Lecture 5

Von-Wun Soo

Department of Computer Science

National Tsing Hua University

- 1.Recurrent neural networks,
- 2.back propagation thru time,
- 3. Long short term memory,
- 4. GRN (2 hr)
- 5.Memory Net, (1 hr)
- 6.Neural Turing Machines (1 hr)

- 實作 3 hr
- Image Captioning with Vanilla RNNs
- Image Captioning with LSTMs

Problem Property of RNN

- Be able to model Sequence to sequence relation

- Must be able to hand long term dependency
- Recurrent_neural_network

Various form of RNN

- Fully recurrent network
- Recursive neural networks

- Hopfield network 1982
- Elman networks and Jordan networks

- Echo state network
- Neural history compressor

- *Long short-term memory (LSTM)
- *Gated recurrent unit (GRU)

RNN models

- Bi-directional RNN
- Continuous-time RNN
- Hierarchical RNN
- Recurrent multilayer Perceptron
- Second order RNN
- Multiple timescales recurrent neural network (MTRNN) model
- Pollack Sequential cascade networks
- Neural Turing Machines (NTM)
- Neural Network Pushdown automata
- Bidiretional associative memory (BAM) (1988)

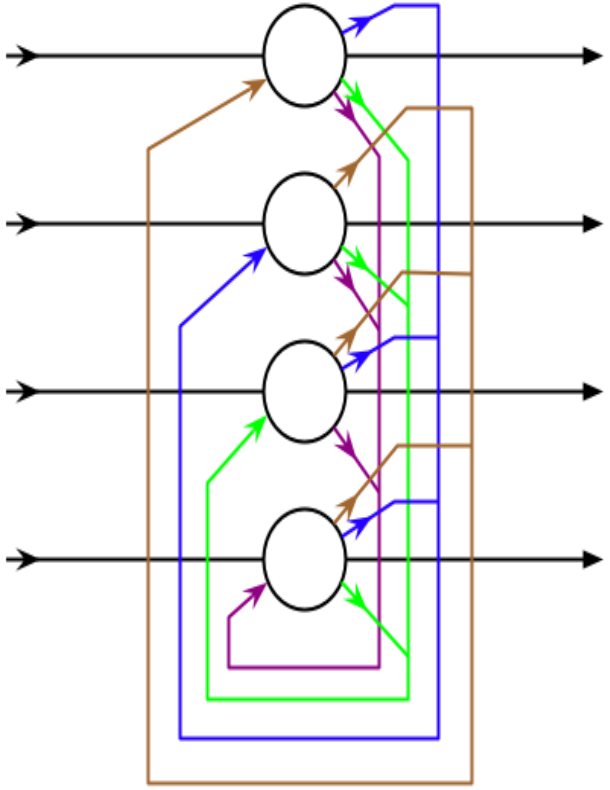
Hopfield network

Every pair of units *i* and *j* in a Hopfield network have a connection that is described by the connectivity weight wij

The connections in a Hopfield net typically have the following restrictions:

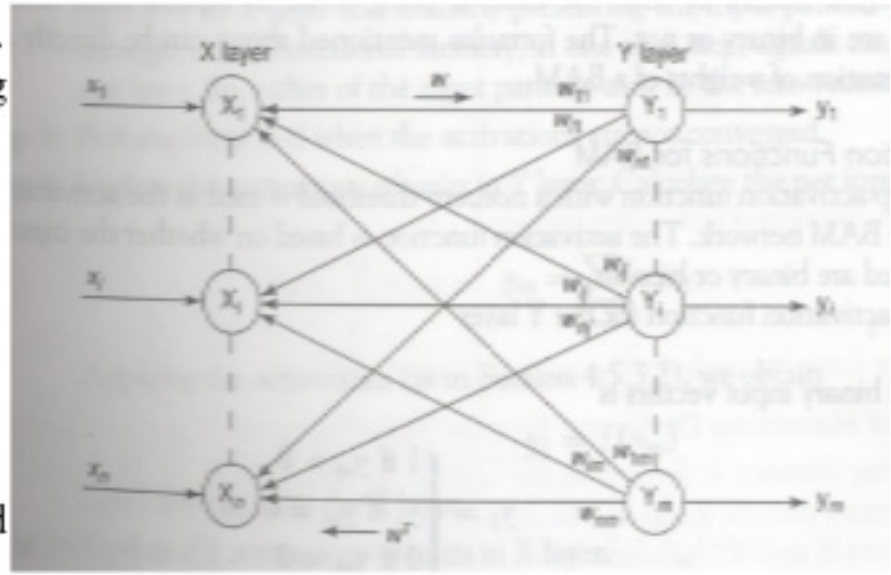
- $w_{ii}=0, \forall i$ (no unit has a connection with itself)
- $w_{ij}=w_{ji}, \forall i,j$ (connections are symmetric)

- Minization
- $E=-1/2\sum_{ij} w_{ij} s_i s_j + \sum \theta_i s_i$

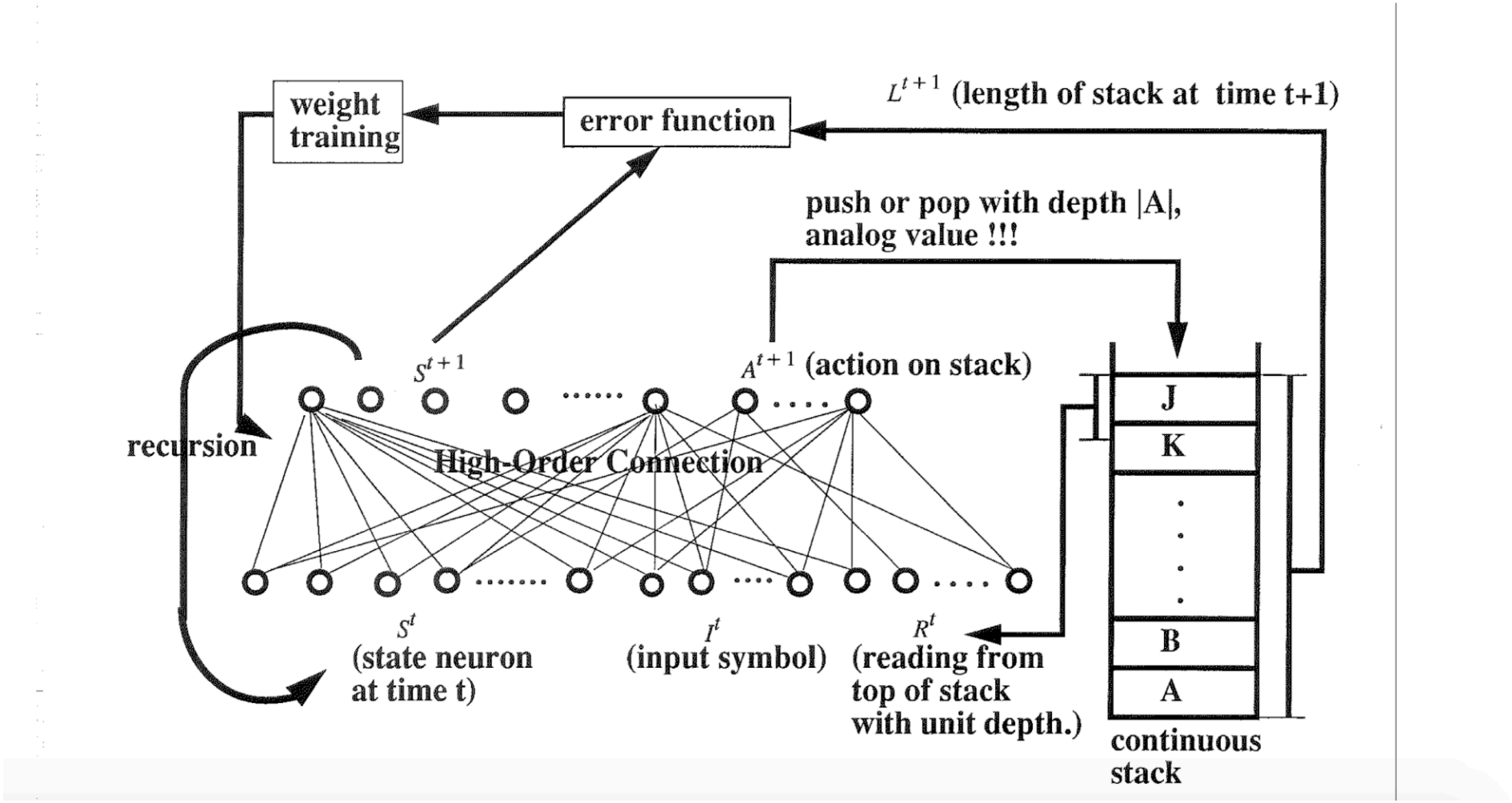


BIDIRECTIONAL ASSOCIATIVE MEMORY(BAM)

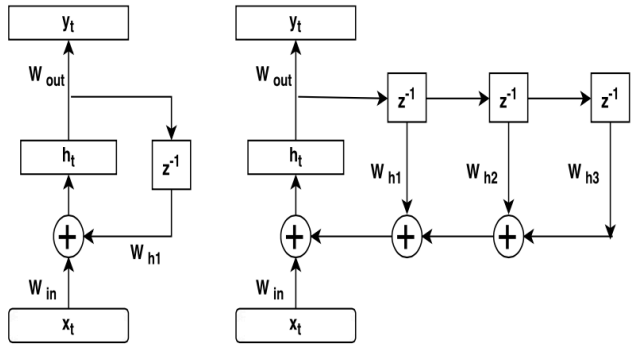
- **Theory**
- Developed by Kosko in the year 1988.
- Performs backward & forward search.
- Encodes binary/bipolar pattern using Hebbian learning rule
- Two types
  - Discrete BAM
  - Continuous BAM
- **Architecture**
- Weights are bidirectional
- X layer has ‘n’ input units
- Y layer has ‘m’ output units.
- Weight matrix from X to Y is W and from Y to X is W^T



Architecture of the Neural Network Pushdown Automata



Higher order recurrent neural networks (3rd order)



Hierarchical RNN for document modeling

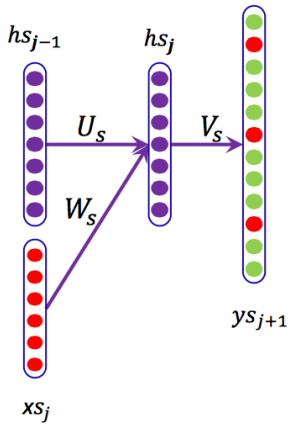


Figure 1: Recurrent Neural Network for Sentence-level Language Modeling

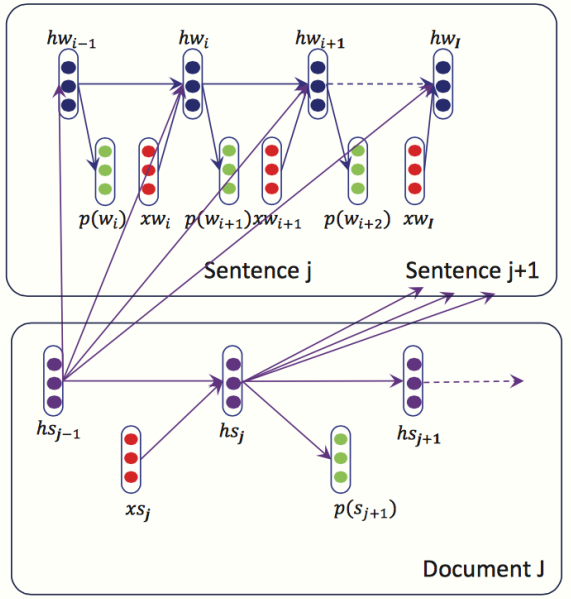


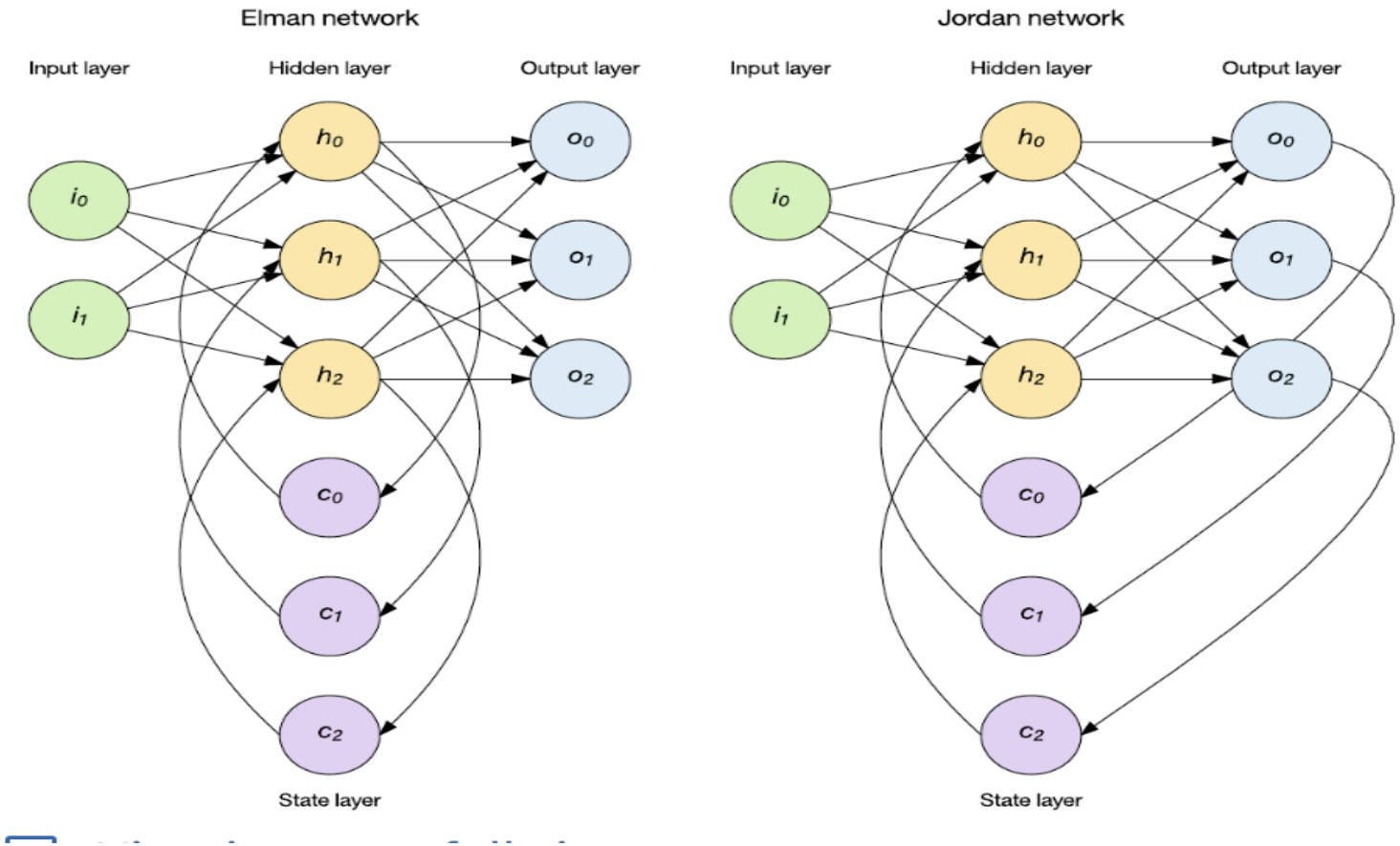
Figure 2: Hierarchical recurrent neural network

