Neural Turing Machines

by

Google DeepMind

article overview
by Ilya Kuzovkin

Conclusion

We have introduced the Neural Turing Machine, a neural network architecture that takes inspiration from both models of biological working memory and the design of digital computers. Like conventional neural networks, the architecture is differentiable end-to-end and can be trained with gradient descent. Our experiments demonstrate that it is capable of learning simple algorithms from example data and of using these algorithms to generalise well outside its training regime."

Turing Machine

Turing Machine

State	Symbol	Write	Move	Goto State

Turing Machine

State	Symbol	Write	Move	Goto State
START	0	0	R	2

Turing Machine

State	Symbol	Write	Move	Goto State
START	0	0	R	2
2	0	0	R	3
2	1	1	R	2

Turing Machine

State	Symbol	Write	Move	Goto State
START	0	0	R	2
2	0	0	R	3
2	1	1	R	2
3	0	—	L	END
3	1	0	L	4

Turing Machine

State	Symbol	Write	Move	Goto State
START	0	0	R	2
2	0	0	R	3
2	1	1	R	2
3	0	_	L	END
3	1	0	L	4
4	0	1	R	2

Turing Machine

State	Symbol	Write	Move	Goto State
START	0	0	R	2
2	0	0	R	3
2	1	1	R	2
3	0	_	L	END
3	1	0	L	4
4	0	1	R	2

What it will do on the following sequence?

↓

___01110110___

Universal Turing Machine

A TM that can simulate any TM on any input.

Universal Turing Machine

A TM that can simulate
any TM on any input.

Reads both from the tape
and then executes.

Universal Turing Machine

A TM that can simulate
any TM on any input.

Reads both from the tape
and then executes.



Universal Turing Machine

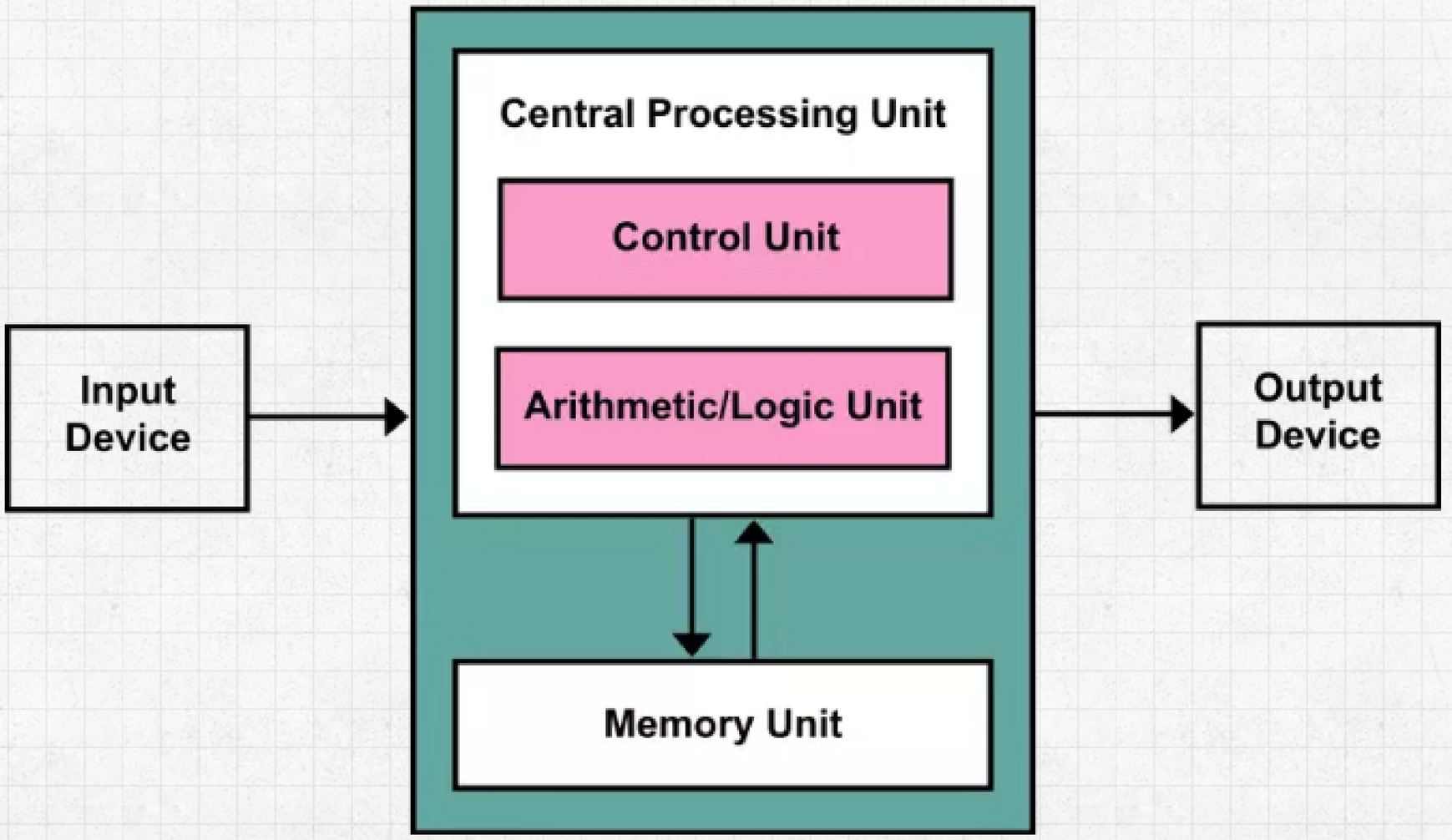
A TM that can simulate
any TM on any input.

Reads both from the tape
and then executes.

1937



Von Neumann architecture



Current Machine Learning

✓ elementary operations

× logical flow control*

× external memory

* what about decision trees?

Current Machine Learning

✓ elementary operations

✗ logical flow control*

✗ external memory

} NTM

* what about decision trees?

Memory In Other Fields

Psychology and Neuroscience (2.1)

- working memory

Memory In Other Fields

Psychology and Neuroscience (2.1)

- working memory
- persistent firing of some neurons in a delay phase of memory-related experiment

Memory In Other Fields

Psychology and Neuroscience (2.1)

- working memory
- persistent firing of some neurons in a delay phase of memory-related experiment
- modeling of explicit tasks

Memory In Other Fields

Psychology and Neuroscience (2.1)

- working memory
- persistent firing of some neurons in a delay phase of memory-related experiment
- modeling of explicit tasks

Cognitive Science and Linguistics (2.2)

- recursive symbol processing

Memory In Other Fields

Psychology and Neuroscience (2.1)

- working memory
- persistent firing of some neurons in a delay phase of memory-related experiment
- modeling of explicit tasks

Cognitive Science and Linguistics (2.2)

- recursive symbol processing
- Fodor and Pylyshyn said that ANN are incapable of variable-binding and thus can't model cognition

Memory In Other Fields

Psychology and Neuroscience (2.1)

- working memory
- persistent firing of some neurons in a delay phase of memory-related experiment
- modeling of explicit tasks

Cognitive Science and Linguistics (2.2)

- recursive symbol processing
- Fodor and Pylyshyn said that ANN are incapable of variable-binding and thus can't model cognition
- Fixed-length vs. variable-length argument

Why RNN?

- state depends on input and current state
  thus enabling context-dependent computations

Why RNN?

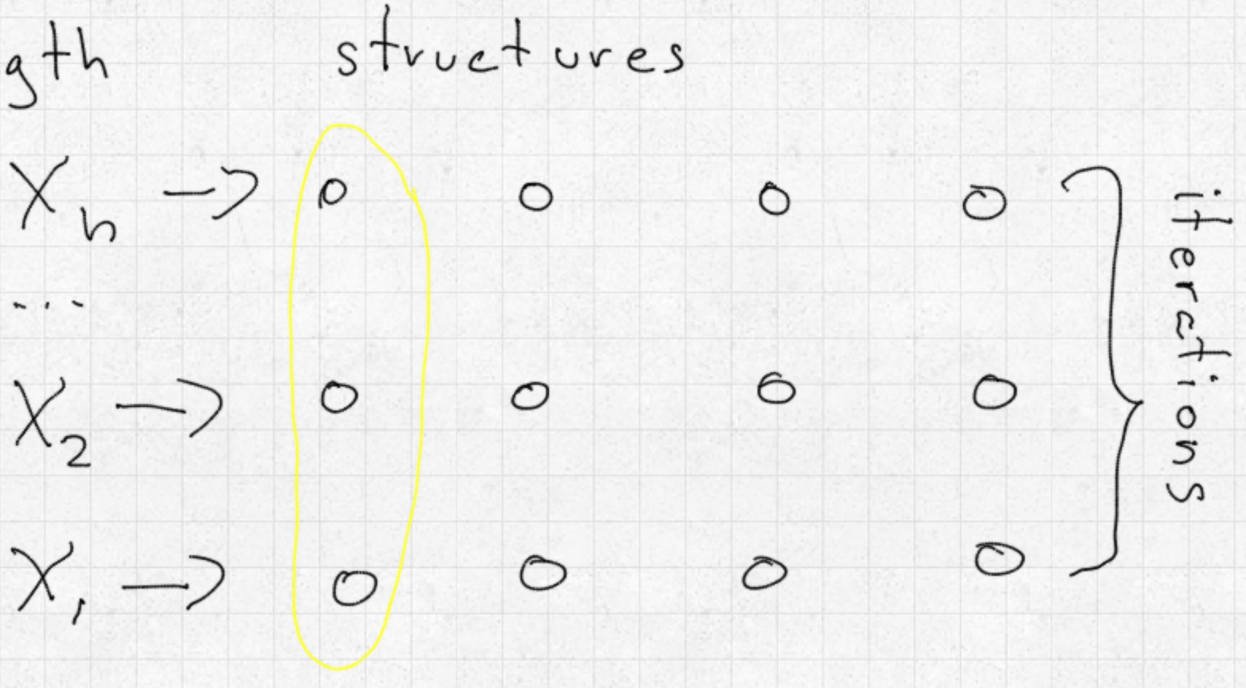
- state depends on input and current state thus enabling context-dependent computations
- distributed state compared to HMM

Why RNN?

- state depends on input and current state thus enabling context-dependent computations
- distributed state compared to HMM
- Long Short-Term Memory (will explain soon)

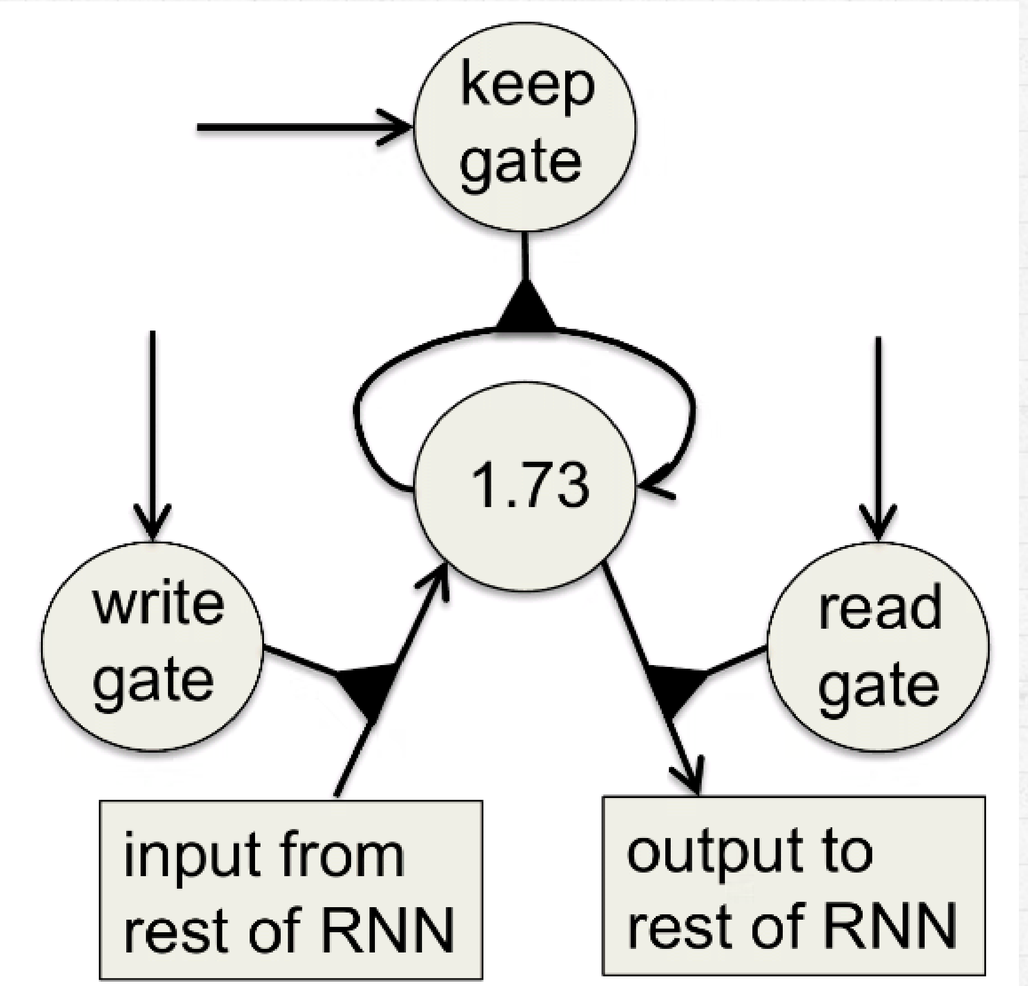
Why RNN?

- state depends on input and current state thus enabling context-dependent computations
- distributed state compared to HMM
- Long Short-Term Memory (will explain soon)
- Variable-length structures



