Neural Turing Machine

Author: Alex Graves, Greg Wayne, Ivo Danihelka

Presented By: Tinghui Wang (Steve)

WASHINGTON STATE UNIVERSITY

World Class. Face to Face.

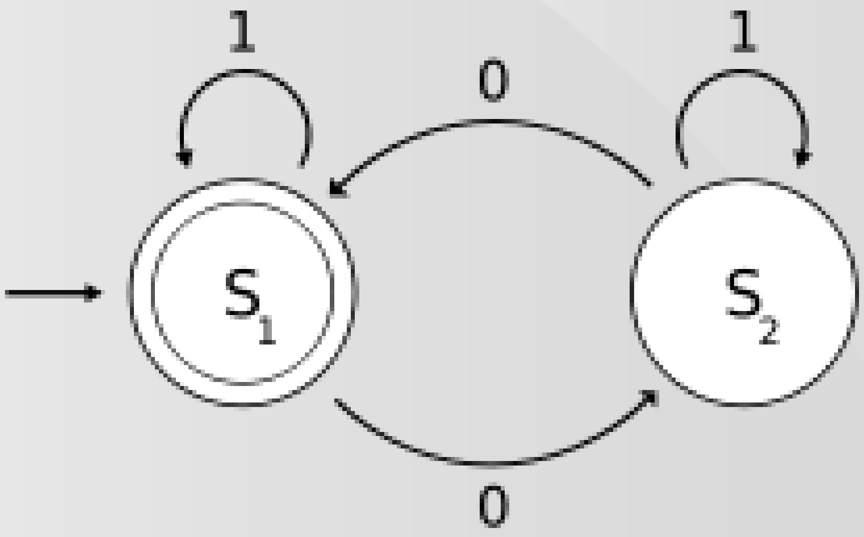
Introduction

- Neural Turning Machine: Couple a Neural Network with external memory resources
- The combined system is analogous to TM, but differentiable from end to end
- NTM Can infer simple algorithms!

In Computation Theory…

Finite State Machine
$(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{\delta}, \boldsymbol{F})$

- $\boldsymbol{\Sigma}$: Alphabet
- $\boldsymbol{S}$: Finite set of states
- $\boldsymbol{s_0}$: Initial state
- $\boldsymbol{\delta}$: state transition function:
  $\boldsymbol{S} \times \boldsymbol{\Sigma} \rightarrow \boldsymbol{P}(\boldsymbol{S})$
- $\boldsymbol{F}$: set of Final States

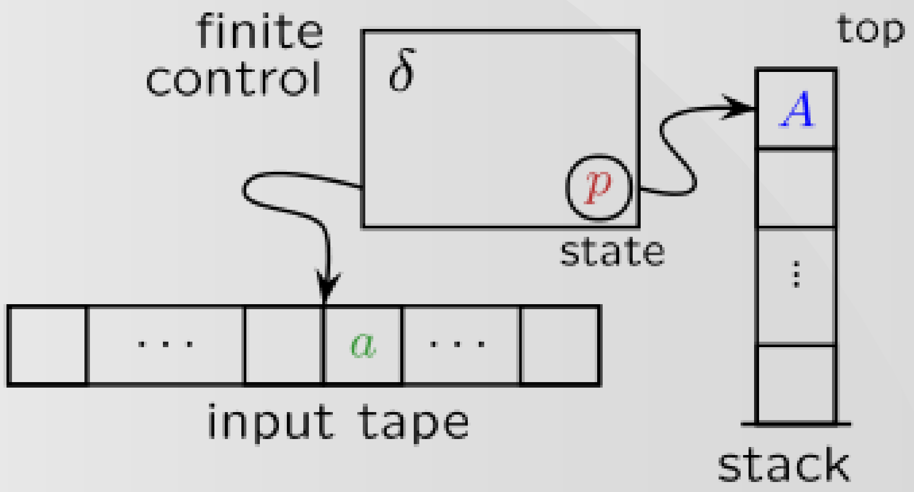


In Computation Theory…

Push Down Automata

$(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$

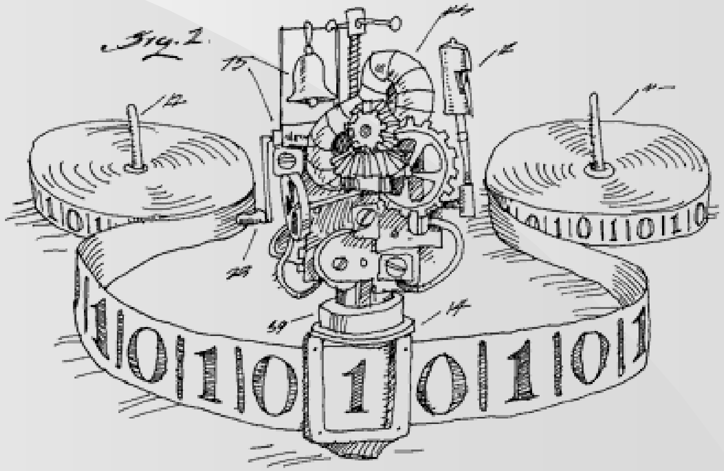
- Σ: Input Alphabet
- Γ: Stack Alphabet
- Q: Finite set of states
- q_0: Initial state
- δ: state transition function:
  $Q \times \Sigma \times \Gamma \rightarrow P(Q) \times \Gamma$
- F: set of Final States



Turing Machine

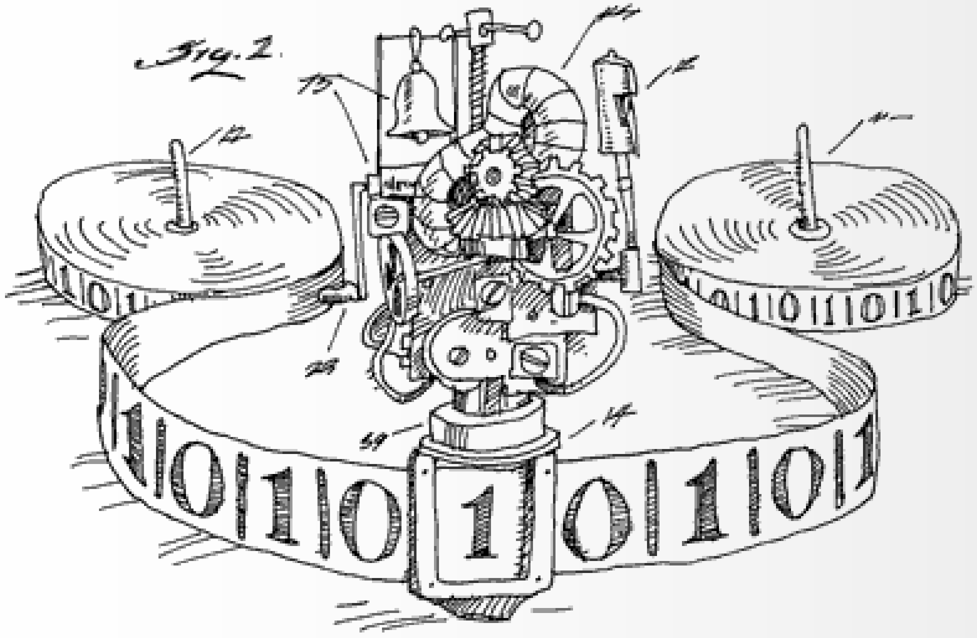
$$(\boldsymbol{Q}, \boldsymbol{\Gamma}, \boldsymbol{b}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$$

- $\boldsymbol{Q}$: Finite set of States
- $\boldsymbol{q_0}$: Initial State
- $\boldsymbol{\Gamma}$: Tape alphabet (Finite)
- $\boldsymbol{b}$: Blank Symbol (occurs infinitely on the tape)
- $\boldsymbol{\Sigma}$: Input Alphabet ($\Sigma = \boldsymbol{\Gamma} \backslash \{b\}$)
- $\boldsymbol{\delta}$: Transition Function
  $(Q \backslash \mathrm{F}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- $\boldsymbol{F}$: Final State (Accepted)

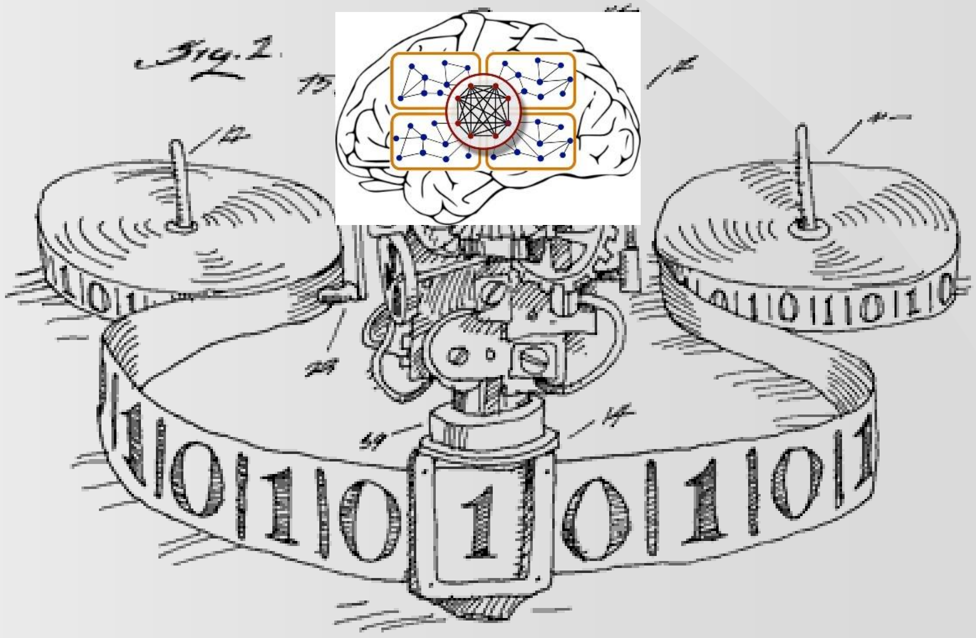


Neural Turing Machine - Add Learning to TM!!