Continuous time RNN

$$k_i \frac{dy_i}{dt} = -y_i + \sigma\left(\sum_j w_{ij} y_j\left(t - \tau_{ij}\right) - \theta_i\right)$$

- $\boldsymbol{\tau_{ij}}$ ***is the time delay*** between neurons i and j
- y is the activation

Elman vs. Jordan recurrent neural networks



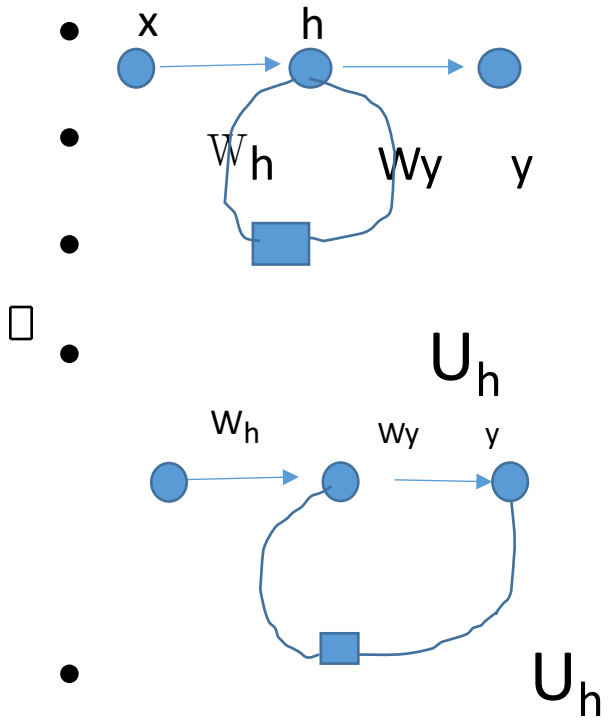
Elman vs Jordan RNN

- Elman network

$$h_t=\sigma_h(W_h x_t+U_h h_{t-1}+b_h)$$
$$y_t=\sigma_y(W_y h_t+b_y)$$

- Jordan network

$$h_t=\sigma_h(W_h x_t+U_h y_{t-1}+b_h)$$
$$y_t=\sigma_y(W_y h_t+b_y)$$



Recurrent networks vs. Recursive neural networks

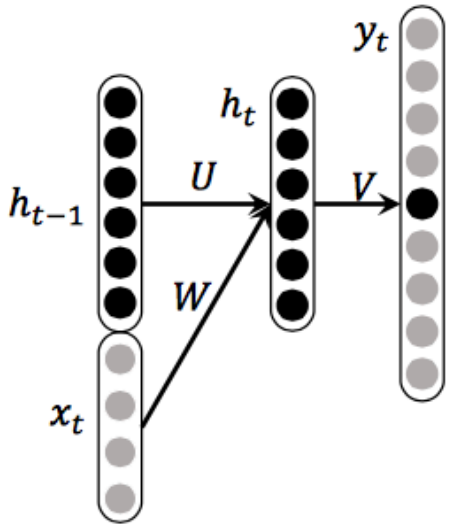


Figure 1: Recurrent neural network

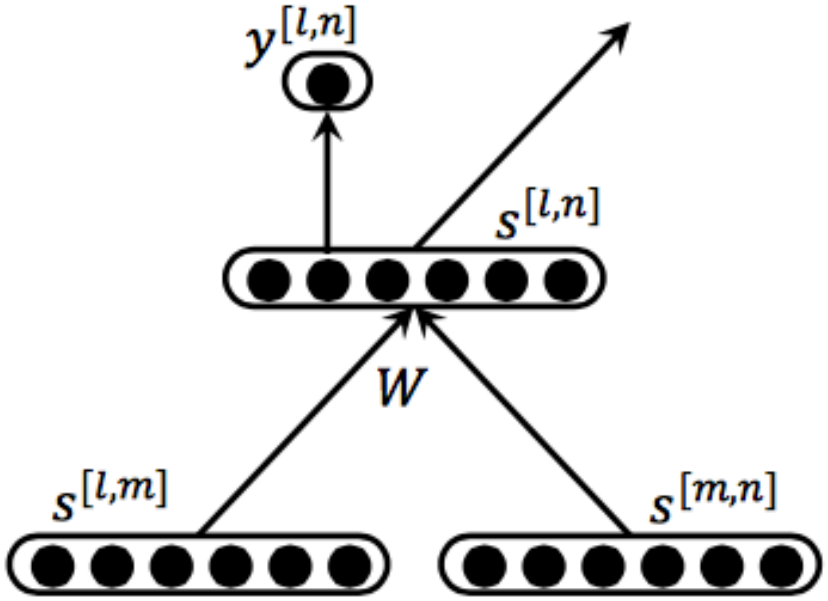


Figure 2: Recursive neural network

Recurrent recursive neural networks

(R^2NN).

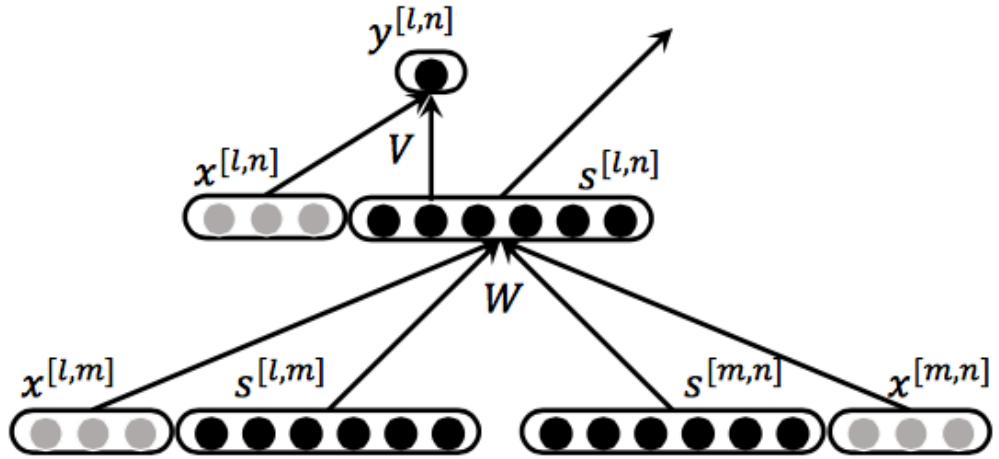


Figure 3: Recursive recurrent neural network

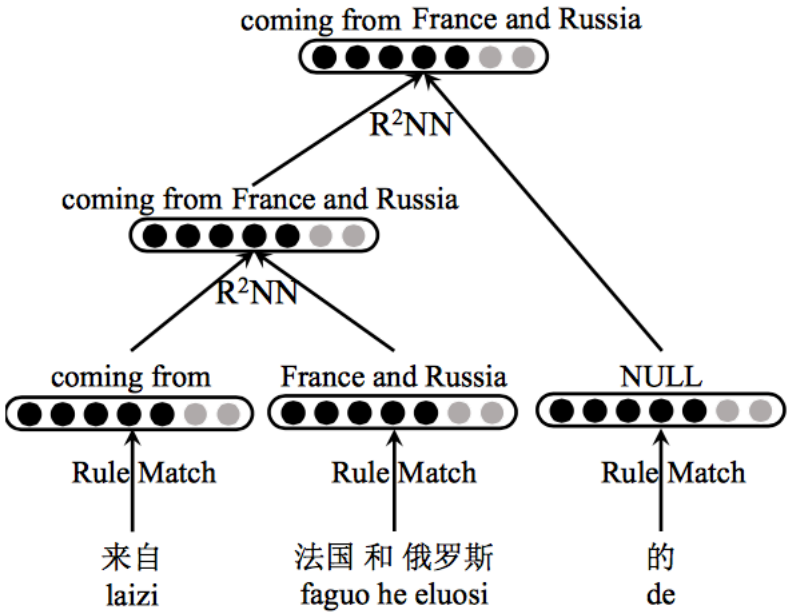


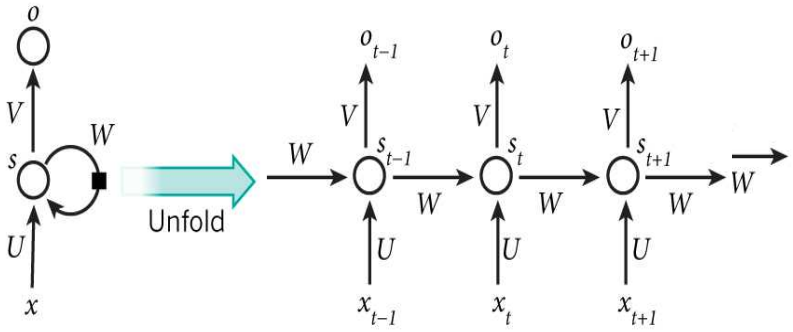
Figure 4: R^2NN for SMT decoding

Echo State Networks

- How to make the recurrent hidden units to capture the history information of input.
- Liquid state machines; reservoid computing
- i.e. map a sequence of inputs up to time t to a fixed- length vector $h^{(t)}$.
- How do we set the input and recurrent weights so that a rich set of history can be represented in the recurrent neural networks?

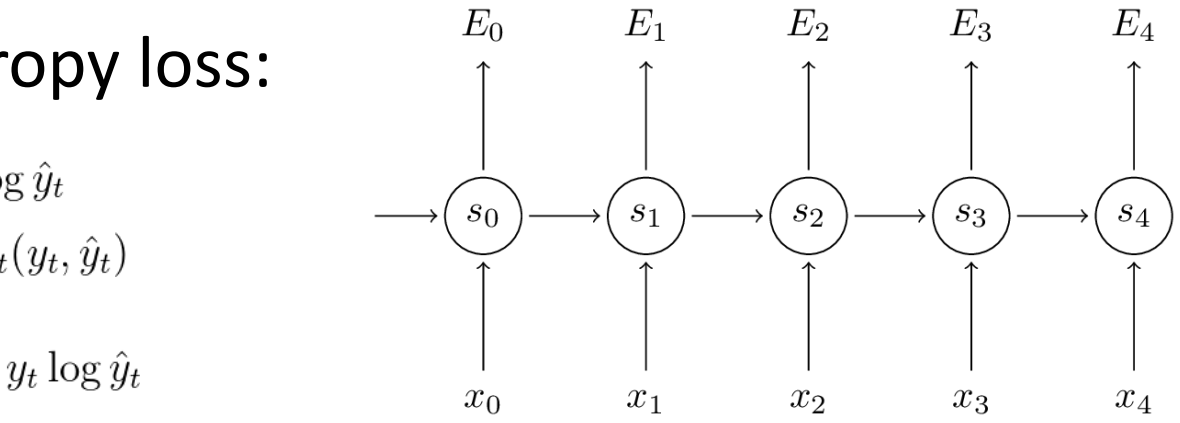
Backpropagation Through Time (BPTT)

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{softmax}(Vs_t)$$



- cross entropy loss:

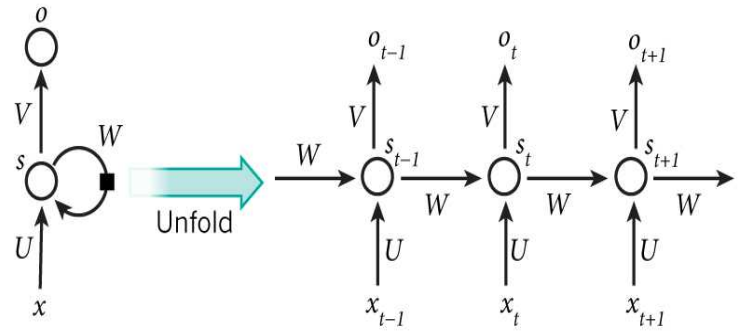
$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$
$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$
$$= -\sum_t y_t \log \hat{y}_t$$



http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

Calculate the gradients of the error with respect to our parameters U, V, and W and then learn good parameters using Stochastic Gradient Descent.

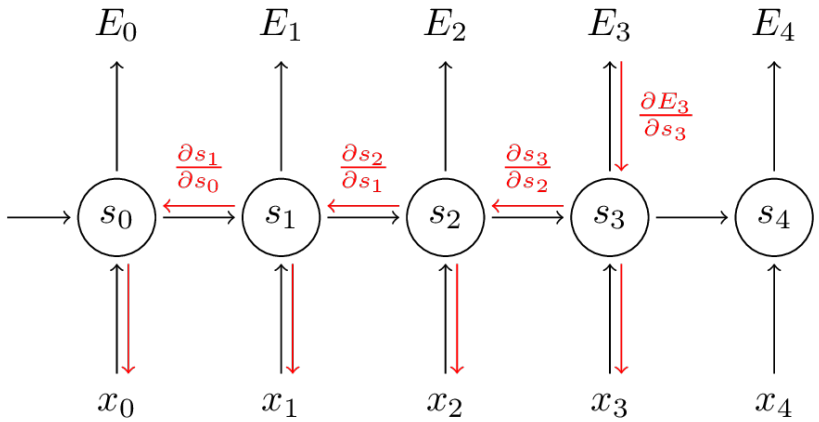
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$



$$
\begin{aligned}
\frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} \\
&= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} \\
&= (\hat{y}_3 - y_3) \otimes s_3
\end{aligned}
$$

$$
\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}
$$

$$
\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}
$$



In the above, $z_3 = V s_3$ and $\otimes$ is th outer product of two vectors.

