Caption images
   B. Translate from English to French
   C. Understand speech
   D. A & B
   E. **A, B, and C**

But first...some clicker questions

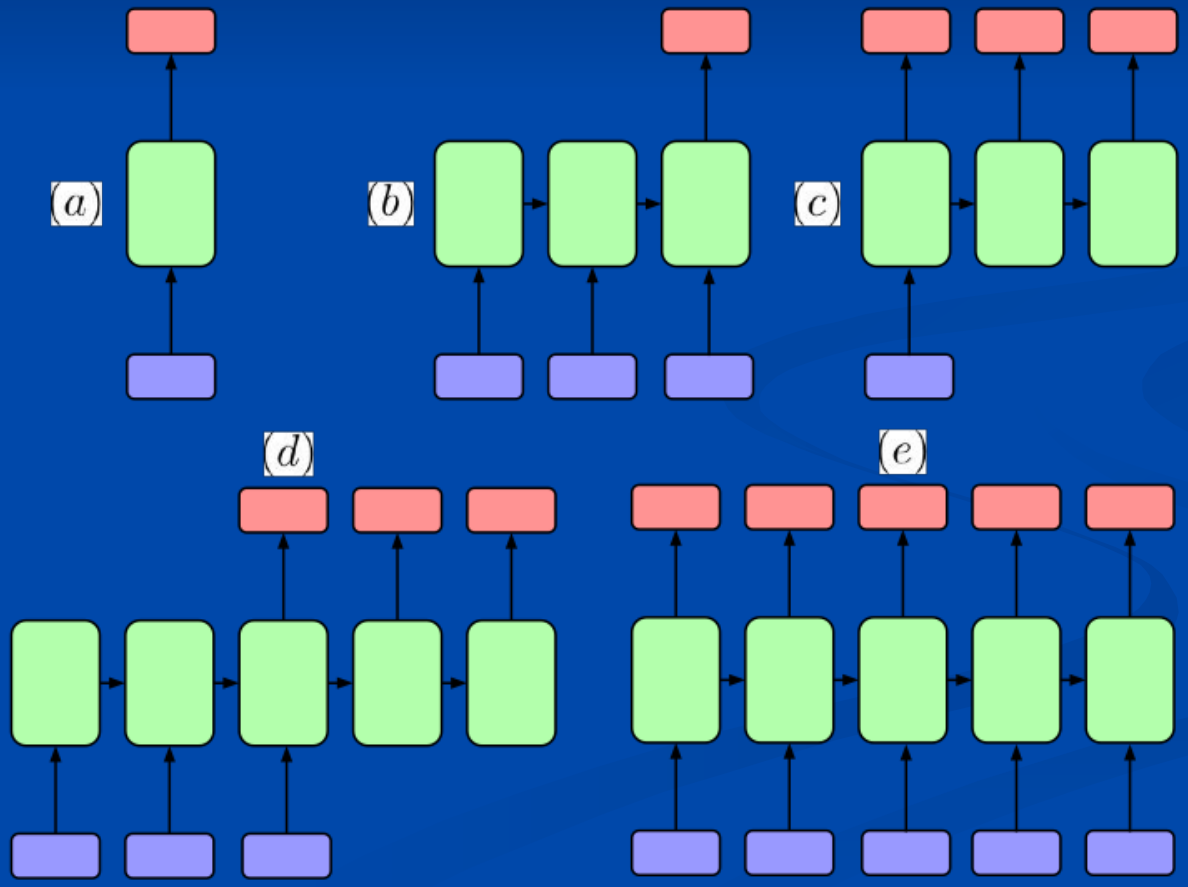
3. Which of the following would be used language translation (after training)?



But first...some clicker questions

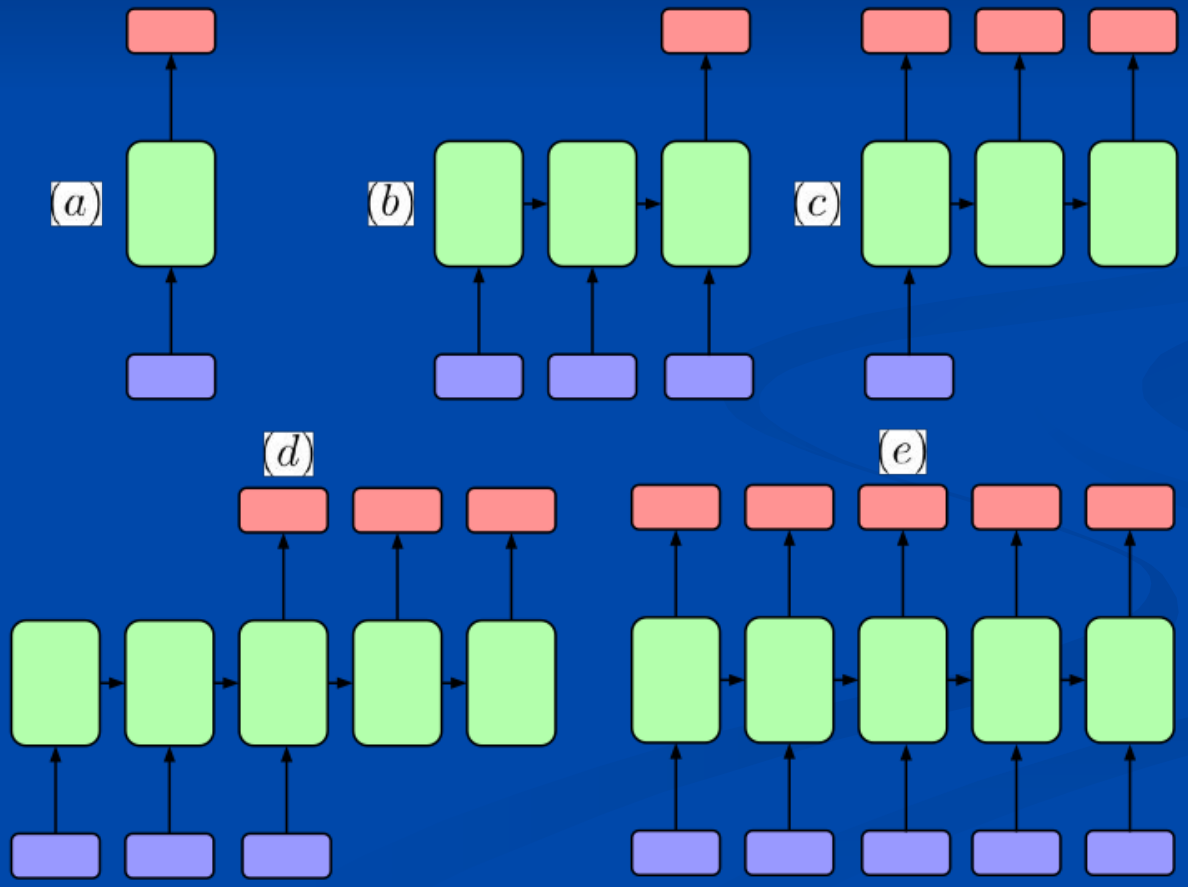
3. Which of the following would be used language translation (after training)?

Answer: D



But first...some clicker questions

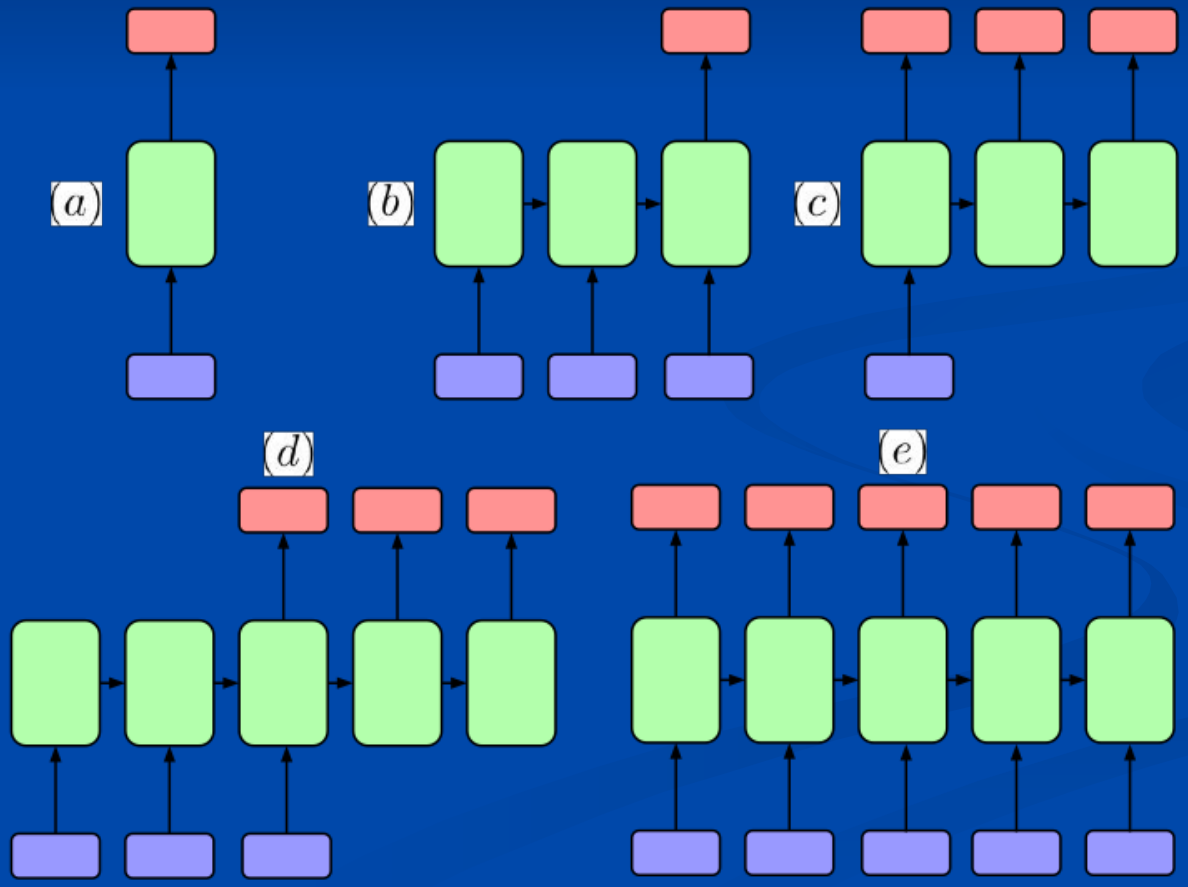
4. Which of the following were used for beer mind?



But first...some clicker questions

4. Which of the following were used for beer mind?

Answer: E

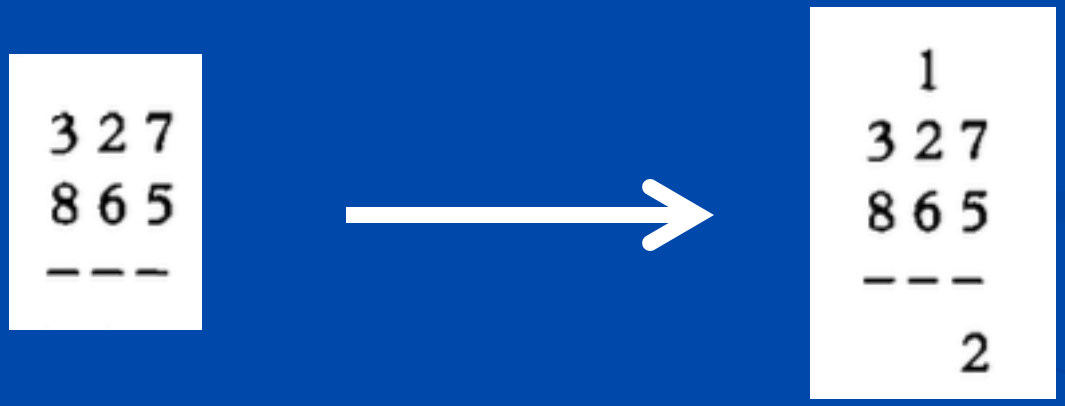


Introduction

- Neural nets have been shown to be Turing-equivalent (Seigelmann & Sontag)
- But can they *learn* programs?
- Yes!

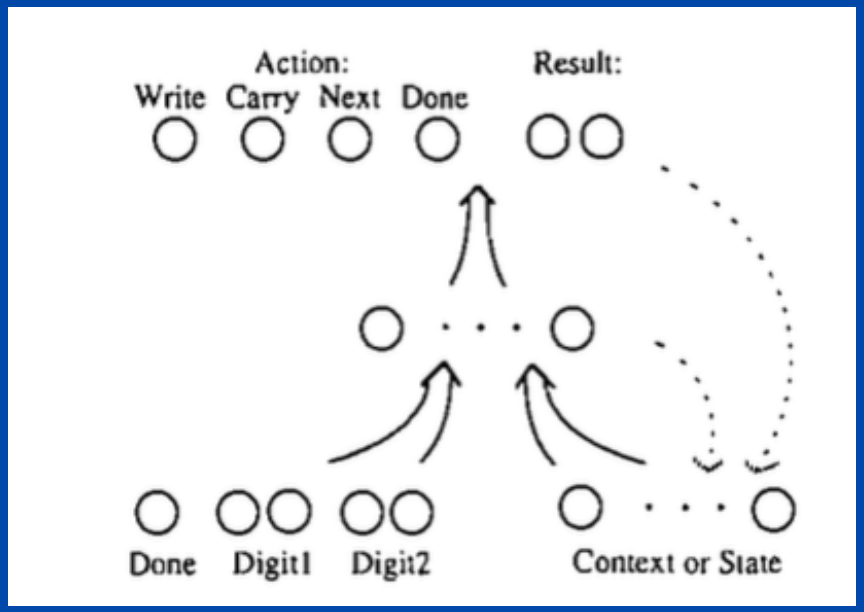
Introduction

- One can explicitly teach a neural net to implement a program – e.g., Tsung & Cottrell (1993), e.g., sequential addition:



Introduction

- One can explicitly teach a neural net to implement a program – e.g. Tsung & Cottrell (1993)