Finite State Machine (Program)



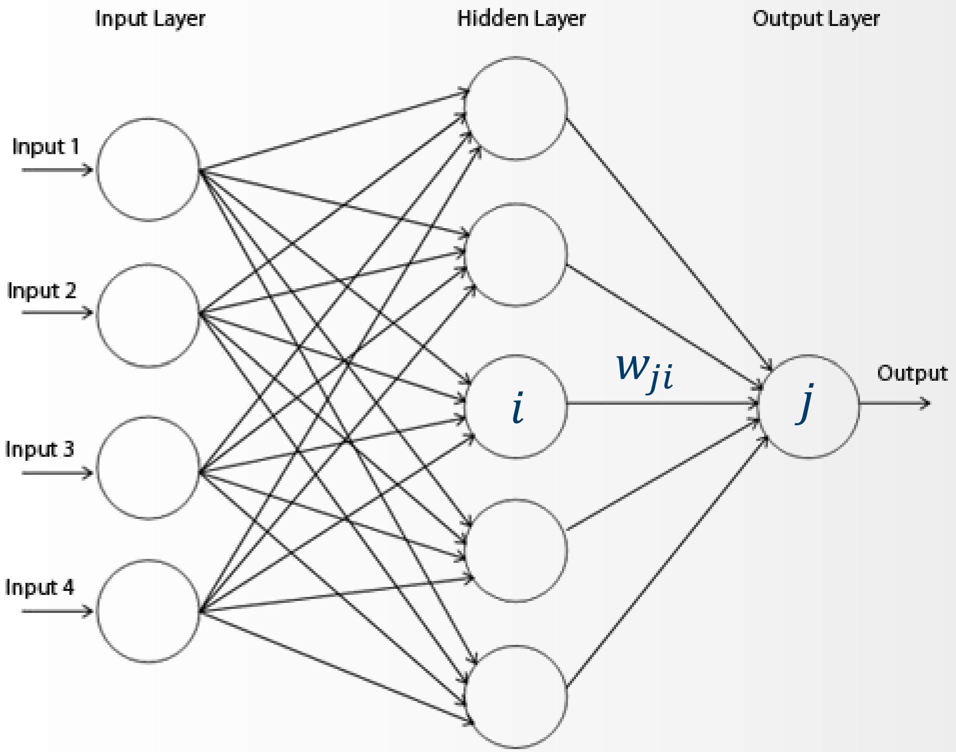
Neural Network (I Can Learn!!)



A Little History on Neural Network

- 1950s: Frank Rosenblatt, Perceptron – classification based on linear predictor
- 1969: Minsky – Proof that Perceptron Sucks! – Not able to learn XOR
- 1980s: Back propagation
- 1990s: Recurrent Neural Network, Long Short-Term Memory
- Late 2000s – now: Deep Learning, Fast Computer

Feed-Forward Neural Net and Back Propagation



- Total Input of unit j:

$$x_j = \sum_i y_i w_{ji}$$

- Output of unit j (logistic function)

$$y_j = \frac{1}{1 + e^{-x_j}}$$

- Total Error (mean square)

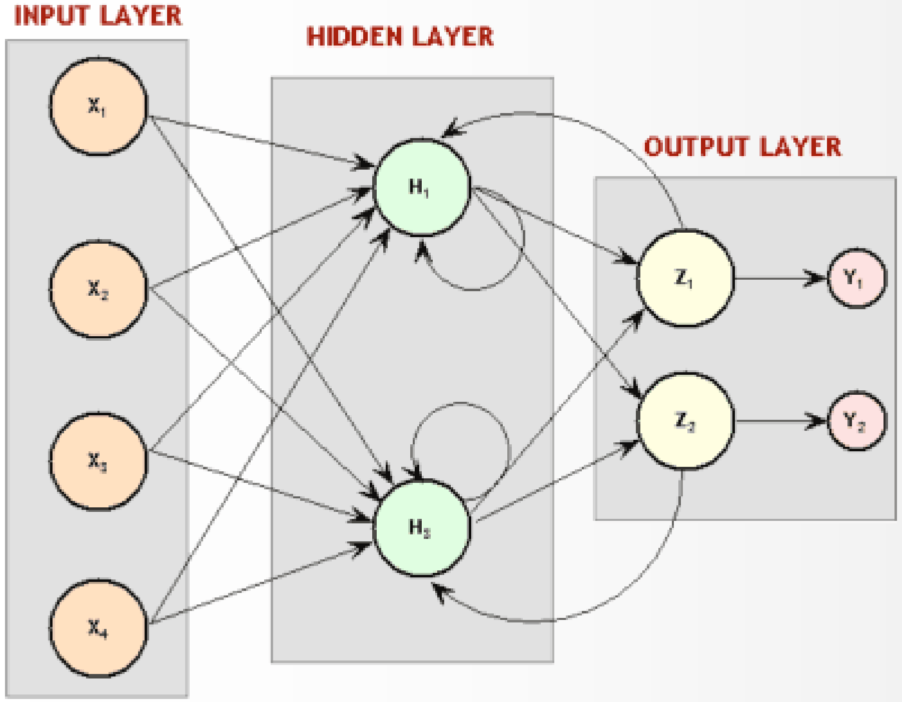
$$E_{total} = \frac{1}{2} \sum_c \sum_j \left(d_{c,j} - y_{c,j}\right)^2$$

- Gradient Descent

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial x_j} \frac{\partial x_j}{\partial w_{ji}}$$

Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". *Nature* **323** (6088): 533–536.

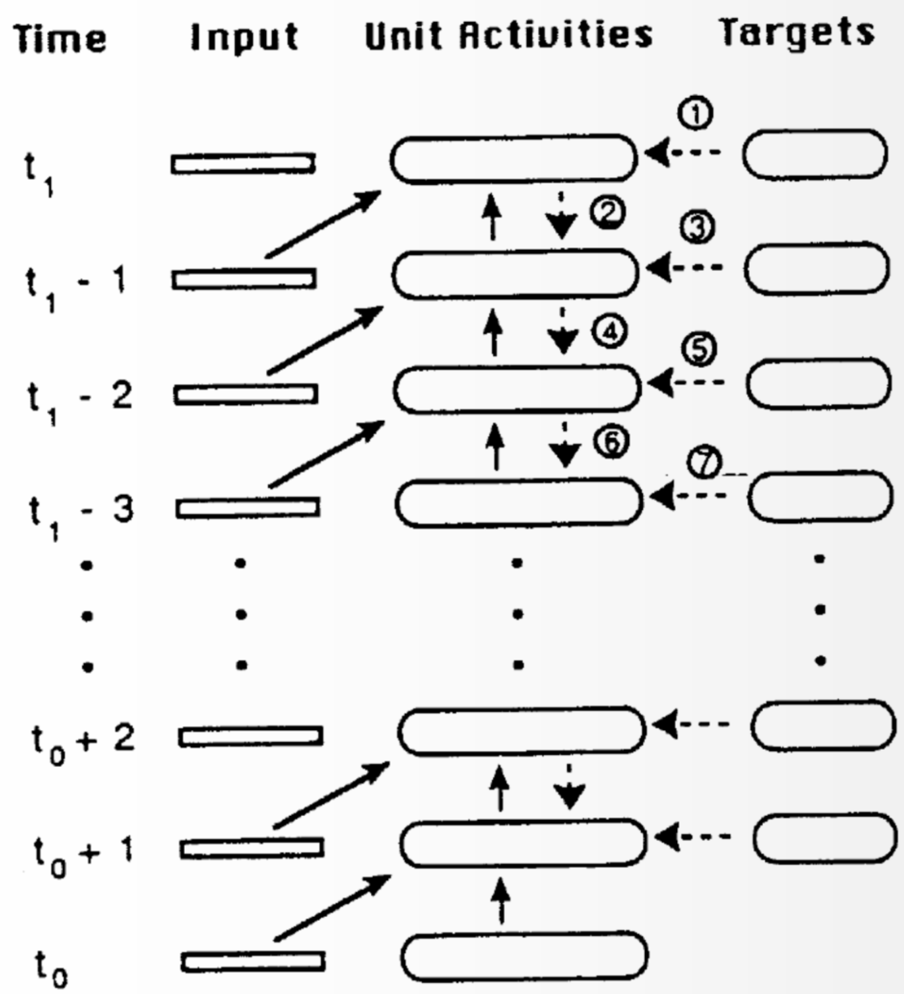
Recurrent Neural Network



R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Back-propagation: Theory, Architectures and Applications. Hillsdale, NJ: Erlbaum, 1994.

- $y(t)$: n-tuple of outputs at time t
- $x^{net}(t)$: m-tuple of external inputs at time t
- U: set of indices k that x_k is output of unit in network
- I: set of indices k that x_k is external input
- $x_k = \begin{cases} x_k^{net}(t), & if\ k \in I \\ y_k(t), & if\ k \in R \end{cases}$
- $y_k(t+1) = f_k\big(s_k(t+1)\big)$ where f_k is a differential squashing function
- $s_k(t+1) = \sum_{i \in I \cup U} w_{ki} x_i(t)$

Recurrent Neural Network – Time Unrolling



- Error Function:

$$J(t) = -\frac{1}{2}\sum_{k\in U}|e_k(t)|^2 = -\frac{1}{2}\sum_{k\in U}|d_k(t) - y_k(t)|^2$$

$$J^{Total}(t', t) = \sum_{\tau=t'+1}^{t} J(\tau)$$

- Back Propagation through Time:

$$\varepsilon_k(t) = \frac{\partial J(t)}{\partial y_k(t)} = e_k(t) = d_k(t) - y_k(t)$$

$$\delta_k(\tau) = \frac{\partial J(t)}{\partial s_k(\tau)} = \frac{\partial J(t)}{\partial y_k(\tau)}\frac{\partial y_k(\tau)}{\partial s_k(\tau)} = f_k'\big(s_k(\tau)\big)\varepsilon_k(\tau)$$

$$\varepsilon_k(\tau - 1) = \frac{\partial J(t)}{\partial y_k(\tau-1)} = \sum_{l\in U}\frac{\partial J(t)}{\partial s_l(\tau)}\frac{\partial s_l(\tau)}{\partial y_k(\tau-1)} = \sum_{l\in U} w_{lk}\delta_l(\tau)$$