Vanishing and exploding gradient problems in RNN

- originally discovered by Sepp Hochreiter in 1991
- In long term dependencies,

$$h^{(t)} = W^T h^{(t-1)}$$

- Eigen decomposition (matrix singular value decomposition SVD in linear algebra)

$$W = QAQ^T$$

$$h^{(t)} = Q^T A^{(t)} Q\, h^{(0)}$$

- Eigen values in A greater than 1, exploding, less than 1, vanishing when t ->∞

Recurrent neural network-- LSTM

- The state whose evolution depends both on the input to the system and on the current state:

$$x(t + 1) = x(t) + g(\text{context})\, i(t)$$

Where g(context) a programmable gate depending on context.

L S T M

- LSTM is normally augmented by recurrent gates called forget gates.
- LSTM RNNs prevent backpropagated errors from vanishing or exploding.
- LSTM has also become very popular in the field of natural language processing.
- Unlike previous models based on HMMs and similar concepts, LSTM can learn to recognise context-sensitive languages.

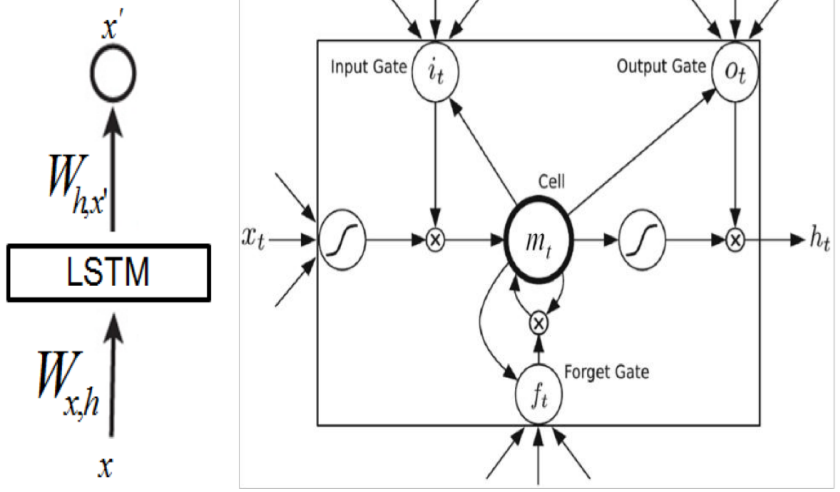
- introduced by Hochreiter & Schmidhuber (1997),

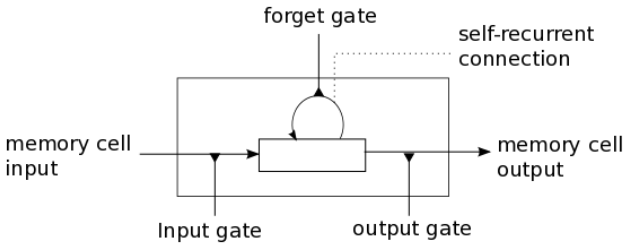
Long distance dependency problems in NLP

For example,

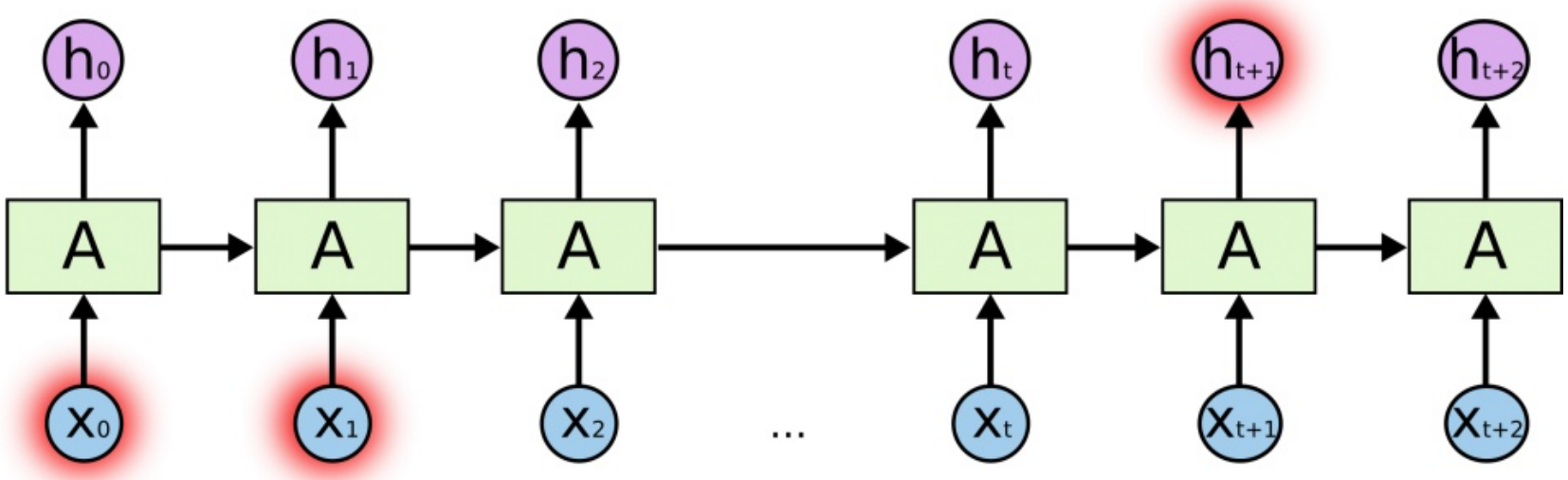
The solution to this problem, Pat said that someone claimed you thought I would never find____.

LSTM

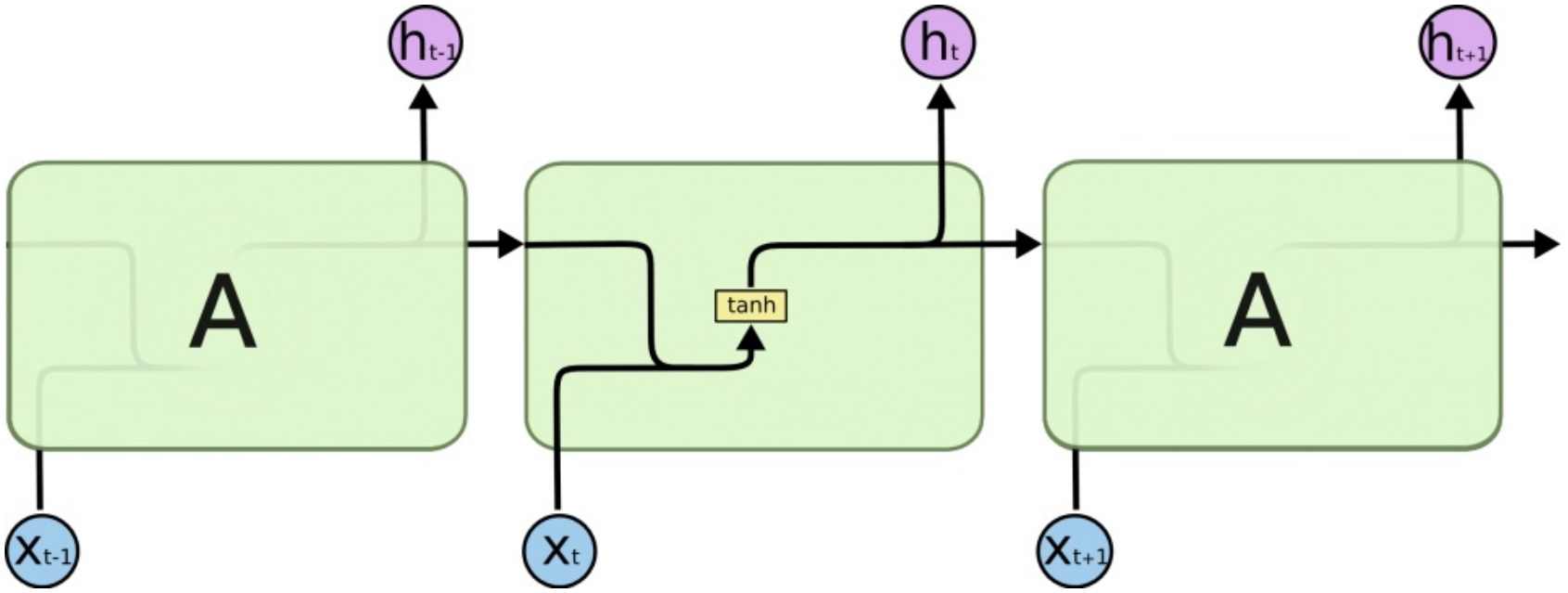

x'
$W_{h,x'}$
LSTM
$W_{x,h}$
x
Input Gate
i_t
Output Gate
o_t
Cell
x_t
m_t
h_t
Forget Gate
f_t


forget gate
self-recurrent
connection
memory cell
input
memory cell
output
Input gate
output gate

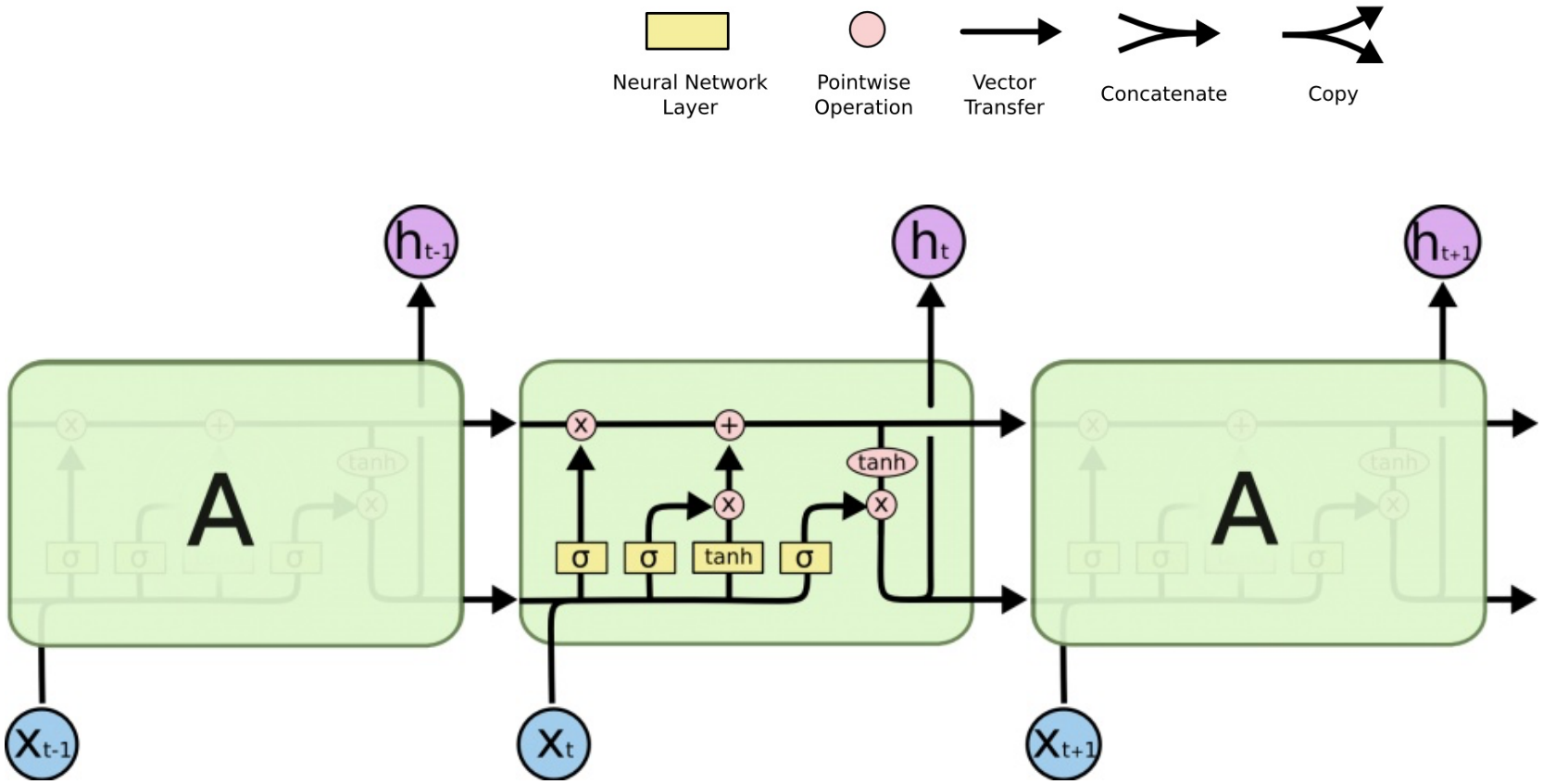
RNN-STM



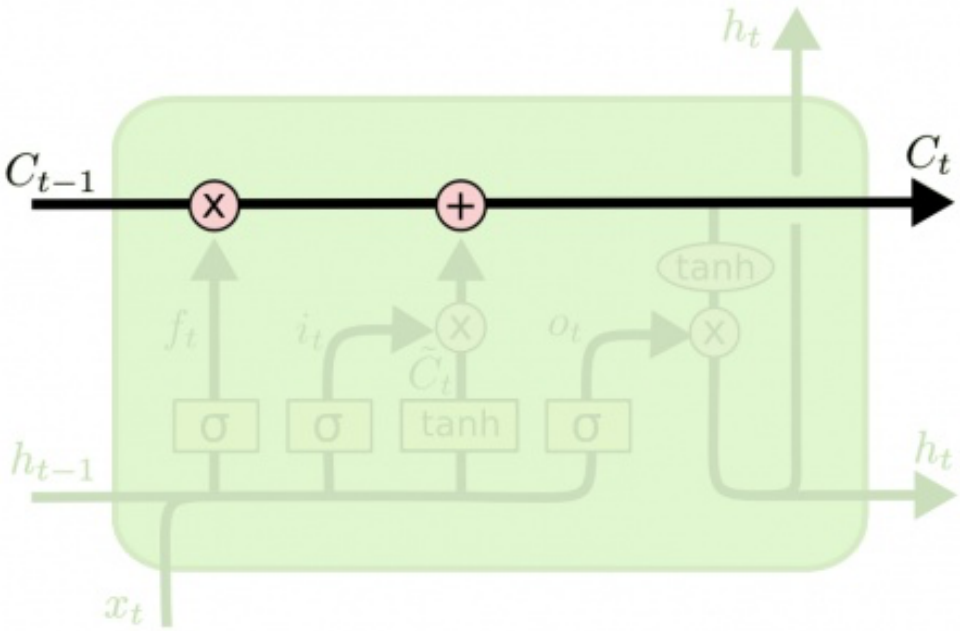
RNN-LSTM



LSTM

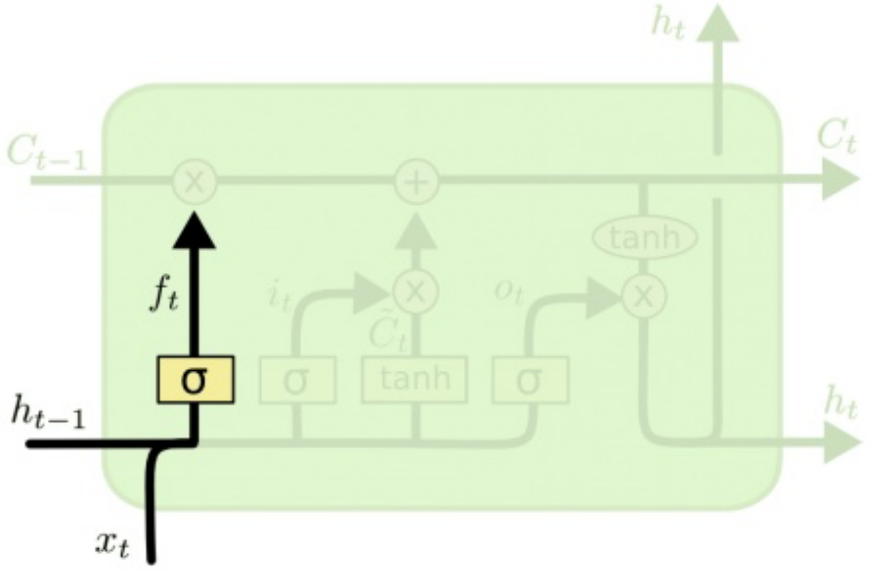

Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy
h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}