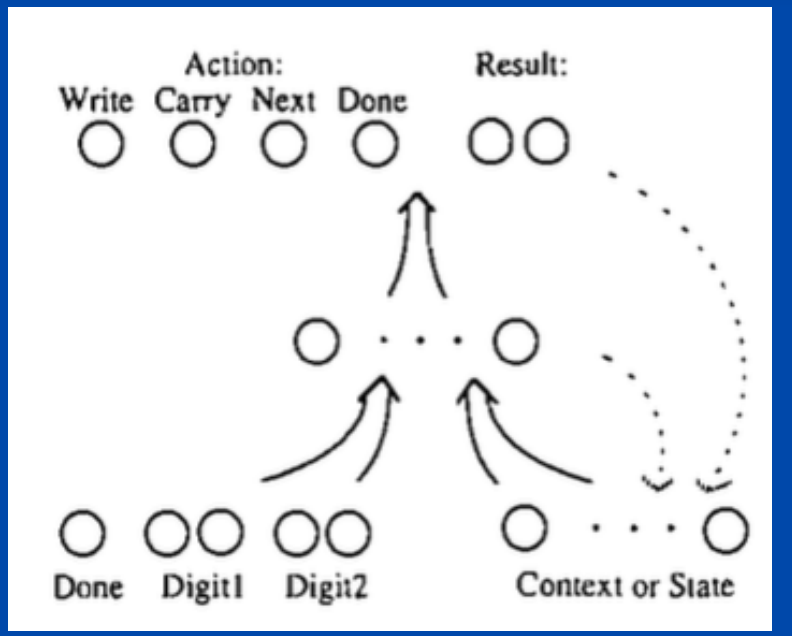


PROGRAM 1

```
while not done do
begin
   output(WRITE, low_order_digit);
   if sum > radix then
      output(CARRY, ???);
   output(NEXT, ???);
end
if carry_on_previous_input then
        output(WRITE, '01');
output(DONE, ???);
```

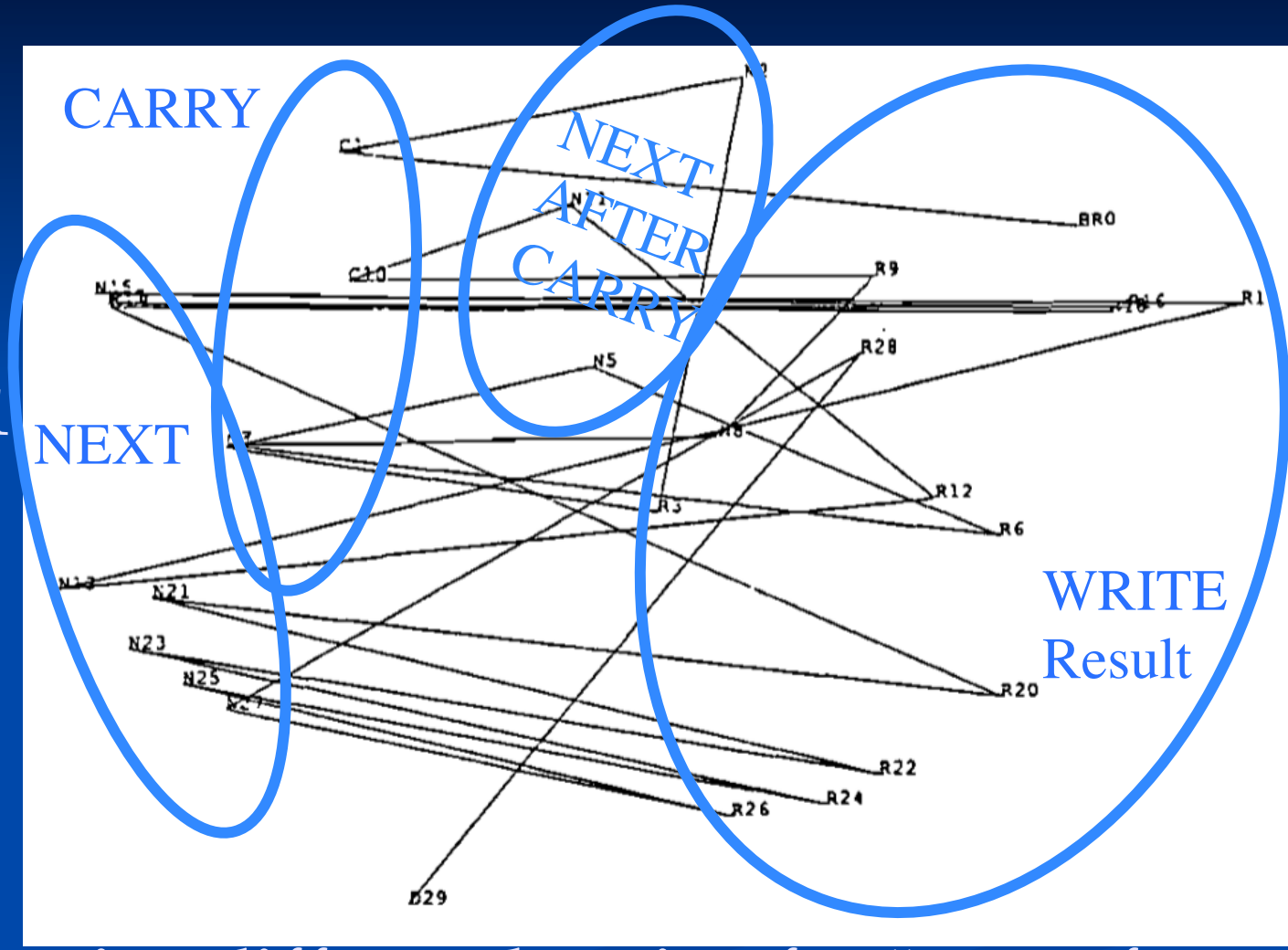
Introduction

- One can explicitly teach a neural net to implement a program – e.g. Tsung & Cottrell (1993)



Input			Output	
			Action	Result
3	2	⑦	1. Write	8
4	8	①	2. Next	—
3	②	7	3. Write	0
4	⑧	1	4. Carry	—
			5. Next	—
③	2	7	6. Write	8
④	8	1	7. Next	—
3	2	7	8. Done	—
4	8	1		

The internal state space: Principal components 1 vs 2 of the hidden unit activations during a 30 step addition



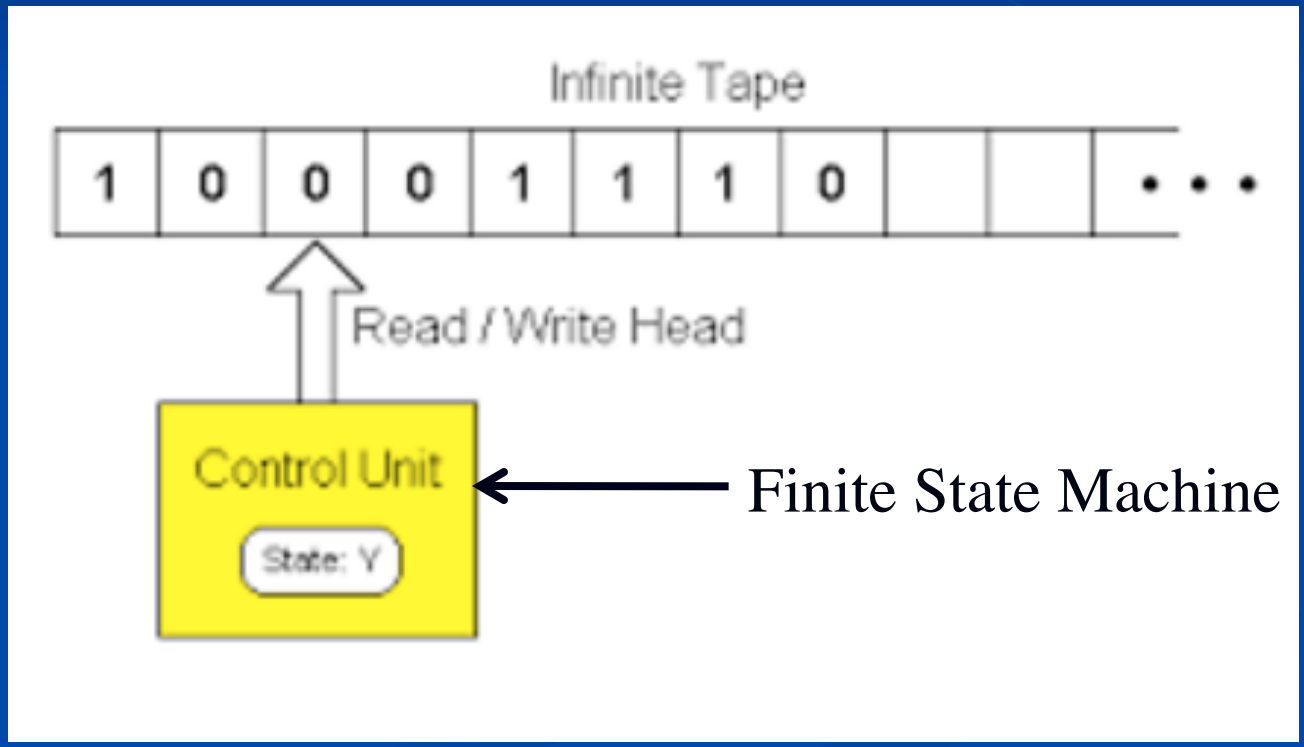
Note that there is a different location for "Next after Carry" – because 1 has to be added to the next Result

Introduction

- One can explicitly teach a neural net to implement a program – e.g. Tsung & Cottrell (1993)
- That's nice, but:
  - The network had to *learn* to remember things in its internal state space – when it had to remember *one* extra bit, it took 5,000 more epochs!
  - We told the model exactly what action to take on every step
- Can a network learn what to do just by seeing input-output examples?

The Neural Turing Machine

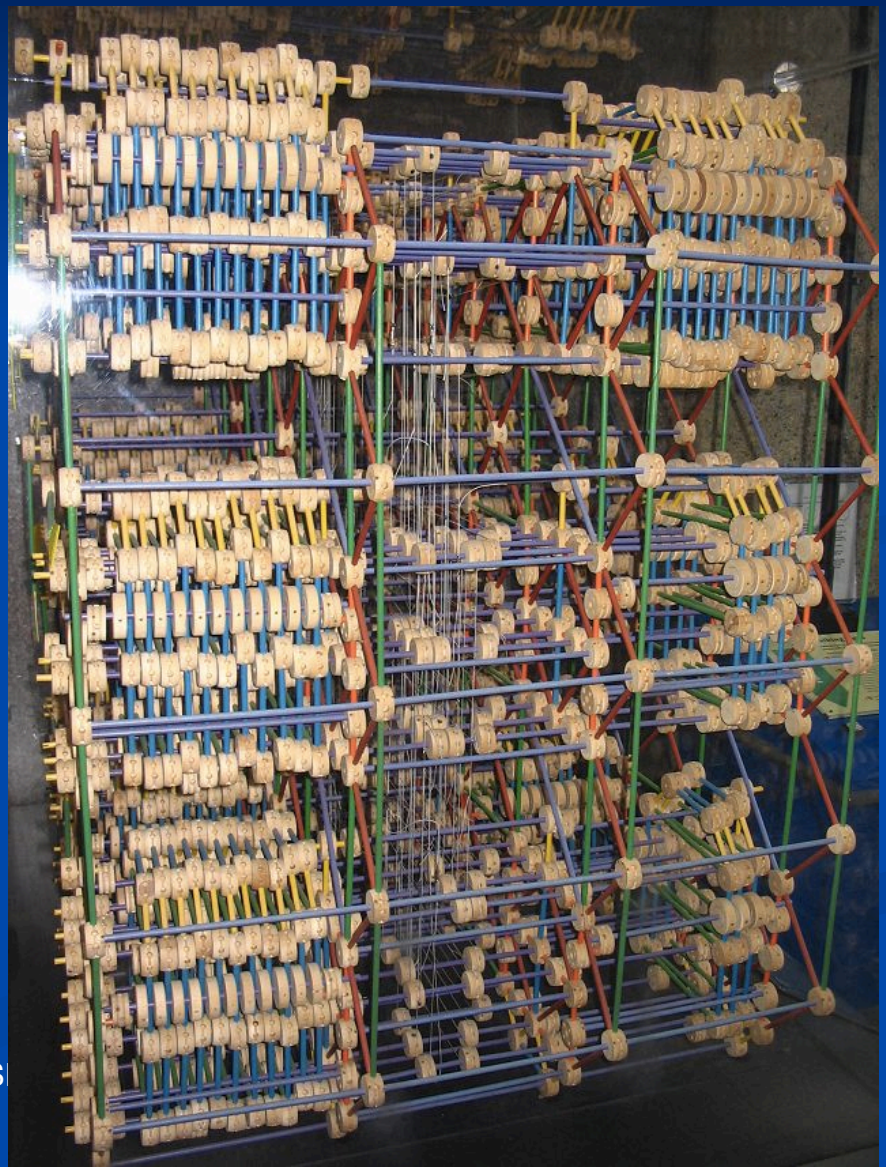
A Turing Machine has a finite state controller, and a tape that it can read from and write to.



The Neural Turing Machine

Can we build one of these out of a neural net?

The idea seems like it might be similar to this one: a Turing Machine made from tinker toys that plays tic-tac-toe

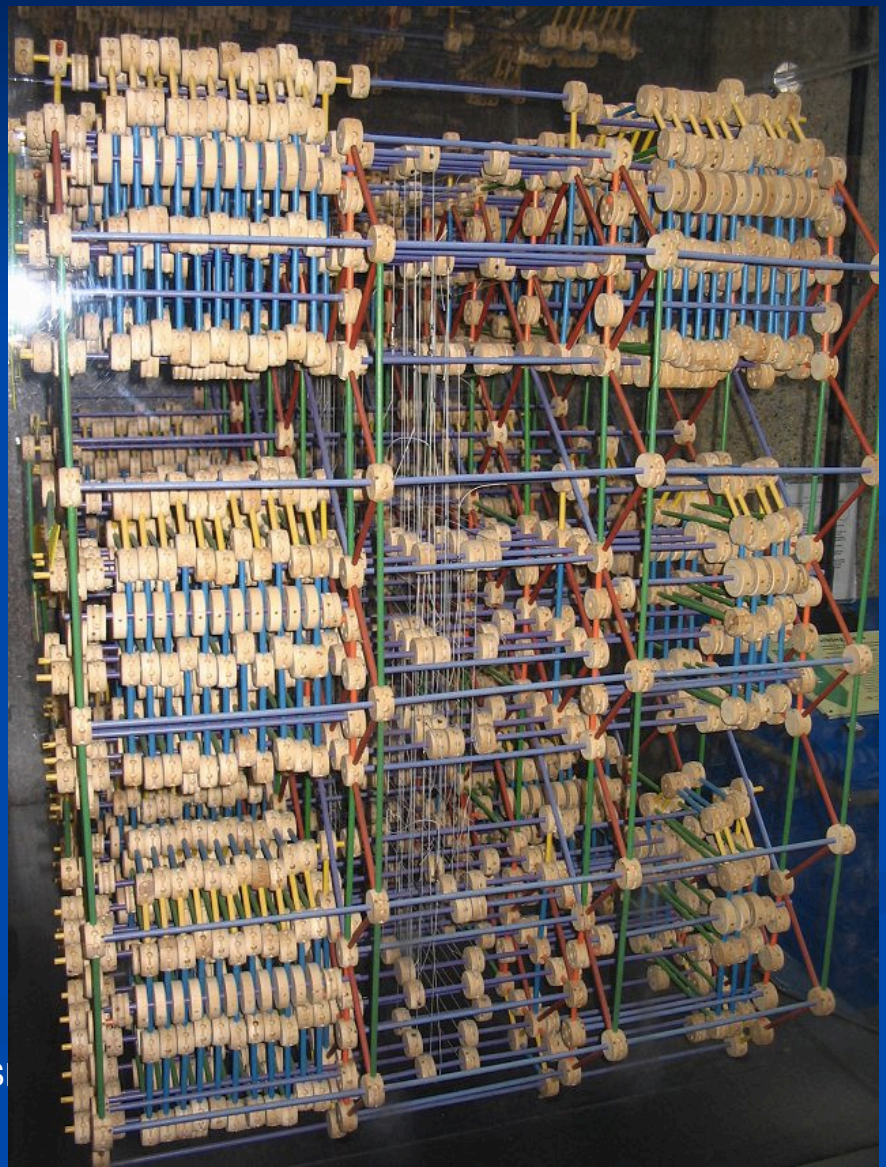


The Neural Turing Machine

It's a nice party trick, but totally impractical.