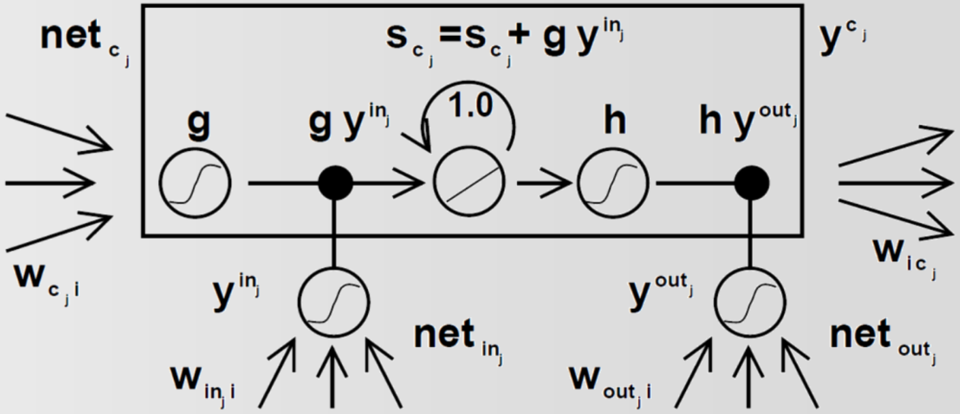
$$\frac{\partial J^{Total}}{\partial w_{ji}} = \sum_{\tau=t'+1}^{t}\frac{\partial J(t)}{\partial s_j(\tau)}\frac{\partial s_j(\tau)}{\partial w_{ji}}$$

RNN & Long Short-Term Memory

- RNN is Turing Complete
  - With Proper Configuration, it can simulate any sequence generated by a Turing Machine
- Error Vanishing and Exploding
  - Consider single unit self loop:

$$\delta_k(\tau) = \frac{\partial J(t)}{\partial s_k(\tau)} = \frac{\partial J(t)}{\partial y_k(\tau)} \frac{\partial y_k(\tau)}{\partial s_k(\tau)} = f_k'\big(s_k(\tau)\big)\varepsilon_k(\tau) = f_k'\big(s_k(\tau)\big) w_{kk} \delta_k(\tau + 1)$$

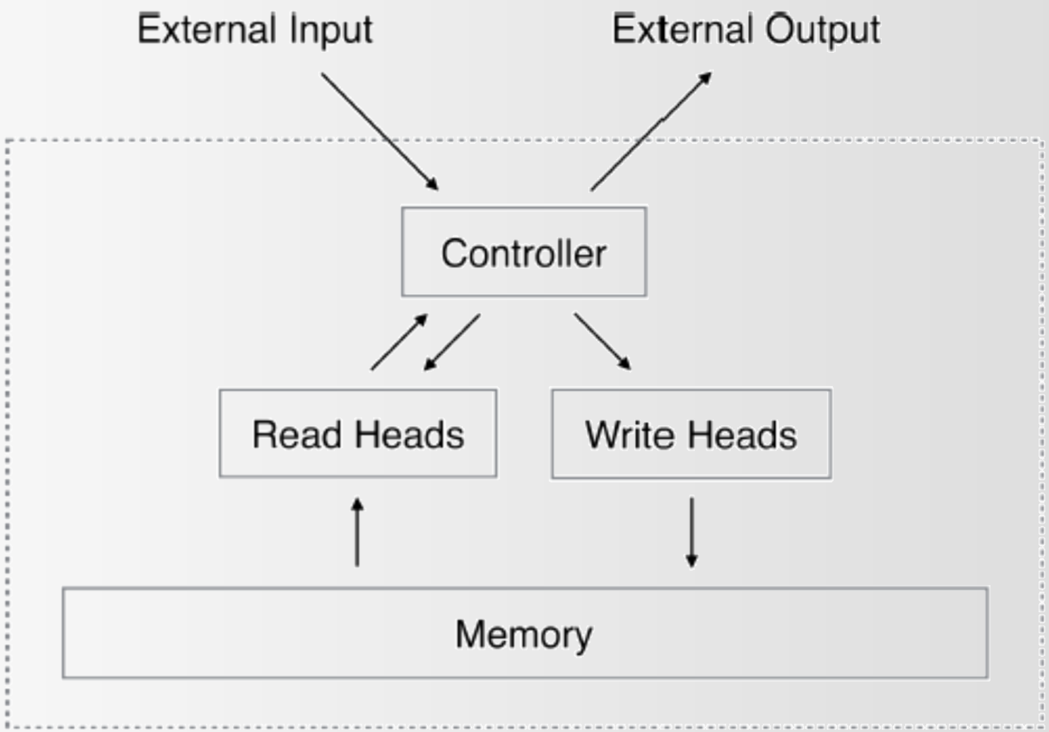
- Long Short-term Memory



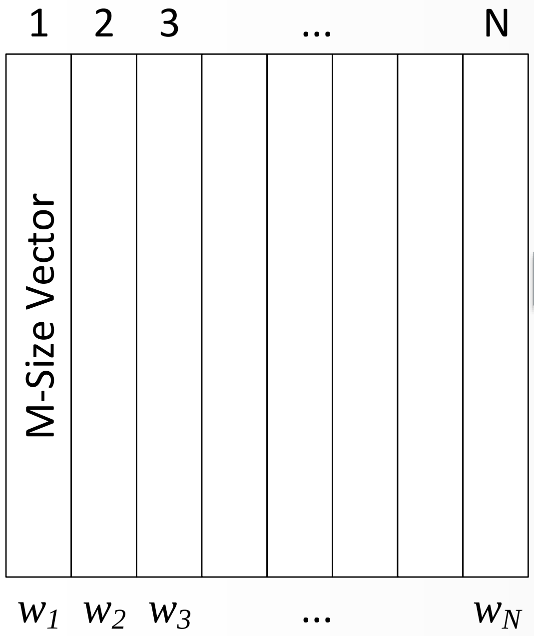
Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8):1735–1780.

Purpose of Neural Turing Machine

- Enrich the capabilities of standard RNN to simplify the solution of algorithmic tasks

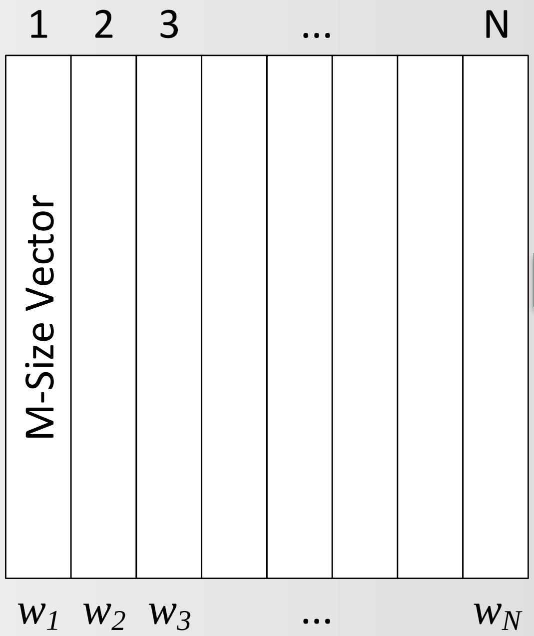


NTM Read/Write Operation



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

$w_t(i)$: vector of weights over the N locations emitted by a read head at time t



Erase:

$$M_t(i) = M_{t-1}(i)[1 - w_t(i)e_t]$$

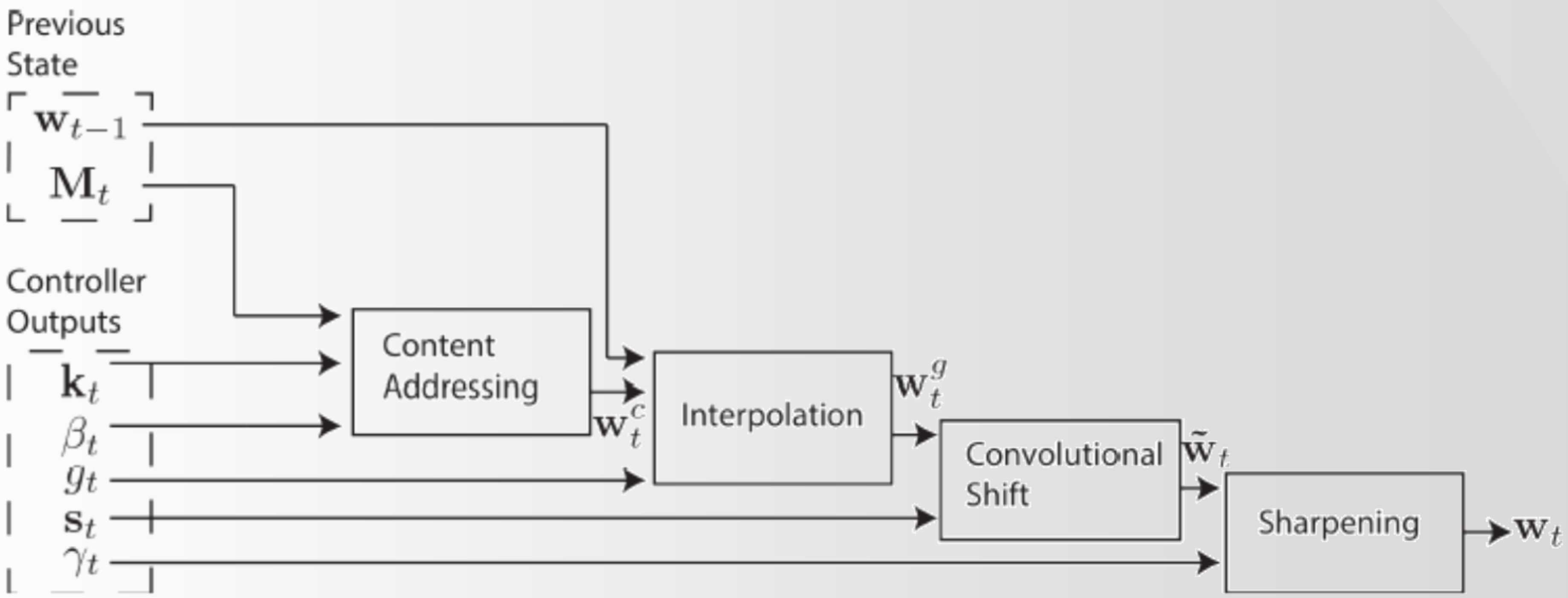
Add:

$$M_t(i) \mathrel{+}= w_t(i)a_t$$

e_t: Erase Vector, emitted by the header
a_t: Add Vector, emitted by the header
w_t: weights vector

Attention & Focus

- Attention and Focus is adjusted by weights vector across whole memory bank!