LSTM

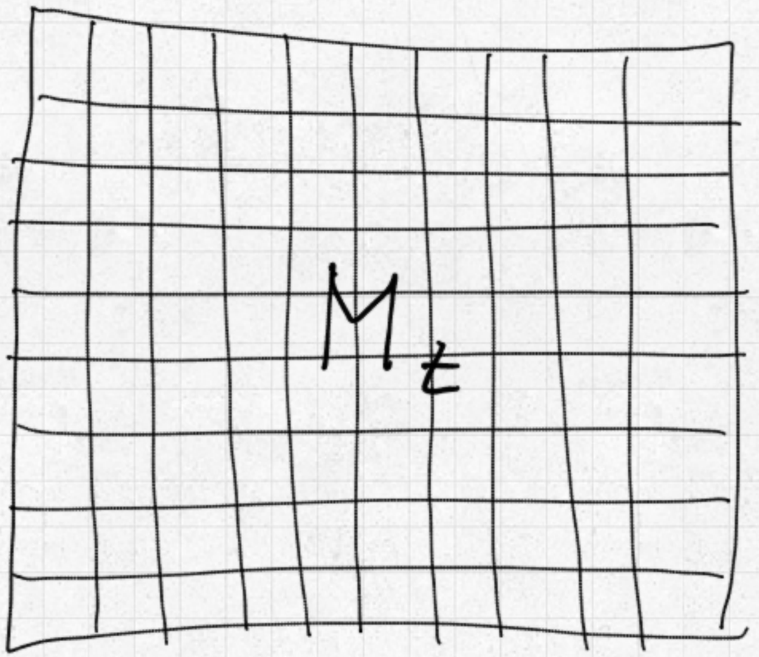
Long Short-Term Memory



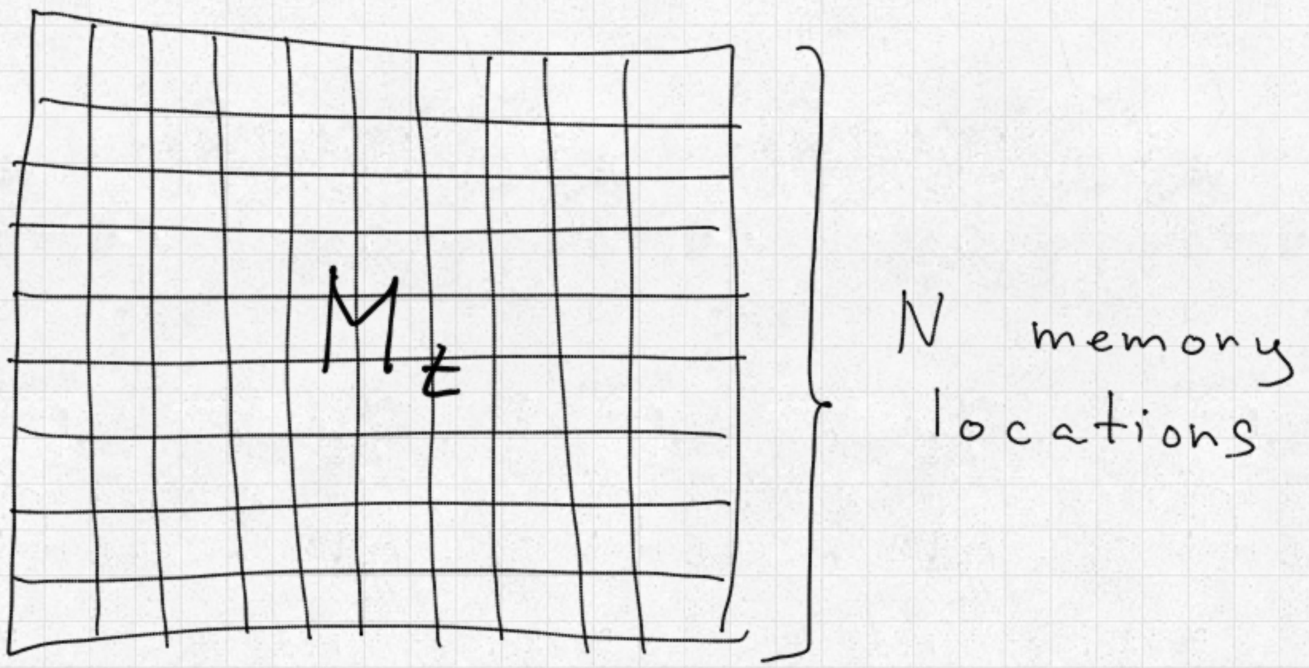
See Hinton's Coursera course Lecture 7

Neural Turing Machine

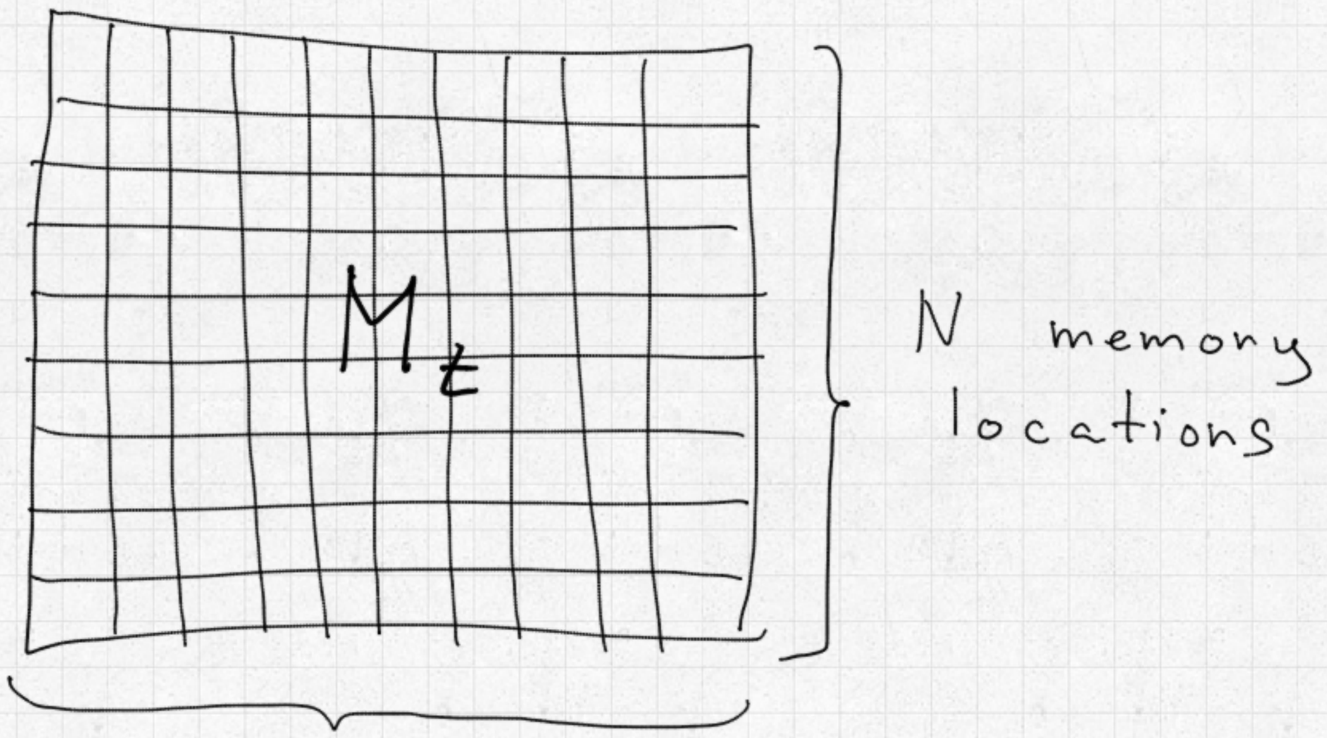
Memory Matrix



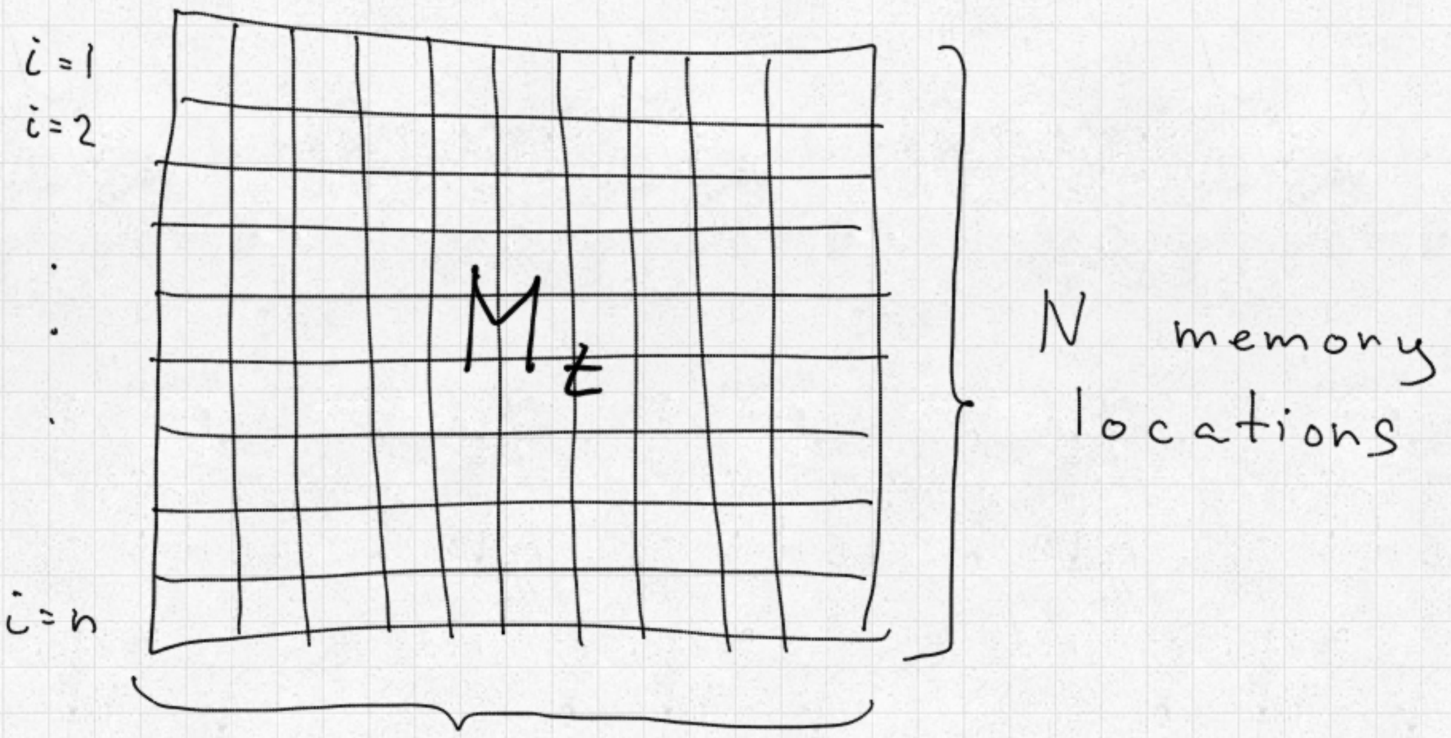
Memory Matrix



Memory Matrix

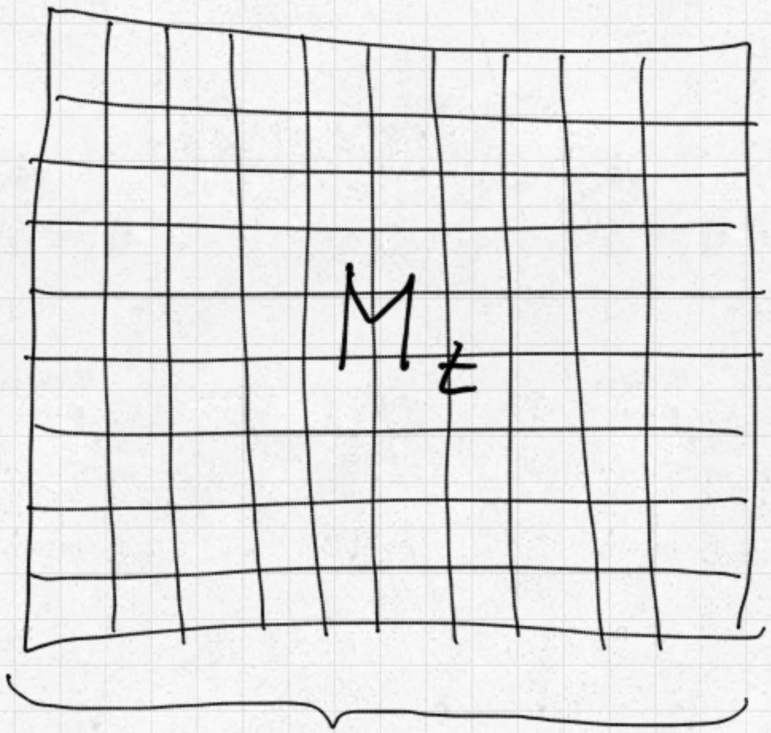


Memory Matrix



Memory Matrix

$$w_t = \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \quad \begin{matrix} i=1 \\ i=2 \\ \\ i=n \end{matrix}$$



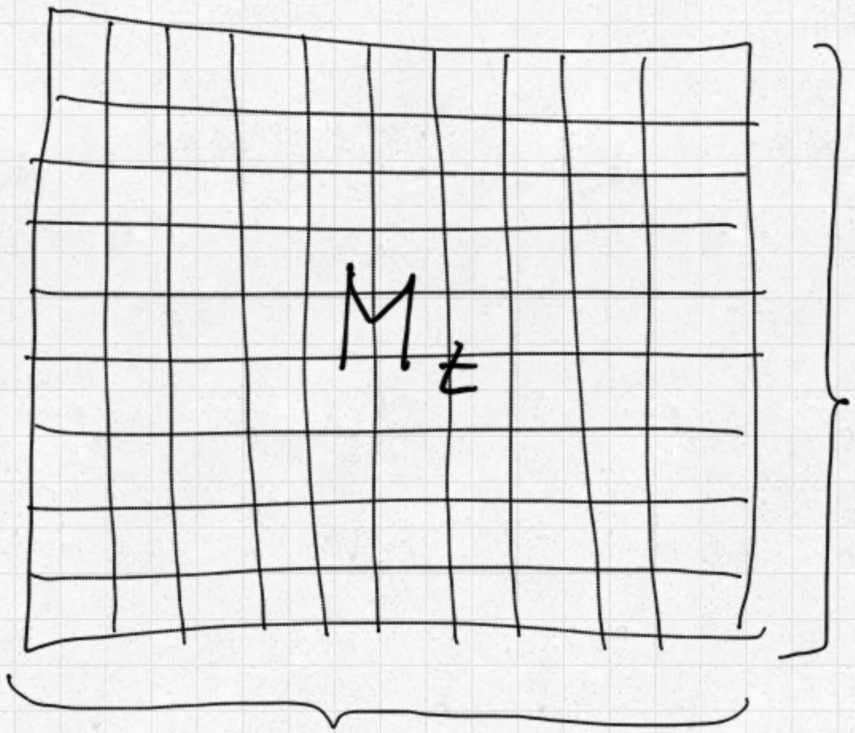
N memory locations

Vector of length M stored at each "location"

Memory Matrix

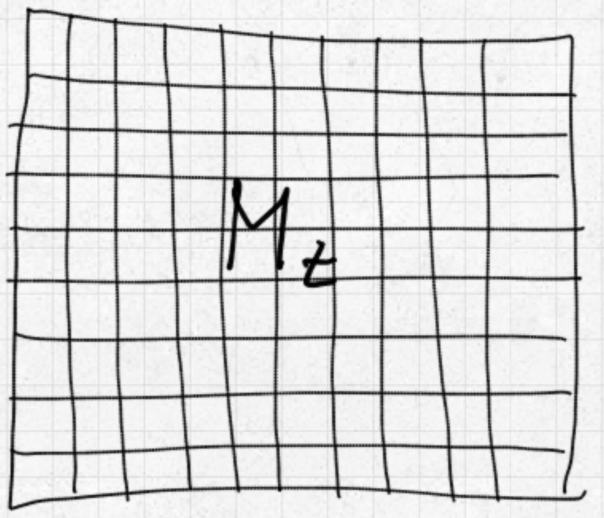
Weightings of memory locations at time t

$$w_t = \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \quad \begin{matrix} i=1 \\ i=2 \\ \\ i=n \end{matrix}$$



Reading (3.1)

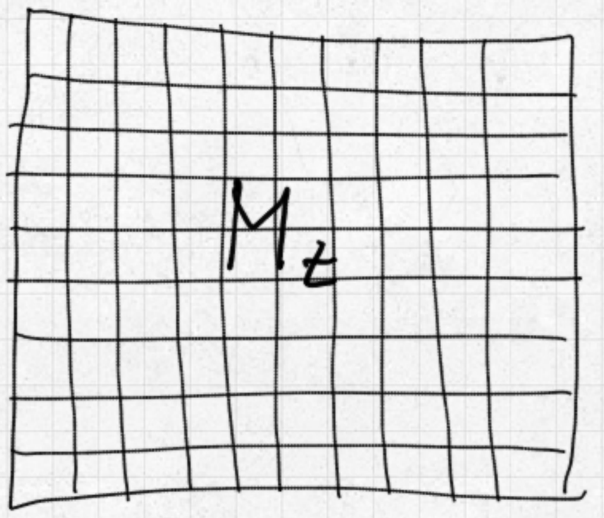
$$w_t = \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \quad \begin{matrix} i=1 \\ i=2 \\ \vdots \\ i=n \end{matrix}$$



Reading (3.1)

$$r_t = \sum_i w_t(i)\, M_t(i)$$

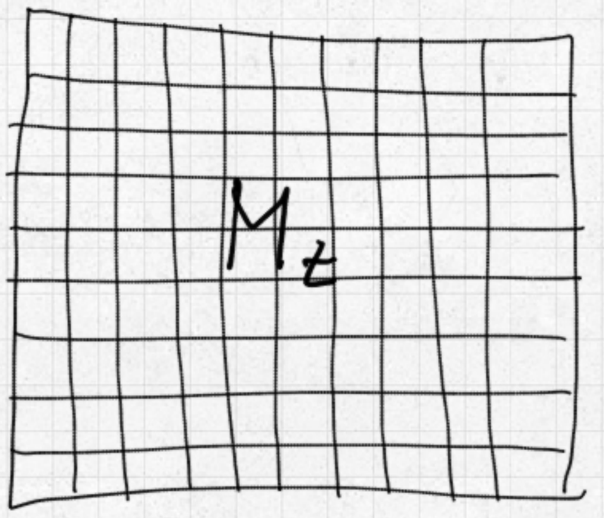
$$w_t = \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \quad \begin{matrix} i=1 \\ i=2 \\ \\ \\ i=n \end{matrix}$$



Reading (3.1)

$$r_t = \sum_i w_t(i)\, M_t(i)$$

$$w_t = \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \quad \begin{matrix} i=1 \\ i=2 \\ \cdot \\ \cdot \\ \cdot \\ i=n \end{matrix}$$

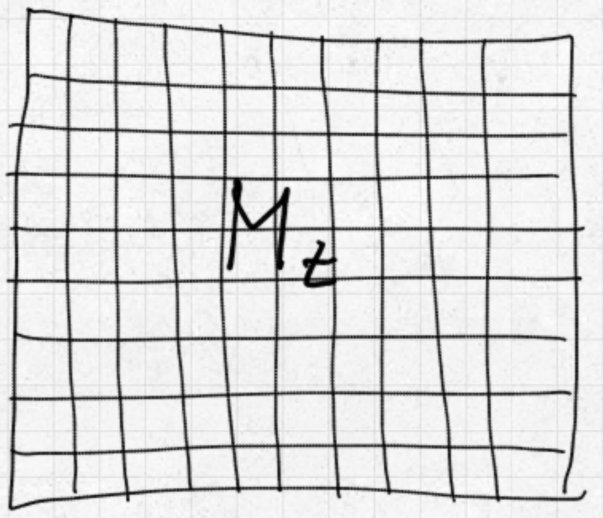


Differentiable w.r.t memory and weighting.

Meaning you can change memory and weight and always obtain some value for r_t

Writing (3.2)

$$w_t \approx \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \begin{matrix} i=1 \\ i=2 \\ \vdots \\ i=n \end{matrix}$$



Writing (3.2)

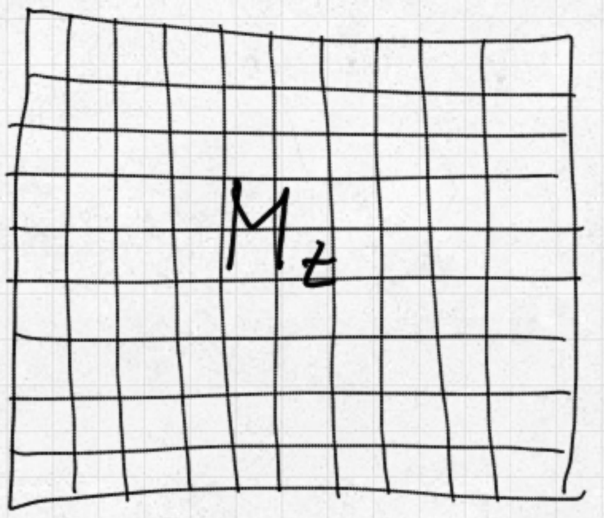
Similar to keep and write gates in LSTM we have

erase vector e_t

and

add vector a_t

$$w_t = \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \begin{matrix} i=1 \\ i=2 \\ \\ \\ i=n \end{matrix}$$



Writing (3.2)

Similar to keep and write gates in LSTM we have

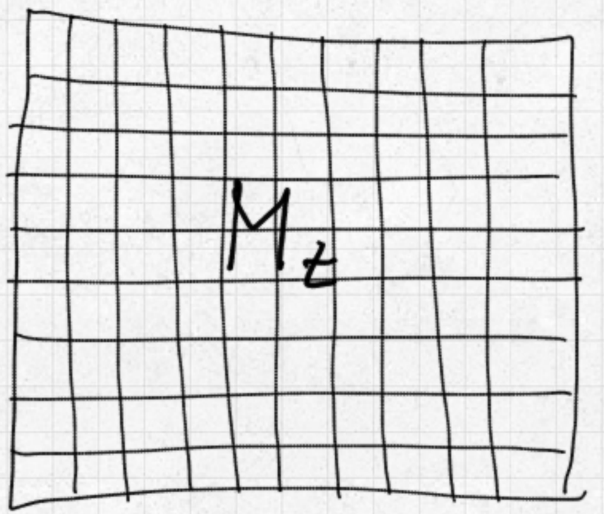
erase vector e_t

and

add vector a_t

both of which are calculated by the RNN.

$$w_t = \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \quad \begin{matrix} i=1 \\ i=2 \\ \\ \\ i=n \end{matrix}$$



$$\tilde{M}_t(i) = M_{t-1}(i)\left[1 - w_t(i)\, e_t\right]$$

$$M_t(i) = \tilde{M}_t(i) + w_t(i)\, a_t$$

Writing (3.2)

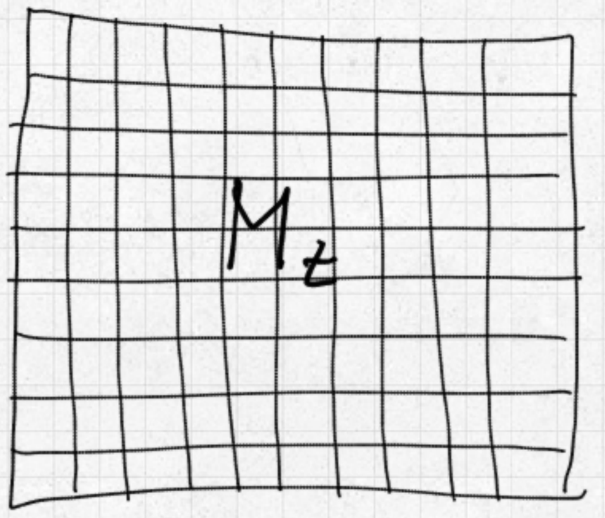
Similar to keep and write gates in LSTM we have

erase vector e_t

and

add vector a_t

$$w_t = \begin{pmatrix} w_t(1) \\ w_t(2) \\ \vdots \\ w_t(n) \end{pmatrix} \begin{matrix} i=1 \\ i=2 \\ \\ \\ i=n \end{matrix}$$



both of which are calculated by the RNN.

$$\tilde{M}_t(i) = M_{t-1}(i)\left[1 - w_t(i)\, e_t\right] \qquad M_t(i) = \tilde{M}_t(i) + w_t(i)\, a_t$$

both operations differentiable → writing differentiable

Addressing (3.3)

One of the most important things in the article: how do we regulate where to write and read from?

Addressing (3.3)

One of the most important things in the article: how do we regulate where to write and read from?

Usually left out from computational models in neuroscience.

Addressing (3.3)

One of the most important things in the article: how do we regulate where to write and read from?

Usually left out from computational models in neuroscience.