Does an NTM have to be the same way?

No!!

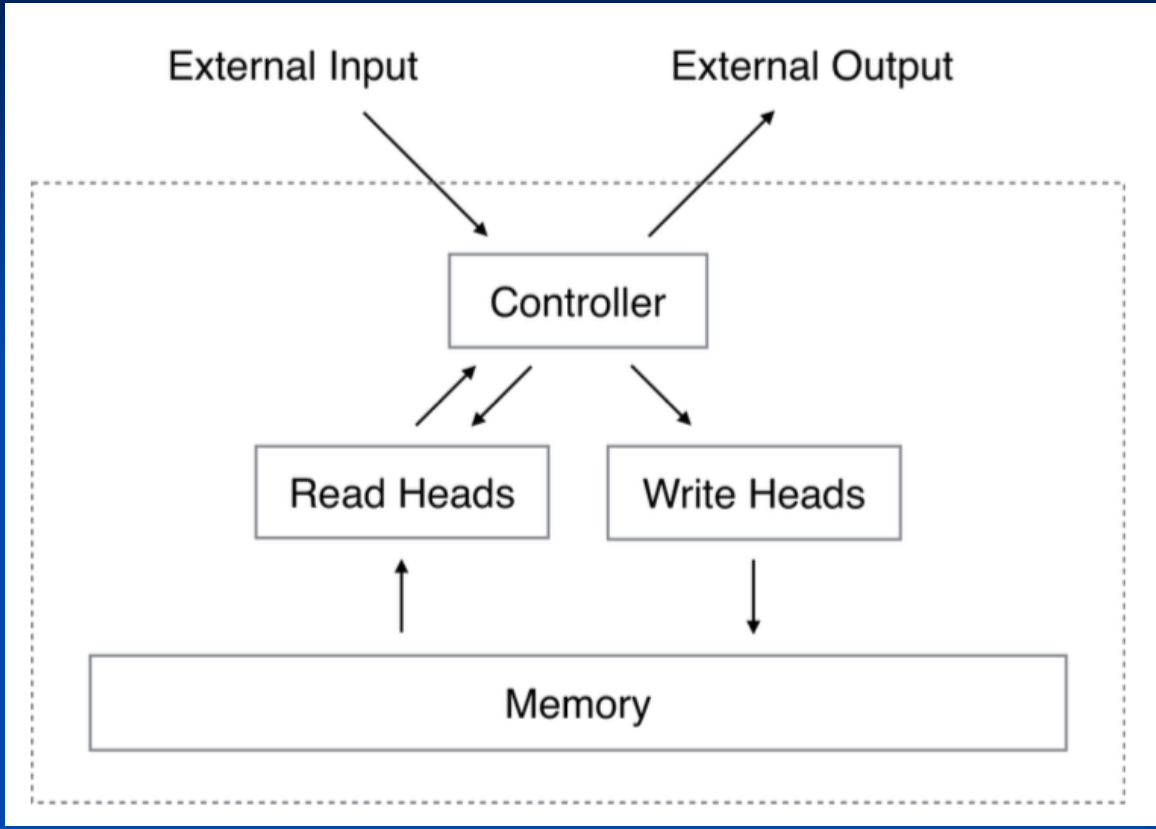


The Neural Turing Machine

The Main Idea:

Add a structured memory to a neural controller that it can write to and read from.

The Neural Turing Machine

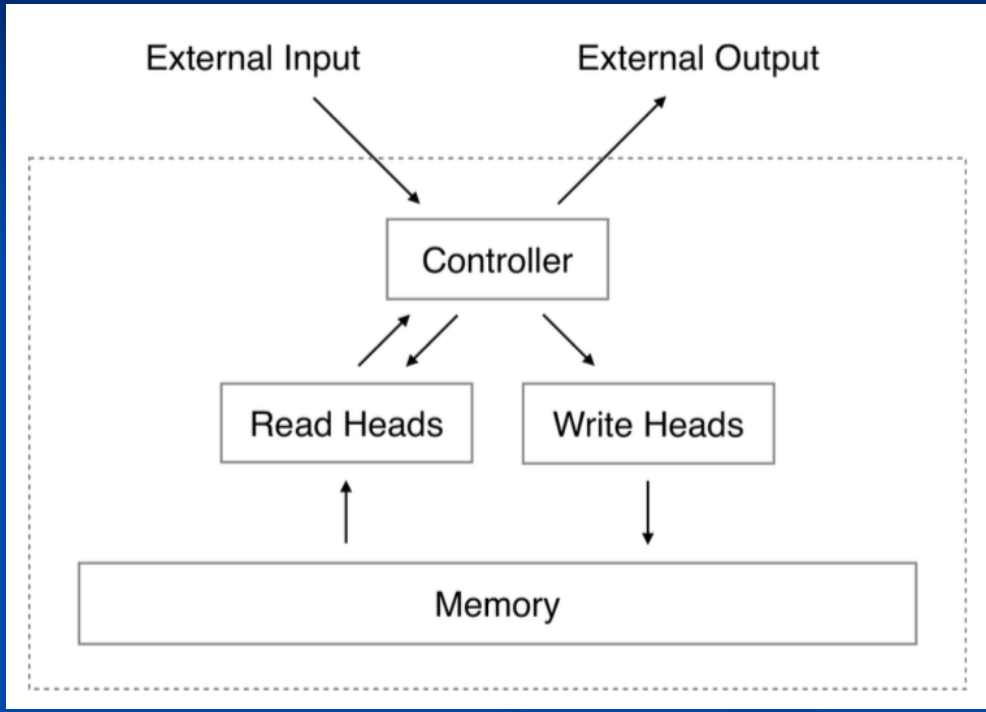


This is completely differentiable from end to end – so it can be trained by BPTT!

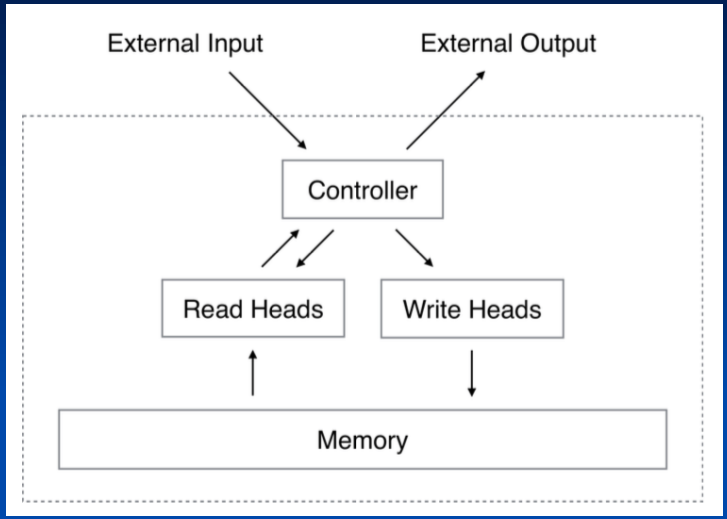
The Neural Turing Machine

The controller can be a feed-forward network or a recurrent one – with LSTM units.

The recurrent one works better.



The Neural Turing Machine



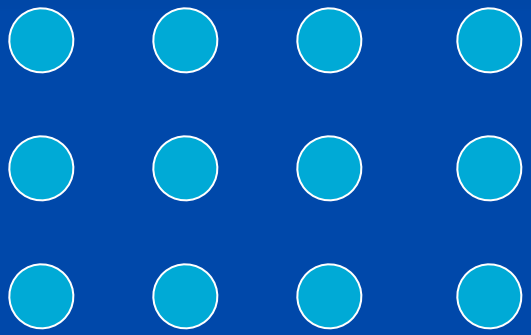
The read and write heads are reasonably simple – the *addressing* mechanism, not so much.

The addressing mechanism allows both content-addressable and location addressing (the normal kind)

It is a highly structured system, but differentiable.

The Neural Turing Machine

The memory is simply a matrix of linear neurons.

M = 

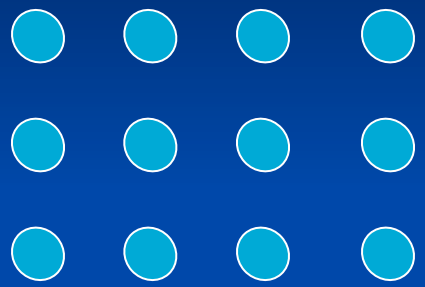
Think of each row as a "word" of memory

We will read and write a row at a time.

So this memory has 3 words of length 4.

The Neural Turing Machine

The memory is simply a matrix of linear neurons.

M = 

Indexing the rows as i=1,2,3, then we can read from this using:

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

Where $\mathbf{r}_t$ is the vector read out from the memory at time t, and w_t is a *softmax column vector* of length 3, which is the address!

Read Example

$$M = \begin{matrix} 3 & 1 & 4 & 1 \\ 5 & 9 & 9 & 7 \\ 2 & 7 & 2 & 8 \end{matrix}$$

Suppose we want the second row. If $w = (0,1,0)^T$ then

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

Yields $\mathbf{r_t}$=(5,9,9,7)

Again, note how w is the address

Read Example

$$M = \begin{matrix} 3 & 1 & 4 & 1 \\ 5 & 9 & 9 & 7 \\ 2 & 7 & 2 & 8 \end{matrix}$$

If $w = (½,½,0)^T$ then

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

Yields (4,5,6.5,4) (average of rows 1 and 2)

So you can even do a simple computation with the addressing mechanism!

Read Example

$$M = \begin{matrix} 3 & 1 & 4 & 1 \\ 5 & 9 & 9 & 7 \\ 2 & 7 & 2 & 8 \end{matrix}$$

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

Note: this expression is clearly differentiable, both with respect to M and w.

Writing to the memory

- Inspired by *gating* from LSTM networks, they use gates to write to memory.
- A *write* operation happens in two steps:
  - *Erase* memories
  - *Add* to them.

Erasing the memory

- A *write* operation happens in two steps:
  - *Erase* memories
  - *Add* to them.
- Erasing memory element i:

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

- Again, w is the address, so if $w_t(i)$ is 1 at time t, and $\mathbf{e}_t$ is an *erase vector*, then the memory is modified. ($\mathbf{e}_t$ is multiplied point-wise times the memory vector.)

Erasing the memory

- Erasing memory element *i*:

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

$$\mathrm{M} = \begin{matrix} 3 & 1 & 4 & 1 \\ 5 & 9 & 9 & 7 \\ 2 & 7 & 2 & 8 \end{matrix}$$

- If $w = (0,1,0)$, and $\mathbf{e}_t = (1,1,1,1)$ then this yields

$$\mathrm{M} = \begin{matrix} 3 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 7 & 2 & 8 \end{matrix}$$

- Obviously, more complex "erases" are possible.

Adding to the memory

- Adding to memory element *i*:

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

$$\mathrm{M} = \begin{matrix} 3 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 7 & 2 & 8 \end{matrix}$$

- If $w = (0,1,0)$, and $\mathbf{a}_t = (1,2,3,4)$ then this yields

$$\mathrm{M} = \begin{matrix} 3 & 1 & 4 & 1 \\ 1 & 2 & 3 & 4 \\ 2 & 7 & 2 & 8 \end{matrix}$$

- Question for you:
  - Why do we have to do a write in two steps?

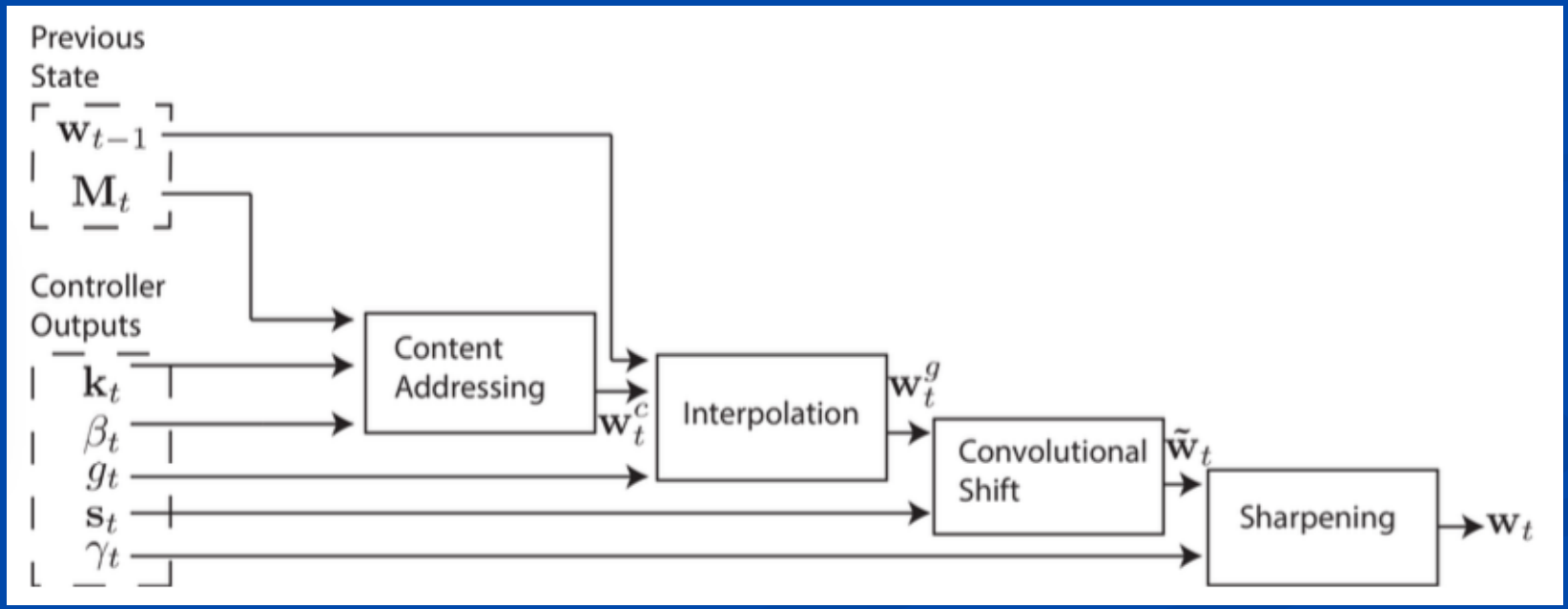
Adding to the memory

- Question for you:
  - Why do we have to do a write in two steps?