Content Based Addressing

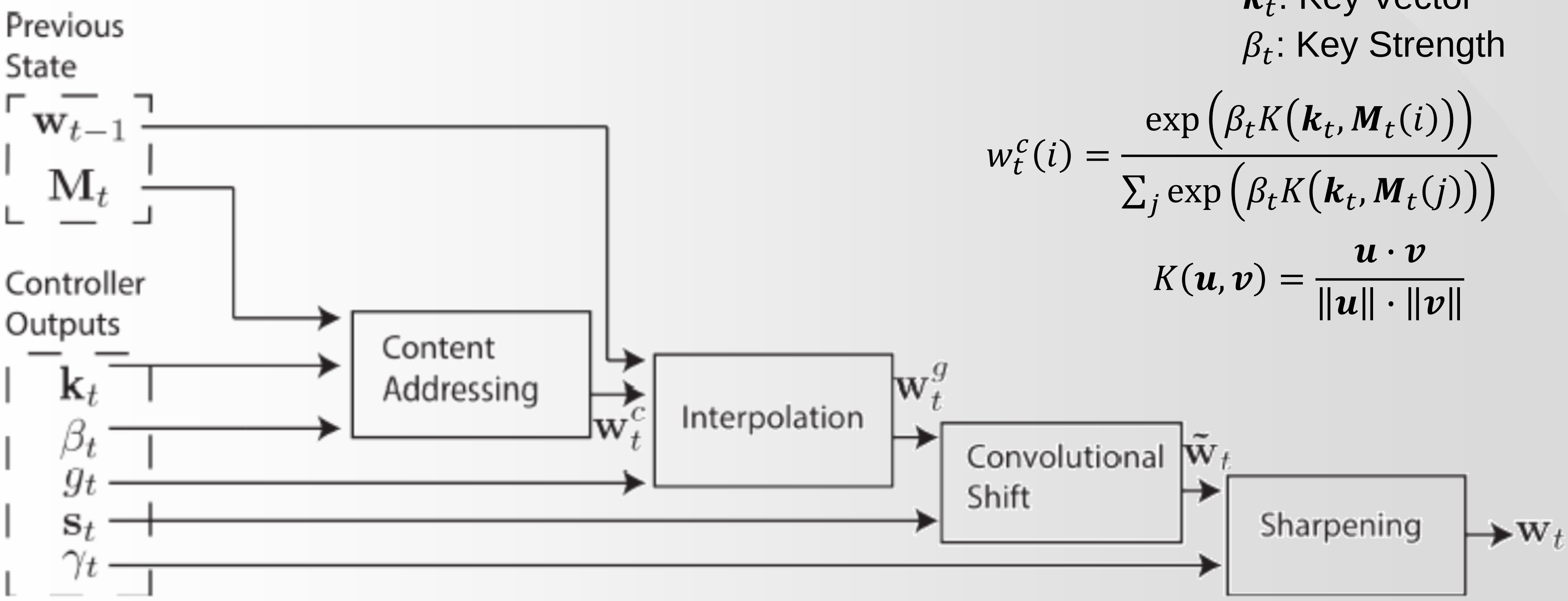
- Find Similar Data in memory

$\boldsymbol{k}_t$: Key Vector

β_t: Key Strength

$$w_t^c(i) = \frac{\exp\left(\beta_t K\left(\boldsymbol{k}_t, \boldsymbol{M}_t(i)\right)\right)}{\sum_j \exp\left(\beta_t K\left(\boldsymbol{k}_t, \boldsymbol{M}_t(j)\right)\right)}$$

$$K(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|}$$

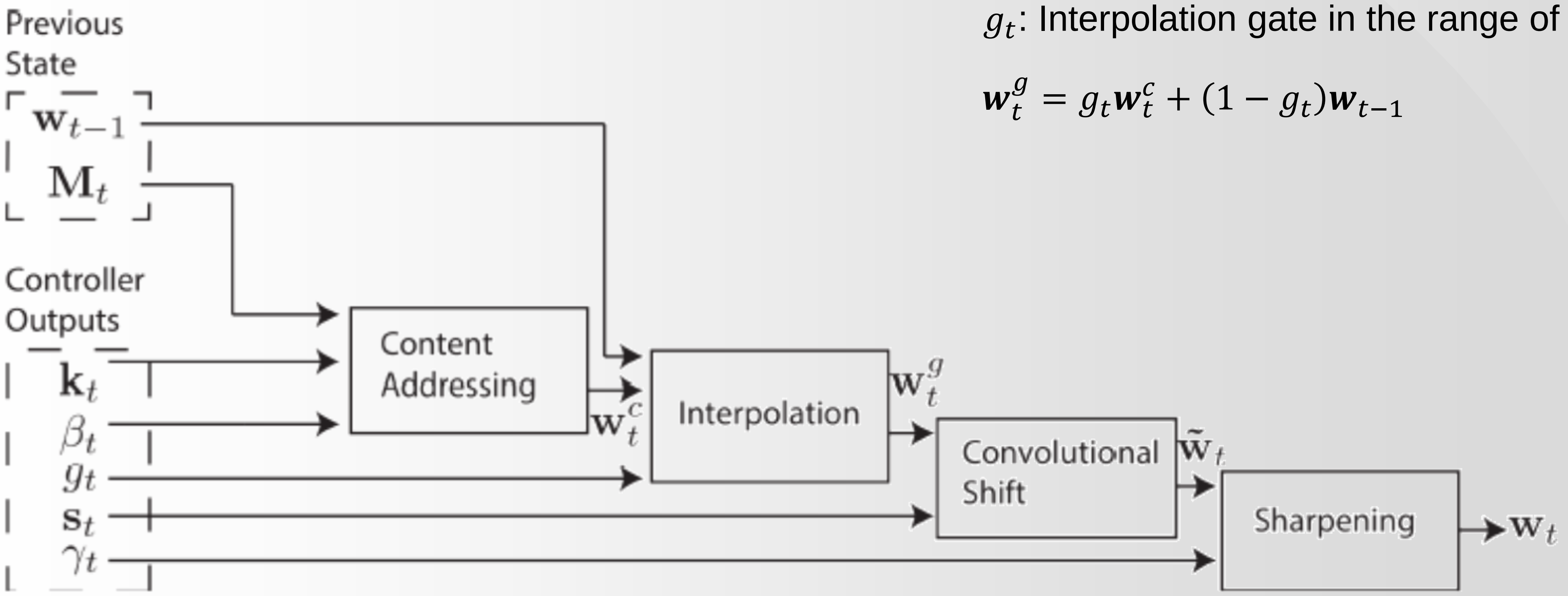


May not depend on content only…

- Gating Content Addressing

g_t: Interpolation gate in the range of (0,1)

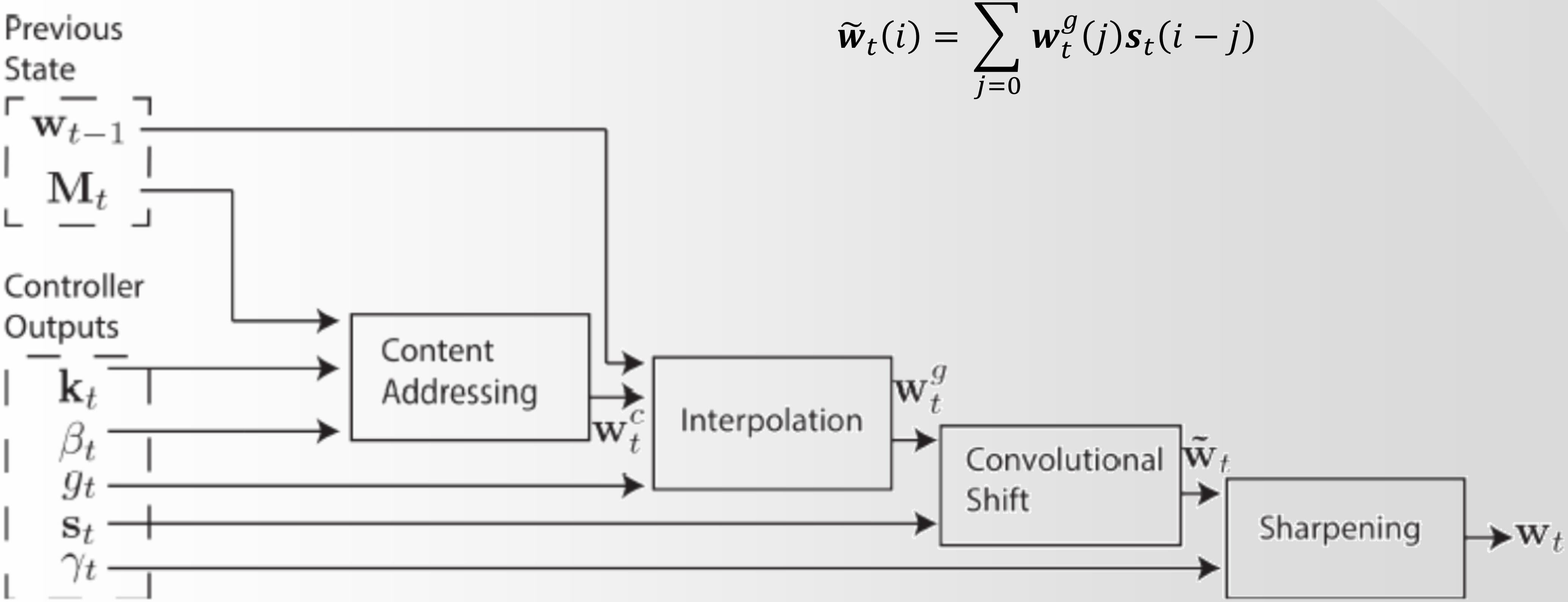
$$\boldsymbol{w}_t^g = g_t \boldsymbol{w}_t^c + (1 - g_t)\boldsymbol{w}_{t-1}$$



Convolutional Shift – Location Addressing

$\boldsymbol{s}_t$: Shift weighting – Convolutional Vector (size N)