Focusing by Content (3.3.1)

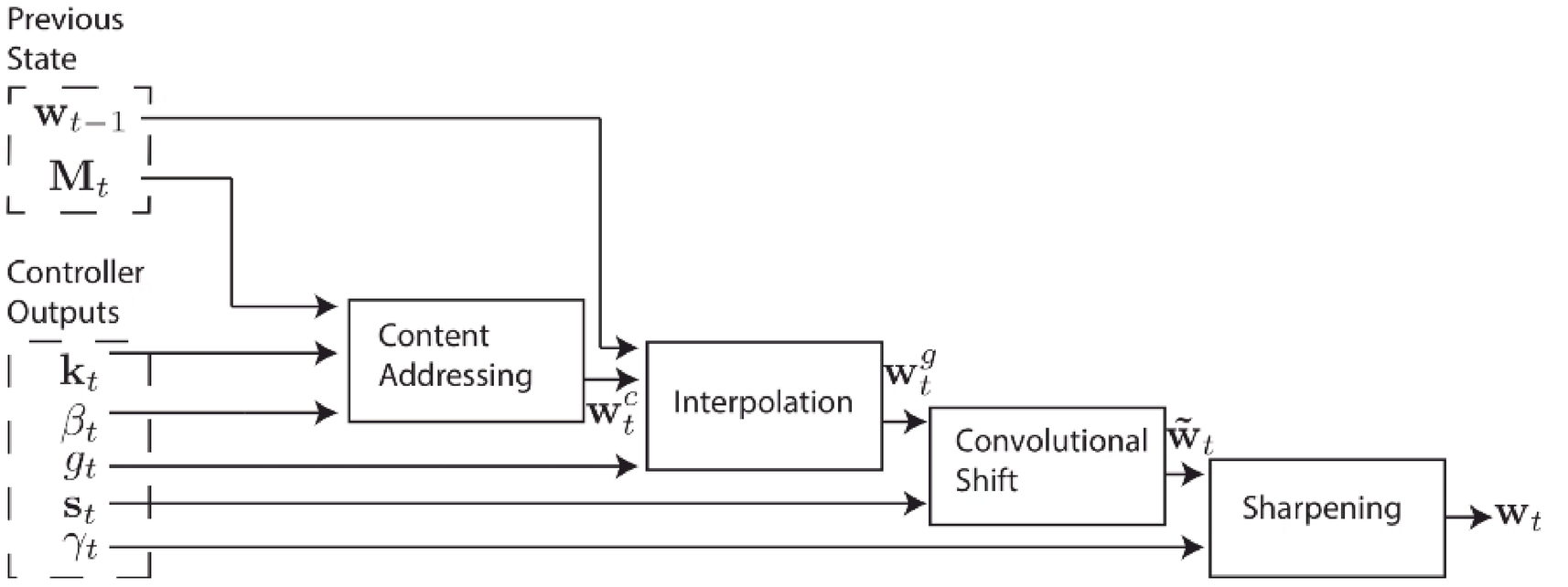
Focusing by Location (3.3.2)

Focusing by Content (3.3.1)

Similar to the Hopfield Network concept.

Focusing by Content (3.3.1)

Similar to the Hopfield Network concept.



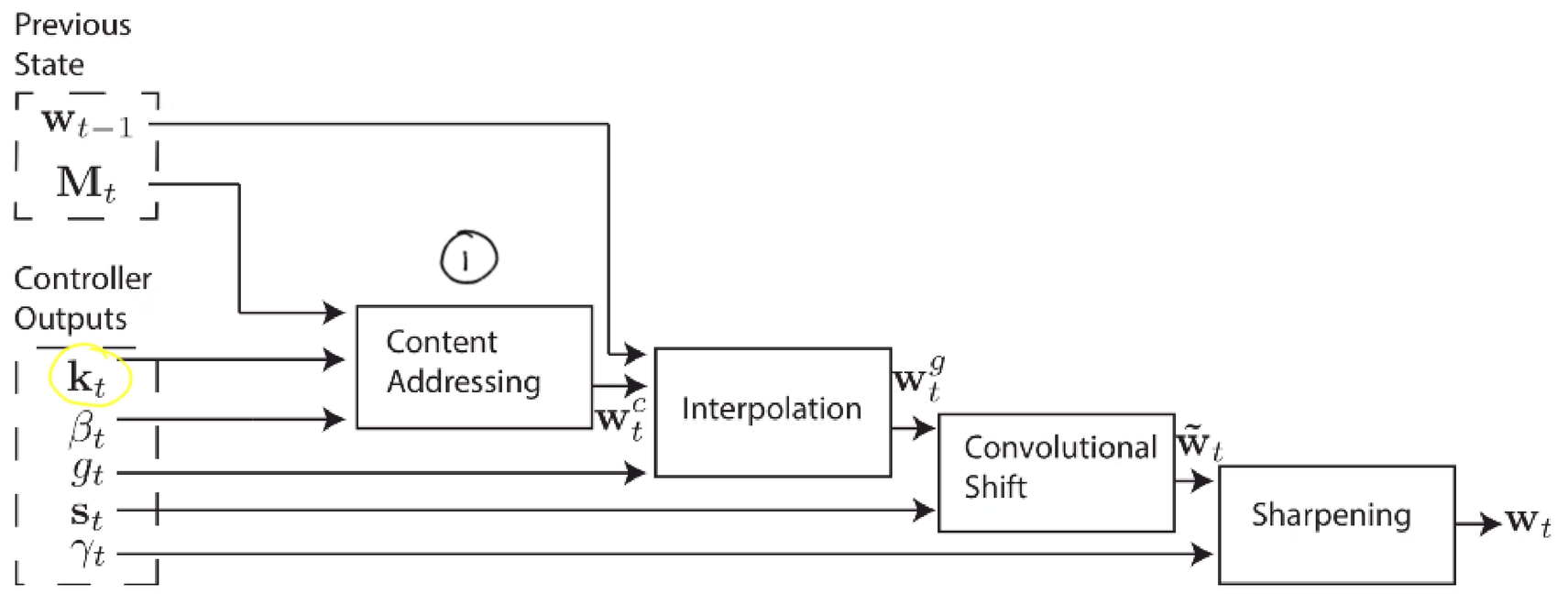
Focusing by Content (3.3.1)

Similar to the Hopfield Network concept.

(1)

Reading or writing head before doing its job will produce vector

k_t



Focusing by Content (3.3.1)

Similar to the Hopfield Network concept.

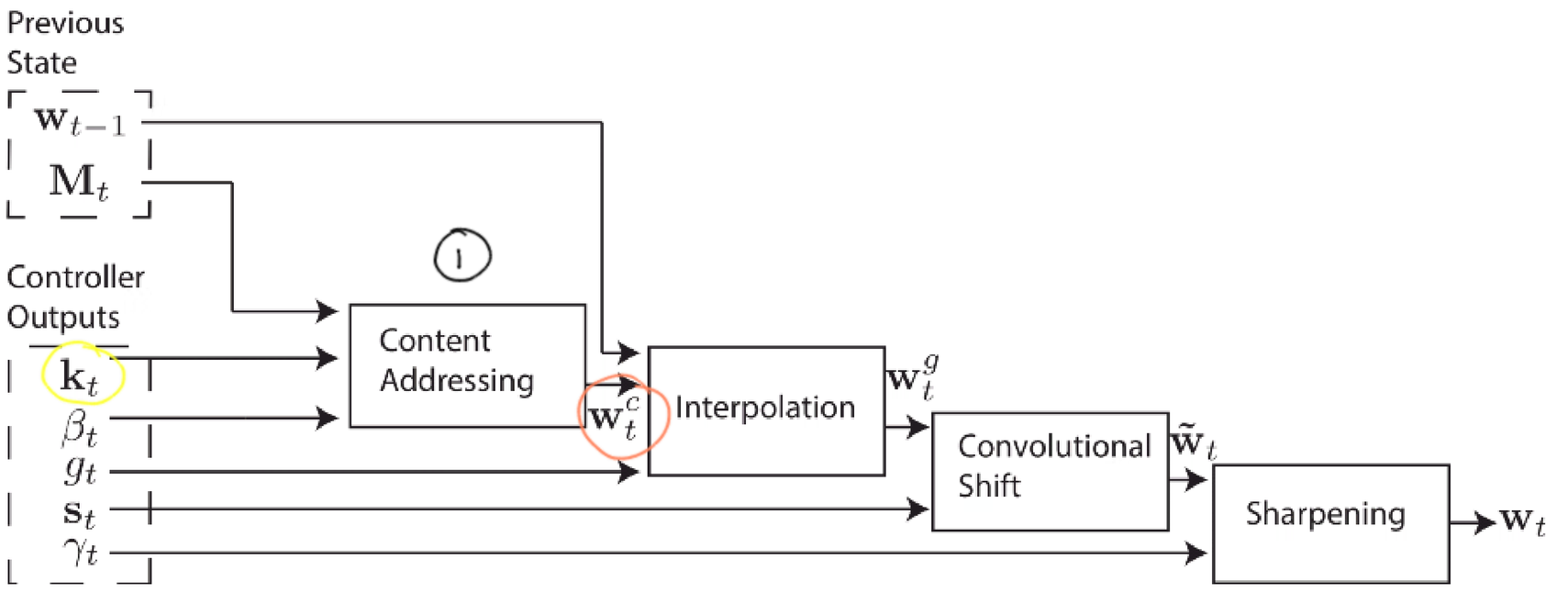
(1)

Reading or writing head before doing its job will produce vector

k_t

compare it to all $M_t(i)$ and output weightings

w_t^c



Focusing by Content (3.3.1)

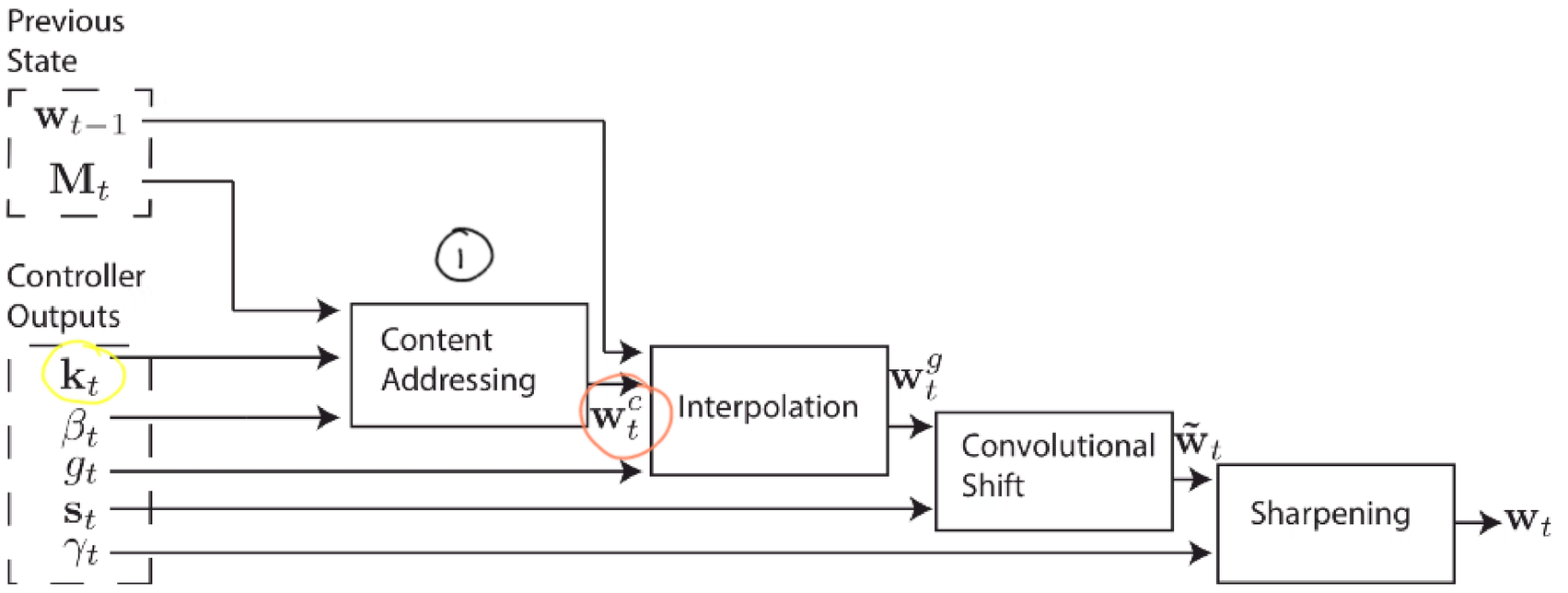
Similar to the Hopfield Network concept.

① Reading or writing head before doing its job will produce vector

k_t

compare it to all $M_t(i)$ and output weightings

$w_t^c(i) =$



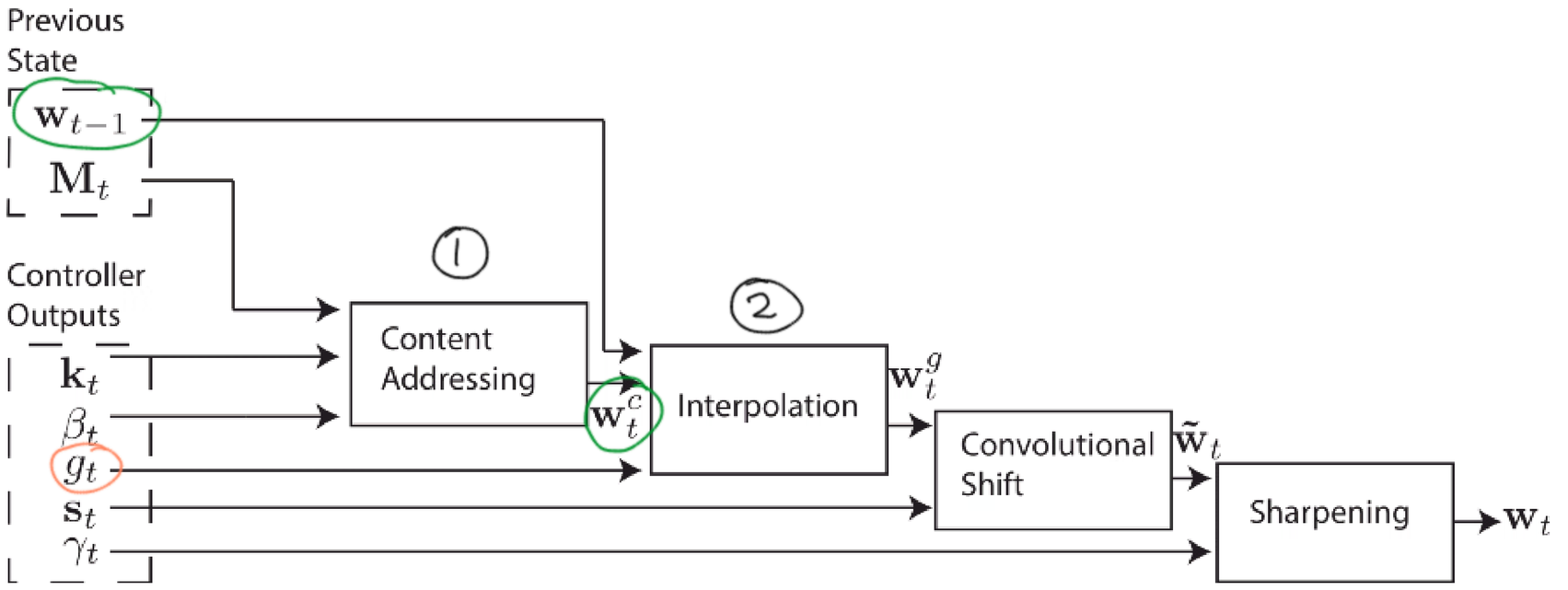
$$\frac{\exp\left(\beta_t K\left[\mathbf{k}_t, \mathbf{M}_t(i)\right]\right)}{\sum_j \exp\left(\beta_t K\left[\mathbf{k}_t, \mathbf{M}_t(j)\right]\right)}.$$

where

$$K\left[\mathbf{u}, \mathbf{v}\right] = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \cdot ||\mathbf{v}||}.$$

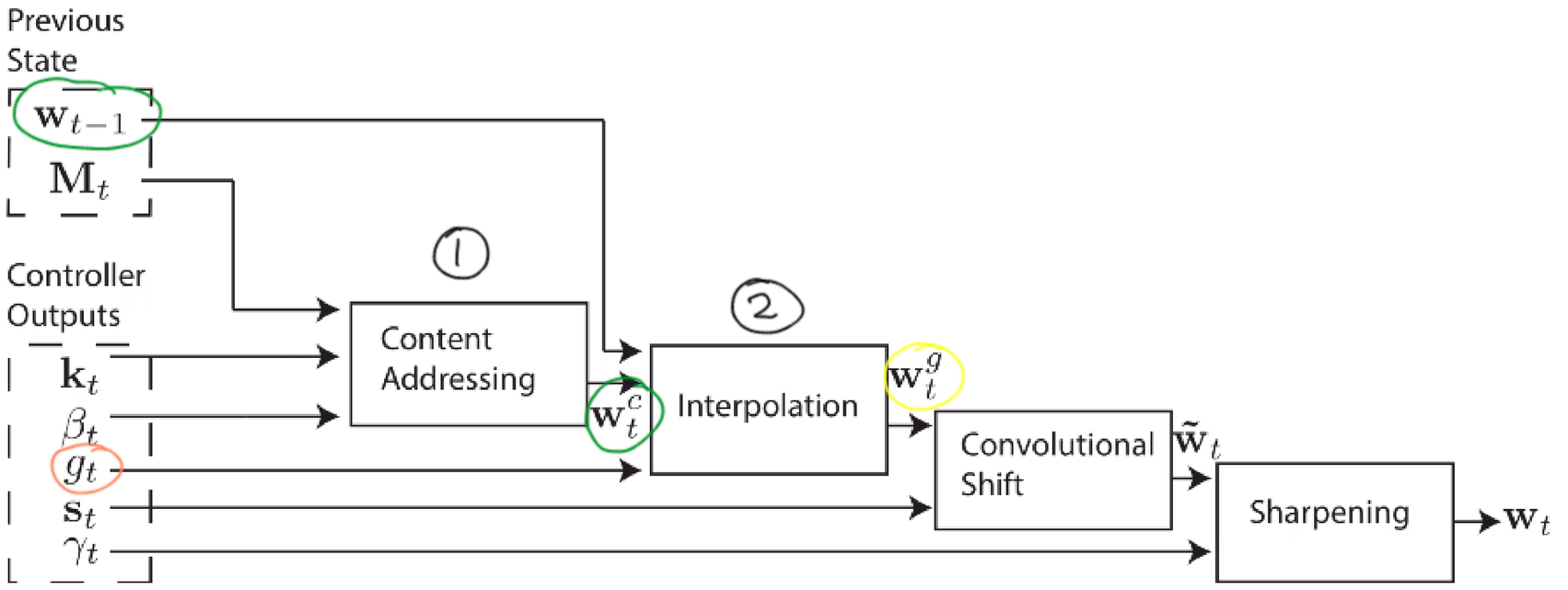
(2)

Interpolation with the weights at the previous step



②

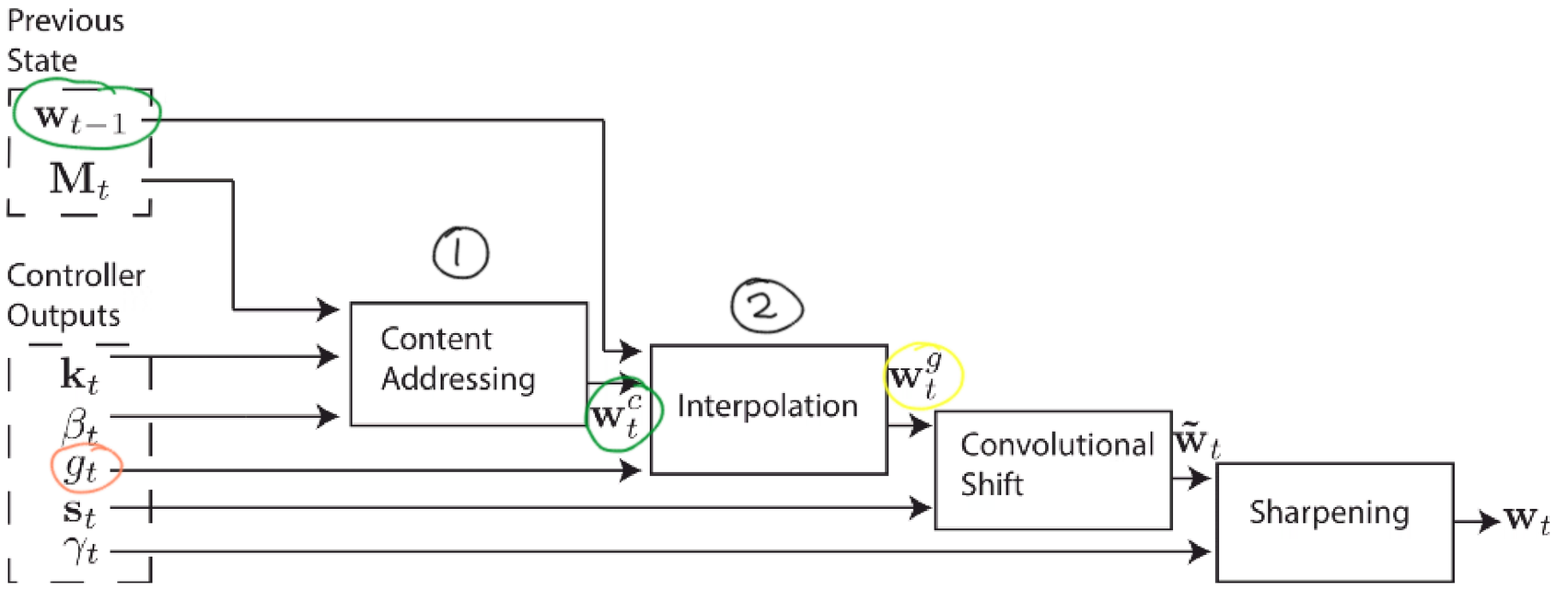
Interpolation with the weights at the previous step



$$w_t^g = g_t w_t^c + (1 - g_t) w_{t-1}$$

(2)