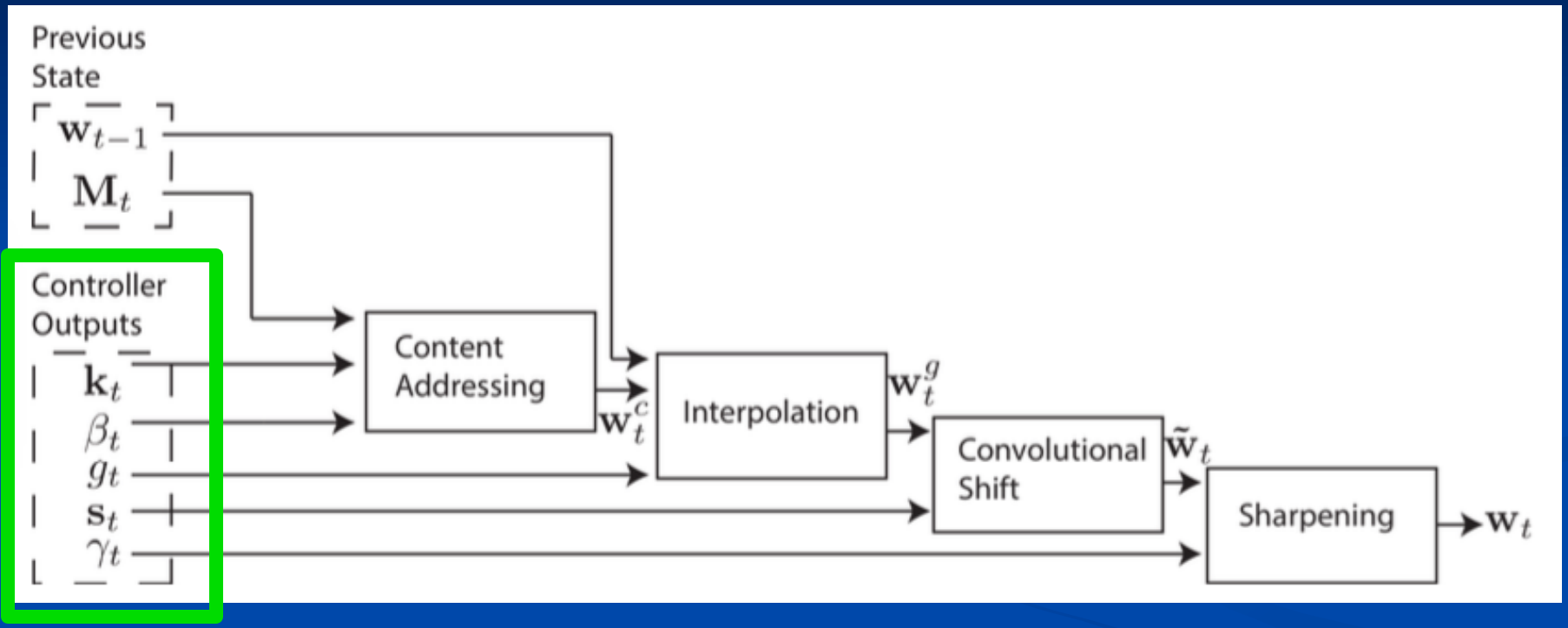
- The memory is a bunch of linear neurons:
  - There is no "overwrite" operation in arithmetic.
  - We could use an add vector that is the new vector minus the old vector, but that requires three steps! (What are they?)
  - We would have to do a *read*, compute the modification vector, and then add it

Computing w, the address

- This is a complicated mechanism, designed to allow both content-addressability as well as location-based addressing

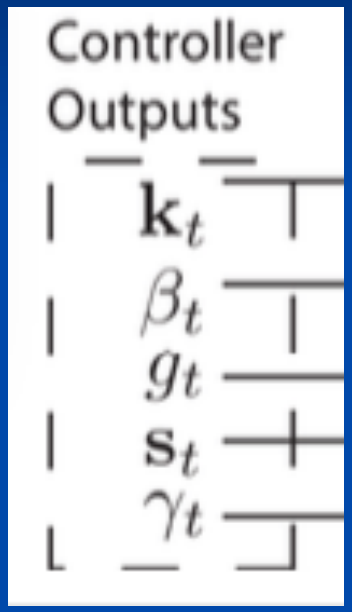


Computing w, the address



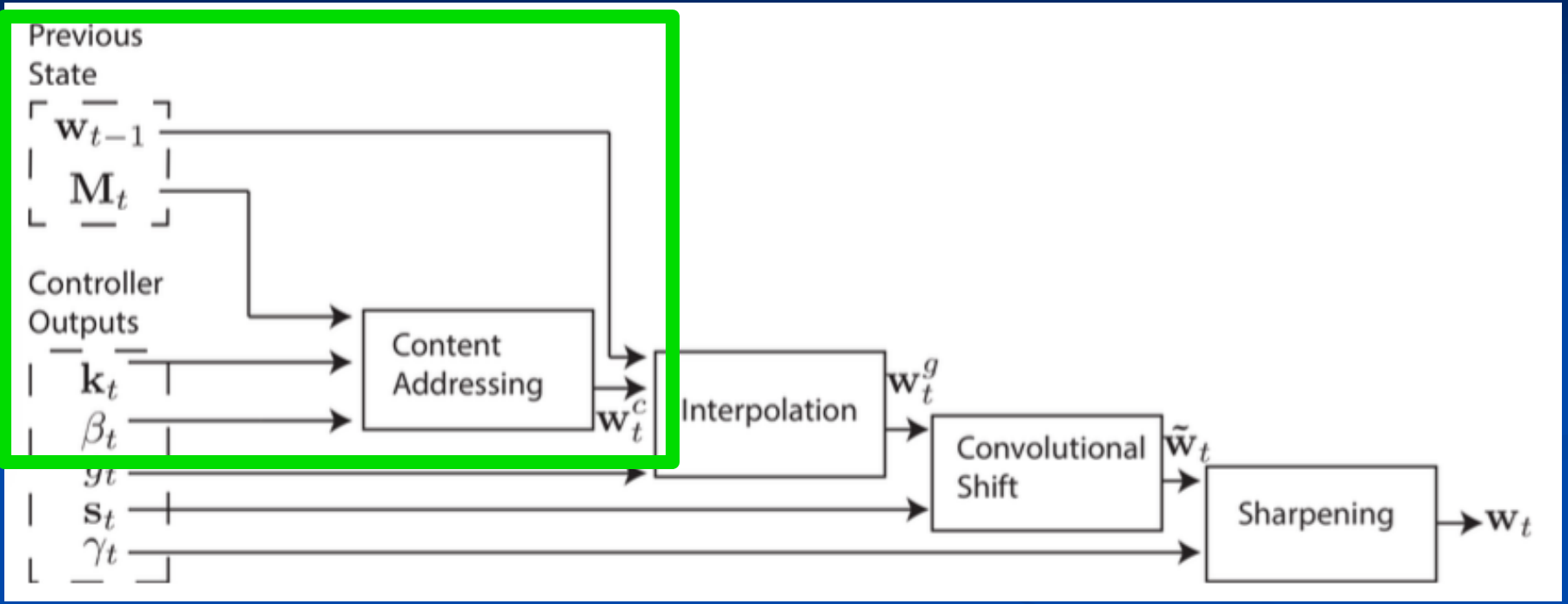
These parameters must be computed by the controller.

Computing w, the address

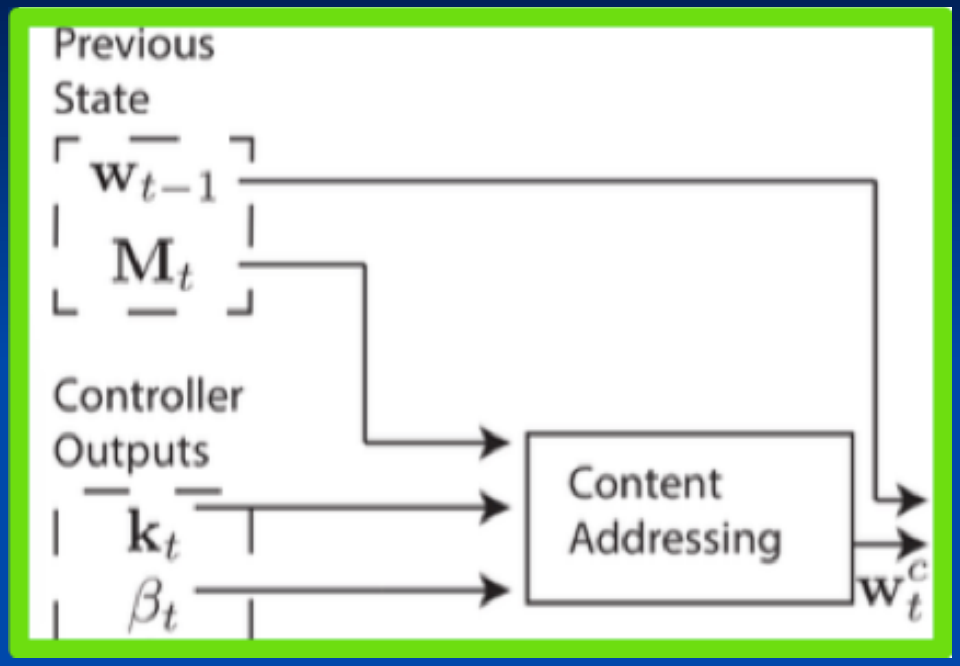


- $\mathbf{k}_t$ is a *key vector* for content-addressable memory - we want the memory element most similar to this vector (for our 3X4 memory, this would be a length 4 vector)
- β_t is a *gain parameter* on the content-match (sharpening it)
- g_t is a switch (gate) between content- and location-based addressing
- $\mathbf{s}_t$ is a *shift vector* that can increment or decrement the address, or leave it alone
- γ_t is *gain parameter* on the softmax address, making it more binary

Content-based addressing



Content-based addressing



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

- $\mathbf{k}_t$ is a *key vector* for content-addressable memory - we want the memory element most similar to this vector
- β_t is a *gain parameter* on the content-match (sharpening it)
- So, we create an address vector where each entry is based on how similar that memory element is to the key.

Where *K* is cosine similarity – so this is a softmax on the similarity

Content-based addressing

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

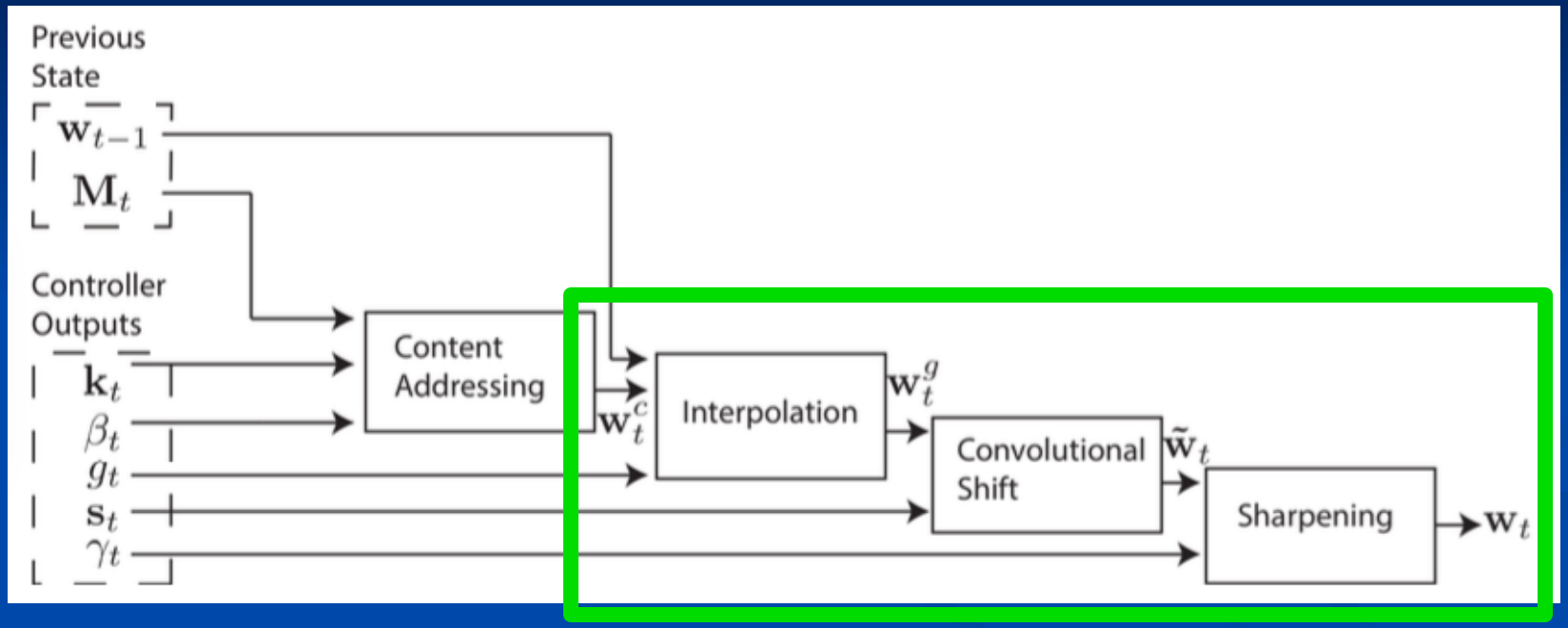
$$\mathrm{M} = \begin{bmatrix} 3 & 1 & 4 & 1 \\ 1 & 2 & 3 & 4 \\ 2 & 7 & 2 & 8 \end{bmatrix}$$