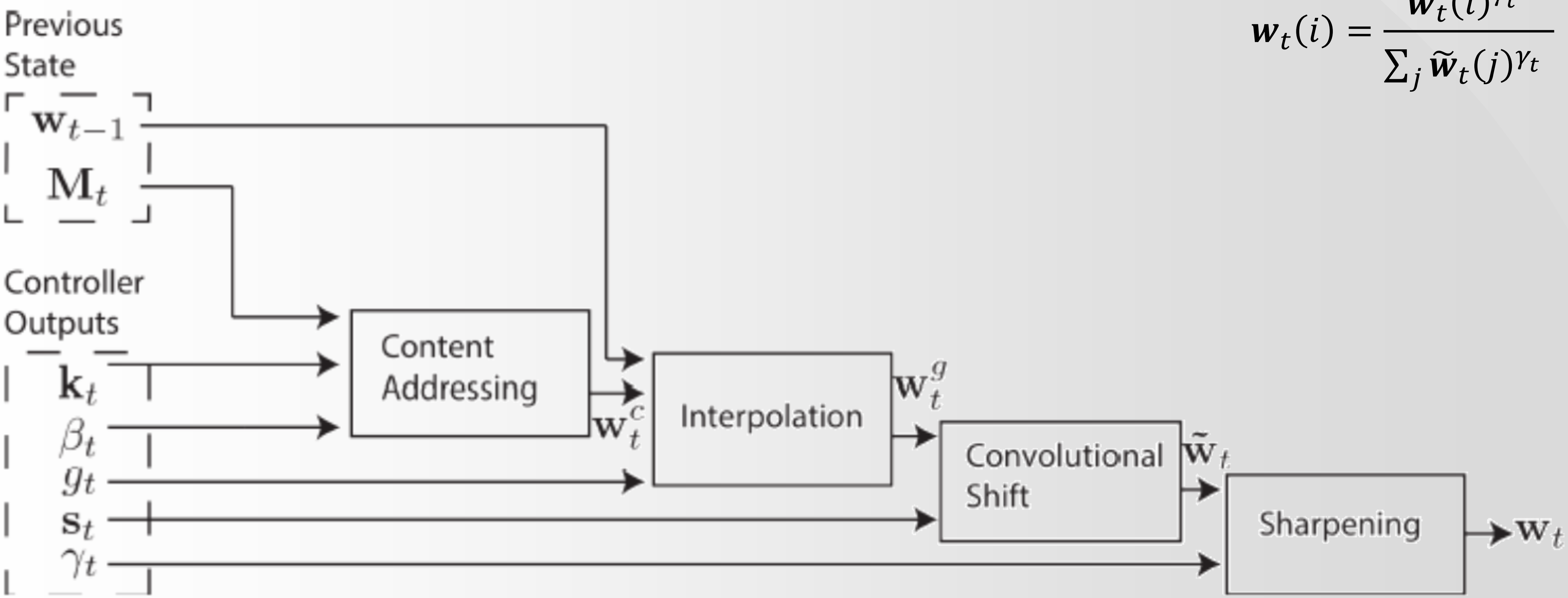
$$\tilde{\boldsymbol{w}}_t(i) = \sum_{j=0}^{N-1} \boldsymbol{w}_t^g(j)\boldsymbol{s}_t(i-j)$$



Convolutional Shift – Location Addressing

γ_t: sharpening factor

$$\boldsymbol{w}_t(i) = \frac{\widetilde{\boldsymbol{w}}_t(i)^{\gamma_t}}{\sum_j \widetilde{\boldsymbol{w}}_t(j)^{\gamma_t}}$$

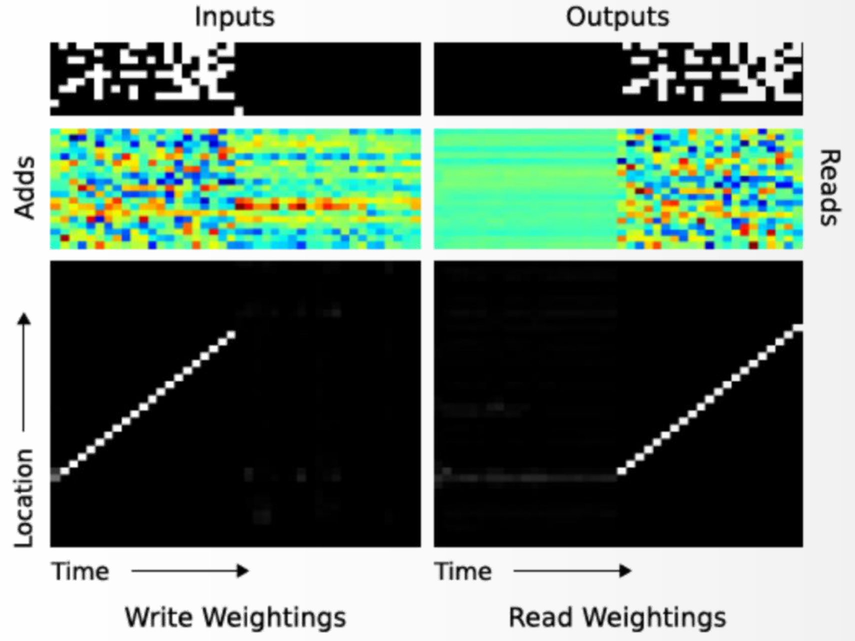


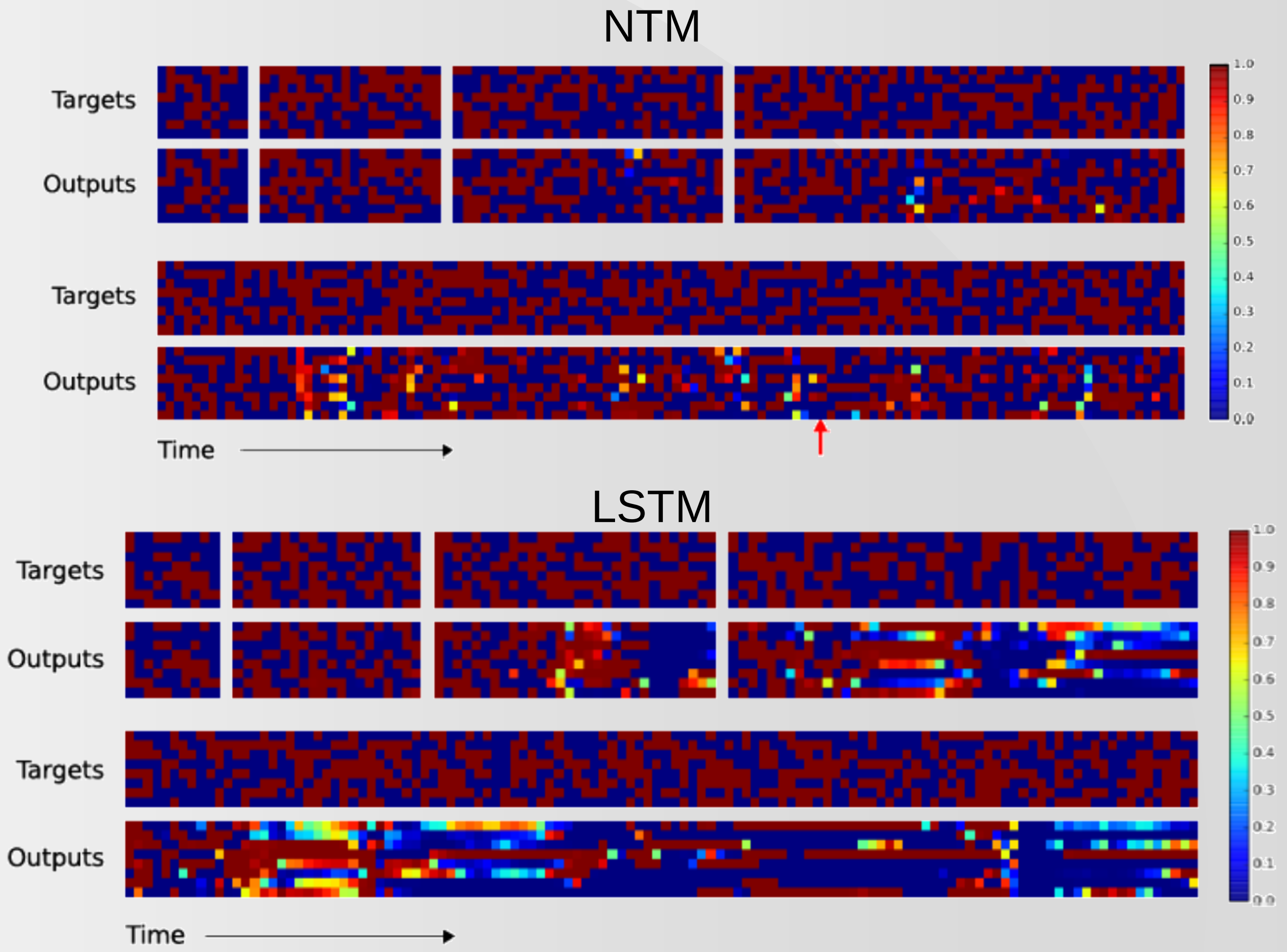
Neural Turing Machine - Experiments

- Goal: to demonstrate NTM is
  - Able to solve the problems
  - By learning compact internal programs
- Three architectures:
  - NTM with a feedforward controller
  - NTM with an LSTM controller
  - Standard LSTM controller
- Applications
  - Copy
  - Repeat Copy
  - Associative Recall
  - Dynamic N-Grams
  - Priority Sort