Interpolation with the weights at the previous step



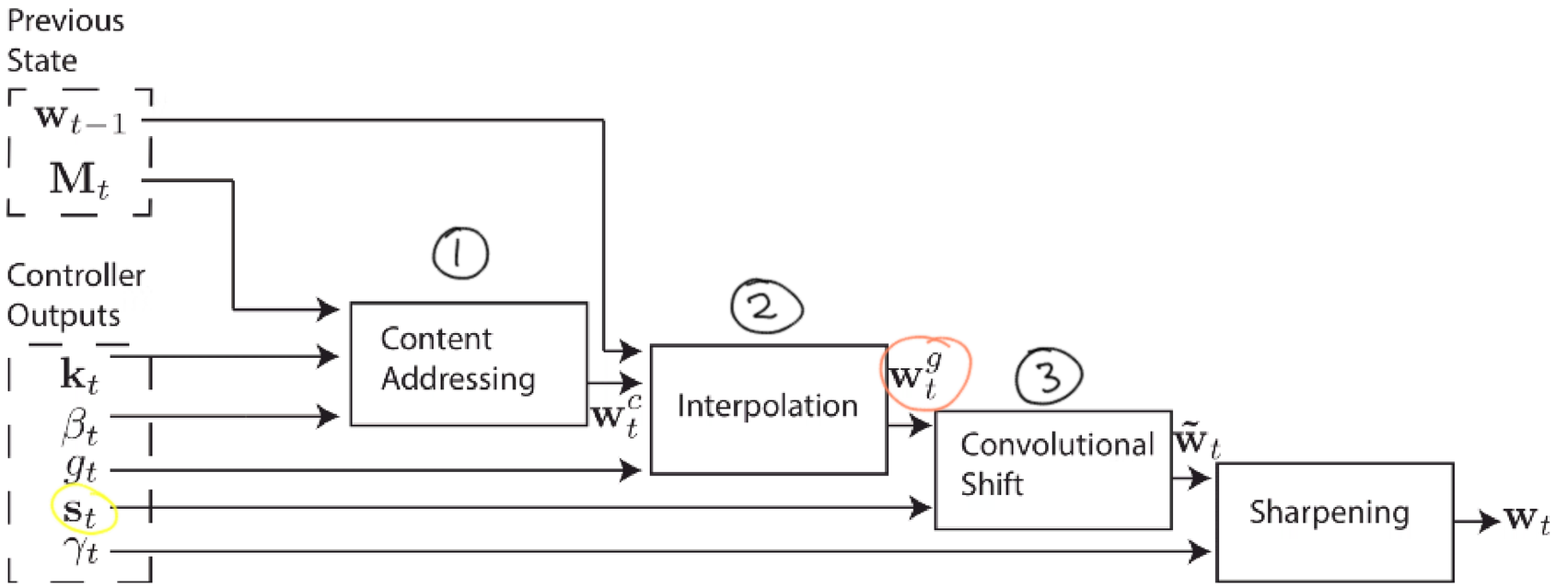
$$w_t^g = g_t w_t^c + (1 - g_t) w_{t-1}$$

the value of g_t is, once again, povided by the magical Controller

③

Shift

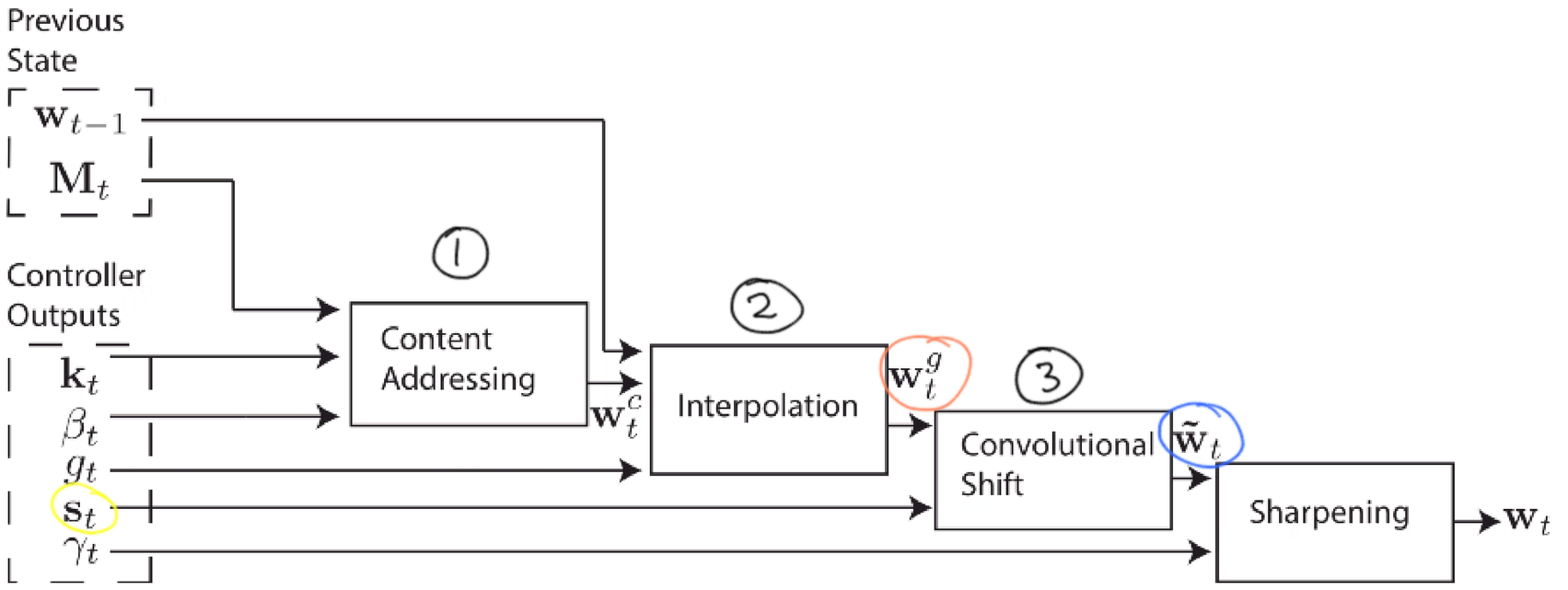
the position we access according to S_t



③

Shift

the position we access according to S_t



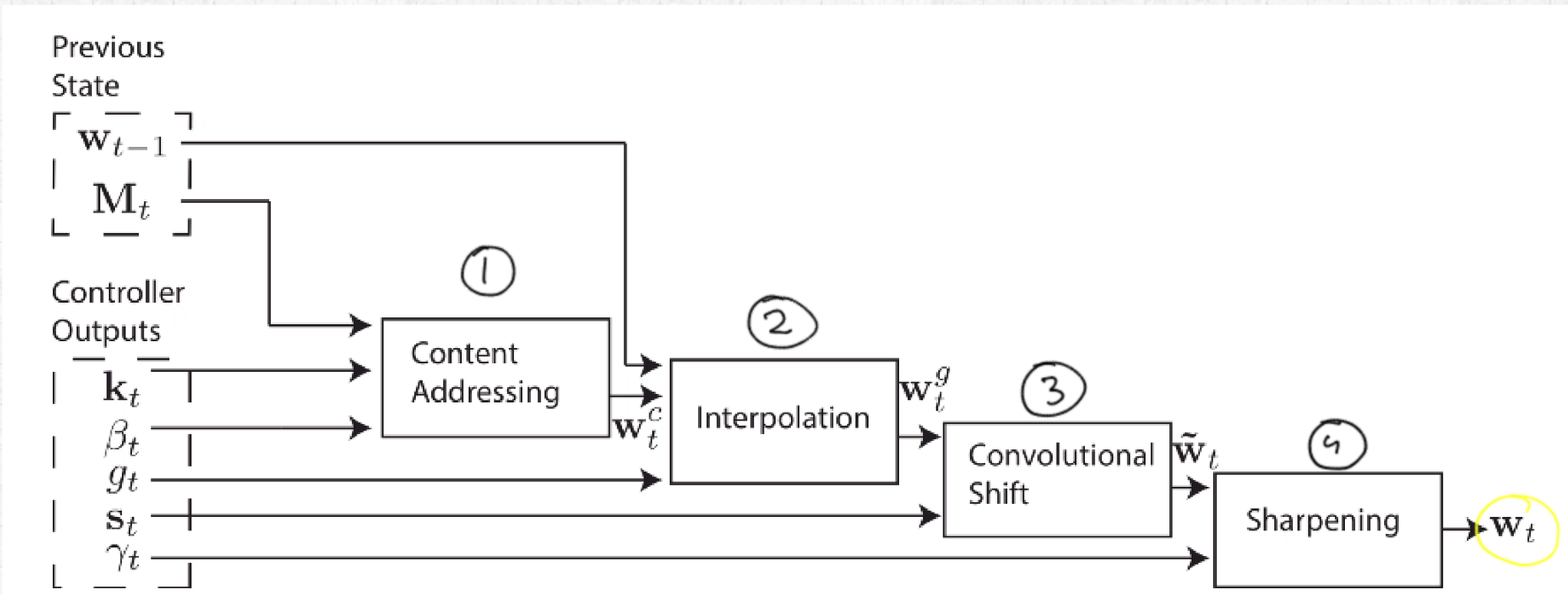
In this way the Controller says how wide area should be affected by the action of the head.

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

④

Sharpening

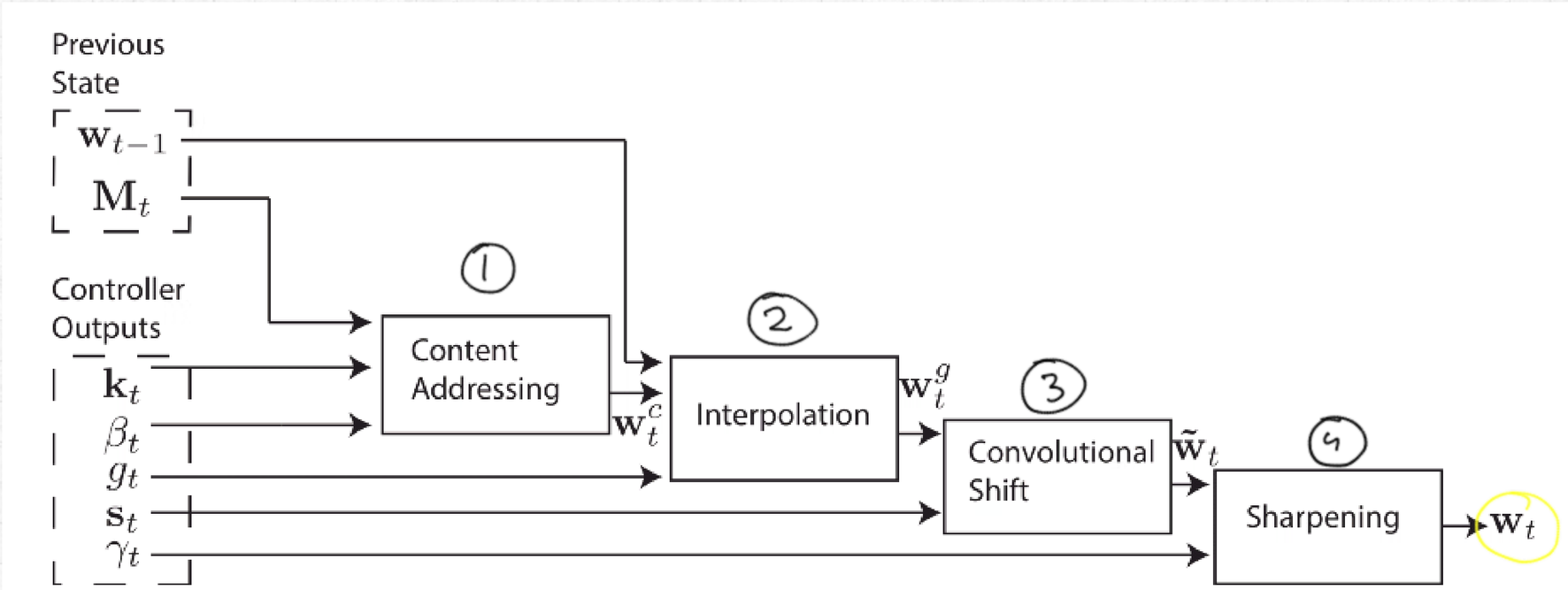
The convolution in the previous step can blur so we just boost thir values



④

Sharpening

The convolution in the previous step can blur so we just boost thir values

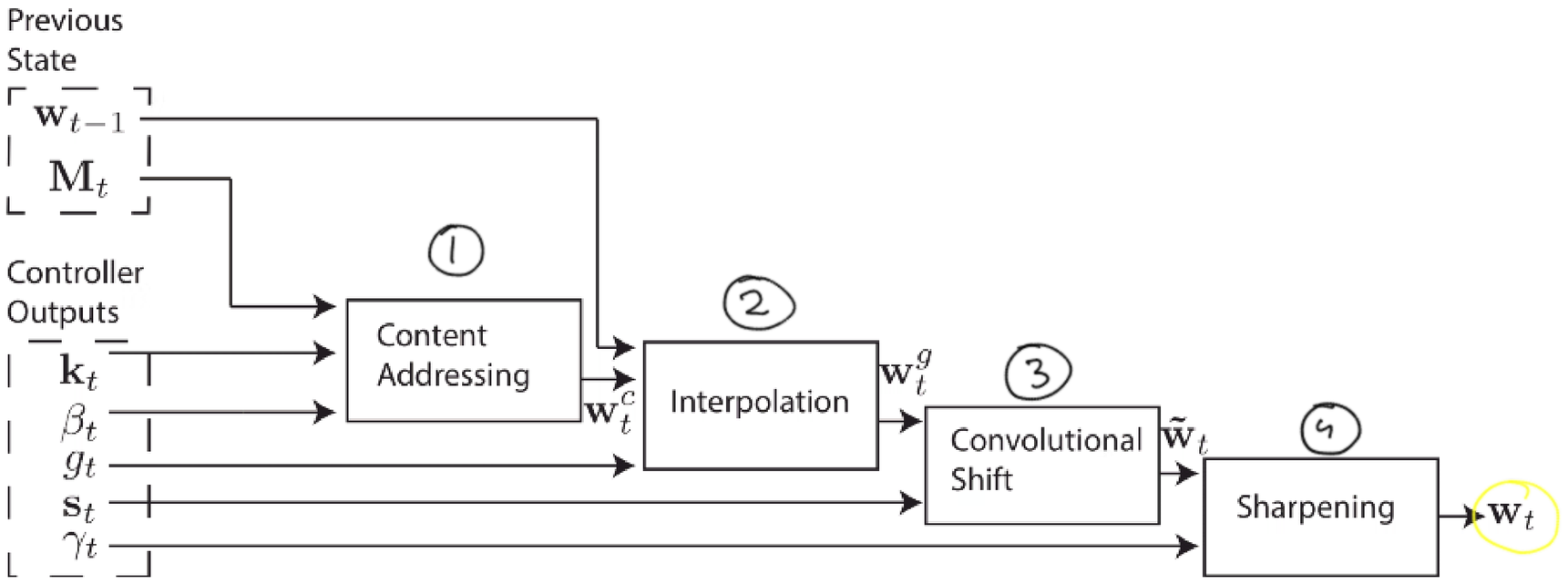


$$w_t(i) = \frac{w_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}}$$

④

Sharpening

The convolution in the previous step can blur so we just boost thir values



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

and finally obtain the address* of the memory Controller thinks we need.

* weight value for each memory location

The Controller

RNN vs. FFNN

The Controller

RNN

+ mix information across multiple steps

vs.

FFNN

+ easier to understand what is going on
- number of heads limits the types of computation

The Controller

RNN vs. FFNN

+ mix information across multiple steps

+ easier to understand what is going on
- number of heads limits the types of computation

RMSProp with momentum 0.9 in all experiments.

The Controller

RNN vs. FFNN

+ mix information across multiple steps

+ easier to understand what is going on

- number of heads limits the types of computation

RMSProp with momentum 0.9 in all experiments.

In the experiments the following controllers are compared:

NTM + FFNN vs LTSM vs NTM + LTSM

Copy (aka Identity) (4.1)

Task: copy 1 to 20 sequencies of 8 bits

Copy (aka Identity) (4.1)

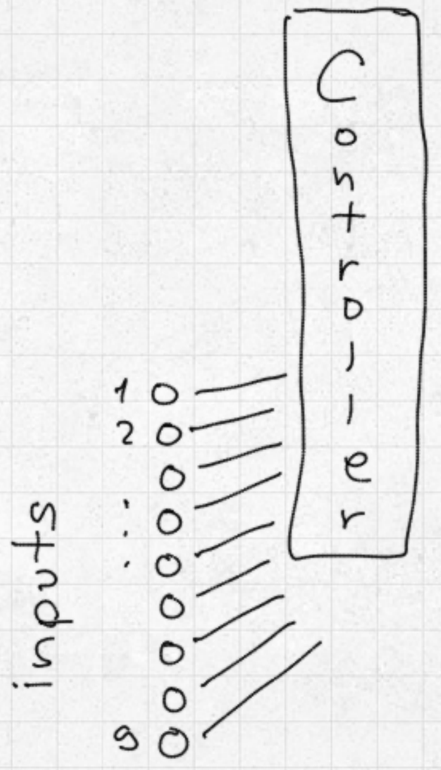
Task: copy 1 to 20 sequencies of 8 bits

inputs

1 00000000
2 ...
9 00000000

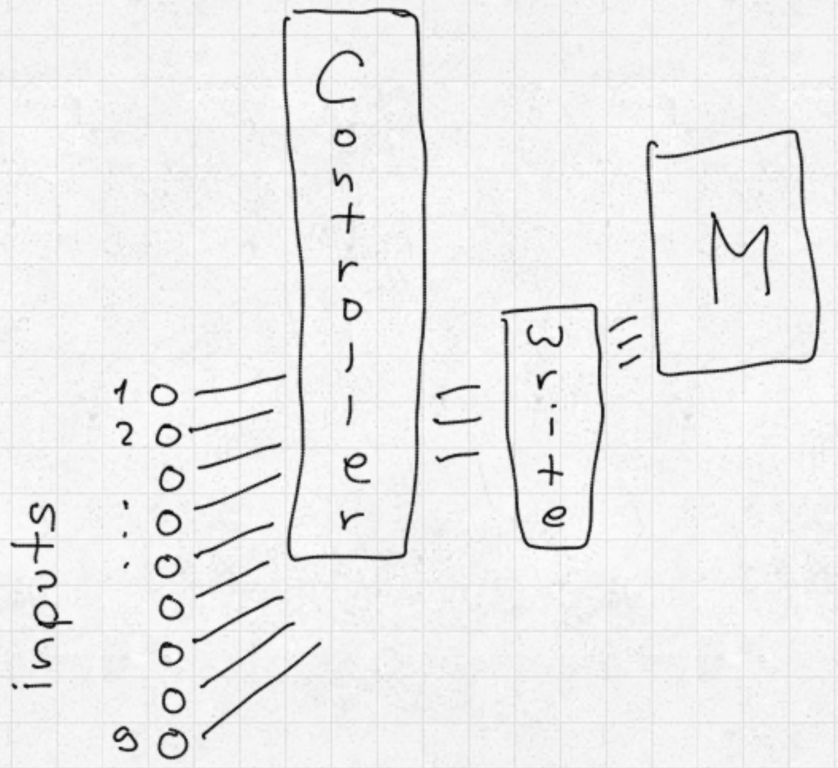
Copy (aka Identity) (4.1)