So, suppose **k** = (3,1,0,0), and suppose the match score (cosine) is 0.8, 0.1,0.1,and β=1,
this operation will produce an address vector of

$$w^c = (0.5, 0.25, 0.25)$$

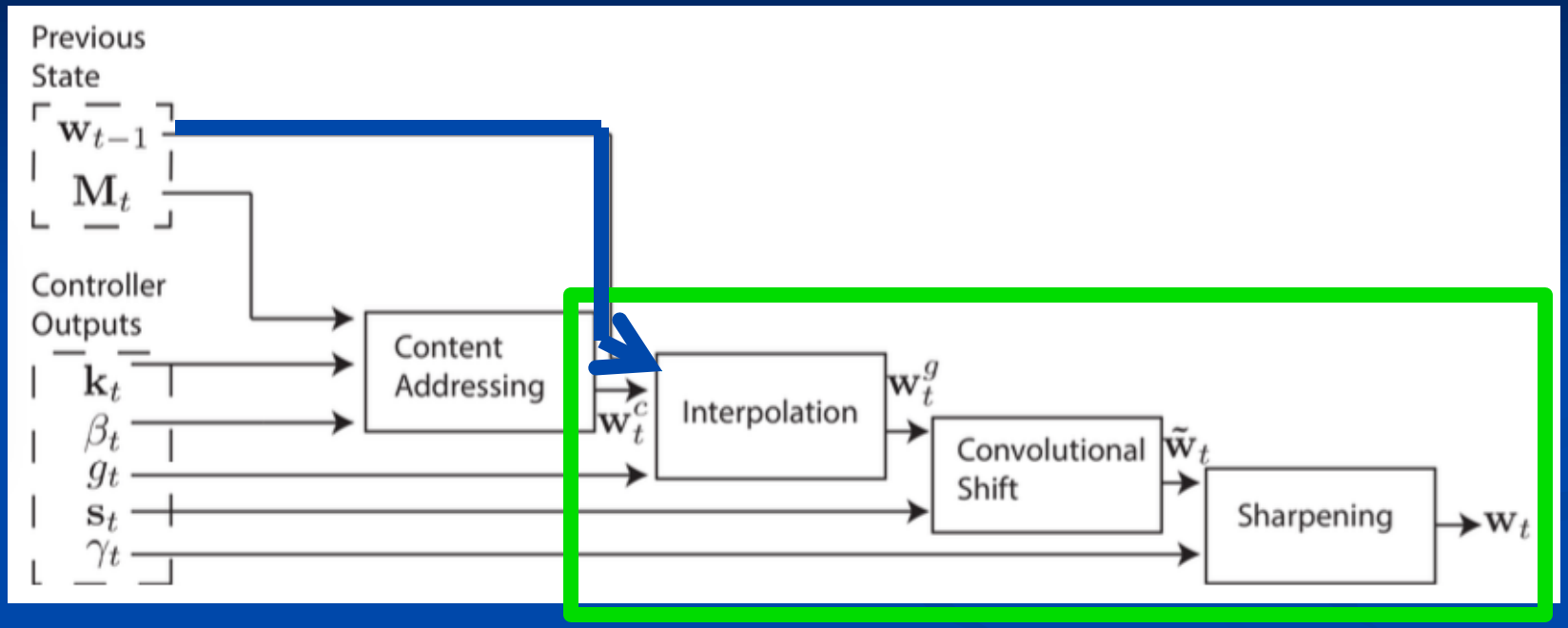
but if β=10, w^c = (0.998, 0.001, 0.001).
So a large β *sharpens* the address.

Location-based addressing



This part of the model takes in the the content-based address, and a *gate variable* g_t, which switches between the content-based addressing and the previous memory address vector (so we can increment or decrement it).

Location-based addressing

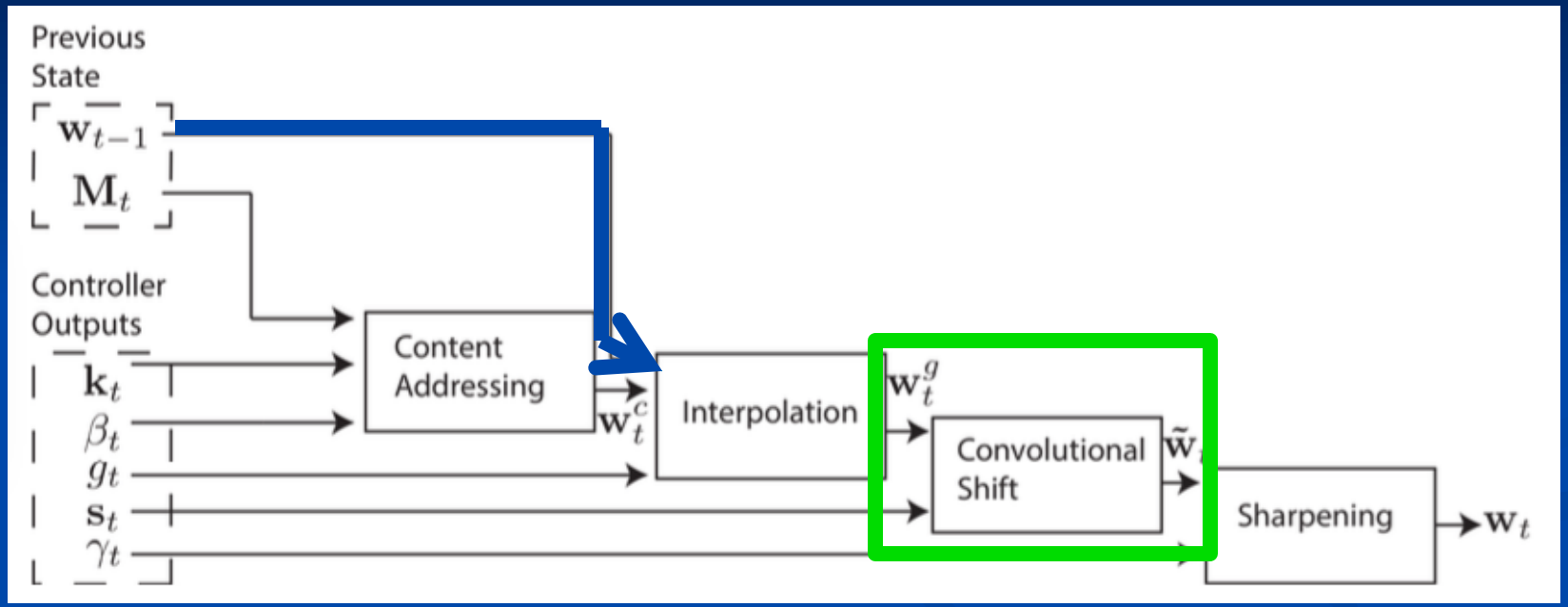


What the "Interpolation" box does: $\mathbf{w}_t^g \longleftarrow g_t \mathbf{w}_t^c + (1 - g_t)\mathbf{w}_{t-1}$

I.e., it if g_t=1, we use the content-based address, if g_t=0, we pass through the *previous* address, $\mathbf{w}_{t-1}$.

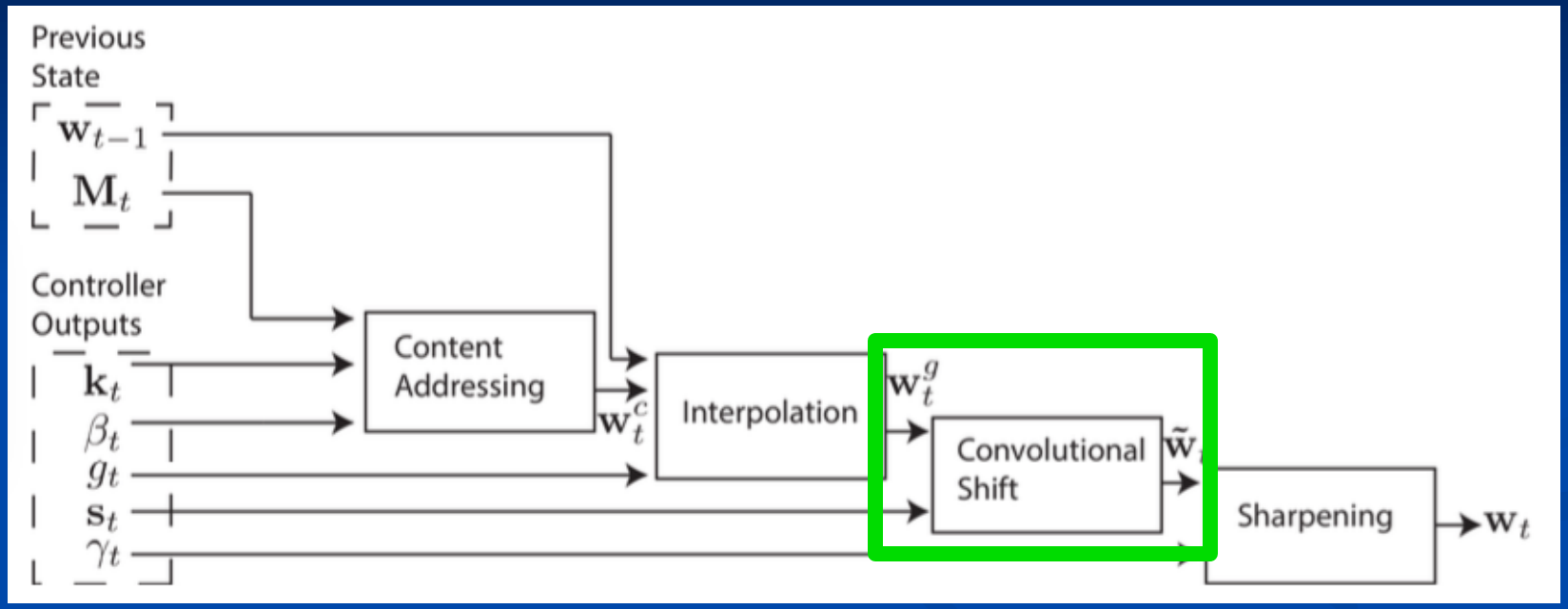
A mixture (e.g., g_t=0.5) doesn't really make sense…

Location-based addressing



The "Convolutional shift" box takes in the content-based or the location-based address, $\mathbf{w}_t^g$ and then can increment it by -1, 0, or +1, based on the $\mathbf{s}_t$ vector (larger ranges are possible)

Location-based addressing

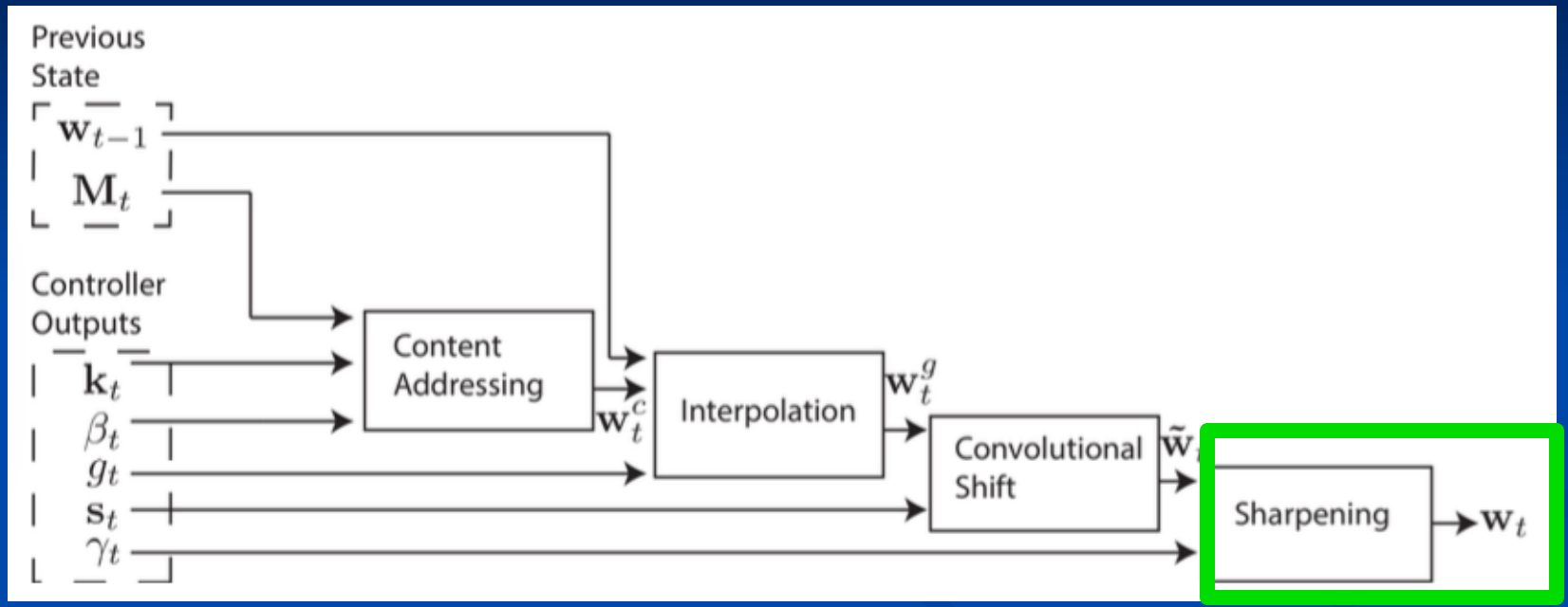


The "Convolutional shift" box operation: *circular convolution*:

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

E.g., if $\mathbf{s}_t$= (0,0,1), this increments the address by 1: (0,1,0) -> (0,0,1)

Location-based addressing



The "Sharpening" box operation again applies a high-gain softmax on the address, based on γ_t:

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

E.g., if γ=2, (.9,.1,0) -> (.99,.01,0)

Computing the address:

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

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

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

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

1. Creating a vector address based on similarity to existing memories
2. Switching between content and location
3. Incrementing or decrementing the address
4. Sharpening the address

Computing the address:

$$w_t^c(i) \longleftarrow \frac{\exp\Big(\beta_t K\big[\mathbf{k}_t, \mathbf{M}_t(i)\big]\Big)}{\sum_j \exp\Big(\beta_t K\big[\mathbf{k}_t, \mathbf{M}_t(j)\big]\Big)}$$

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

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

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

All of these operations are differentiable with respect to all of their parameters, which means we can backprop through them to change the parameters!

Ok, what can we do with this?