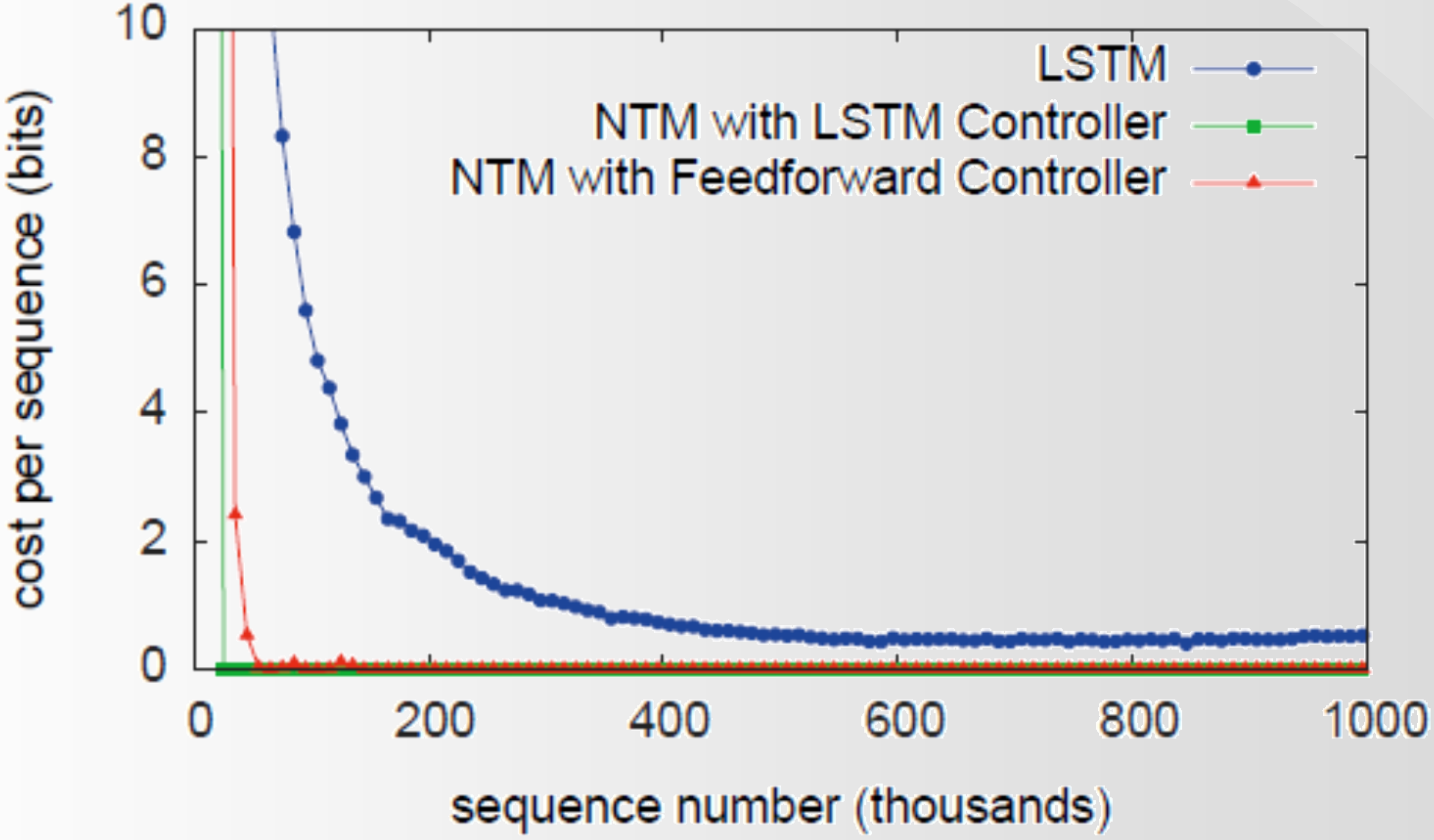
NTM Experiments: Copy

- Task: Store and recall a long sequence of arbitrary information
- Training: 8-bit random vectors with length 1-20
- No inputs while it generates the targets