Task: copy 1 to 20 sequencies of 8 bits



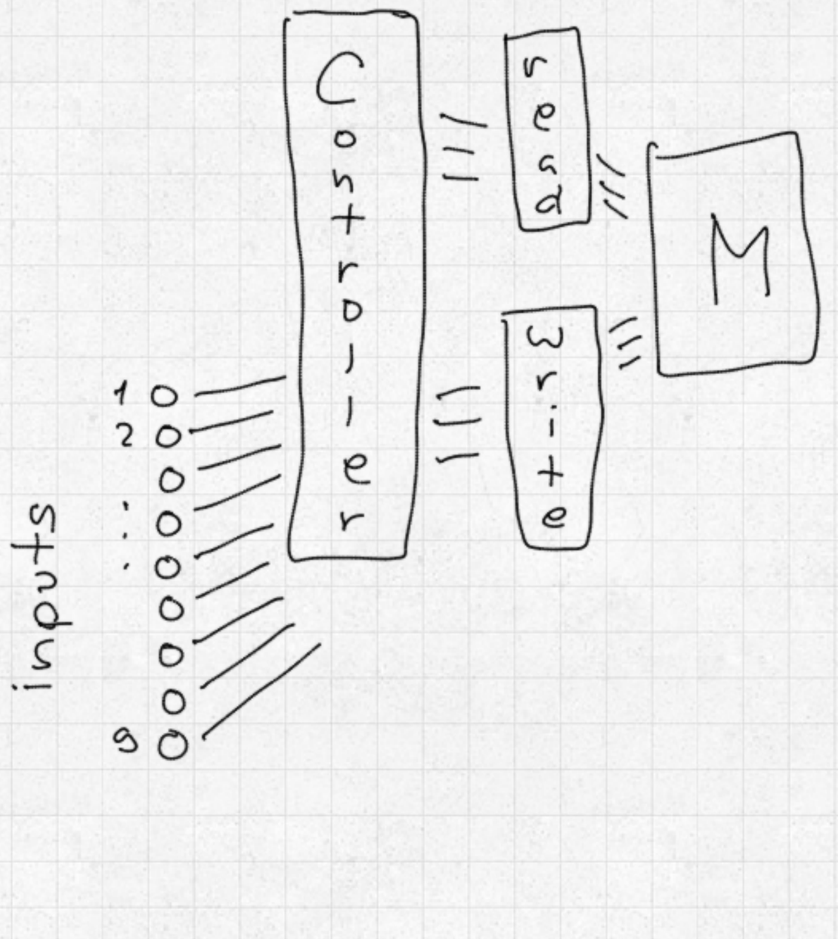
Copy (aka Identity) (4.1)

Task: copy 1 to 20 sequencies of 8 bits



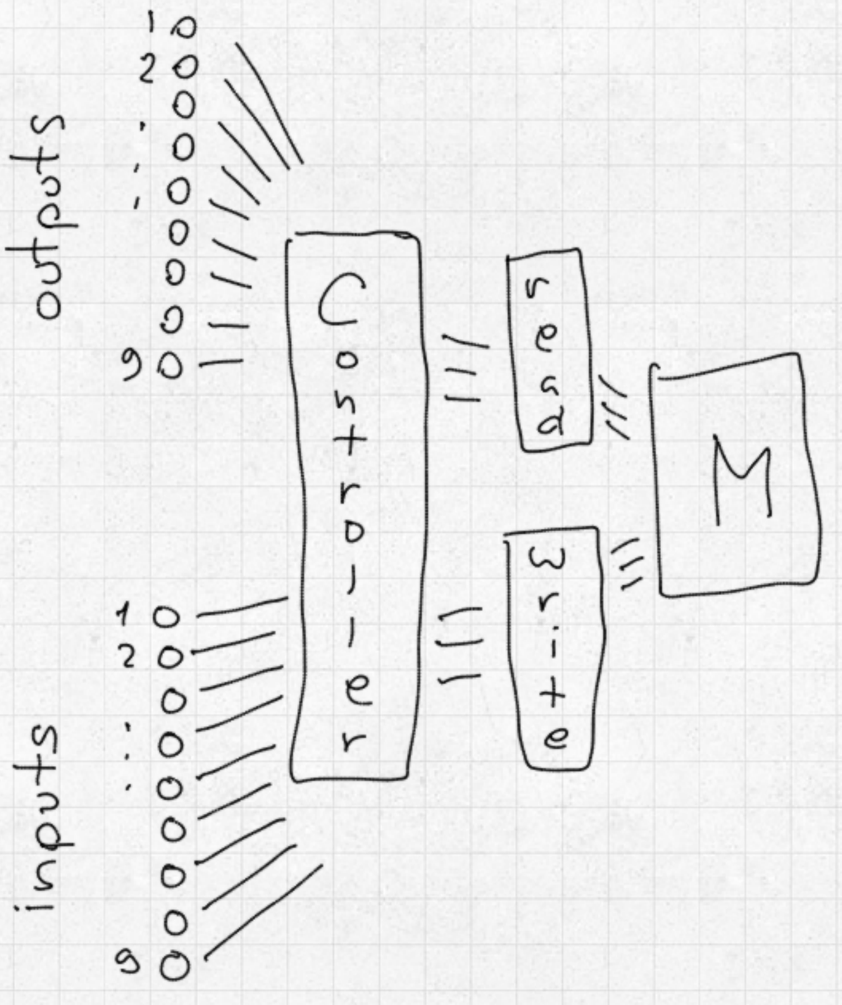
Copy (aka Identity) (4.1)

Task: copy 1 to 20 sequencies of 8 bits



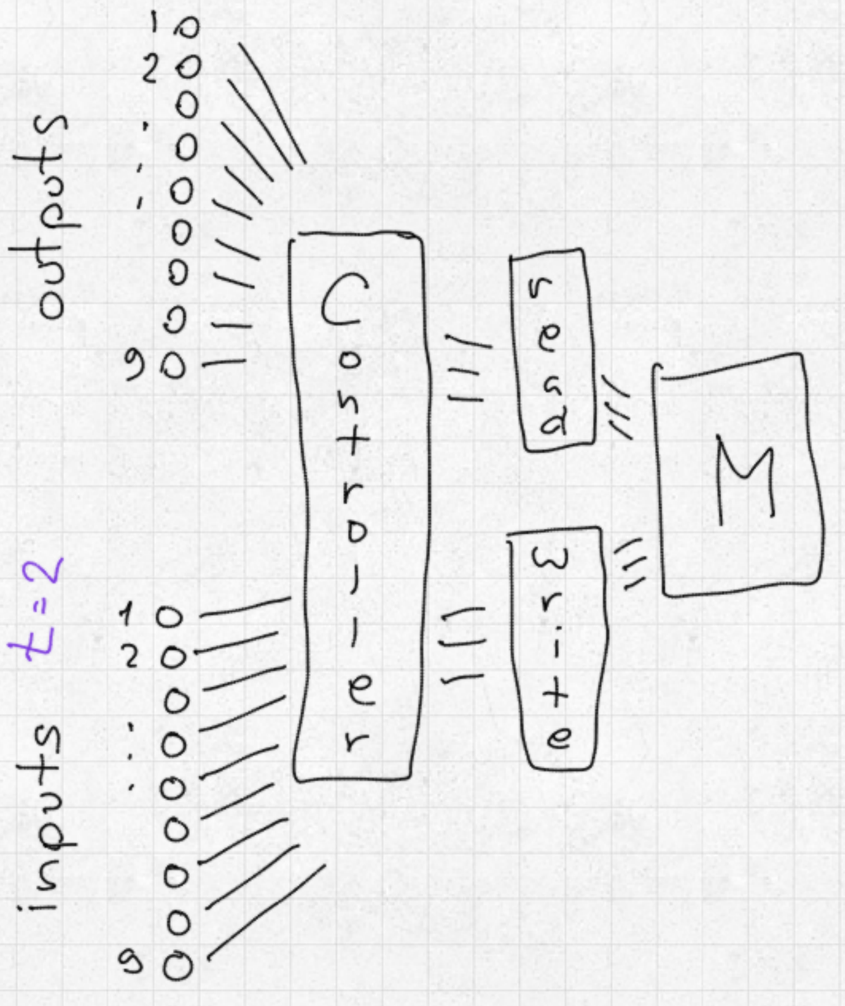
Copy (aka Identity) (4.1)

Task: copy 1 to 20 sequencies of 8 bits



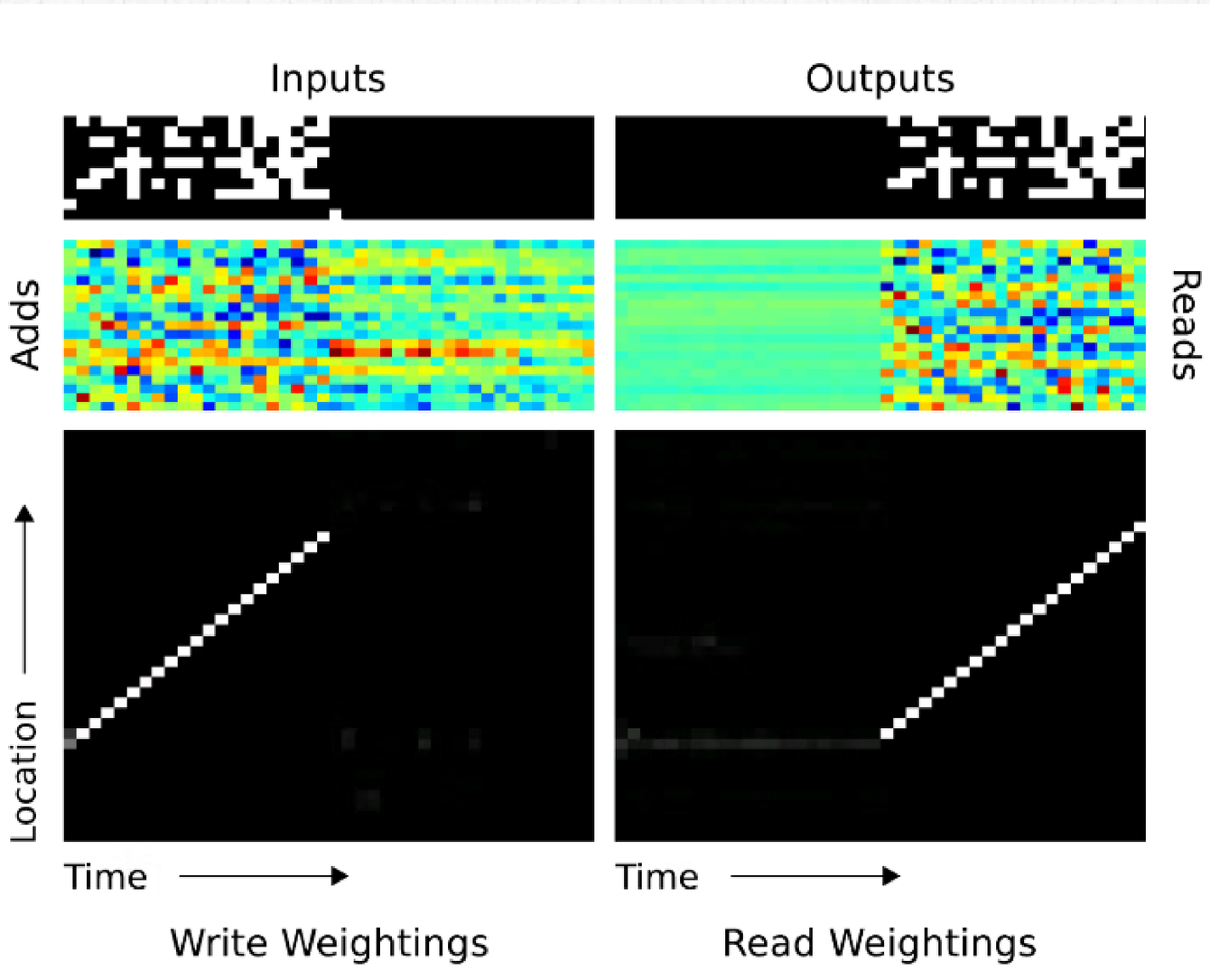
Copy (aka Identity) (4.1)

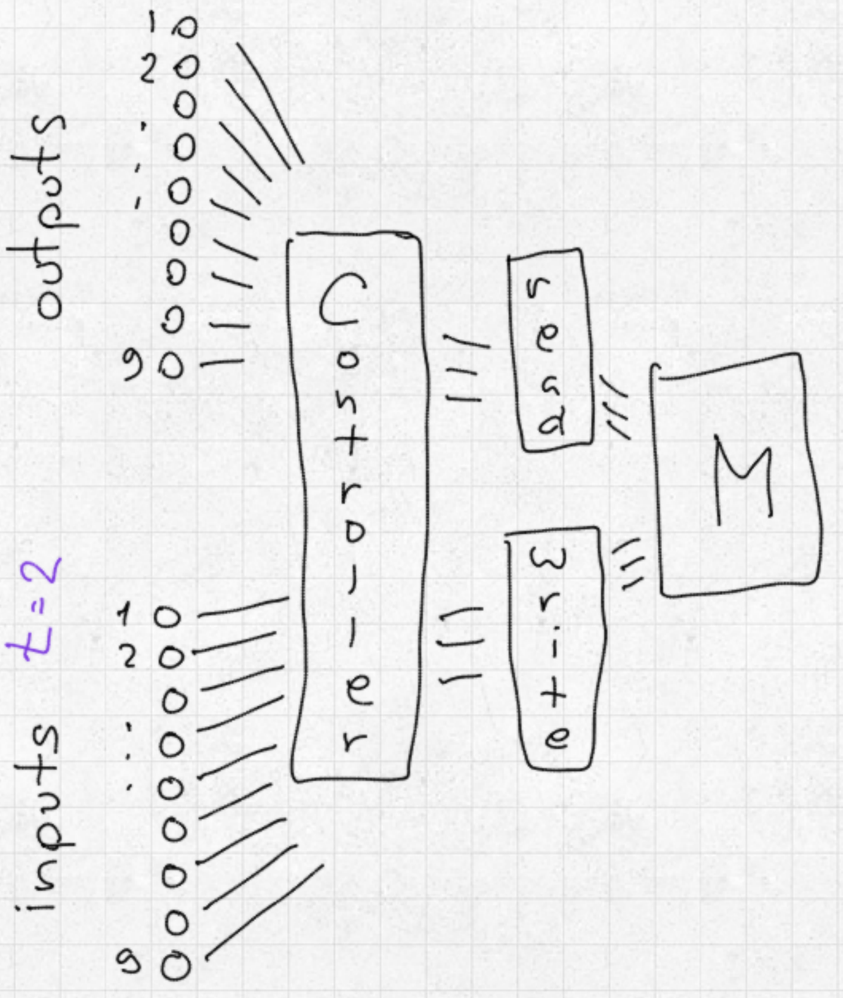
Task: copy 1 to 20 sequencies of 8 bits



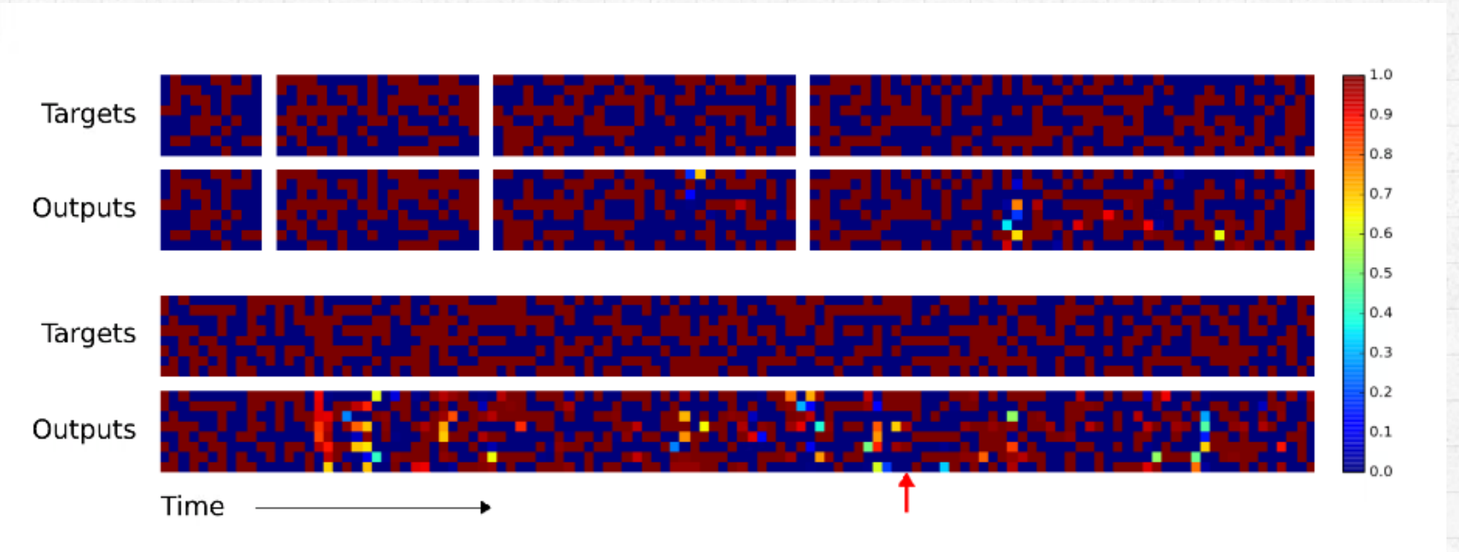
Copy (aka Identity) (4.1)

Task: copy 1 to 20 sequencies of 8 bits



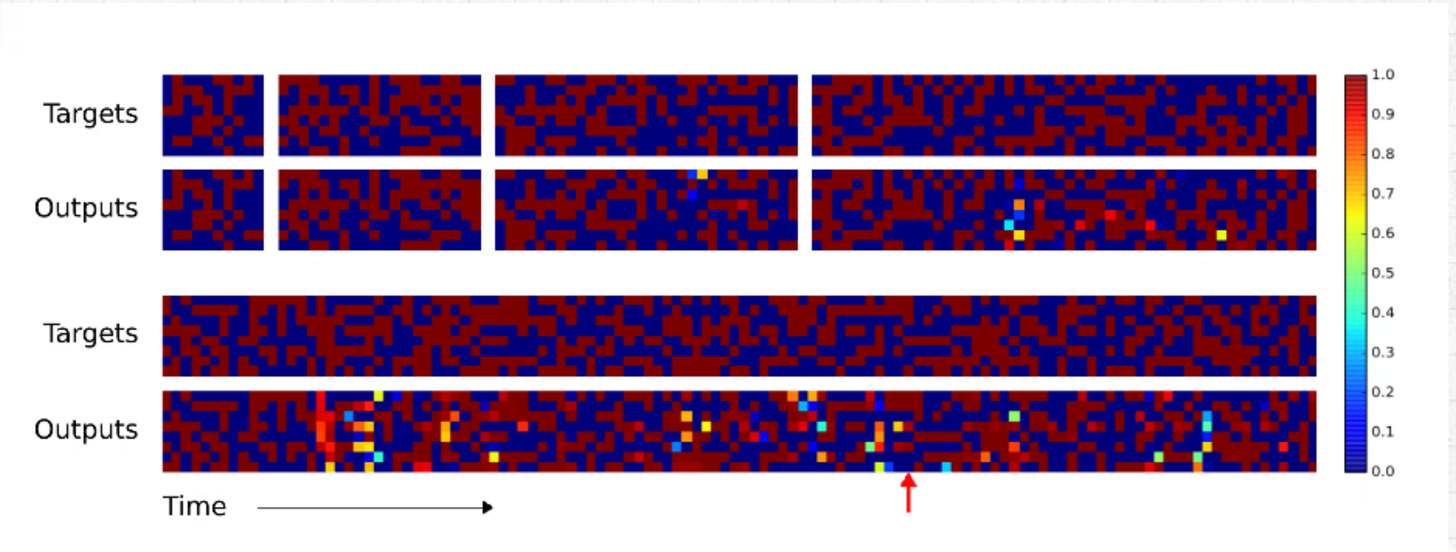


Copy (aka Identity) (4.1)