- Copy
- Repeated Copy
- Associative recall
- Dynamic N-grams
- Priority Sort

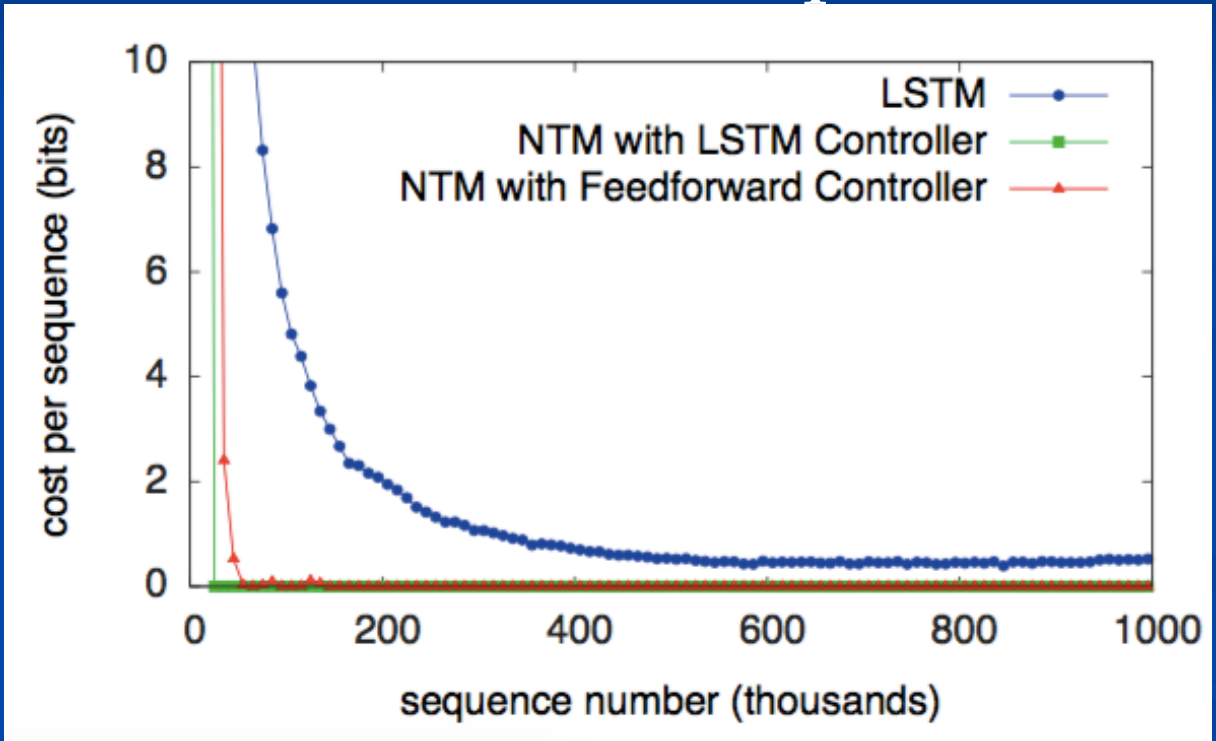
They compare a feed-forward controller, an LSTM controller, and a vanilla LSTM net on these tasks

Ok, what can we do with this?

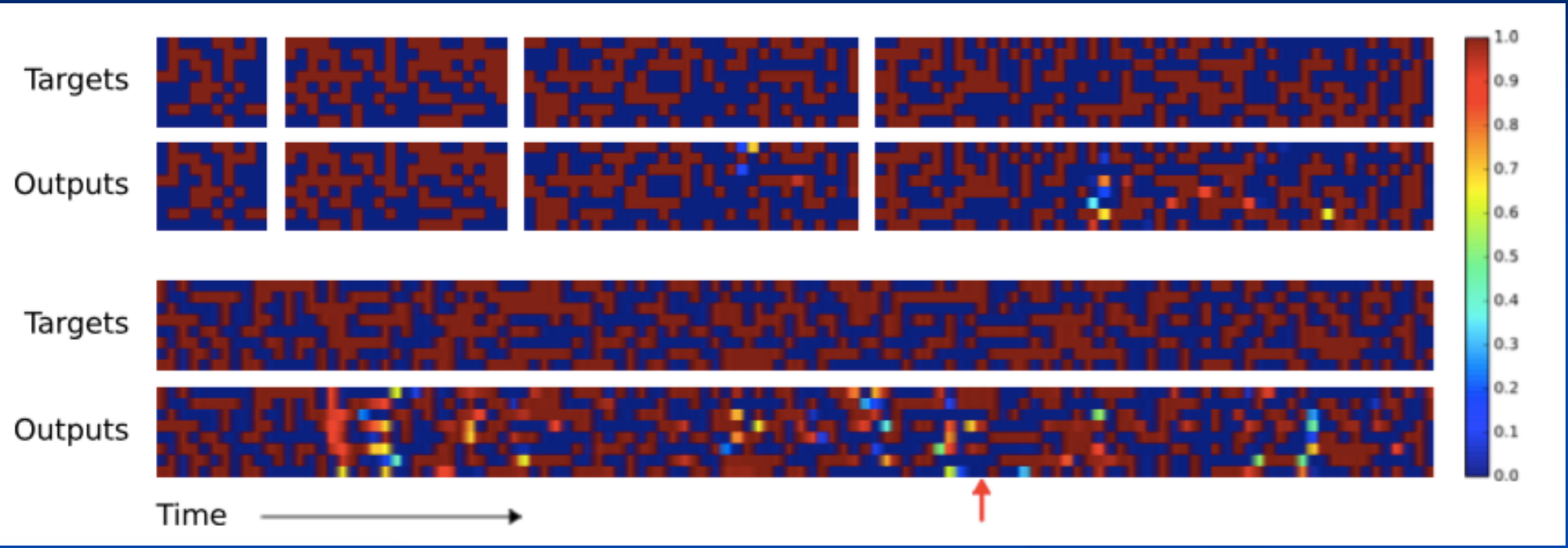
- Copy
- Repeated Copy
- Associative recall
- Dynamic N-grams
- Priority Sort

Copy

- Input: some sequence, e.g., 1,2,3,4
- Target: after the end of the sequence, 1,2,3,4

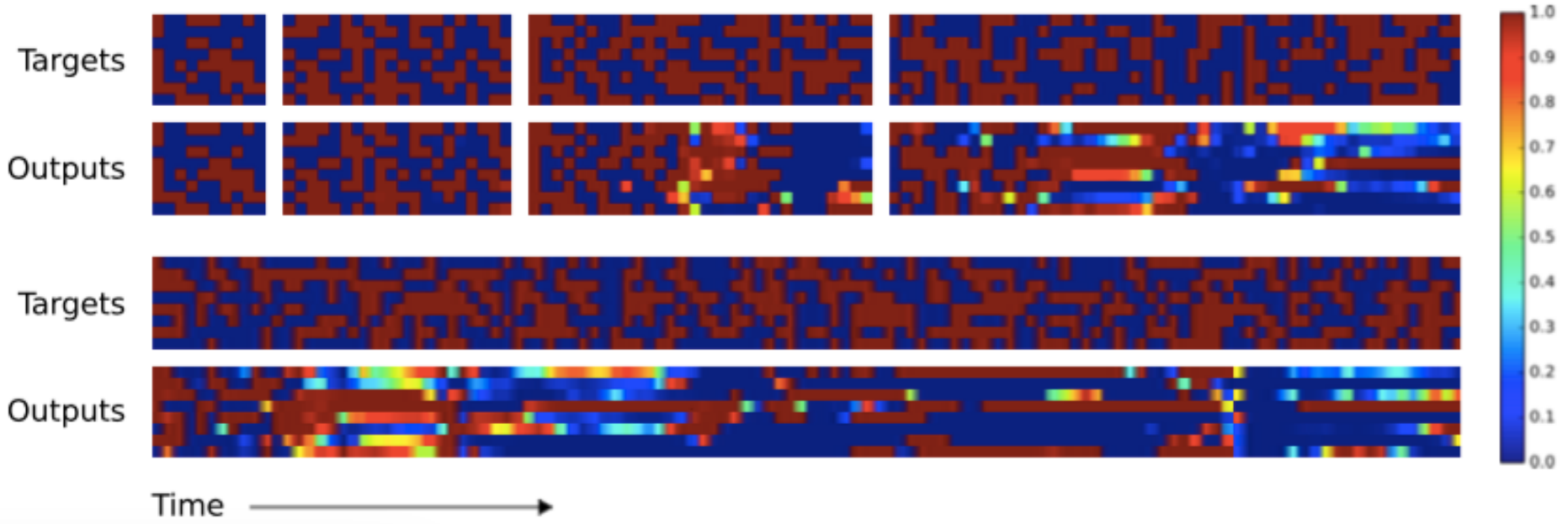


Copy: Results (LSTM controller)



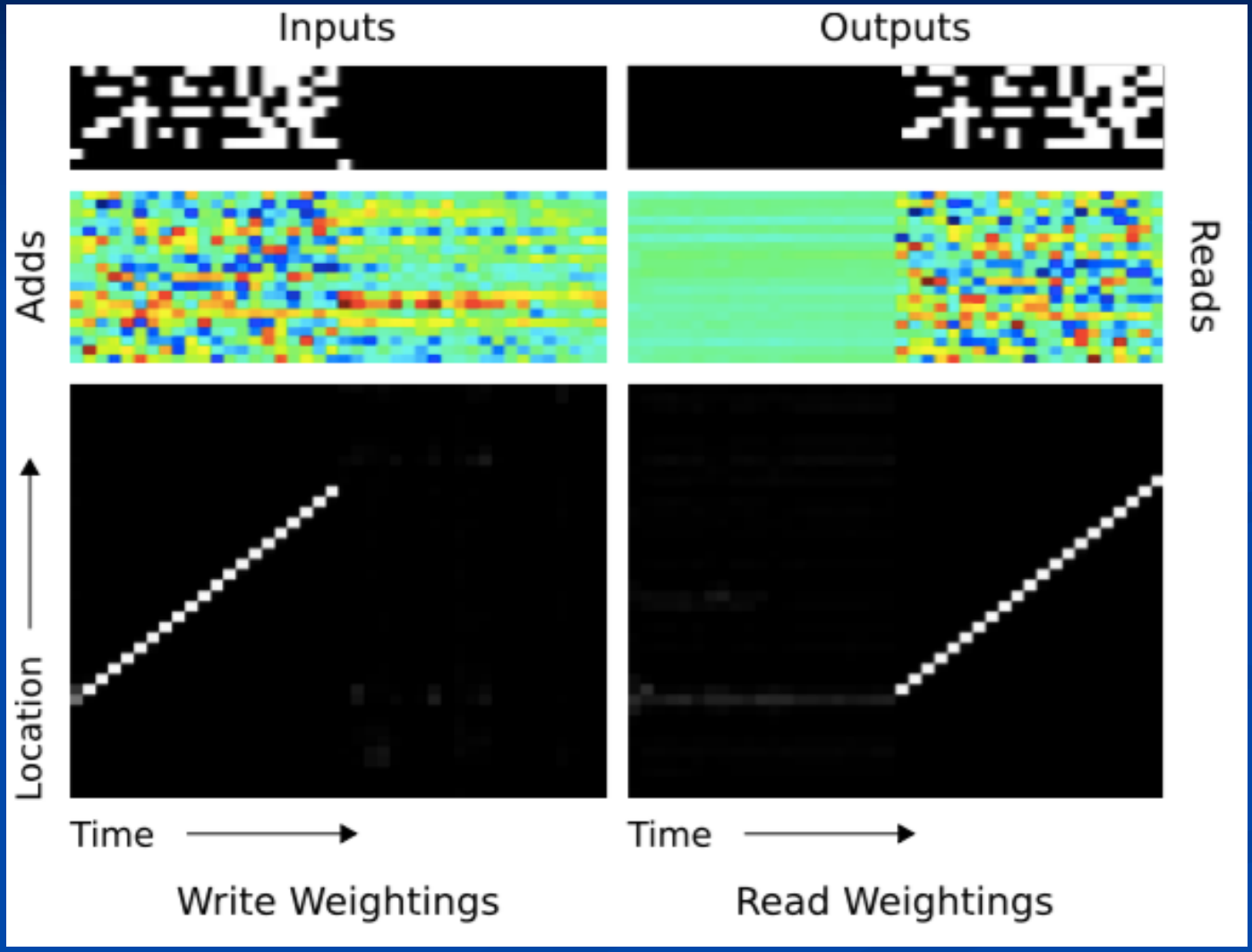
- Note it was only trained on length 20 sequences

Copy: Results (LSTM network)



- Note it was only trained on length 20 sequences

Copy: Results (LSTM NTM)



Copy: The program it learned

```
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
```

Repeat Copy

- For this problem, the network is given a sequence of random bit vectors, followed by a scalar representing the number N of copies to make.
- The network then repeats the sequence N times.