Main connection line



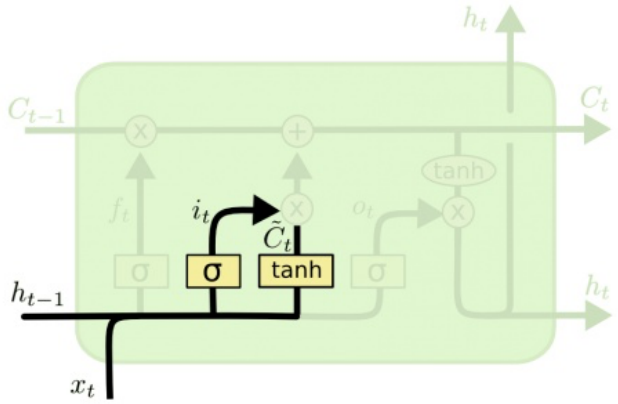
Forget gate

- f_t=0 forget; f_t=1 remain completely



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

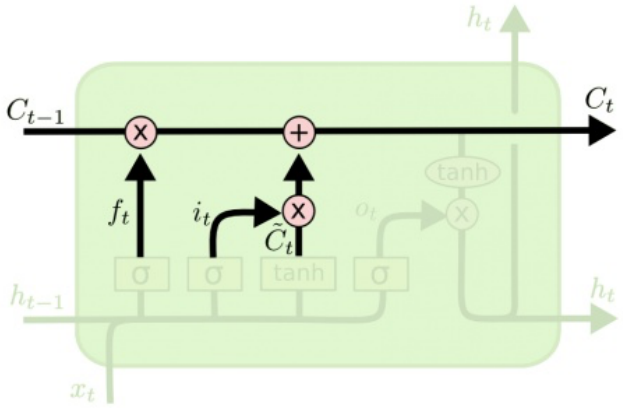
Input gate



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

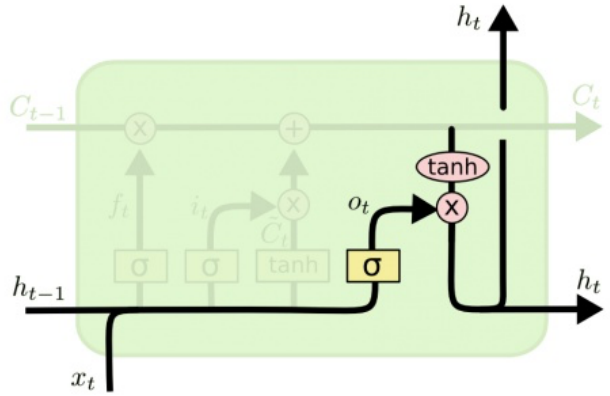
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Input gate connect with forget gate



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output gate

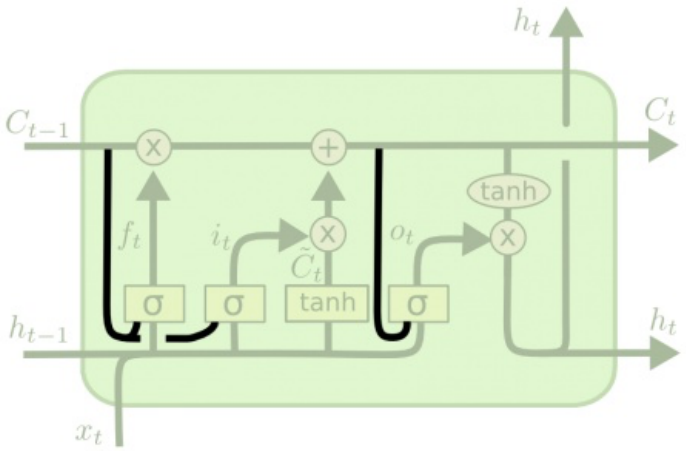


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

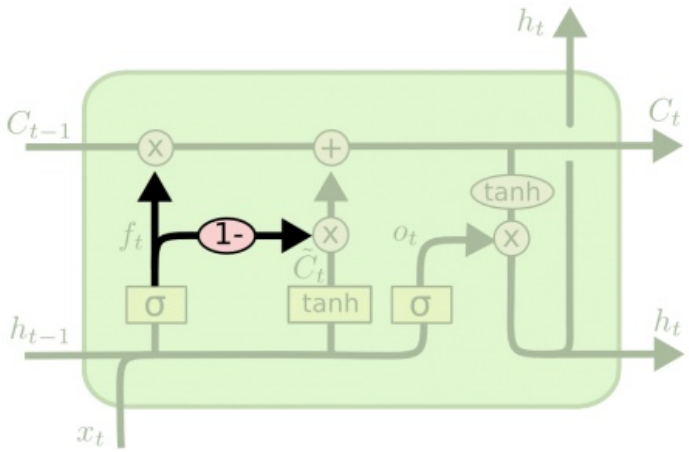
Peephole connection

- Gers & Schmidhuber (2000) introduced a popular LSTM variant by，adding a peephole connection that allows input and forget gates to connect to the main connection C_{t-1} while output gate connecting to C_t



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

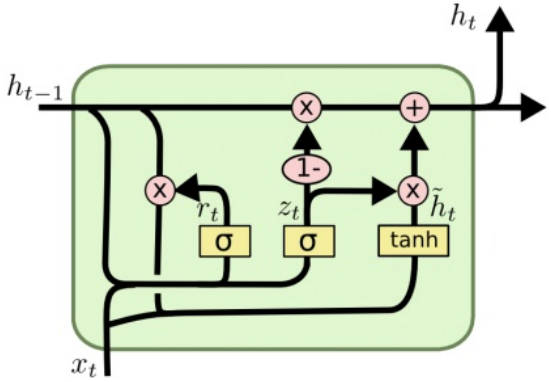
Peephole connection



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

Gated Recurrent Unit (GRU)

- Cho, et al(2014), combine forget gate and input gate into an update gate; combine neuron's state and hidden state and other change.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

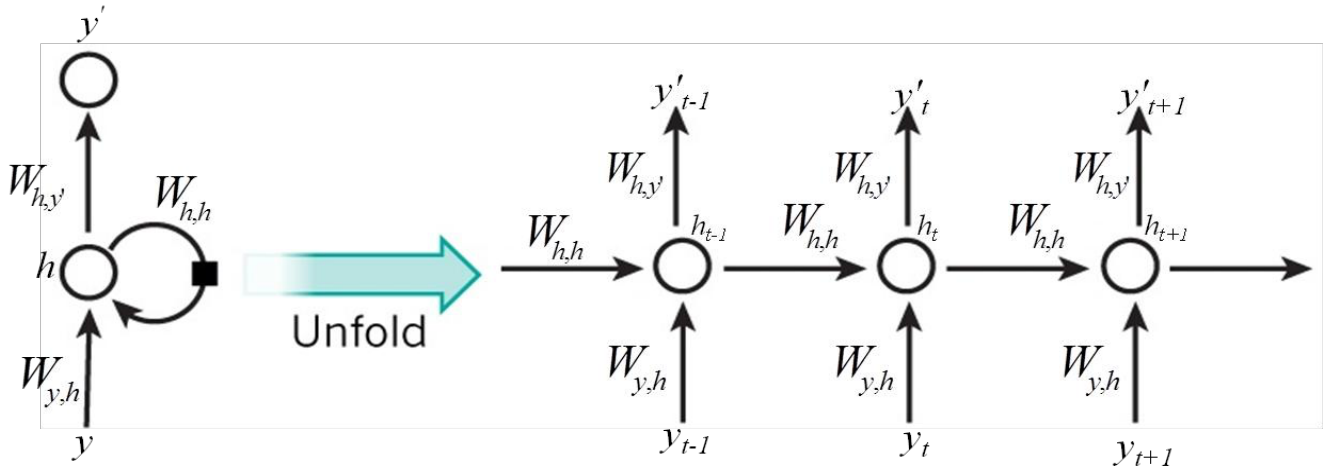
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

原文網址：https://read01.com/N6n3Ba.html

Gated RNN 2014

- **Gated recurrent unit**s (**GRU**s) are a gating mechanism in recurrent neural networks, introduced in 2014.
- Their performance on polyphonic music modeling and speech signal modeling was found to be similar to that of long short-term memory.[1]
- They have fewer parameters than LSTM, as they lack an output gate.

Gated recurrent NN



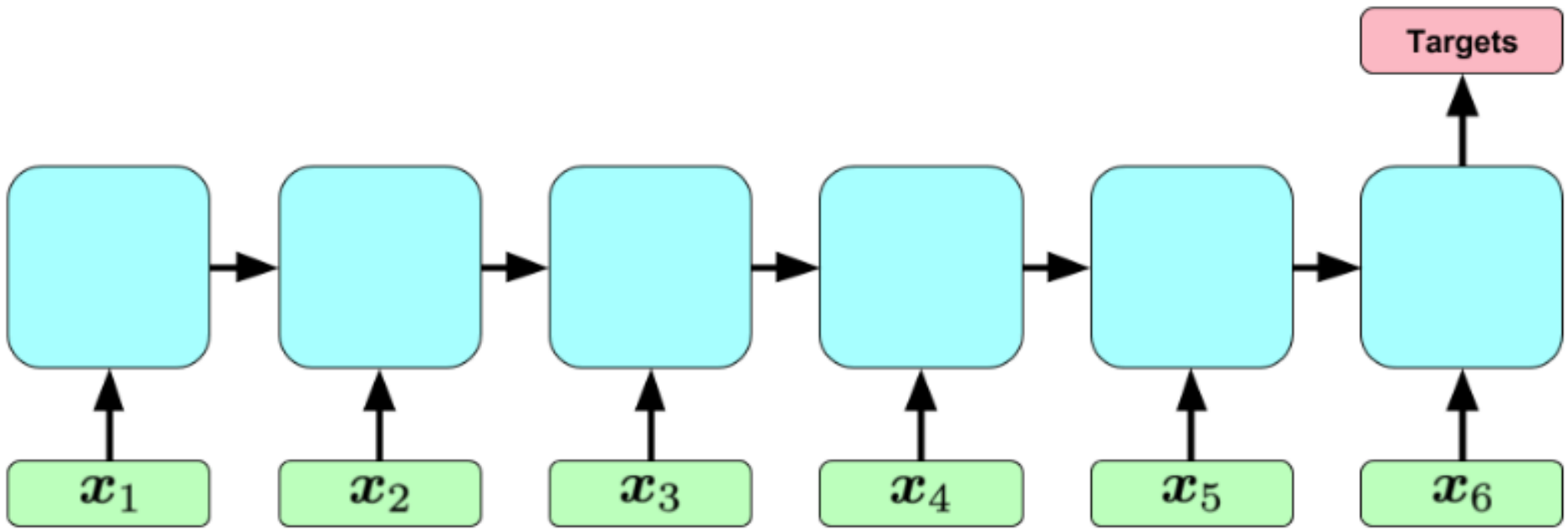
$h_t = F(W_{y,h}y_t + W_{h,h}h_{t-1} + b_h)$

$y'_t = G(W_{h,y'}h_t + b_{y'})$

Difference between GRU and LSTM

- A GRU has two gates, an LSTM has three gates.
- GRUs don’t possess an internal memory that is different from the exposed hidden state. They don’t have the output gate that is present in LSTMs.
- The input and forget gates are coupled by an update gate and the reset gate is applied directly to the previous hidden state. Thus, the responsibility of the reset gate in a LSTM is really split up into both r and z.
- We don’t apply a second nonlinearity when computing the output.

$$\text{loss}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{|L|} \sum_{l=1}^{l=|L|} -(y_l \cdot \log(\hat{y}_l) + (1 - y_l) \cdot \log(1 - \hat{y}_l)).$$



$$\alpha \cdot \frac{1}{T} \sum_{t=1}^{T} \text{loss}(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}) + (1 - \alpha) \cdot \text{loss}(\hat{\boldsymbol{y}}^{(T)}, \boldsymbol{y}^{(T)})$$

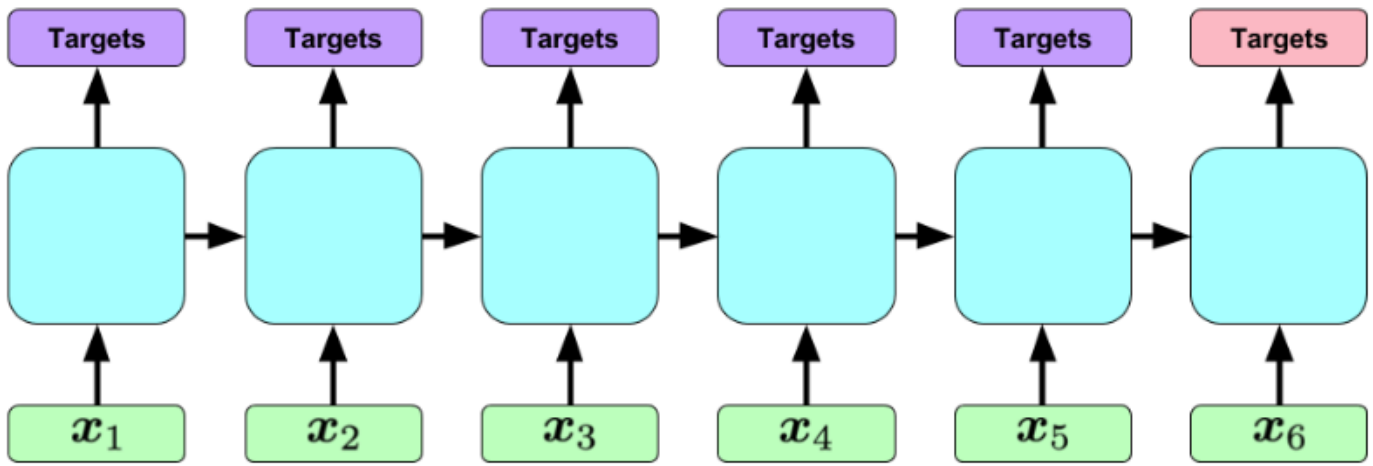


Figure 2: An RNN classification model with *target replication*. The primary target (depicted in red) at the final step is used at prediction time, but during training, the model back-propagates errors from the intermediate targets (purple) at every sequence step.