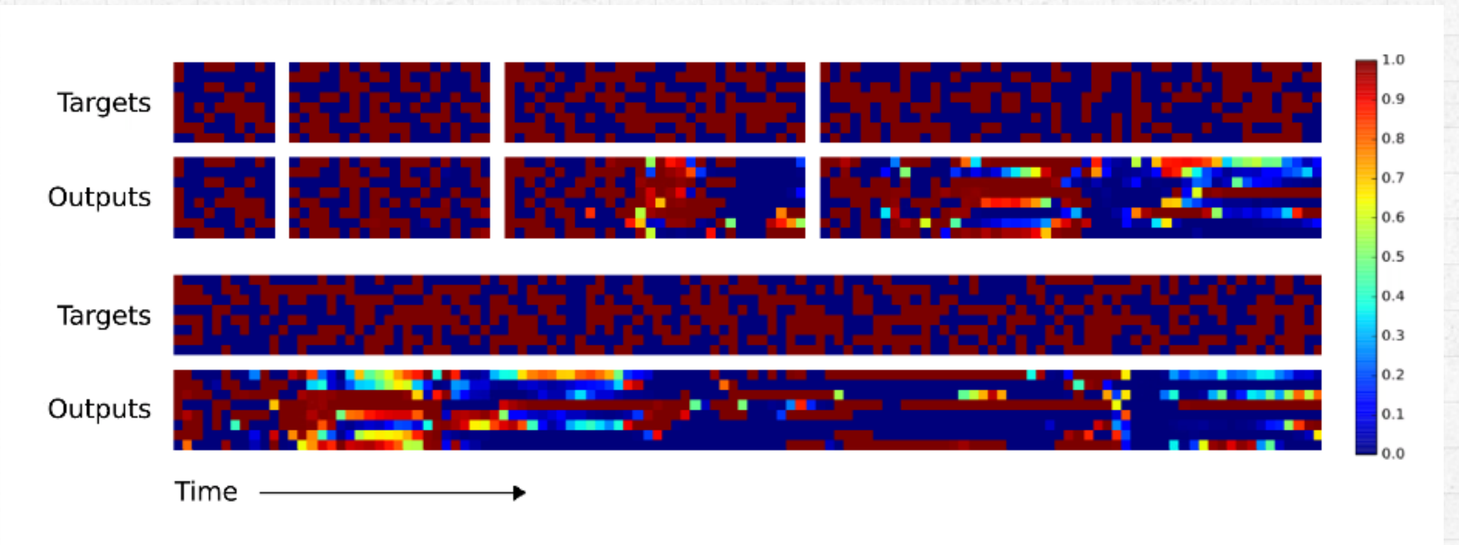


NTM

Copy (aka Identity) (4.1)



NTM



LSTM

Copy (aka Identity) (4.1)

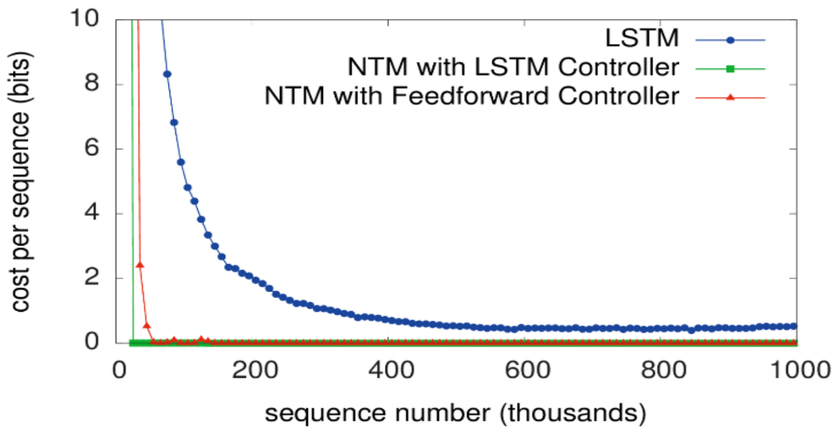
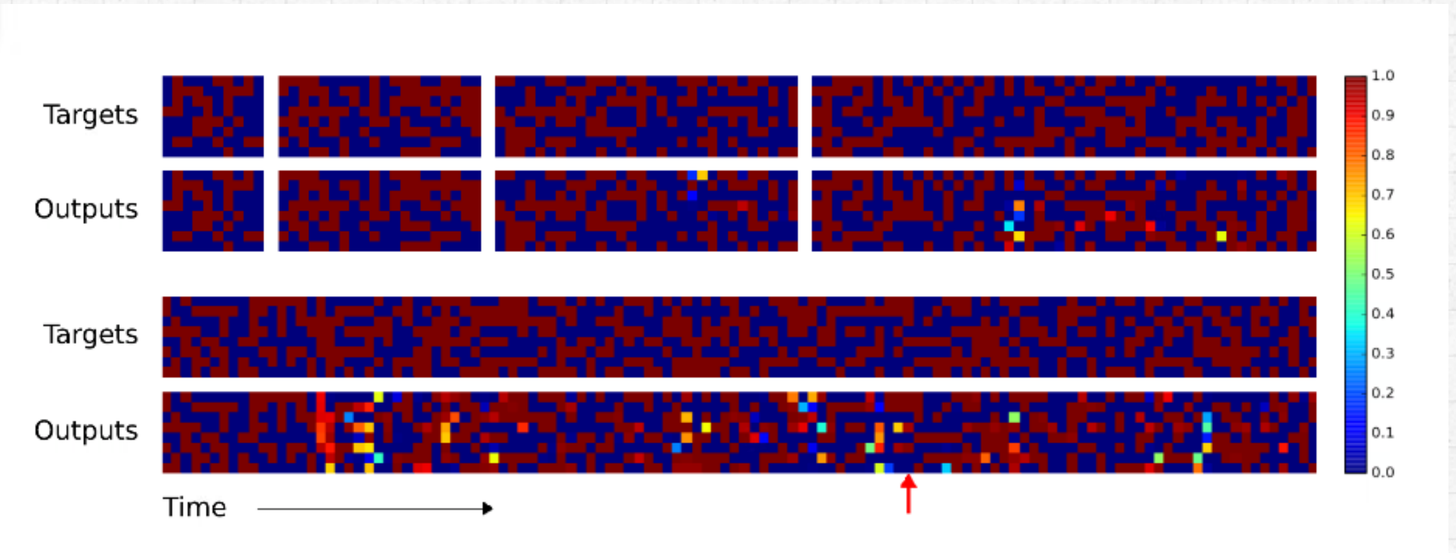
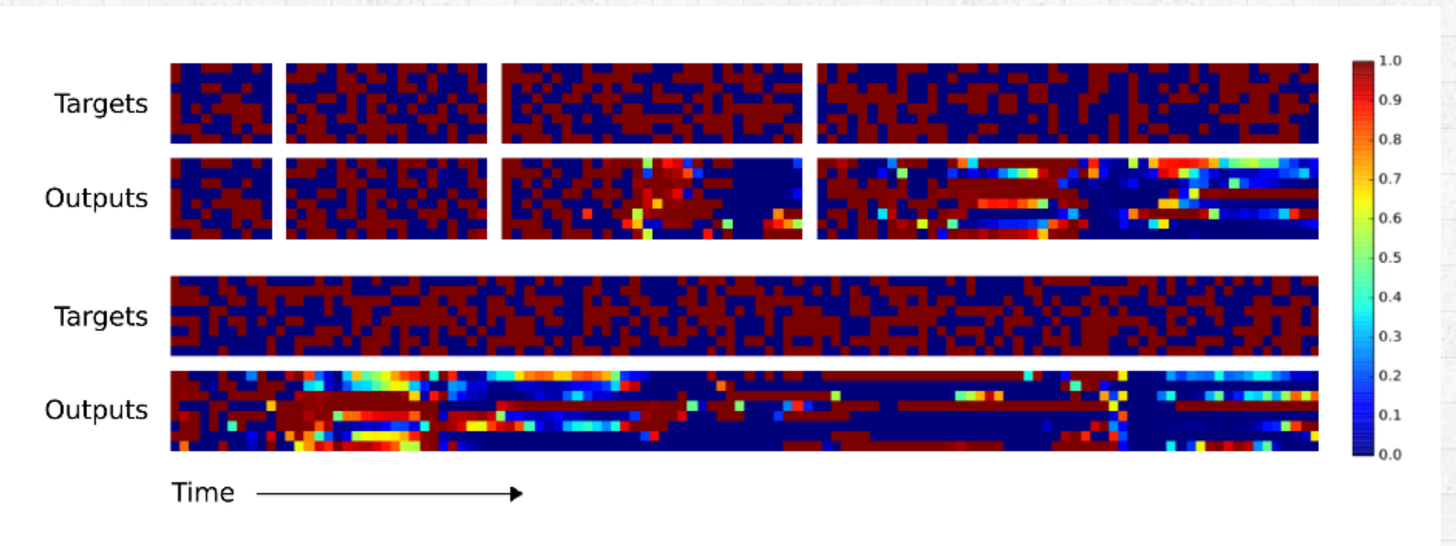


Figure 3: Copy Learning Curves.



NTM



LSTM

Copy (aka Identity) (4.1)

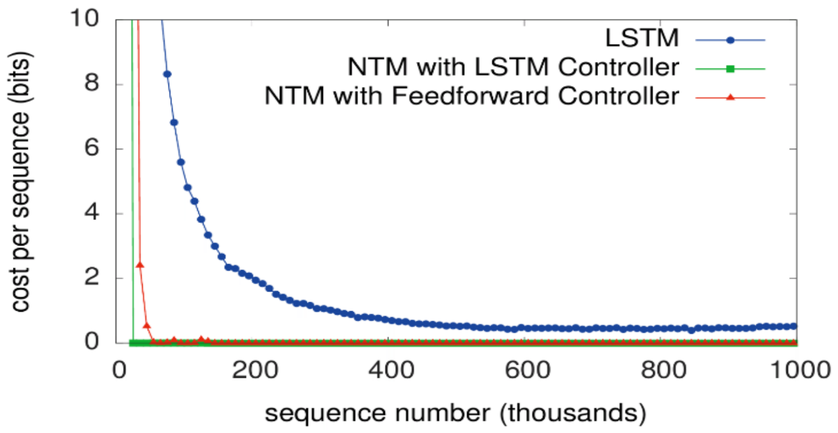
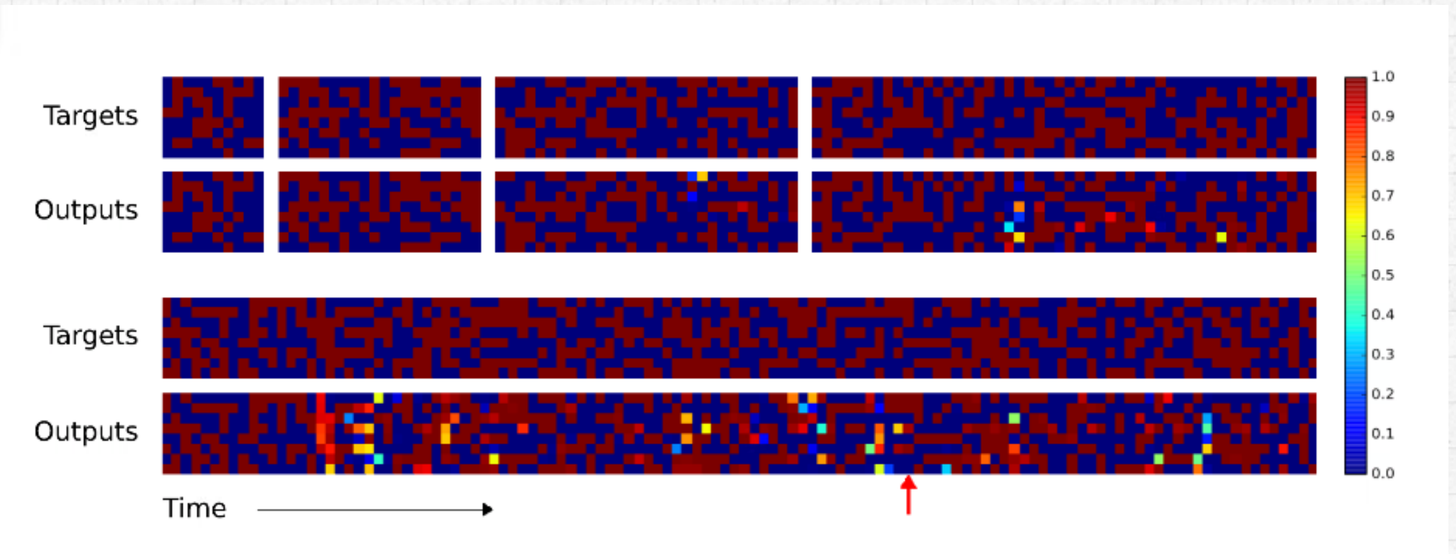
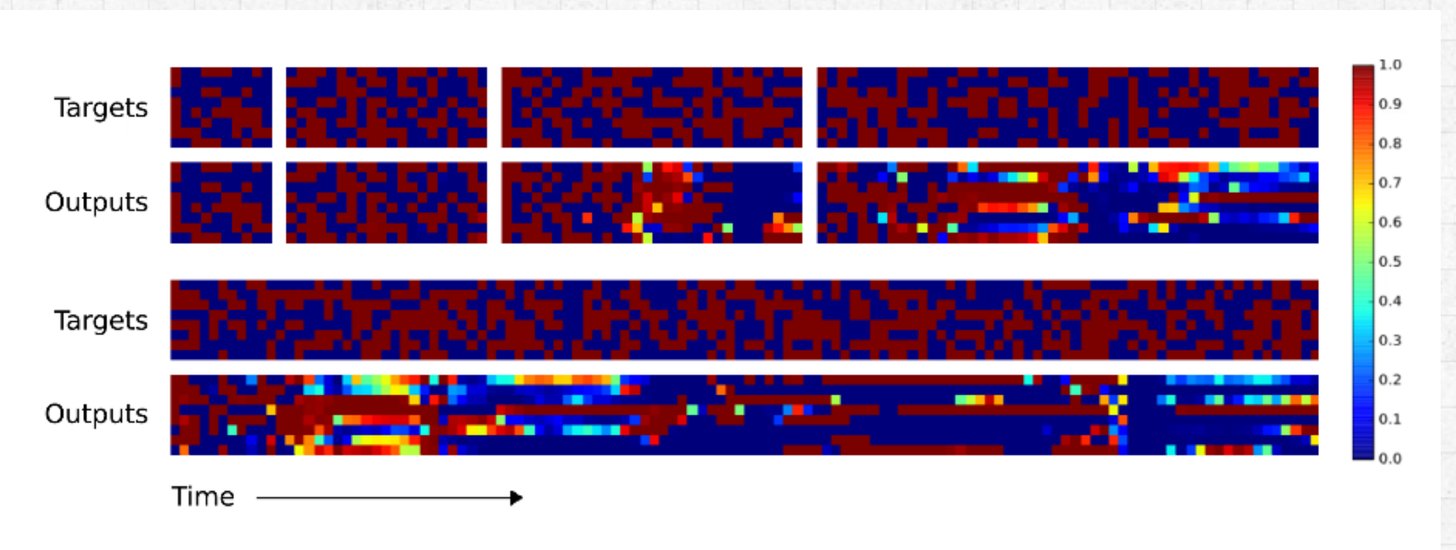


Figure 3: Copy Learning Curves.