Repeat copy learning curves

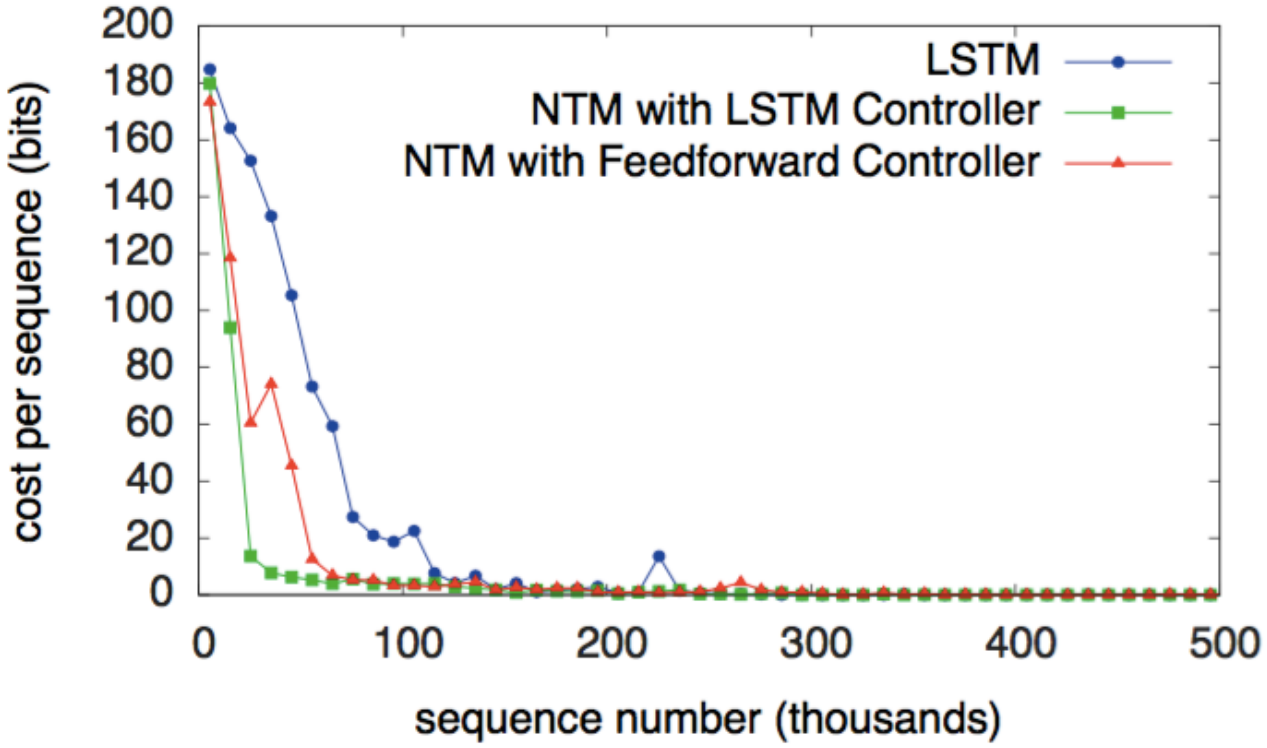
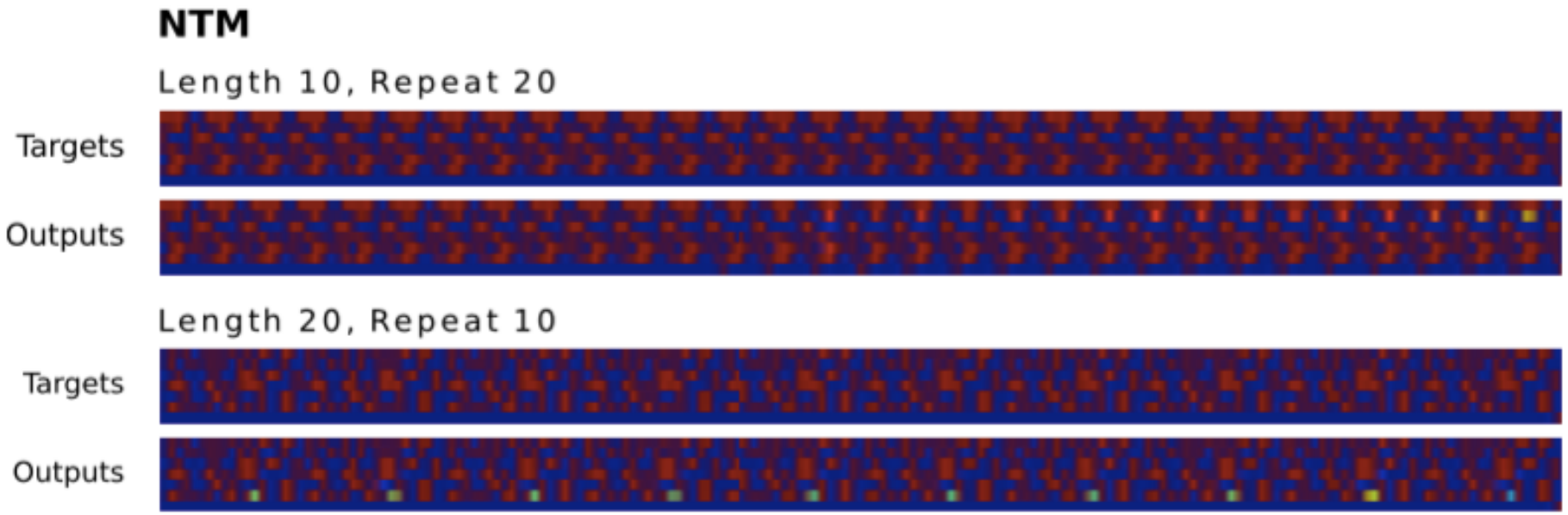


Figure 7: Repeat Copy Learning Curves.

Repeat Copy Results

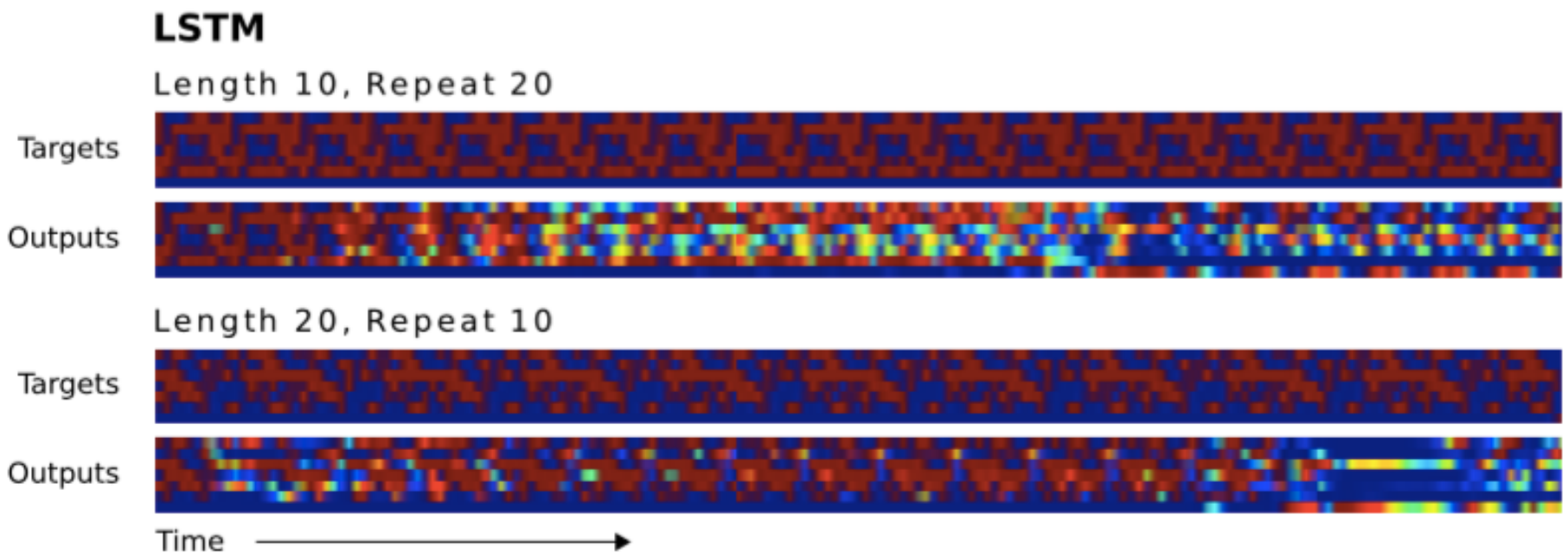
- The network is tested for generalization:
- First the size of the pattern to be copied are doubled (to check generalization)
- And then the number of copies is doubled.

Repeat Copy Results



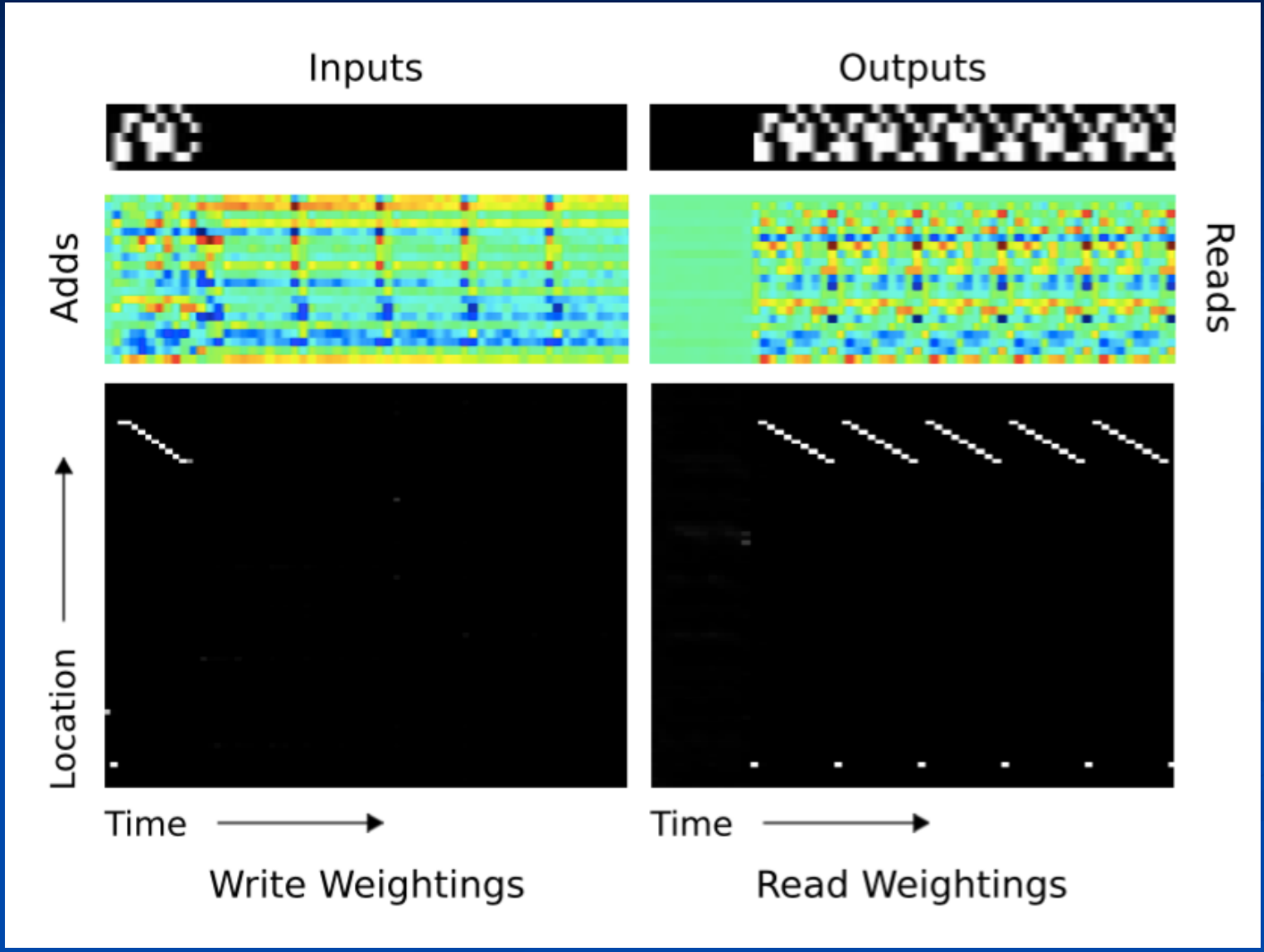
The NTM generalizes very well!

Repeat Copy Results



The vanilla LSTM network, not so much…

Memory use during repeat copy

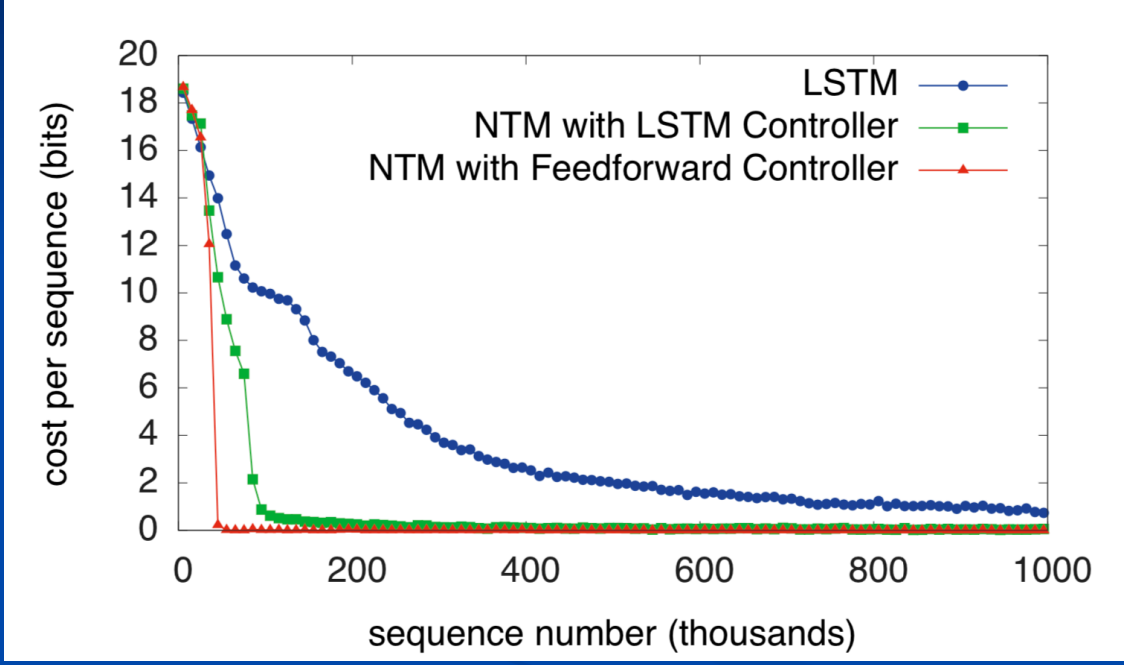


Associative Recall

- Network is presented with 2 to 6 A-B pairs
- Separated by delimiters (7th row)
- Afterwards the network is queried with A and asked to produce B.
- Note 8th row bits indicate this is a query