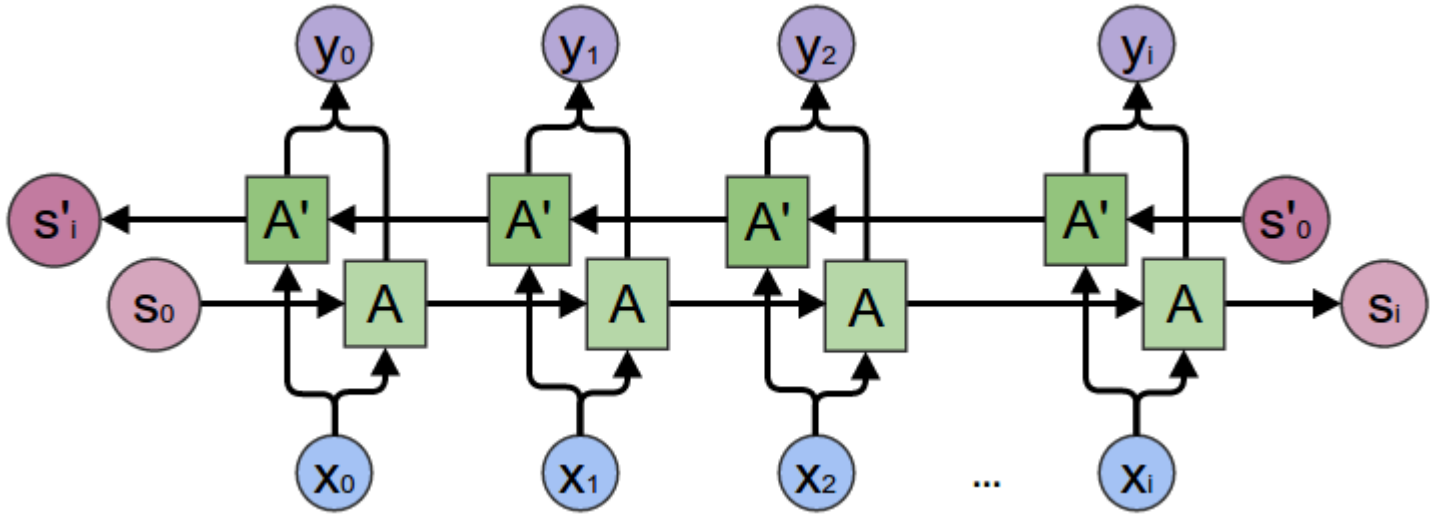
Why bidirectional RNN?

- The output layer can get information from past and future states.
- BRNN are especially useful when the context of the input is needed
- For example:

- 我今天不舒服，我打算 __ 一天。
- 只根据‘不舒服‘，可能推出我打算‘去醫院‘，‘睡覺‘，‘請假‘等等，但如果加上後面的‘一天‘，能選擇的範圍就變小了，‘去醫院‘就不能選了，而‘請假‘‘休息‘之類的被選概率就會更大。

Bi-directional RNN

Fig 1: General Structure of Bidirectional Recursive Neural Networks. Source: colah’ s blog



Simple RNN vs. bi-directional RNN

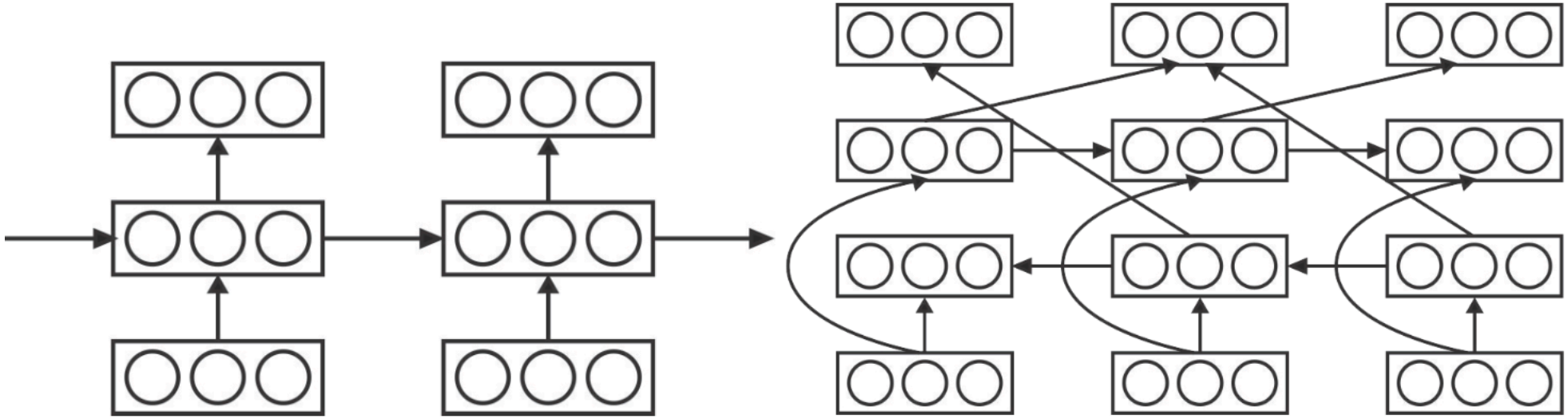


Figure 1: Structure of the simple RNN (left) and the bidirectional RNN (right).

Simple RNN vs bidirectional RNN

- Simple RNN

$$P\left(\mathbf{y}_t \mid \{\mathbf{x}_d\}_{d=1}^{t}\right) = \phi\left(\mathbf{W}_{\mathrm{y}}\mathbf{h}_t + \mathbf{b}_{\mathrm{y}}\right)$$

$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{\mathrm{h}}\mathbf{h}_{t-1} + \mathbf{W}_{\mathrm{x}}\mathbf{x}_t + \mathbf{b}_{\mathrm{h}}\right)$$

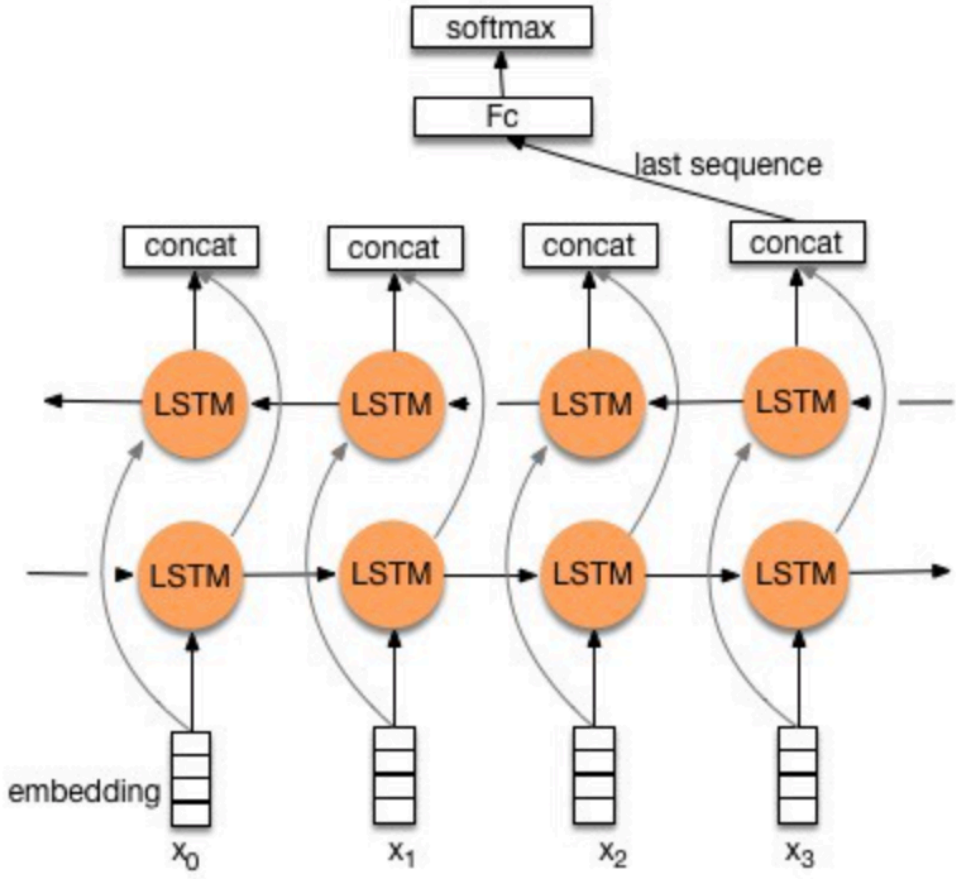
- Bi-directional RNN

$$P\left(\mathbf{y}_t \mid \{\mathbf{x}_d\}_{d\neq t}\right) = \phi\left(\mathbf{W}_{\mathrm{y}}^{\mathrm{f}}\mathbf{h}_{t-1}^{\mathrm{f}} + \mathbf{W}_{\mathrm{y}}^{\mathrm{b}}\mathbf{h}_{t+1}^{\mathrm{b}} + \mathbf{b}_y\right)$$

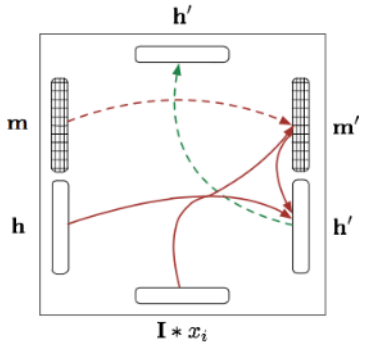
$$\mathbf{h}_t^{\mathrm{f}} = \tanh\left(\mathbf{W}_{\mathrm{h}}^{\mathrm{f}}\mathbf{h}_{t-1}^{\mathrm{f}} + \mathbf{W}_{\mathrm{x}}^{\mathrm{f}}\mathbf{x}_t + \mathbf{b}_{\mathrm{h}}^{\mathrm{f}}\right)$$

$$\mathbf{h}_t^{\mathrm{b}} = \tanh\left(\mathbf{W}_{\mathrm{h}}^{\mathrm{b}}\mathbf{h}_{t+1}^{\mathrm{b}} + \mathbf{W}_{\mathrm{x}}^{\mathrm{b}}\mathbf{x}_t + \mathbf{b}_{\mathrm{h}}^{\mathrm{b}}\right).$$

Bi-directional LSTM

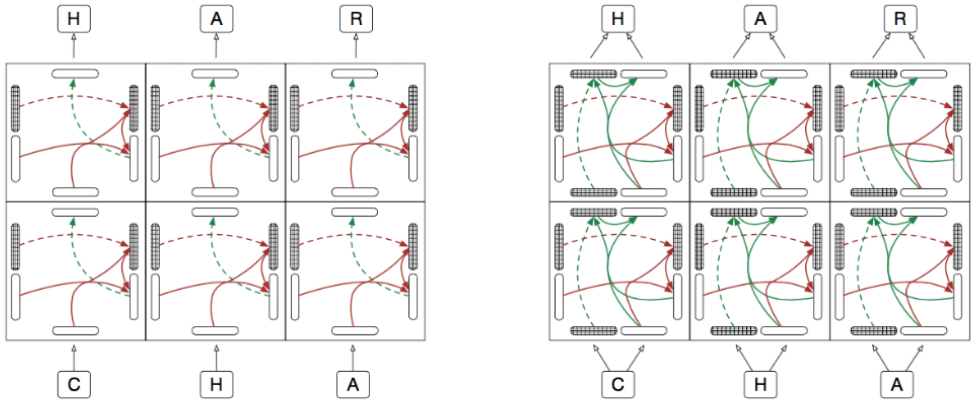


Grid LSTM



Standard LSTM block

Figure 1



Stacked LSTM 2d Grid LSTM

Figure 2

$$
\begin{aligned}
g^u &= \sigma(W^u H) \\
g^f &= \sigma(W^f H) \\
g^o &= \sigma(W^o H) \\
g^c &= tanh(W^c H) \\
m' &= g^f \odot m + g^u \odot g^c \\
h' &= tanh(g^o \odot m')
\end{aligned}
$$

The past inputs x_1, ..., x_{i-1} determine the state of the network that comprises a *hidden* vector $h \in R_d$ and a *memory* vector $m \in R_d$