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



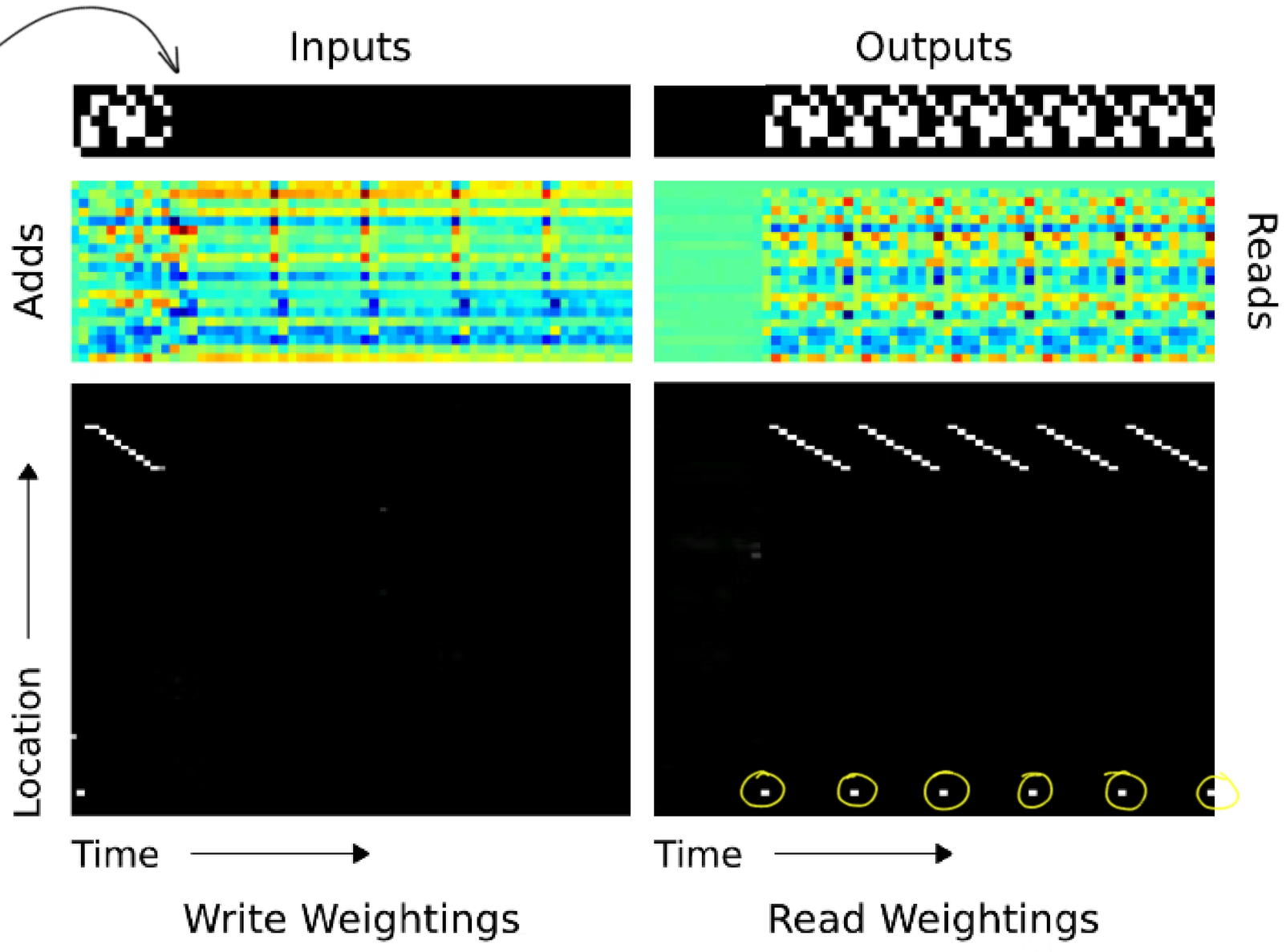
NTM



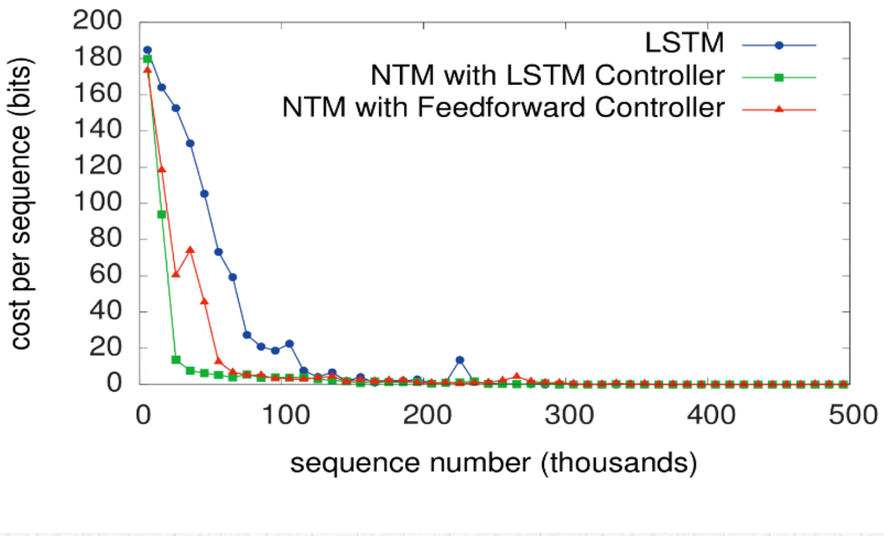
LSTM

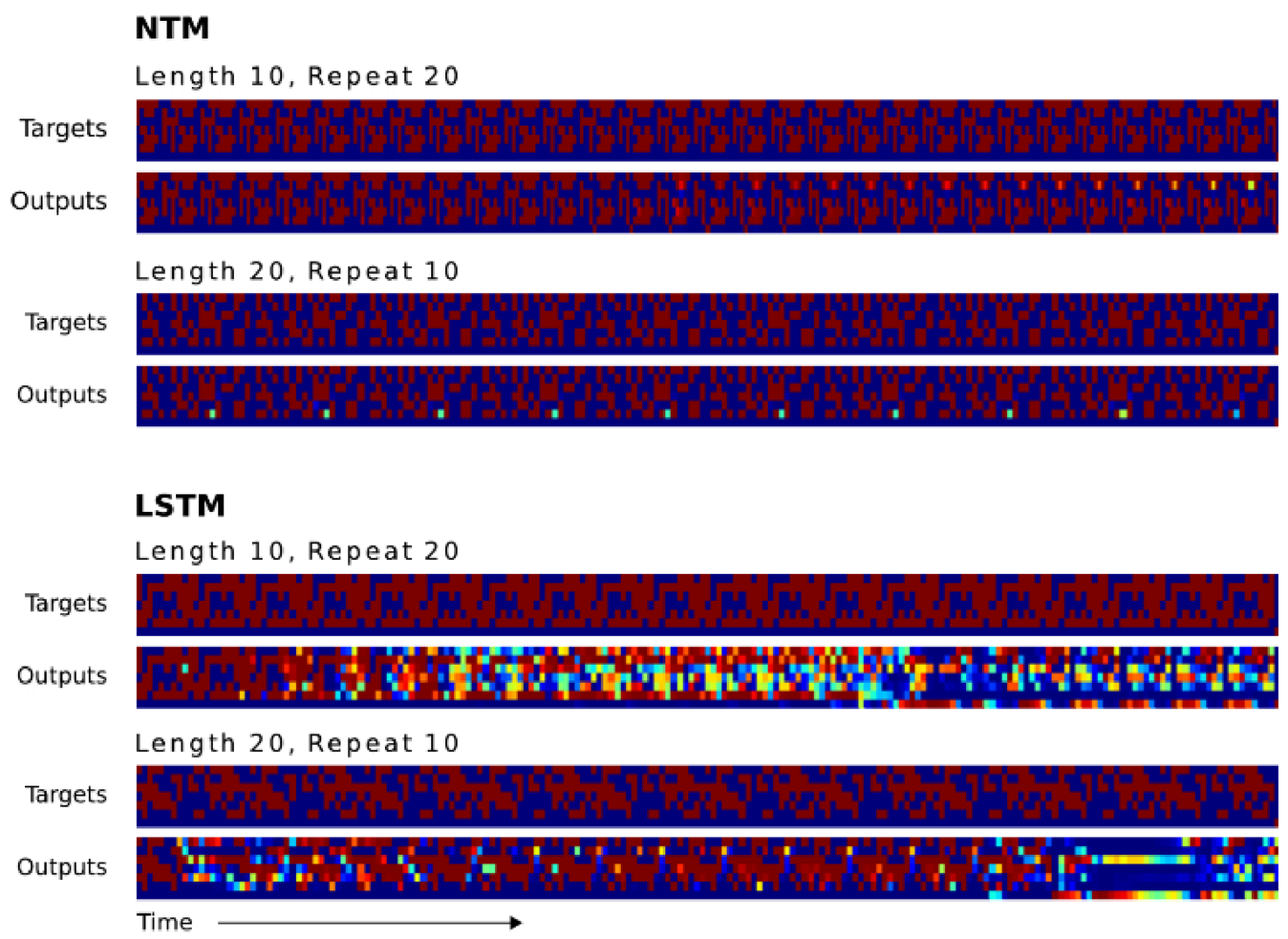
Repeat copy (aka For-loop) (4.2)

additional input: number of repeats

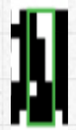


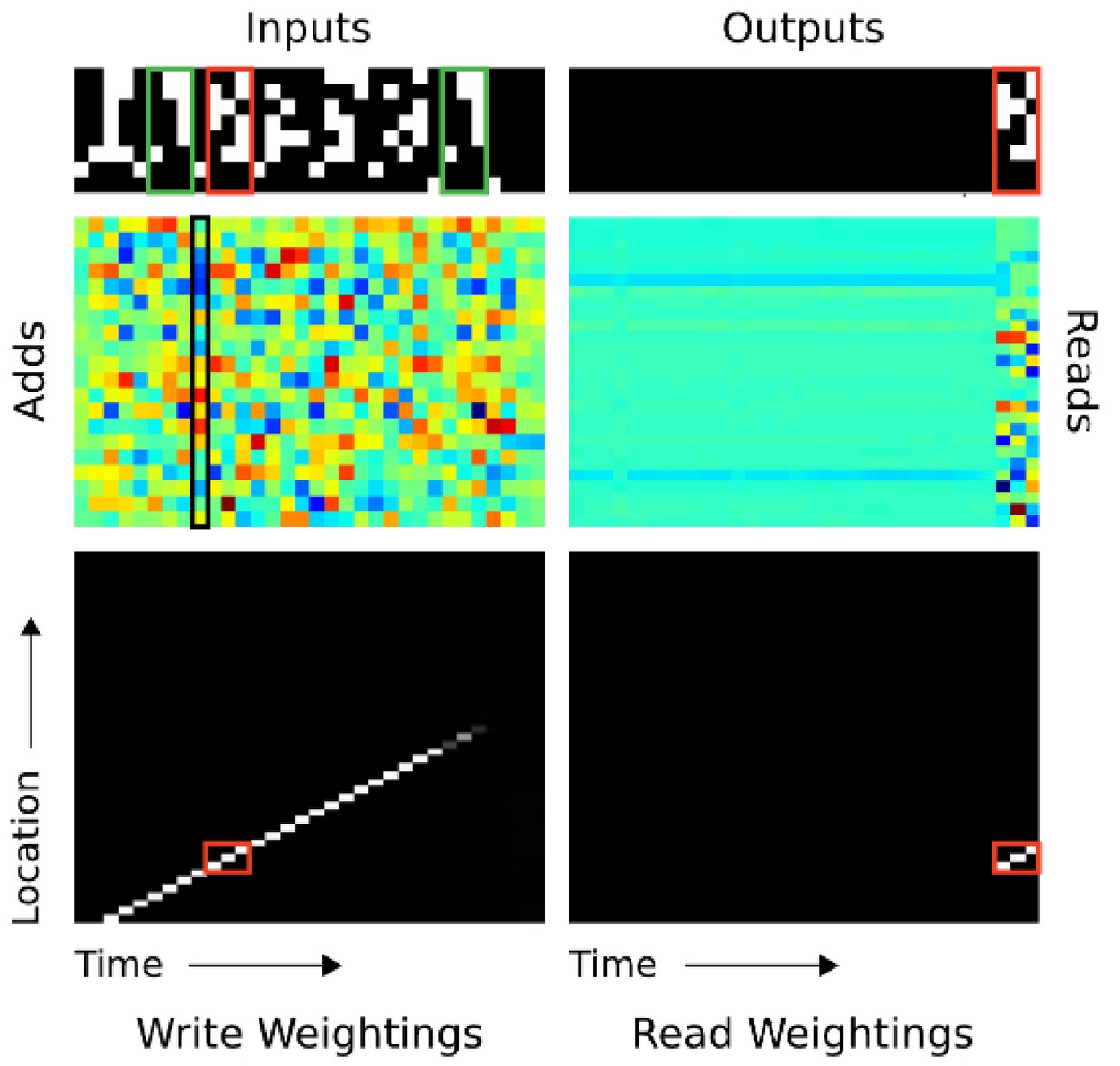
Repeat copy (aka For-loop) (4.2)





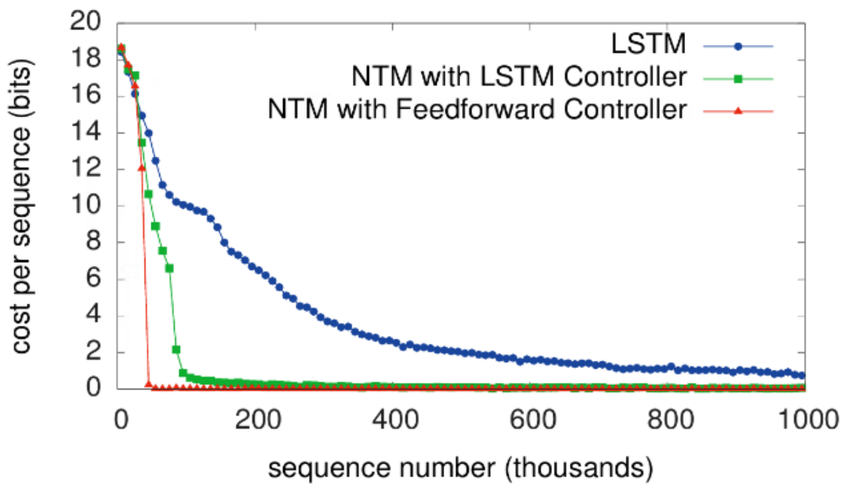
Associative Recall (4.3)

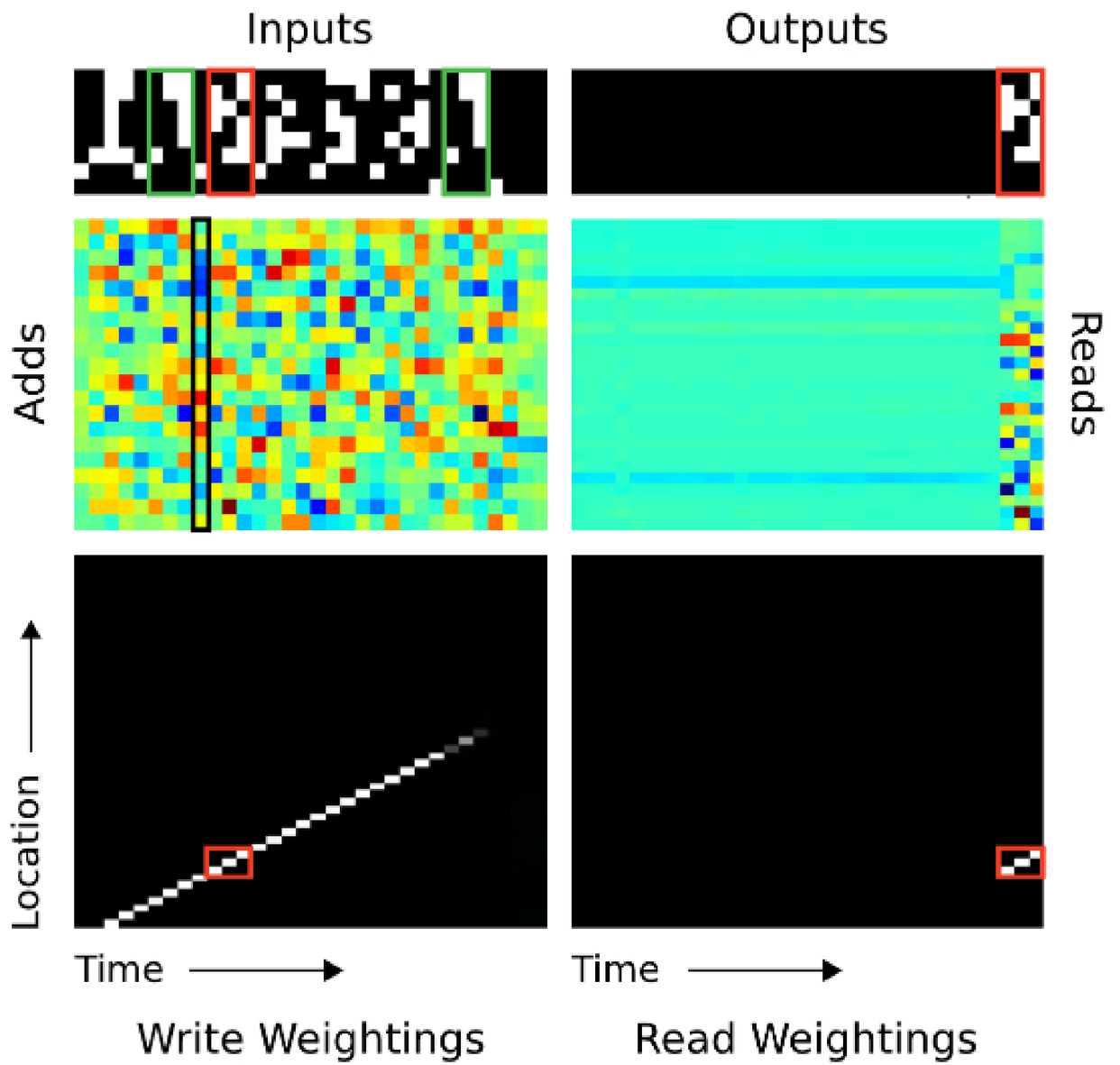
What was the item after  ?



Associative Recall (4.3)

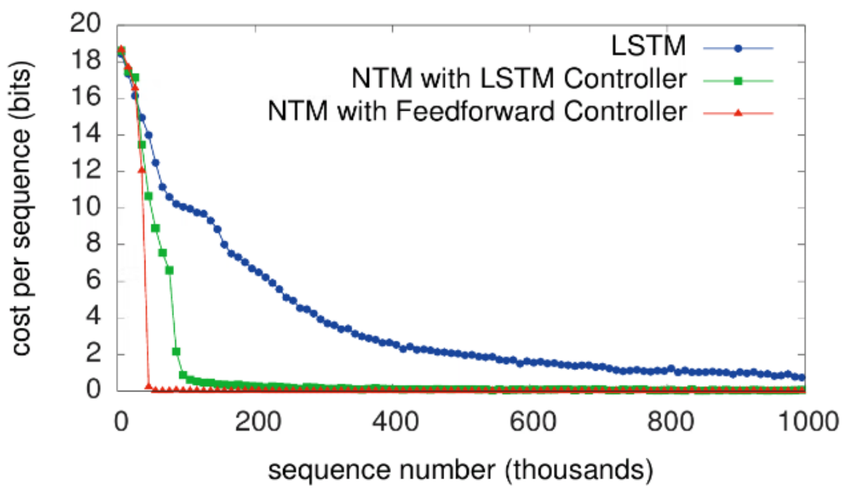
What was the item after  ?



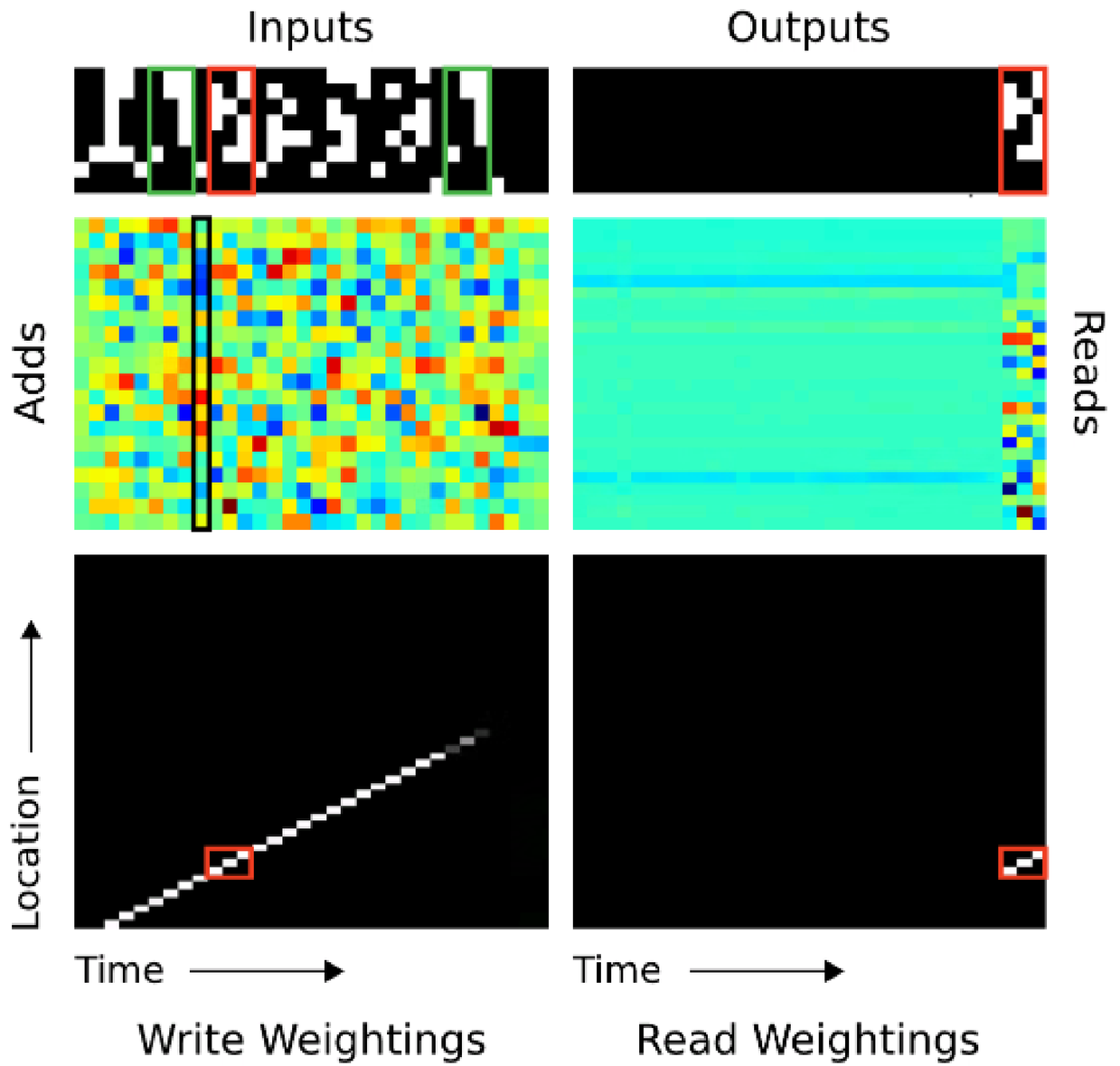


Associative Recall (4.3)

What was the item after  ?



Content addressing and the shift via location addressing



Dynamic N-Grams (4.4)

6-Grams: given 01101 what is the probability that next is 1?

Dynamic N-Grams (4.4)

6-Grams: given 01101 what is the probability that next is 1?

Knowledge is stored as 32-element vector of probabilities.

Dynamic N-Grams (4.4)

6-Grams: given 01101 what is the probability that next is 1?

Knowledge is stored as 32-element vector of probabilities.

Updated based on observations.

Dynamic N-Grams (4.4)

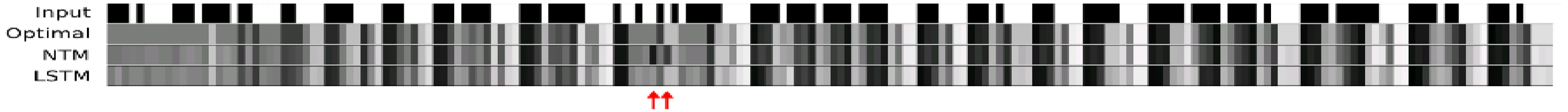
6-Grams: given 01101 what is the probability that next is 1?

Knowledge is stored as 32-element vector of probabilities.