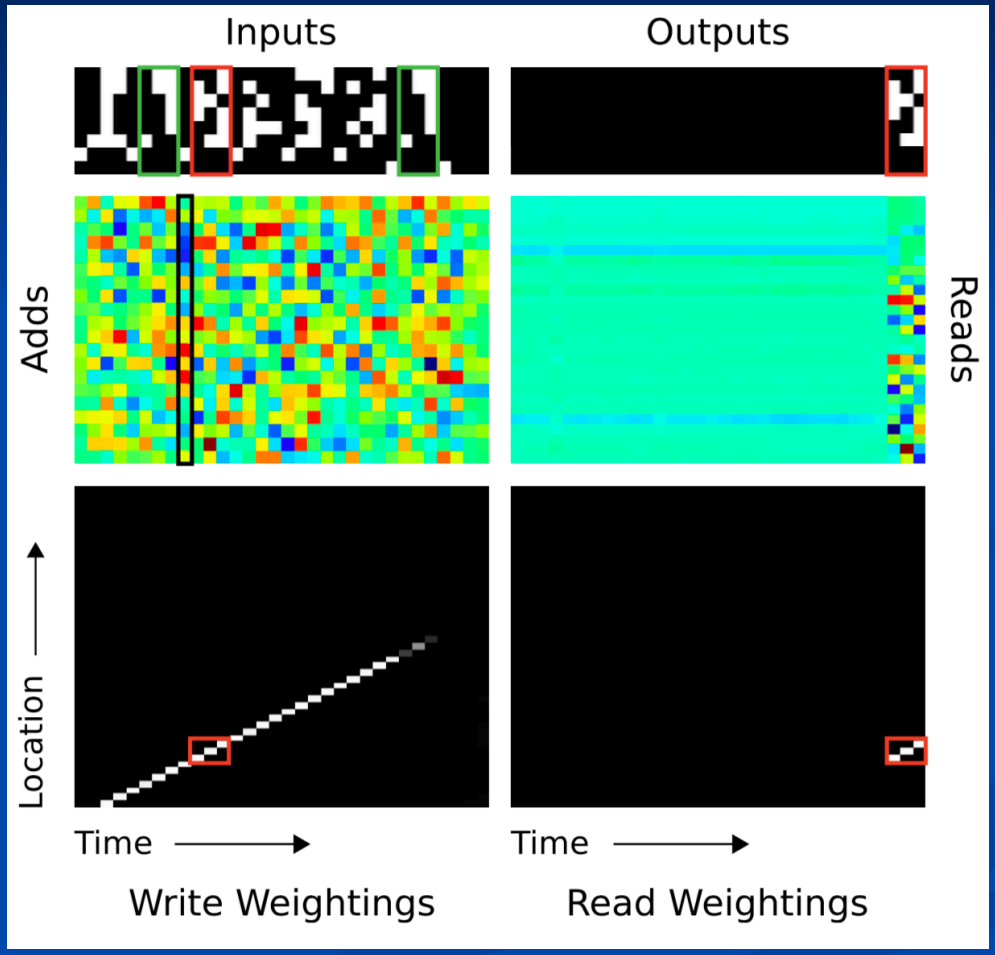
Associative Recall learning curves

- NTM learns perfectly early on
- Vanilla LSTM never gets perfect

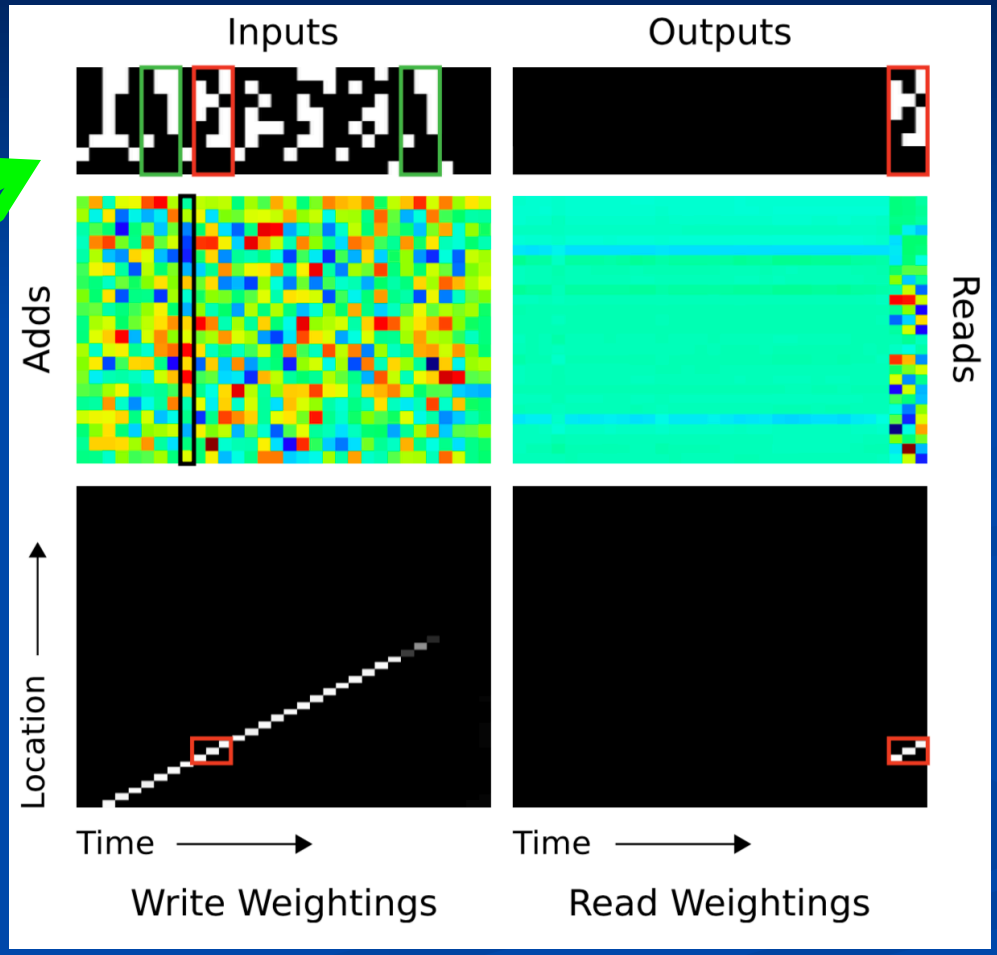


Example run

- Network store

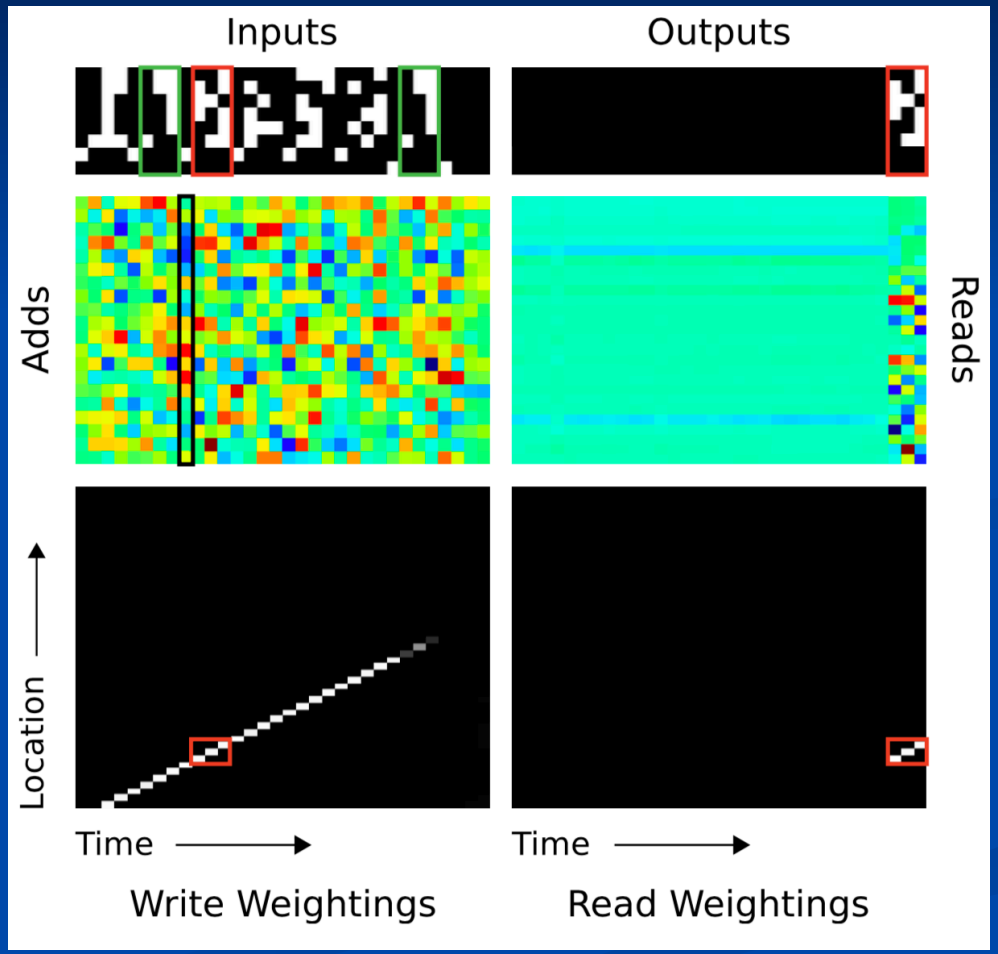


Example run

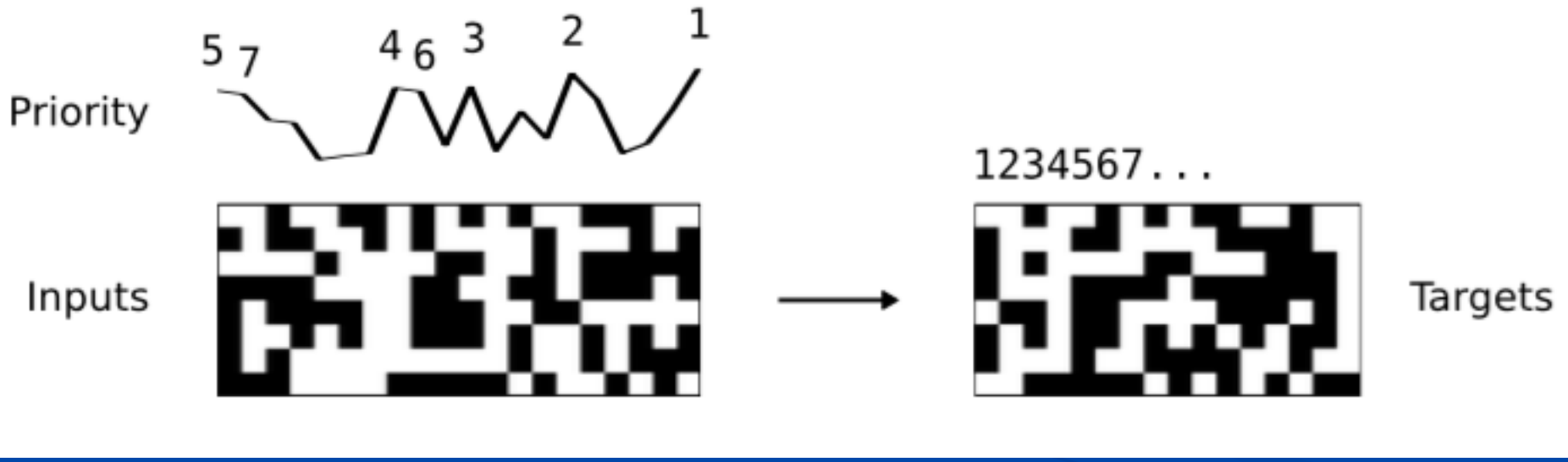

Inputs
Outputs
Adds
Reads
Location
Time
Time
Write Weightings
Read Weightings

How it works

- Stores a compressed version of A
- Followed by B
- Uses ***content-based addressing*** of ***A followed by an increment*** to get B
- I.e., it combines two addressing mechanisms



Priority Sort: the training



Priority Sort: the results

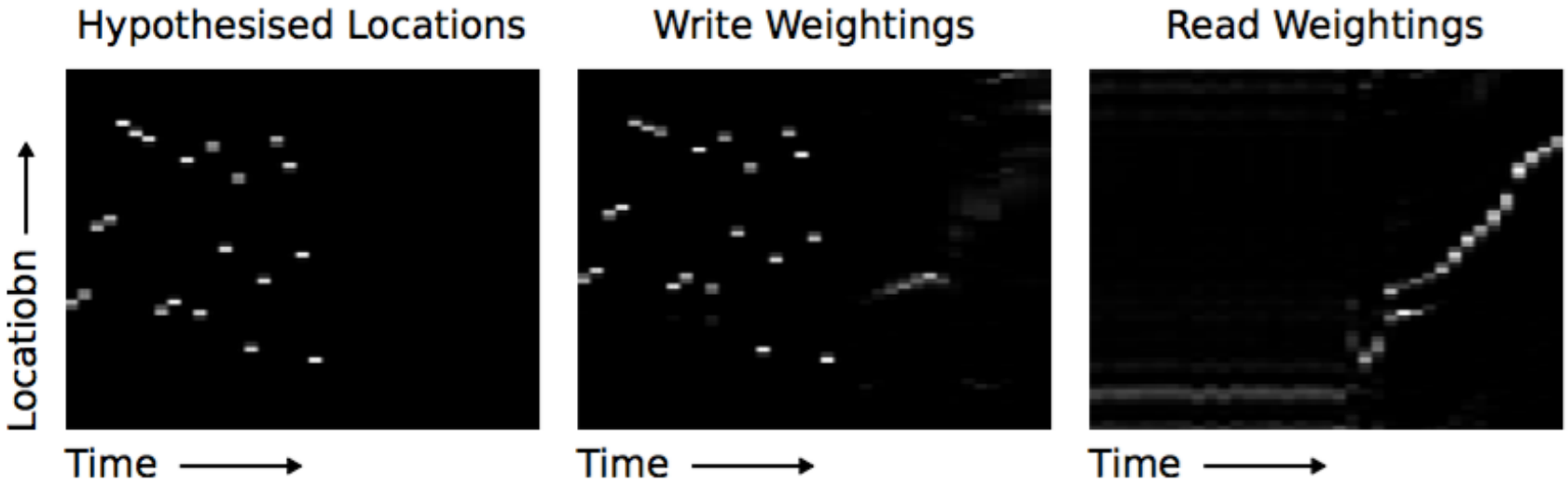


Figure 17: NTM Memory Use During the Priority Sort Task. Left: Write locations returned by fitting a linear function of the priorities to the observed write locations. Middle: Observed write locations. Right: Read locations.

"We hypothesize that it uses the priorities to determine the relative location of each write. To test this hypothesis we fitted a linear function of the priority to the observed write locations. Figure 17 shows that the locations returned by the linear function closely match the observed write locations. It also shows that the network reads from the memory locations in increasing order, thereby traversing the sorted sequence."

Summary

- The Neural Turing Machine is a recurrent neural network augmented by a memory.
- The network (given some addressing mechanisms) can learn to use the memory – because it is end-to-end differentiable, so it can be trained completely by BPTT
- The NTM can learn algorithms from examples of desired input-output sequences!
- Recent work has extended this paradigm...

The Big Questions

- How long before this research leads to networks that can reason
- That can *reflect*
- That can build internal models of itself and other people...
- Before it can have what psychologists call "Theory of Mind" – I know that you know that I know...

The Big Questions

- And at what point do we call that consciousness?
- I'll probably be dead – but this will happen in your lifetimes.
- You had better be ready for it!