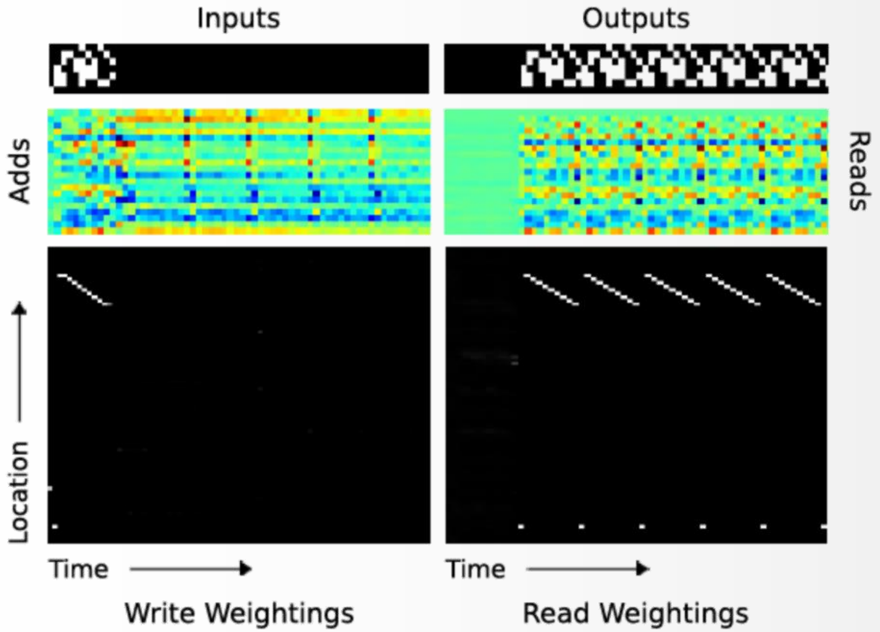


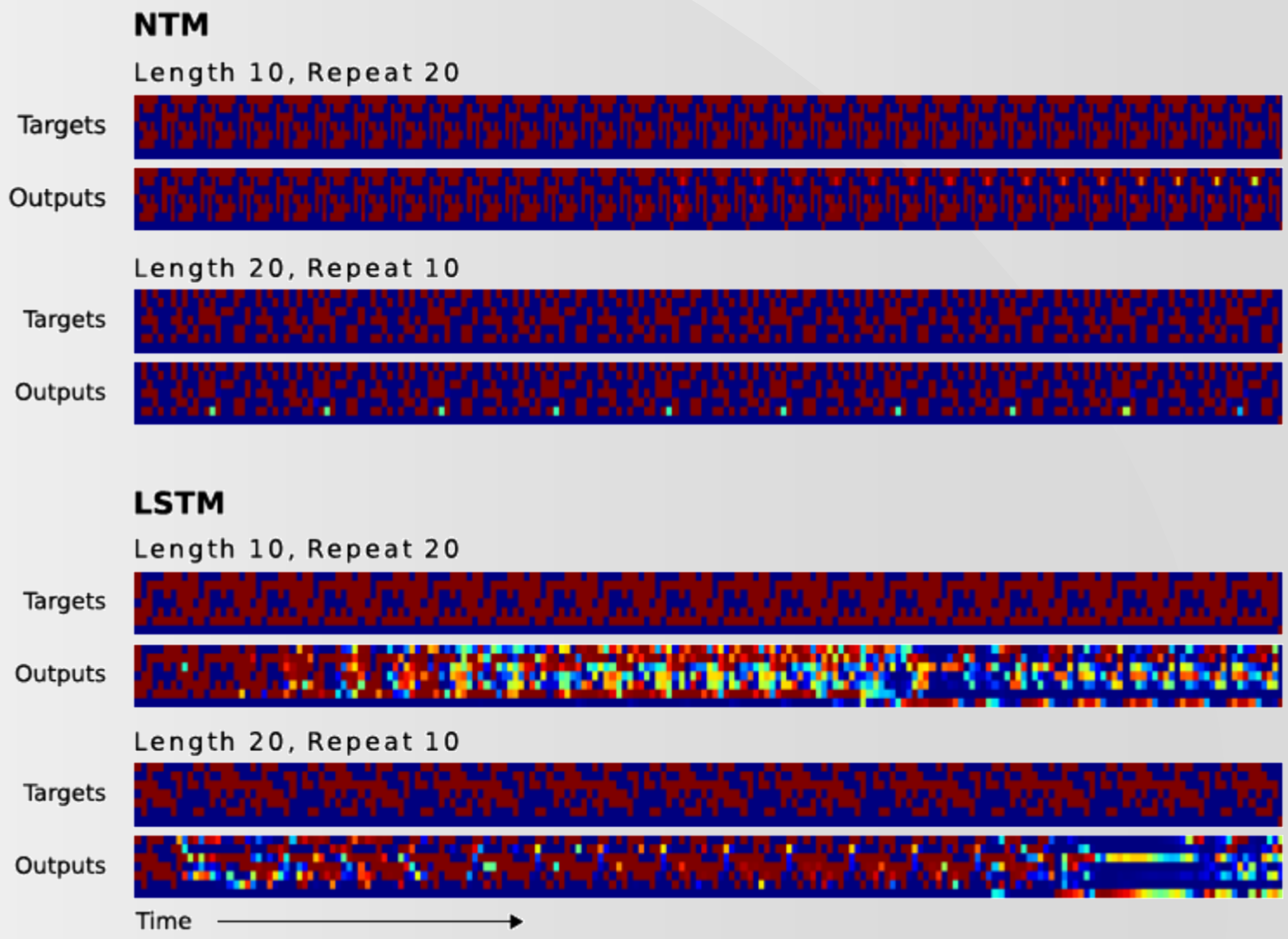
NTM Experiments: Copy – Learning Curve



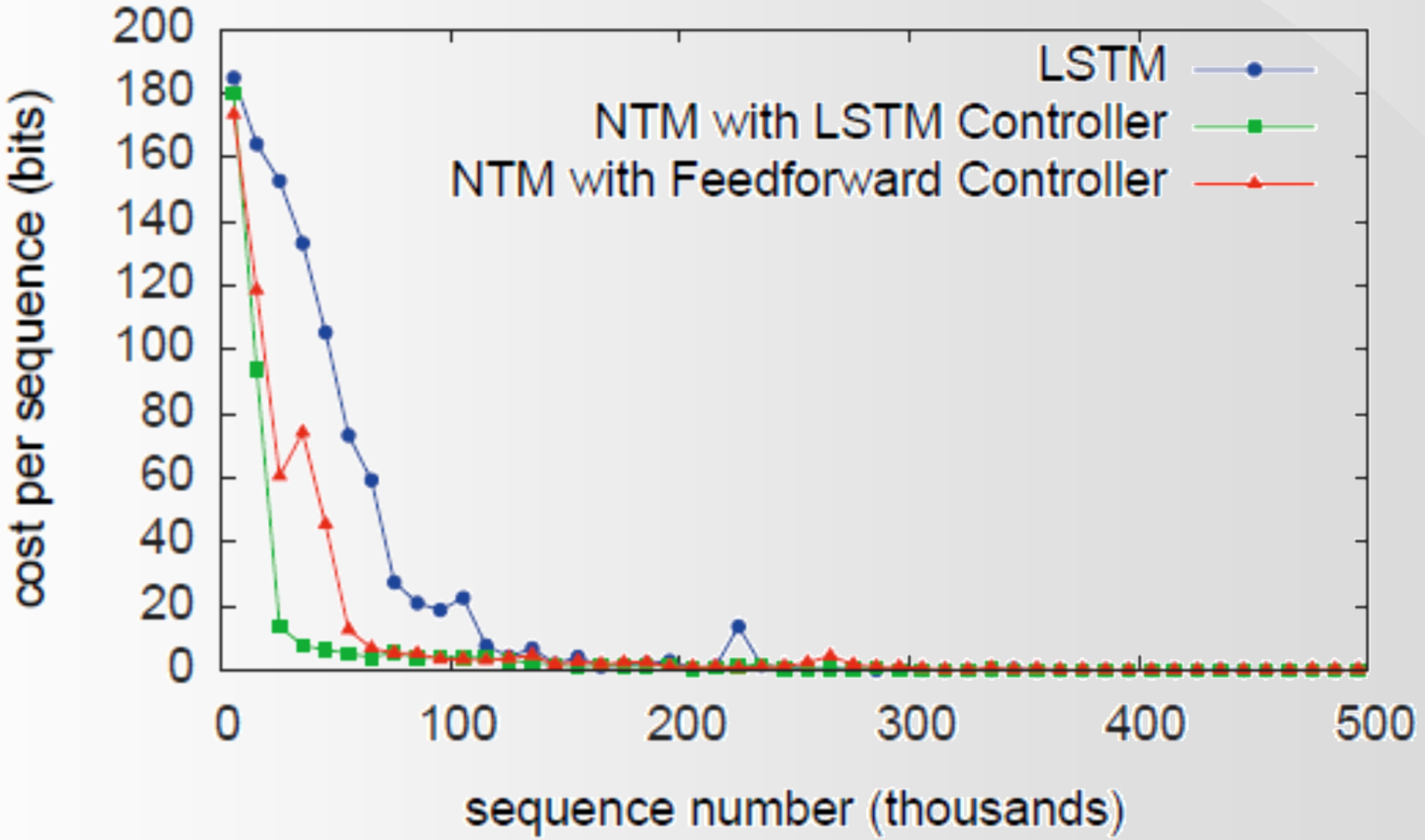
NTM Experiments: Repeated Copy

- Task: Repeat a sequence a specified number of times
- Training: random length sequences of random binary vectors followed by a scalar value indicating the desired number of copies



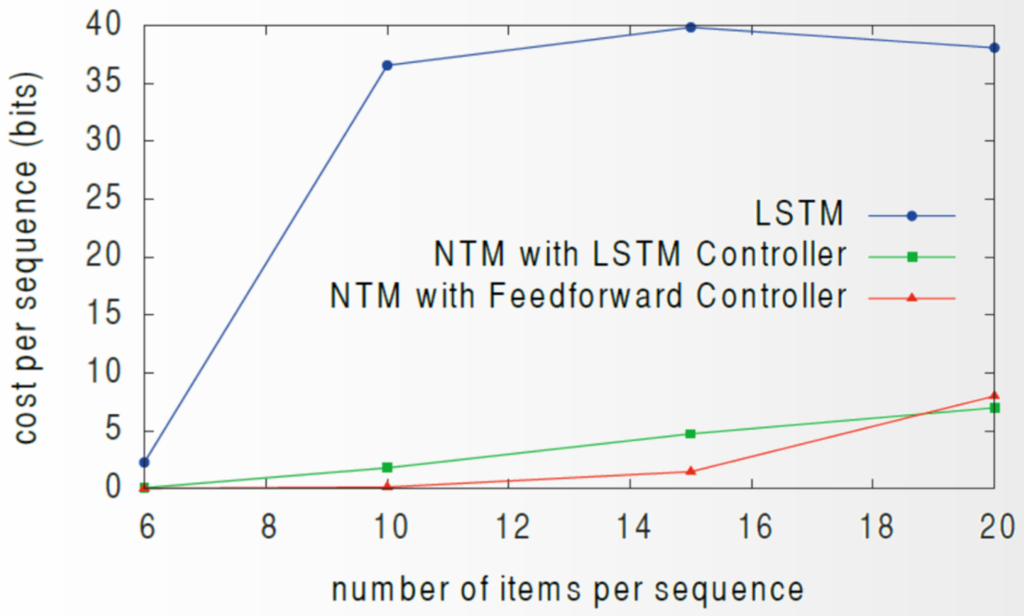


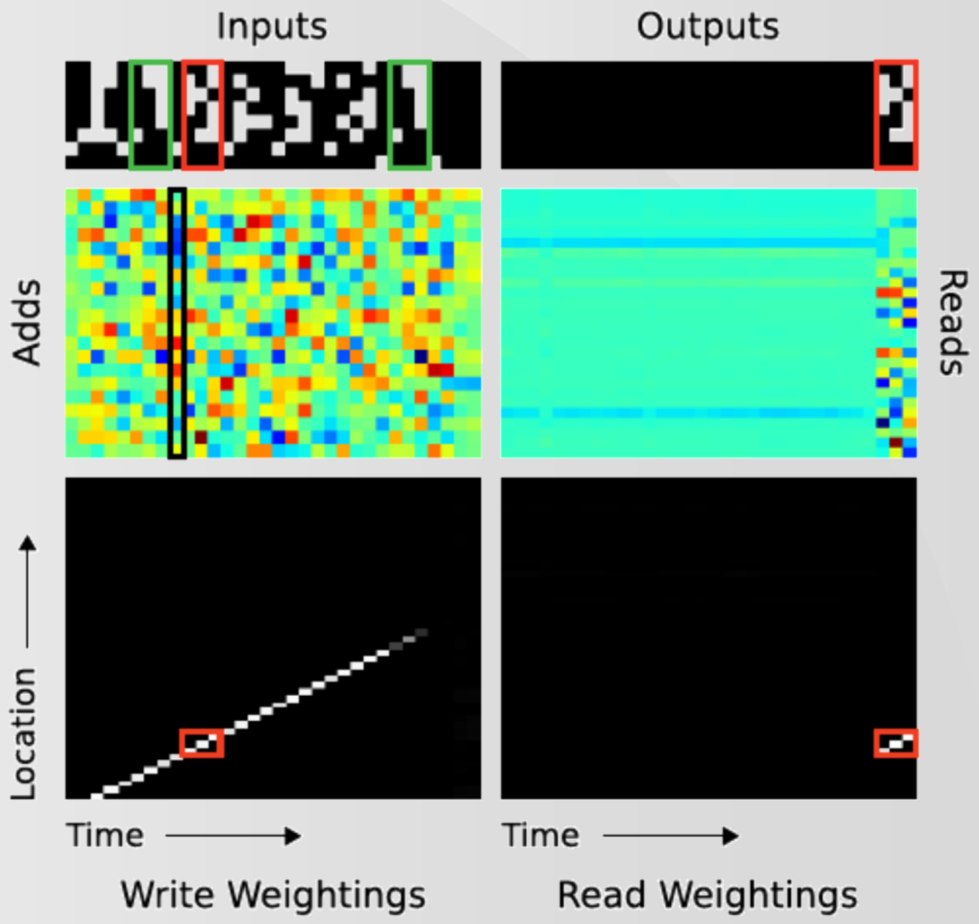
NTM Experiments: Repeated Copy – Learning Curve



NTM Experiments: Associative Recall

- Task: ask the network to produce the next item, given current item after propagating a sequence to network
- Training: each item is composed of three six-bit binary vectors, 2-8 items every episode

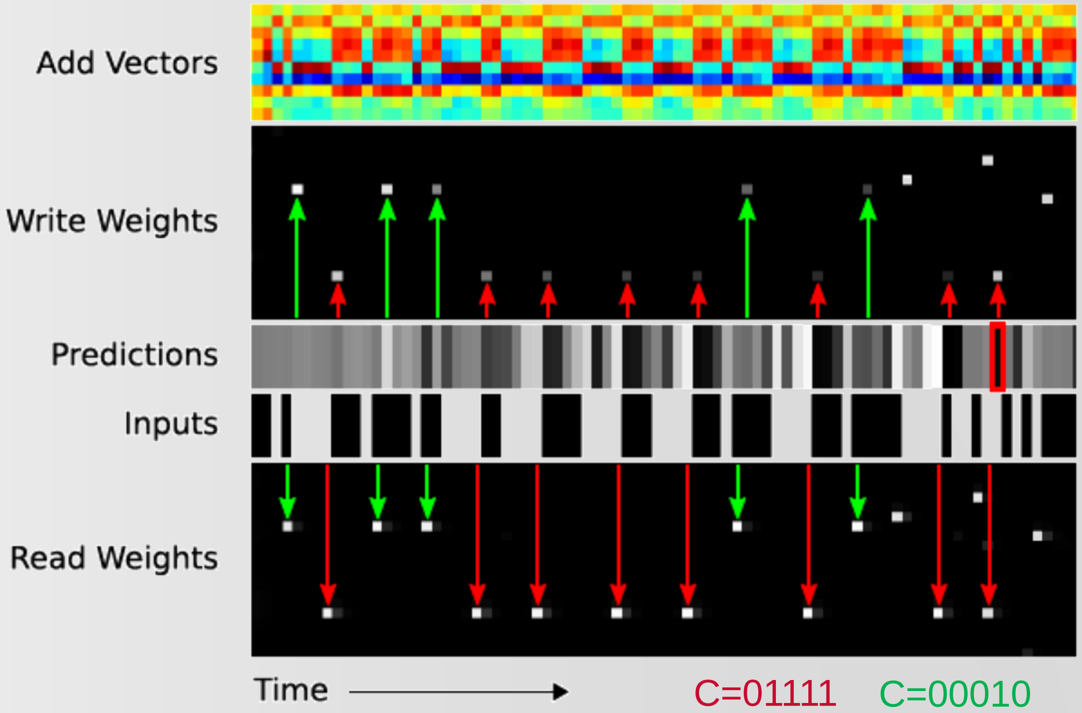


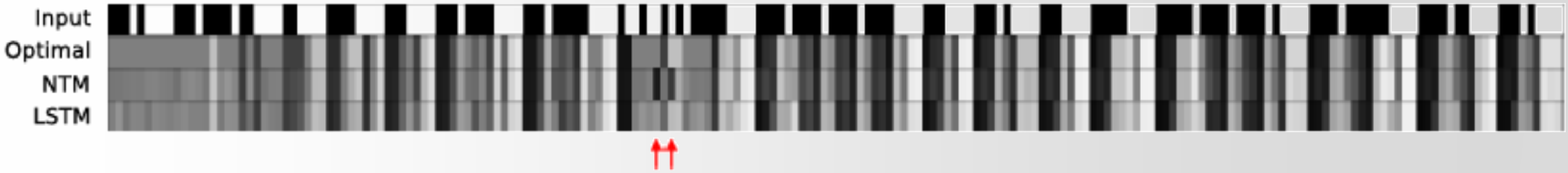


NTM Experiments: Dynamic N-Grams

- Task: Learn N-Gram Model – rapidly adapt to new predictive distribution
- Training: 6-Gram distributions over binary sequences. 200 successive bits using look-up table by drawing 32 probabilities from Beta(.5,.5) distribution
- Compare to Optimal Estimator:

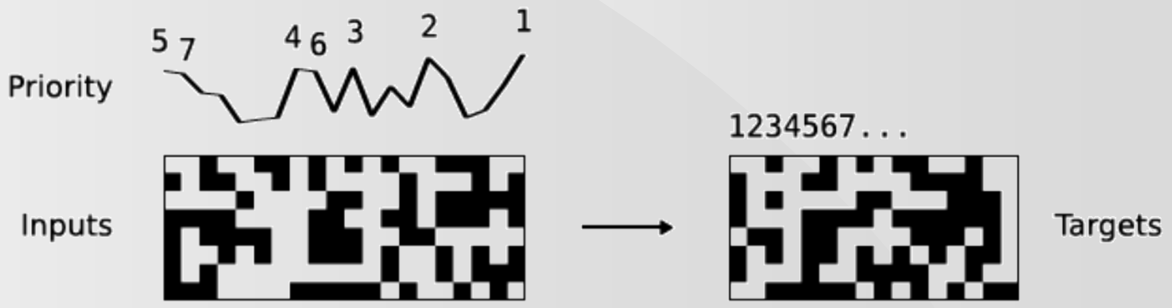
$$P(B=1|N_1,N_2,\boldsymbol{c})=\frac{N_1+0.5}{N_1+N_0+1}$$

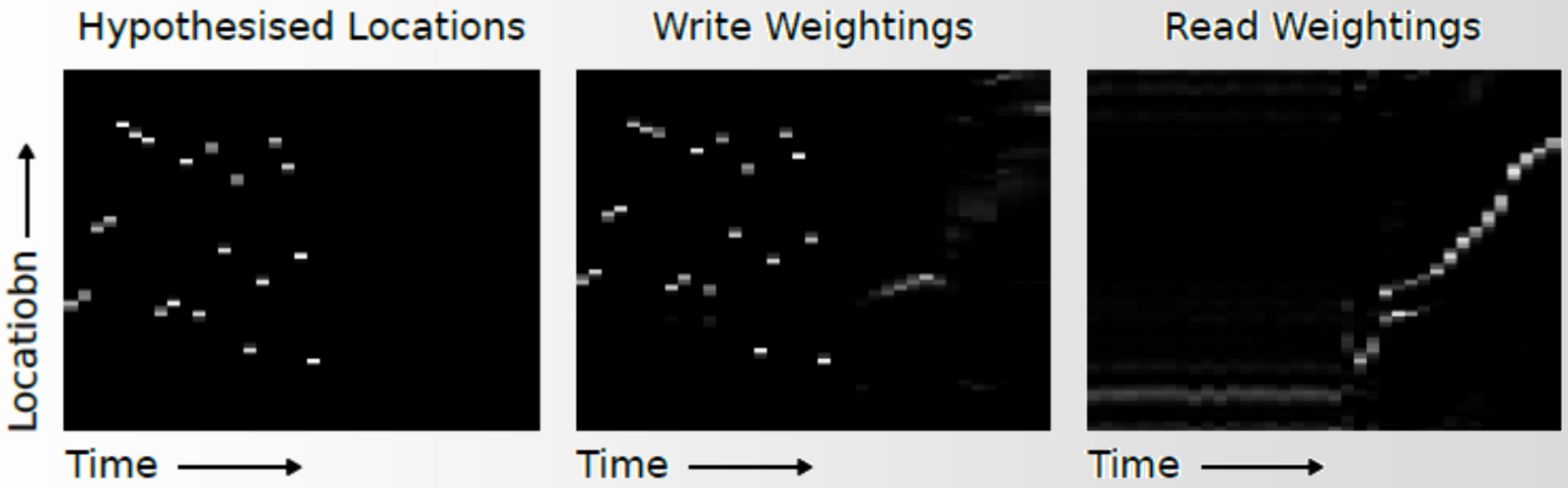




NTM Experiments: Priority Sorting

- Task: Sort Binary Vector based on priority
- Training: 20 binary vectors with corresponding priorities, output 16 highest-priority vectors





NTM Experiments: Learning Curve

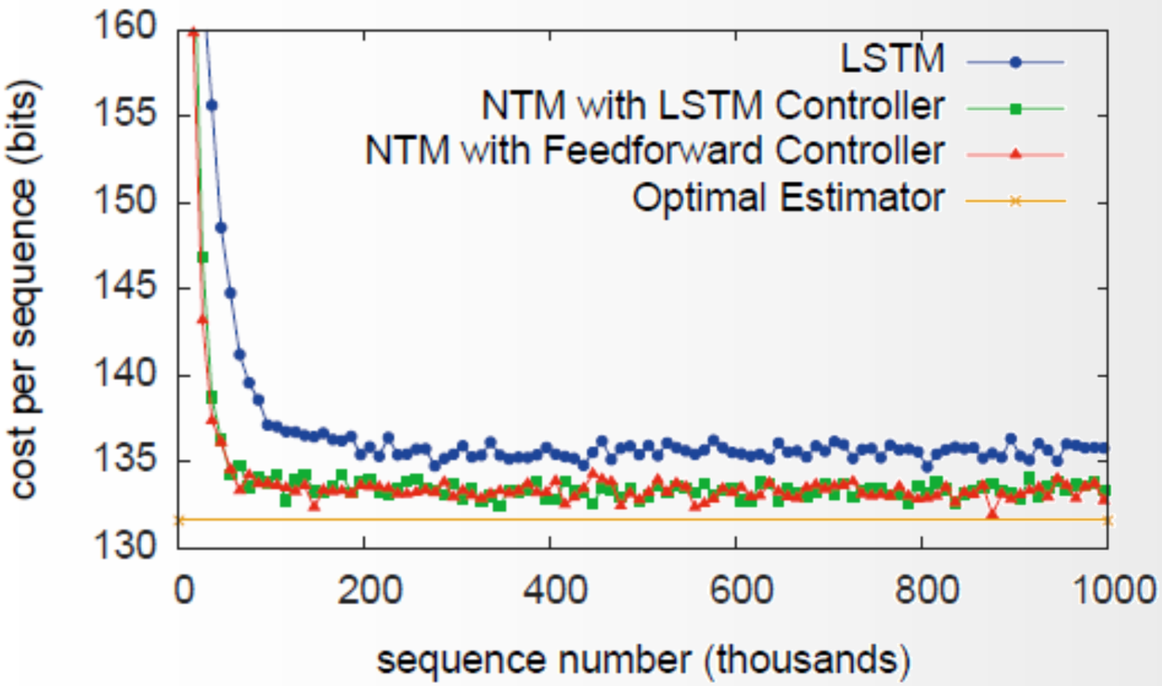


Figure 13: Dynamic N-Gram Learning Curves.

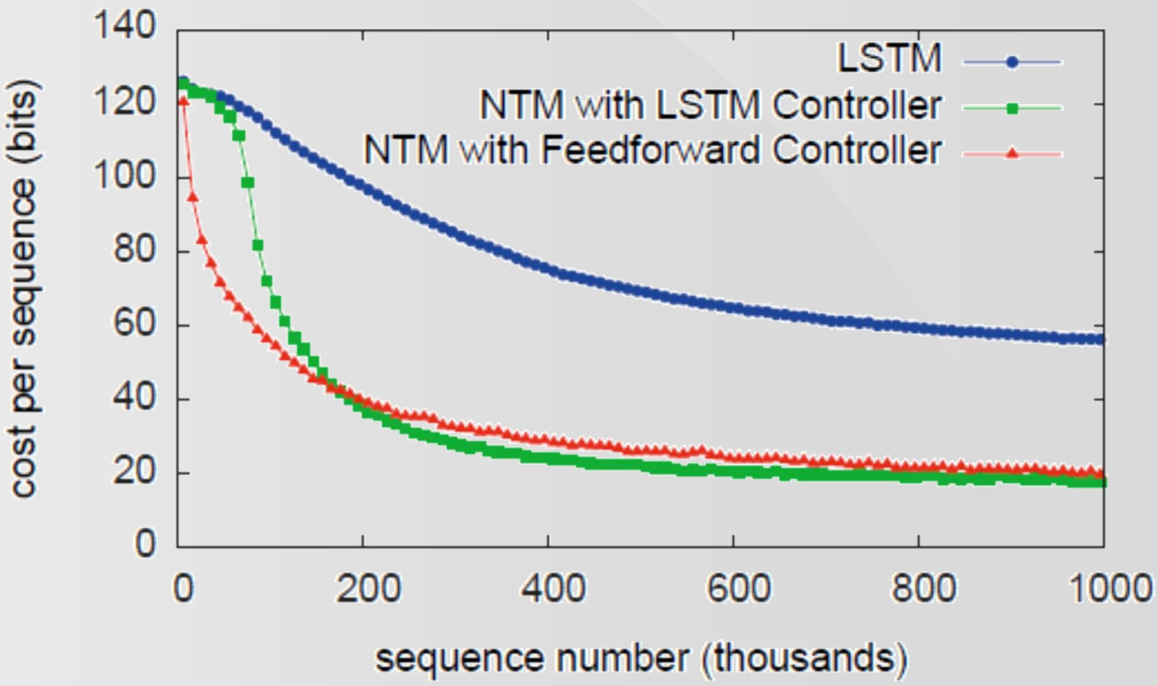


Figure 18: Priority Sort Learning Curves.

Some Detailed Parameters

NTM with Feed Forward Neural Network

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

NTM with LSTM

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

LSTM

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

Additional Information

- Literature Reference:
  - Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". *Nature* **323** (6088): 533–536
  - R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Back-propagation: Theory, Architectures and Applications. Hillsdale, NJ: Erlbaum, 1994.
  - Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8):1735–1780
  - Y. Bengio and Y. LeCun, Scaling Learning Algorithms towards AI, *Large-Scale Kernel Machines*, Bottou, ., Chapelle, O., DeCoste, D., and Weston, J., Eds., MIT Press, 2007.
- NTM Implementation
  - https://github.com/shawntan/neural-turing-machines
- NTM Reddit Discussion
  - https://www.reddit.com/r/MachineLearning/comments/2m9mga/implementation_of_neural_turing_machines/


Need Some Rest After All these Math!!!
Thank You!