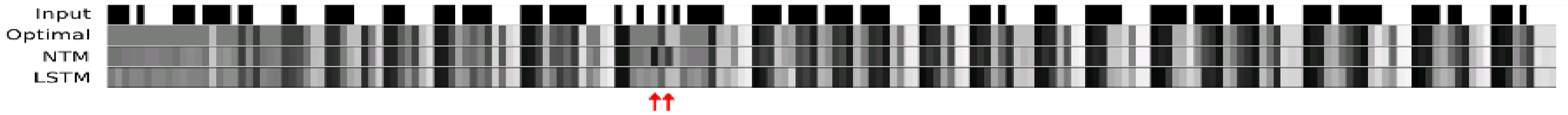
Updated based on observations.

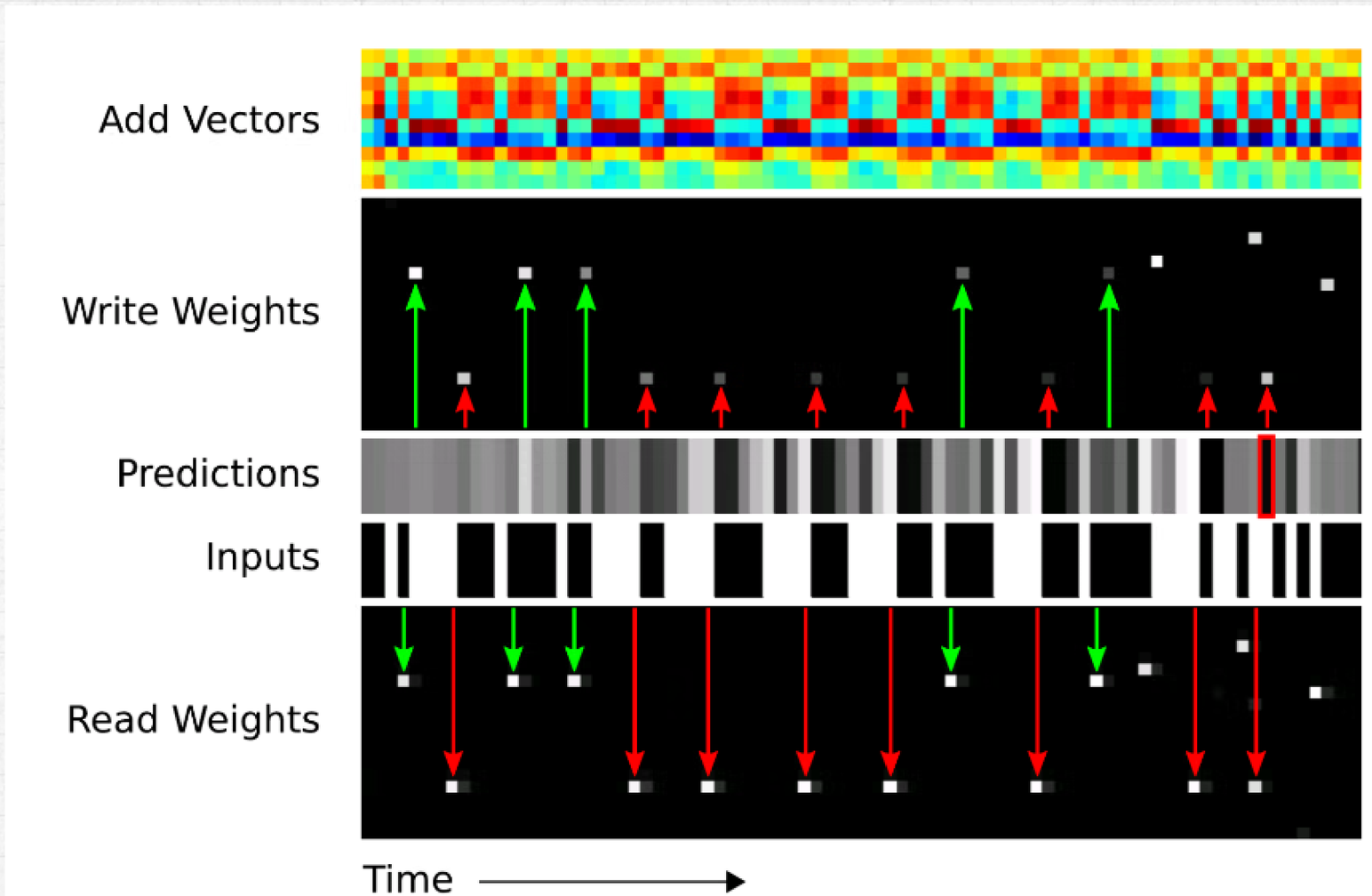
Training : 200 Successive bits
Testing : 1000 of 200-bit sequencies

Dynamic N-Grams (4.4)

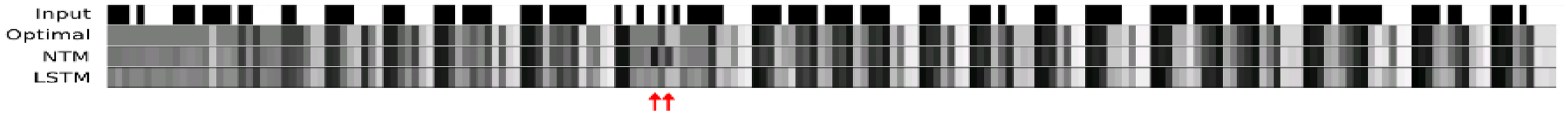


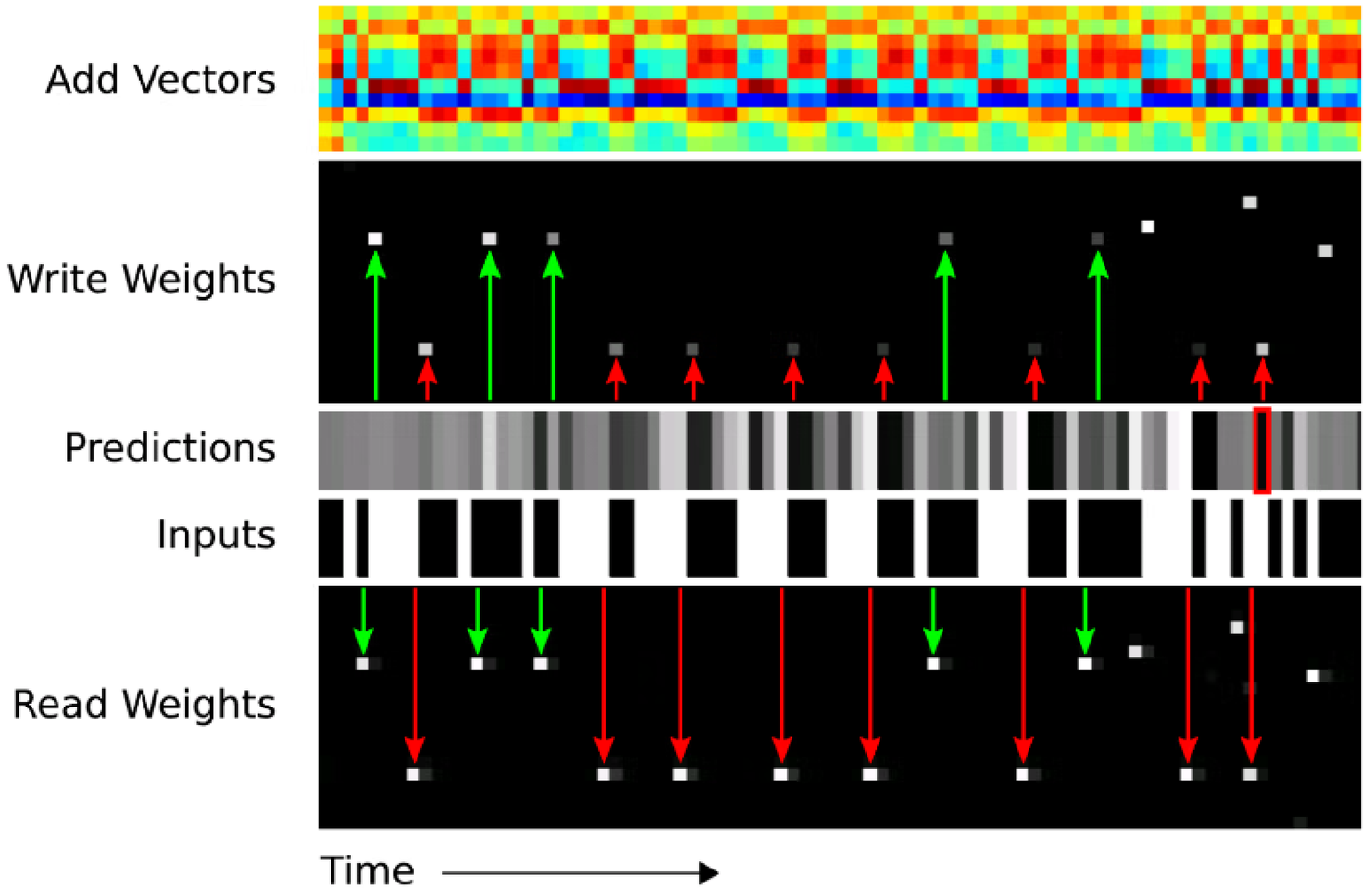
Dynamic N-Grams (4.4)





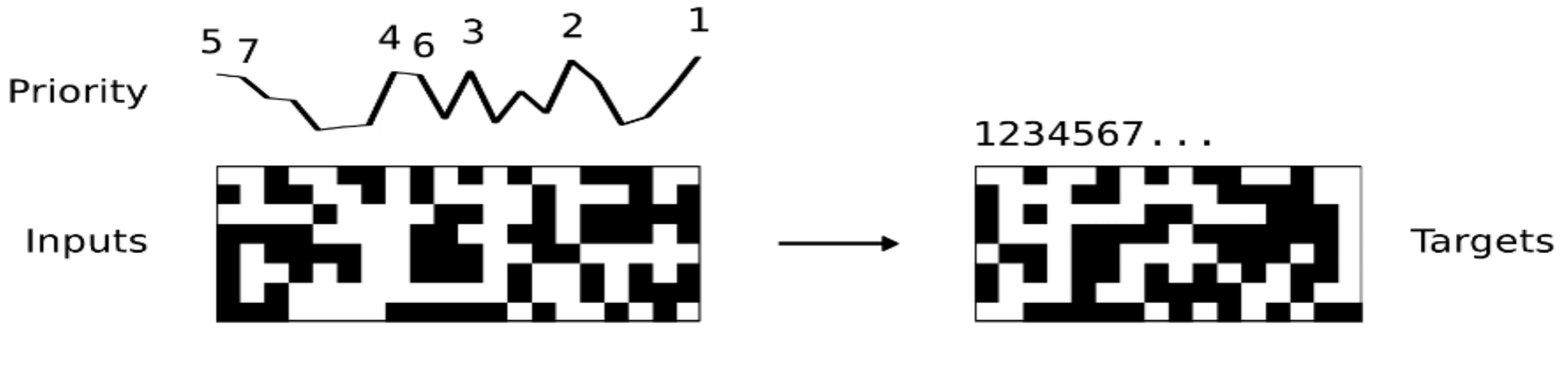
Dynamic N-Grams (4.4)



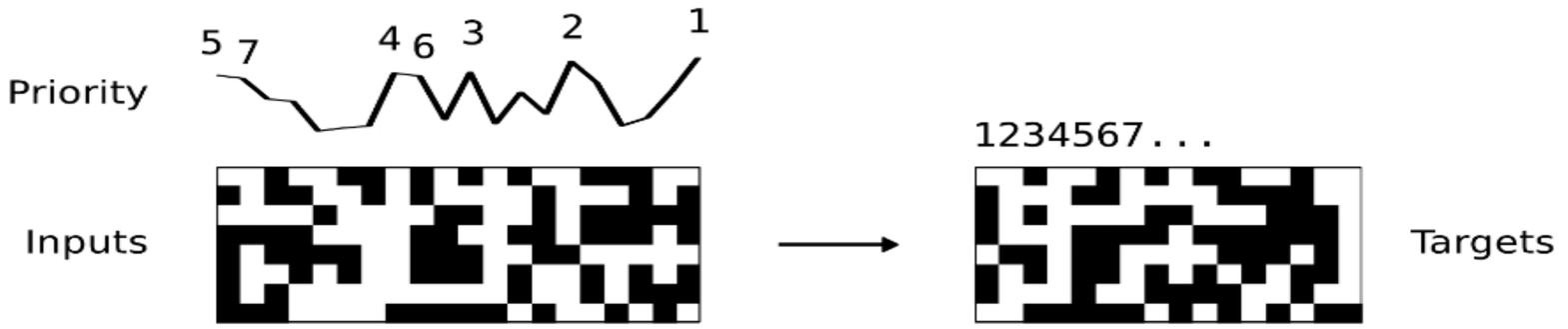


It seems that the network count "1"s by writing after each 5 bits

Priority Sort (4.5)



Priority Sort (4.5)



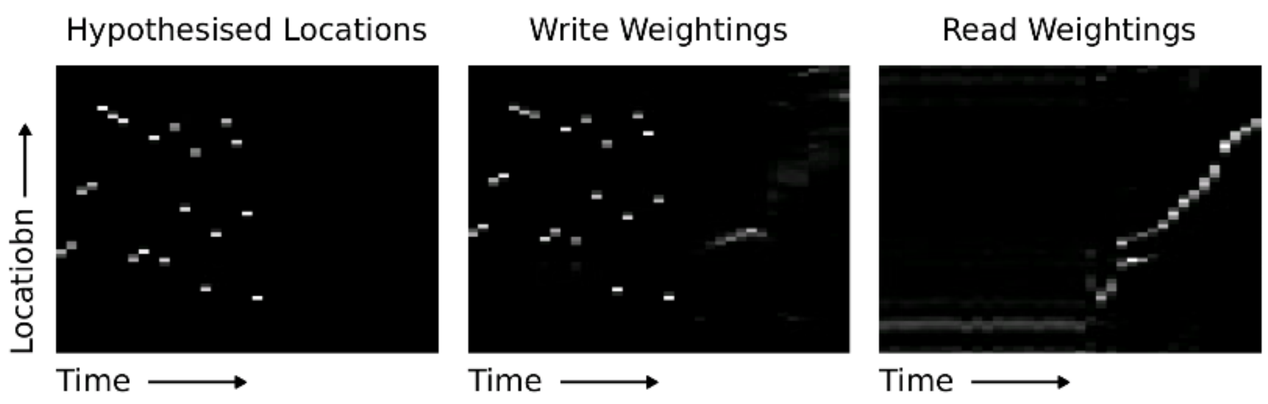
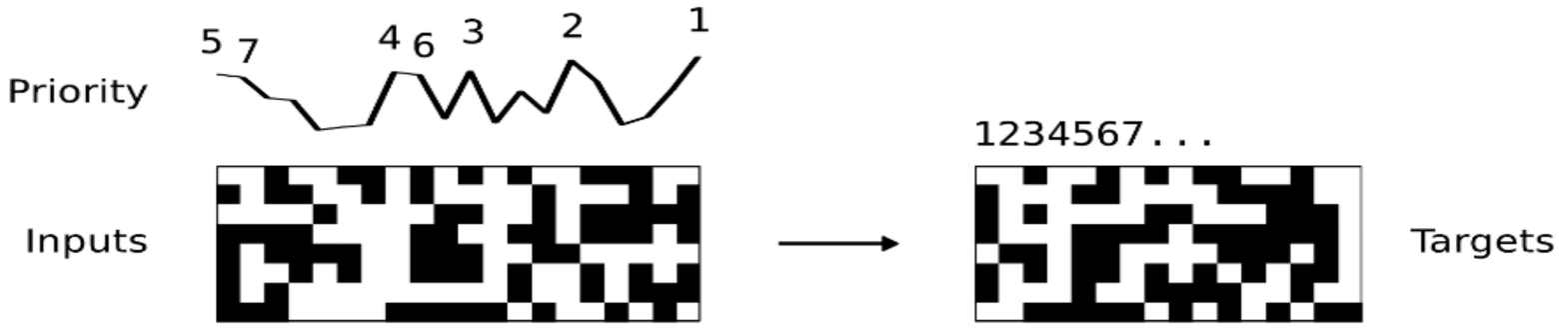


Figure 17: NTM Memory Use During the Priority Sort Task. Left: Write locations returned by fitting a linear function of the priorities to the observed write locations. Middle: Observed write locations. Right: Read locations.

Priority Sort (4.5)



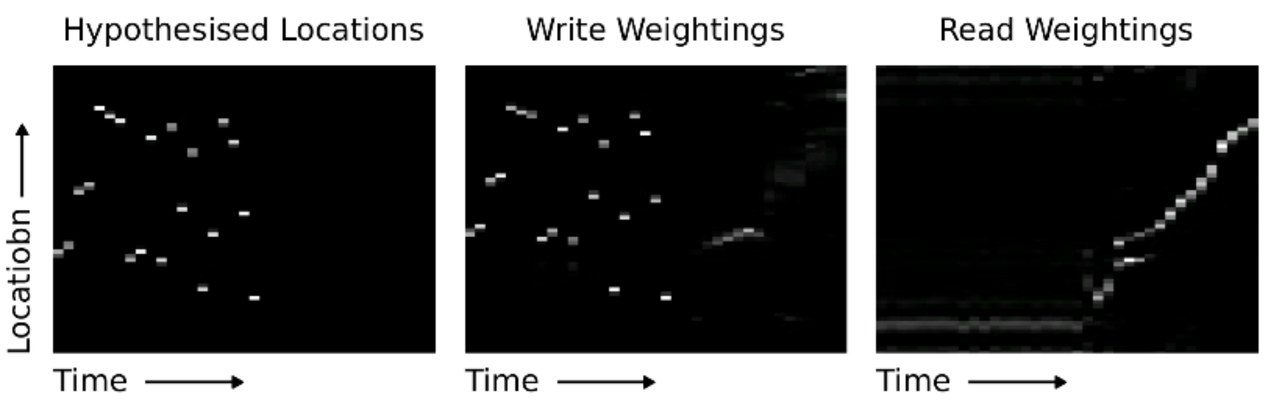
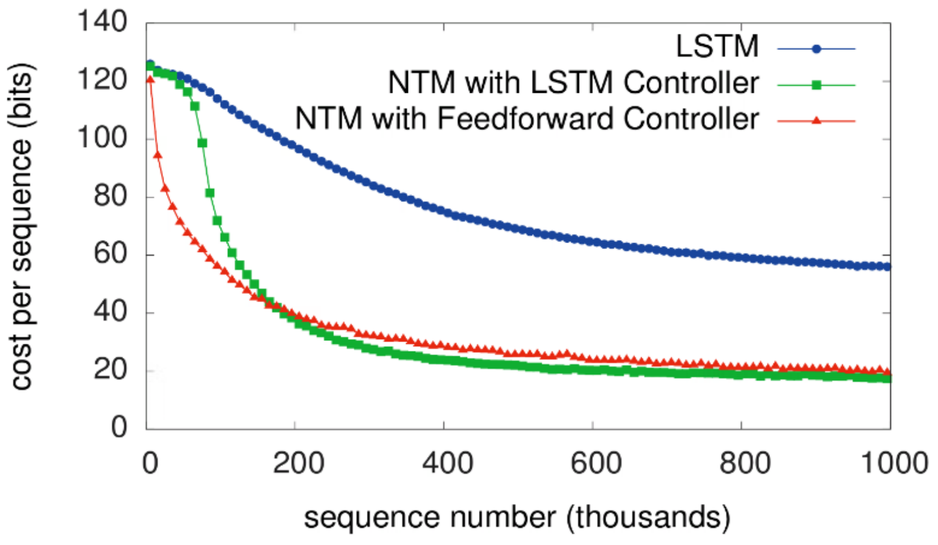


Figure 17: NTM Memory Use During the Priority Sort Task. Left: Write locations returned by fitting a linear function of the priorities to the observed write locations. Middle: Observed write locations. Right: Read locations.



Demo

github.com/shawtan/neural-turing-machines

blog.wtf.sg/category/neural-turing-machines