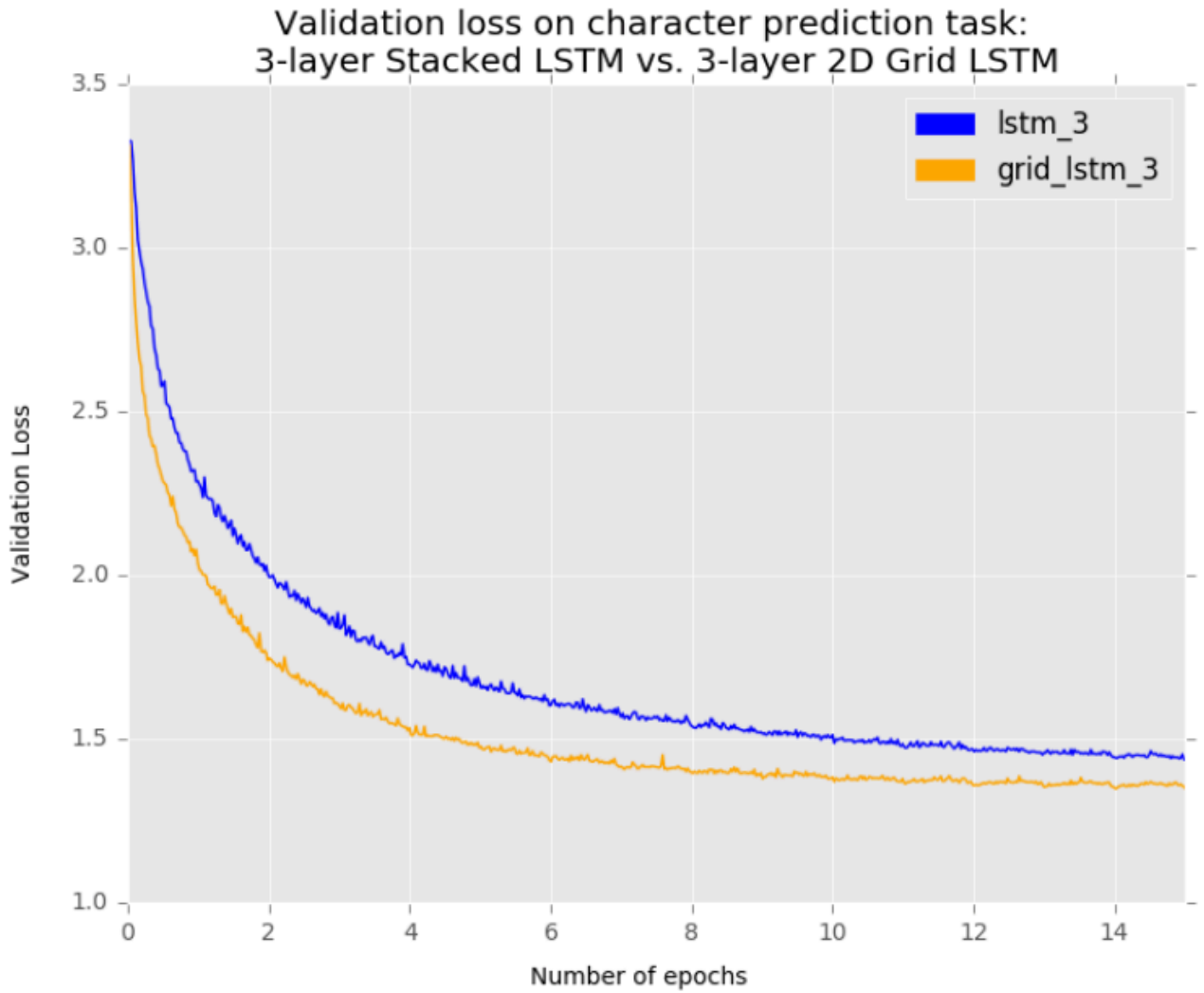

Validation loss on character prediction task:
3-layer Stacked LSTM vs. 3-layer 2D Grid LSTM
lstm_3
grid_lstm_3
Validation Loss
3.5
3.0
2.5
2.0
1.5
1.0
0
2
4
6
8
10
12
14
Number of epochs

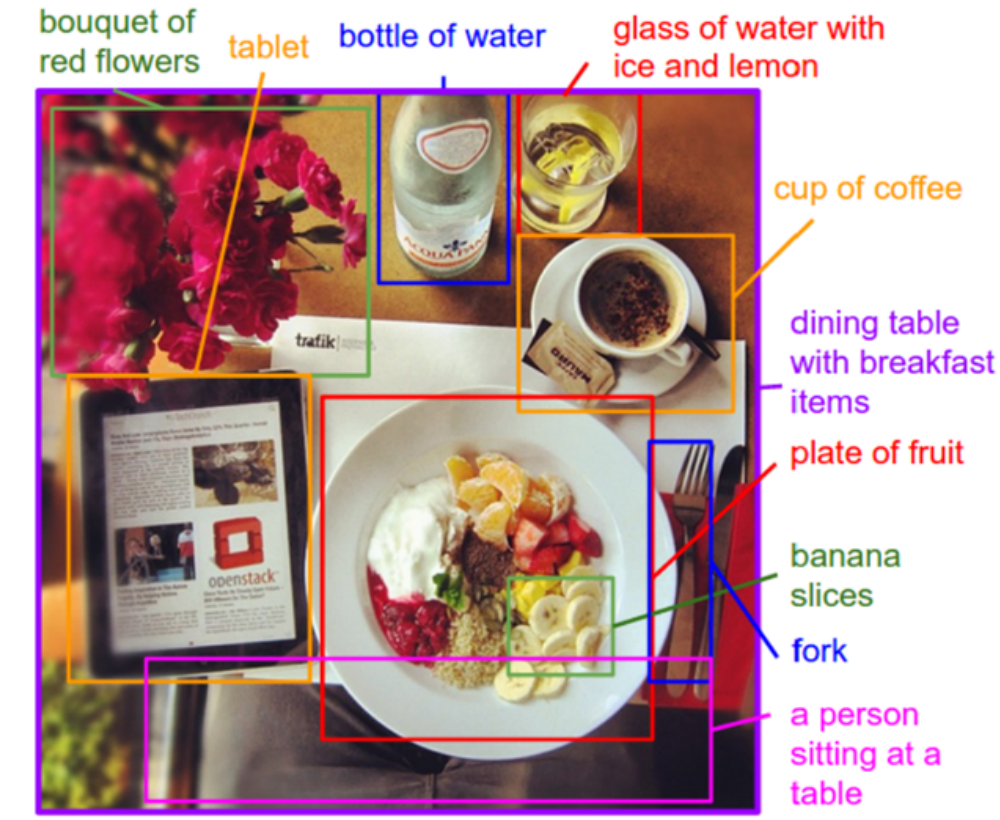
Figure 4

Generating Image Descriptions (2014)

- Andrej Karpathy and Fei-Fei Li
- combination of CNNs and bidirectional RNNs

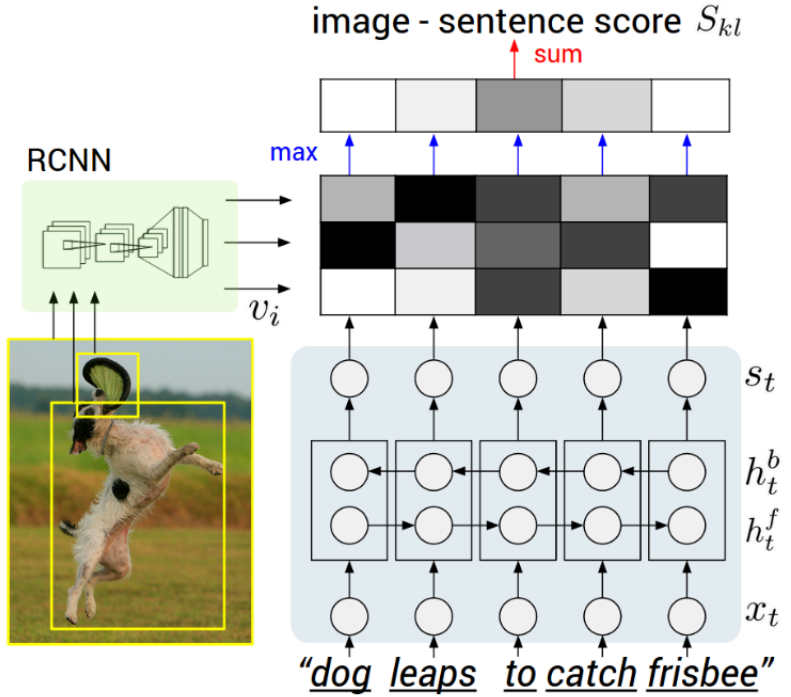
  (Recurrent Neural Networks)

- To generate natural language description



Example output of the model

$$x_t = W_w \mathbb{I}_t$$
$$e_t = f(W_e x_t + b_e)$$
$$h_t^f = f(e_t + W_f h_{t-1}^f + b_f)$$
$$h_t^b = f(e_t + W_b h_{t+1}^b + b_b)$$
$$s_t = f(W_d(h_t^f + h_t^b) + b_d)$$



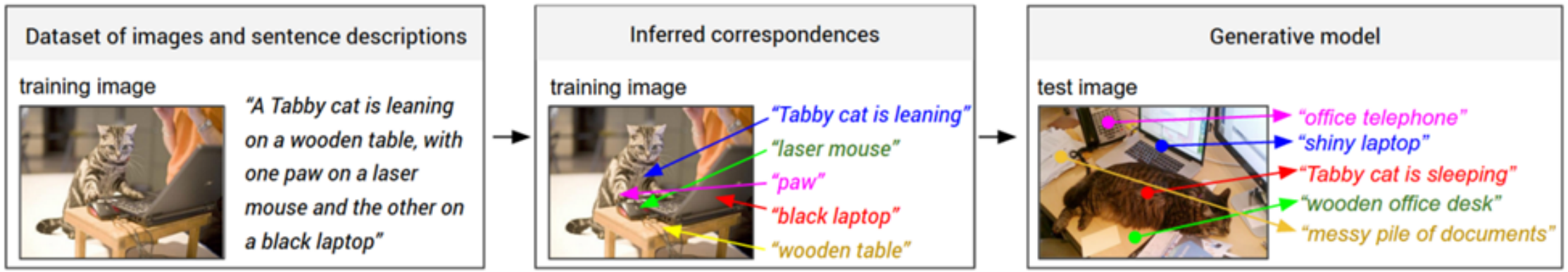
- Here, $\mathbb{I}_t$ is an indicator column vector that has a single one at the index of the t-th word in a word vocabulary. The weights W_w specify a word embedding matrix that we initialize with 300-dimensional word2vec weights and keep fixed due to overfitting concerns.

- Using this training data, a deep neural network “infers the latent alignment between segments of the sentences and the region that they describe”
- Alignment Model: measure of similarity between i^{th} region and t^{th} word
- g_k is the set of image fragments in image k and
- g_l is the set of sentence fragments in sentence *l*.

$$S_{kl} = \sum_{t \in g_l} \max_{i \in g_k} v_i^T s_t$$

- Generation Model:

Workflow of generating image description



Workflow of alignment and generative model

Datasets

- Flickr8K: 8,000 images ; Flickr30K: 31,000 images
- MSCOCO: 123,000 images
- Each image is annotated with 5 sentences using Amazon Mechanical Turk
- For Flickr8K and Flickr30K, 1,000 images for validation, 1,000 images for testing, and the rest for training.
- For MSCOCO we use 5,000 images for both validation and testing
- Filter words to those that occur at least 5 times in the training set, which results in 2538, 7414, and 8791 words for Flickr8K, Flickr30K, and MSCOCO datasets, respectively.

BLEU score

- Evaluate the quality of descriptive sentences of a bag of words w.r.t. ground true annotations

- BLEU = $\min(1, \frac{output\ length}{reference\ length})(\prod_{i=1}^{4} precision_i)^{1/4}$

$$\text{BLEU} = \min\left(1, \frac{output\ length}{reference\ length}\right)\left(\prod_{i=1}^{4} precision_i\right)^{1/4}$$

SYSTEM A: Israeli officials (2-GRAM MATCH) responsibility of airport (1-GRAM MATCH) safety

REFERENCE: Israeli officials are responsible for airport security

SYSTEM B: airport security (2-GRAM MATCH) Israeli officials are responsible (4-GRAM MATCH)

Metric	System A	System B
precision (1gram)	3/6	6/6
precision (2gram)	1/5	4/5
precision (3gram)	0/4	2/4
precision (4gram)	0/3	1/3
brevity penalty	6/7	6/7
BLEU	0%	52%

Meteor score

Precision = m/w_t

Recall = m/w_r

- *m* is the number of unigrams in the candidate translation that are also found in the reference translation,
- w_t, w_r number of unigrams in candidate and reference respectively
- Harmonic mean of precision and recall where recall weighted 9 times more than precision:

$$F_{mean} = 10 * P*R/(R + 9P)$$

Chunk penalty p: The penalty p is computed as follows, $p = 0.5 (c / u_m)^3$
Where *c* is the number of chunks, and u_m is the number of unigrams that have been mapped.

- $M= F_{mean}(1 - p)$

CIDEr (Consensus-based Image Description Evaluation)

- To evaluate for image I_i how well a candidate sentence c_i matches the consensus of a set of image descriptions $S_i = \{s_{i1}, \ldots, s_{im}\}$.
- The number of times an n-gram ω_k occurs in a reference sentence s_{ij} is denoted by $h_k(s_{ij})$ or $h_k(c_i)$ for the candidate sentence c_i .
- We compute the TF-IDF weighting $g_k(s_{ij})$ for each n-gram ω_k using:

$$g_k(s_{ij}) = \frac{h_k(sij)}{\sum_{w_l \in \Omega}} \log\left(\frac{|I|}{\sum_{I,p \in I} \min(1, \sum_q h_k(spq))}\right)$$

where Ω is the vocabulary of all n-grams and I is the set of all images in the dataset.

CIDErn score

- CIDErn score for n-grams of length n is computed using the average cosine similarity between the candidate sentence and the reference sentences, which accounts for both precision and recall:

$$\text{CIDEr}_n(c_i, S_i) = 1/m \sum_j \frac{g_n(ci) \cdot gn(sij)}{||g_n(ci)|| \cdot ||gn(sij)||},$$

where $g_n(c_i)$ is a vector formed by $g_k(c_i)$ corresponding to all n-grams of length n and $||g_n(c_i)||$ is the magnitude of the vector $g_n(c_i)$. j is indexed as a sentence for each image i and m as the total number of sentences.

- Similarly for $g_n(s_{ij})$.

- $\text{CIDEr}(c_i,S_i) = \sum_{n=1}^{N} w_n CIDEr_n(ci, Si)$

  where uniform weight $w_n = 1/N$ works the best.

ROUGE score for text summary (2004)

- ROUGE-N: N-gram based co-occurrence statistics
- ROUGE-L: LCS-based statistics
- ROUGE-W: Weighted LCS-based statistics that favors consecutive LCSes
- ROUGE-S: Skip-bigram-based co-occurrence statistics
- ROUGE-SU: Skip-bigram plus unigram-based co-occurrence statistics

- Example:
  1. *police killed the gunman*
  2. police kill the gunman
  3. the gunman kill police
- ROUGE-N: S2=S3 (“police”, “the gunman”)
- ROUGE-L:
  - S2=3/4 (“police the gunman”)
  - S3=2/4 (“the gunman”)
  - S2>S3

Automatic evaluation scores critiques

- Ignore relevance of words
  (names and core concepts more important than determiners and punctuation)
- Operate on local level
  (do not consider overall grammaticality of the sentence or sentence meaning)
- Scores are meaningless
  (scores very test-set specific, absolute value not informative)
- Human translators score low on BLEU
  (possibly because of higher variability, different word choices)

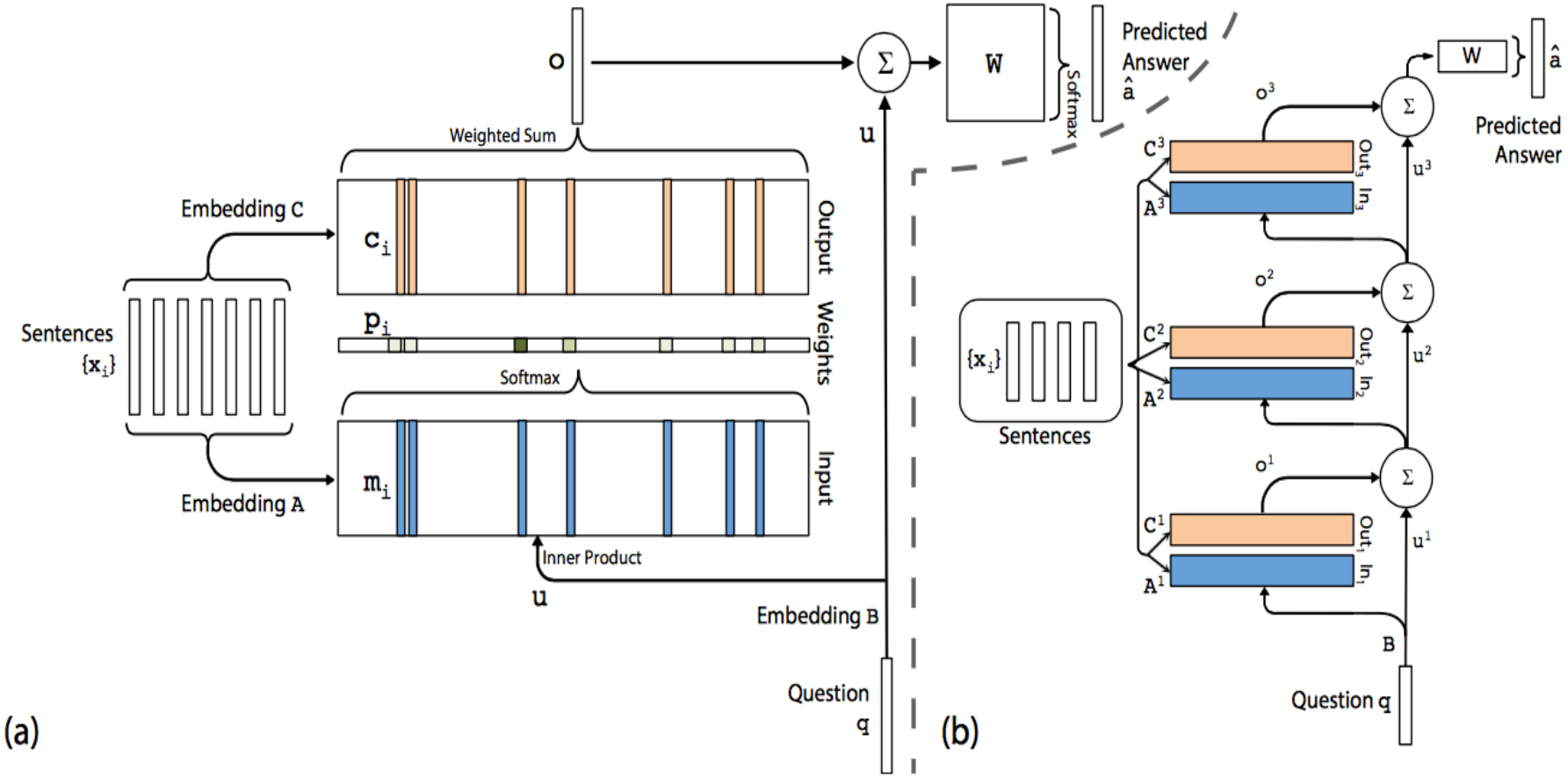
MemoryNet

- Attention mechanism
- Mapping into semantic vectors
- Q/A matching
- Hopping

- Children story book

- Sam walks into the kitchen.
- Sam picks up an apple.
- Sam walks into the bedroom.
- Sam drops the apple.
- Q: Where is the apple?
- A. Bedroom

Deep learning approach for Q/A: Memory networks



Single layer memory net

- The query q is embedded matrix B to obtain an internal state u.

  $u = Bq$

  $m_i = Ax_i$

  $c_i = Cx_i$

  $p_i = \text{Softmax}(u^T m_i)$. [$\text{Softmax}(z_i) = e^{z_i} / \Sigma_j e^{z_j}$.]

  $o = \Sigma_i p_i c_i$.

- Answer: $\hat{a} = \text{Softmax}(W(o + u))$
- Three embedding matrices: A, B and C, as well W

Multi-layer memory net (dealing with hopping)

- $u_{k+1} = u_k + o_k$.

The input to W also combines the input and the output of the top memory layer:

$$\hat{a} = \text{Softmax}(W u_{K+1}) = \text{Softmax}(W (o_K + u_K)).$$

Q/A Dealing with hopping

- Sam walks into the kitchen.
- Sam picks up an apple.
- Sam walks into the bedroom.
- Sam drops the apple.
- Q: Where is the apple?
- A. Bedroom

- Mary journeyed to the den.
- Mary went back to the kitchen.
- John journeyed to the bedroom.
- Mary discarded the milk.
- Q: Where was the milk before the den?
- A. Hallway

- Brian is a lion.
- Julius is white.
- Julius is a lion.
- Bernhard is green.
- Q: What color is Brian?
- A : White

Turing Machine

- A Turing machine is a finite state machine with the ability to read and write data to a tape.
- Turing machine is a general purpose computer that can move read-write head, to store program
- Despite the model's simplicity, given any computer algorithm, a Turing machine capable of simulating that algorithm's logic can be constructed.[3]

Neural Turing Machines 2014

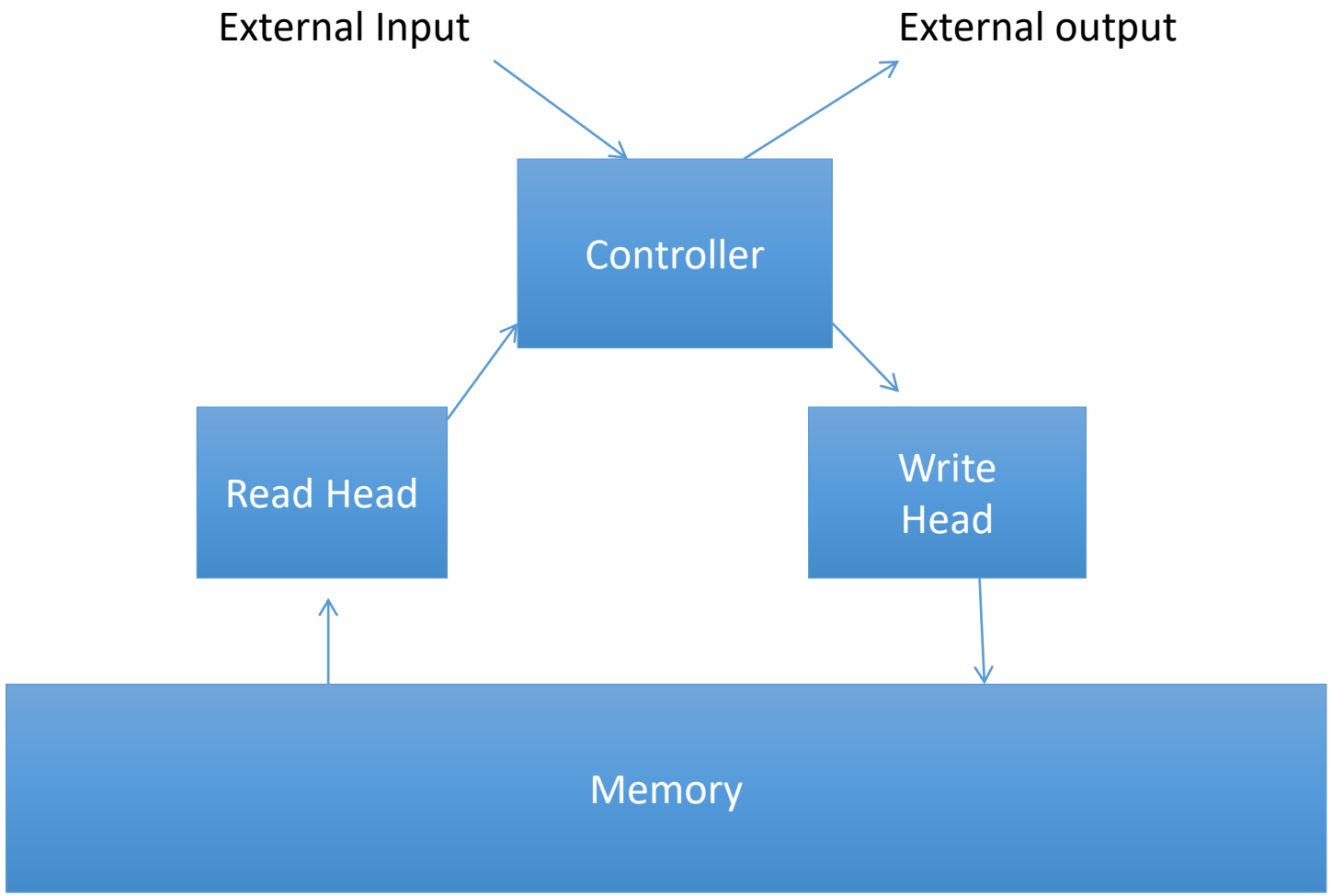
- *Graves, Alex; Wayne, Greg and Danihelka, Ivo (2014)*

- NTM resembles a working memory system, as it is designed to solve tasks that require the application of approximate rules to “rapidly-created variables.
- NTM architecture uses an attentional process to read from and write to memory selectively.

Alex's remark

- To combine the fuzzy pattern matching capabilities of neural networks with the algorithmic power of programmable computers.
- A neural network controller is given read/write access to a memory matrix of floating point numbers, allow it to store and iteratively modify data.
- Neural Turing Machines can infer algorithms from input and output examples alone.
- In other words, they can learn how to program themselves.

NTM


External Input
External output
Controller
Read Head
Write Head
Memory

Characteristics of NTM

- Every component of the architecture is differentiable, making it straightforward to train with gradient descent.

- The controller interacts with the external world via input and output vectors.
- Unlike a standard network, it also interacts with a memory matrix using selective read and write operations.

- The degree of blurriness is determined by an attentional “focus” mechanism that constrains each read and write operation to interact with a small portion of the memory, while ignoring the rest.
- Therefore interaction with the memory is highly sparse.

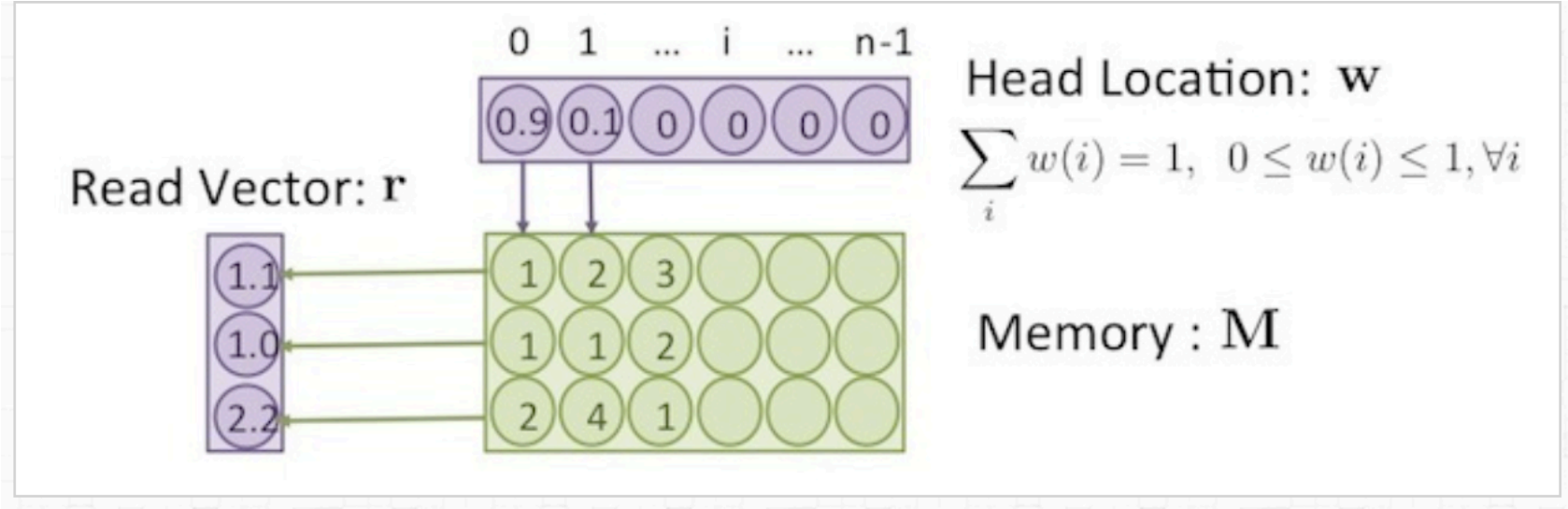
Mathematics of NTM

- Reading
- writing
- Addressing: content addressing; interpolation;
  convolutional shift; sharpening

Reading

- $0 \leq w_t(i) \leq 1$
- $\Sigma^R{}_{i=1} w_t(i)=1$ (R rows of normalized weighting)
- Return vector r with normalized weight w from memory M_t
- $r_t \leftarrow \sum_i^R w_t(i) M_t(i)$

NTM READ EXAMPLE



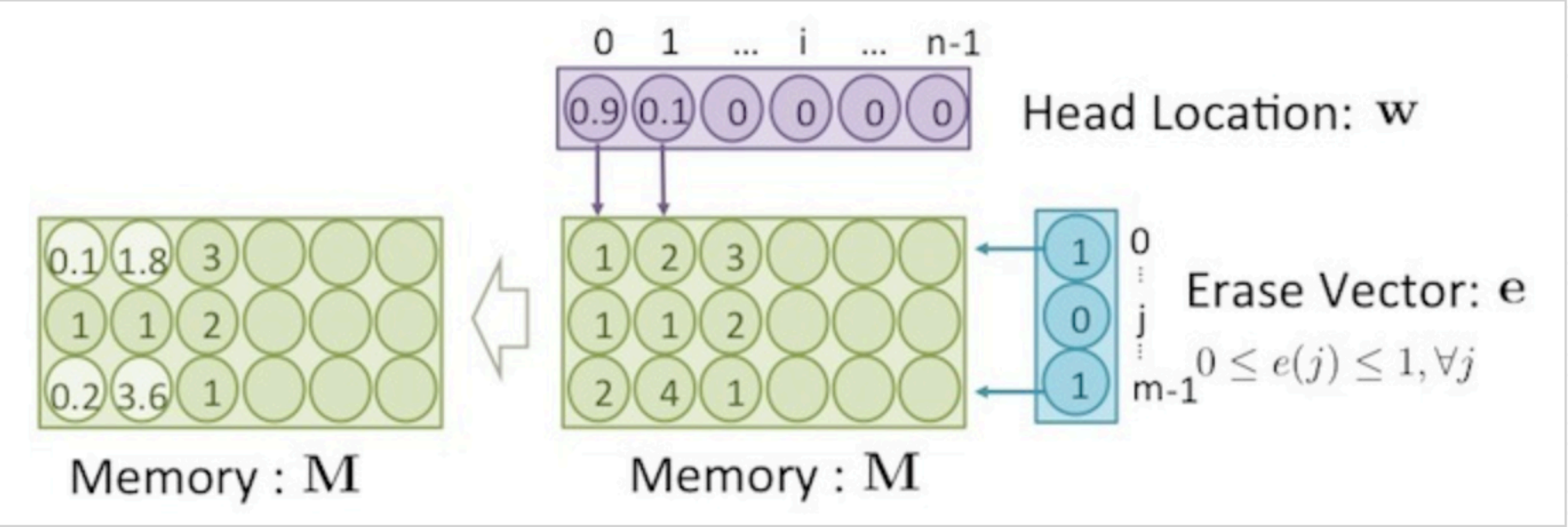
Writing

- Erase operation
- $M^{erased}_t(i) \leftarrow M_{t-1}(i)[\mathbf{1} - w_t(i)e_t]$

- Add operation
- Add vector a_t to memory
- $M_t(i) \leftarrow M^{erased}_t(i) + w_t(i)a_t$

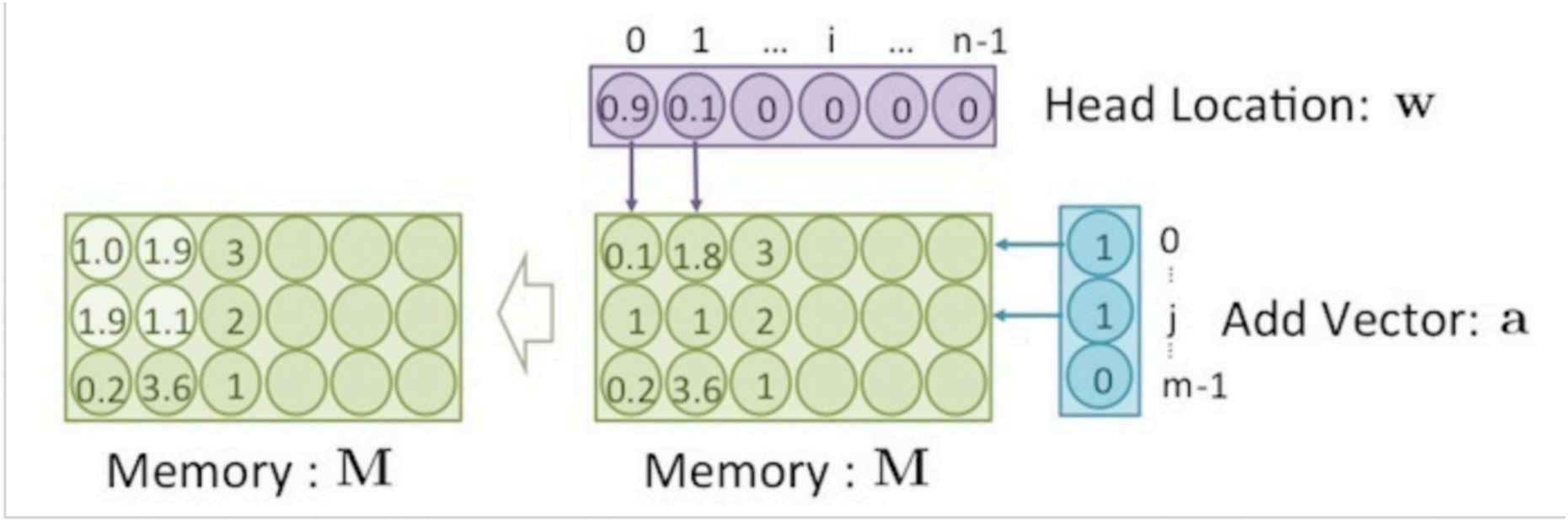
NTM ERASE OPERATION EXAMPLE

- $M^{erased}_t(i) \leftarrow M_{t-1}(i)[\mathbf{1} - w_t(i)e_t]$

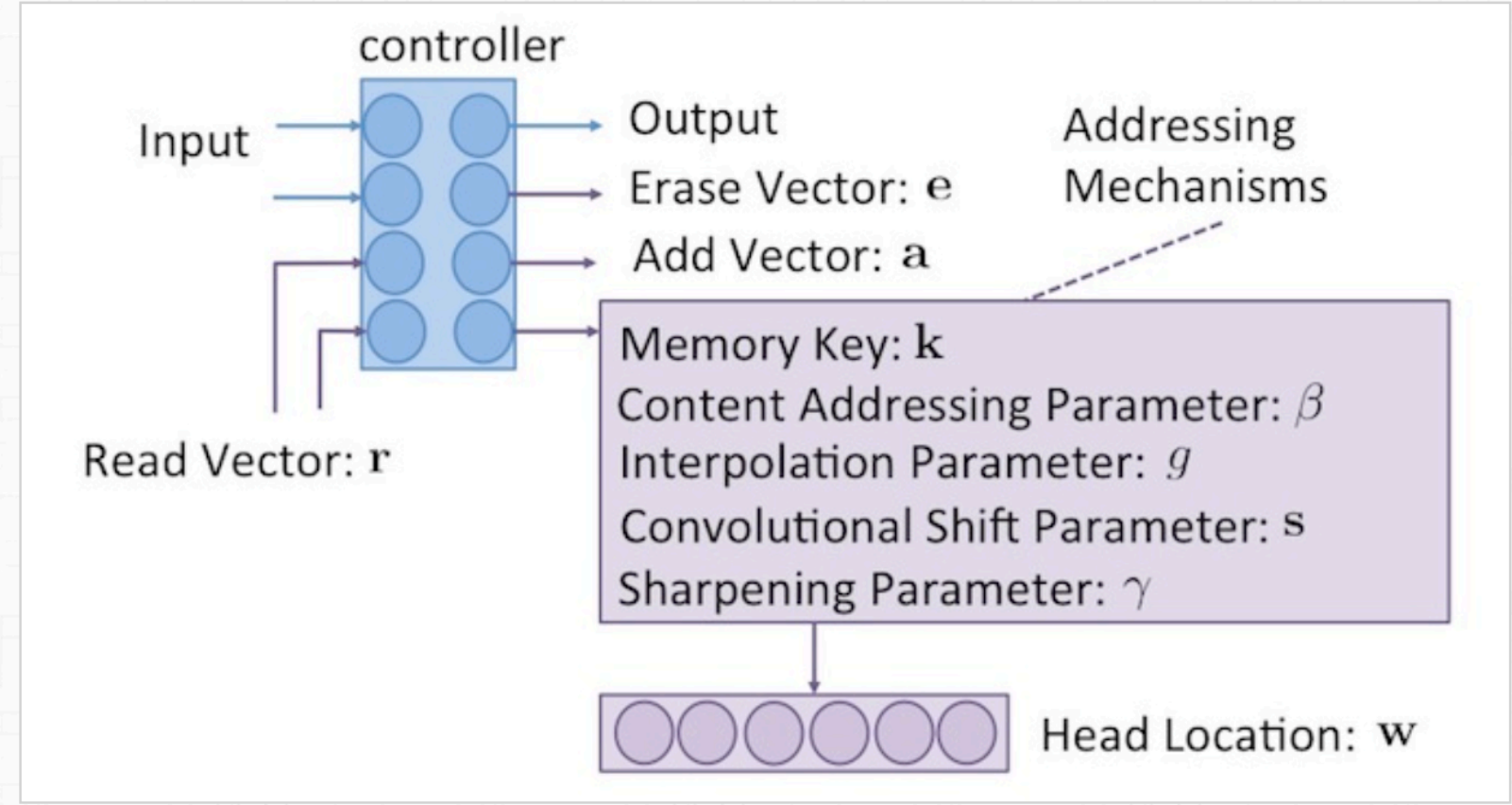


NTM ADD OPERATION EXAMPLE

- $M_t(i) \leftarrow M_t^{erased}(i) + w_t(i)a_t$



Controller: can be a neural networks



http://cpmarkchang.logdown.com/posts/279710-neural-network-neural-turing-machine

- *controller* 根據外部環境的輸入值 *input*，以及 *read vector* **r** ，經過其內部運算，會輸出 *output* 值到外在環境，還有 *erase vector* **e** 和 *add vector* **a** ，來控制記憶體的清除與寫入。但還缺少了讀寫頭向量 **w** 。
- 如果要產生讀寫頭向量 **w** ， 需要透過一連串的 *Addressing Mechanisms* 的運算，最後即可得出讀寫頭位置。而 *controller* 則負責產生出 *Addressing Mechanisms* 所需的參數。

Addressing

- Focusing by Content
- Content similarity addressing:

  $K(u,v) = u \bullet v / ||u|| \, ||v||$
- Interpolation:

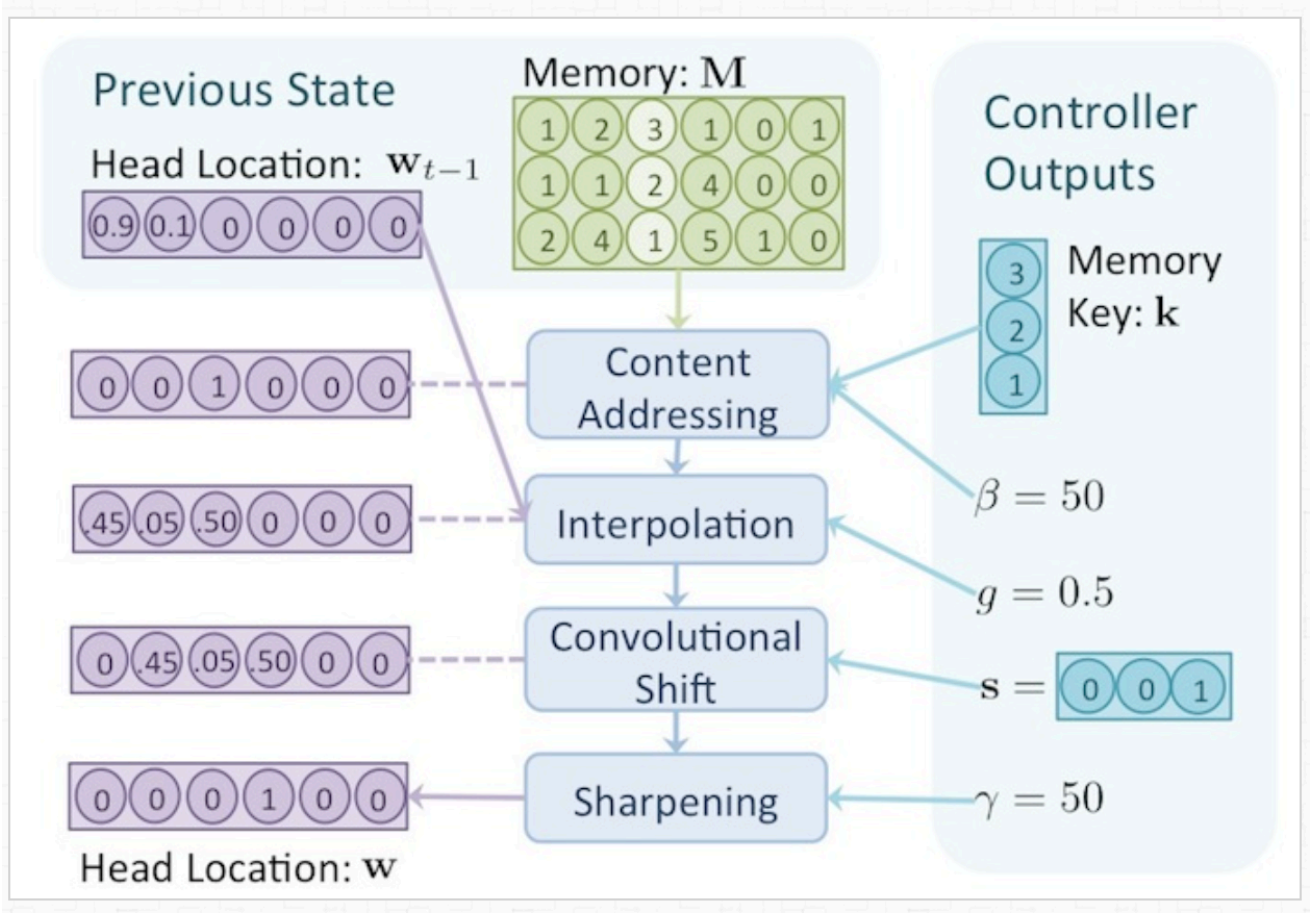
  $w^c_t(I) = \exp(\beta_t K(k_t, M_t(i))) / \sum_j \exp(\beta_t K(k_t, M_t(j)))$

- Focusing by location
- $w_t^g \leftarrow g_t w_t^c + (1 - g_t) w_{t-1}$.

- Convolution: $W_t(i) \leftarrow \sum_{j=0}^{R-1} w^g_t(j) s_t(i-j)$

- Shapening: $w_t(i) \leftarrow w'_t(i)^{\gamma t} / \sum_j w'_t(j)^{\gamma t}$

Addressing mechanism



Content address

$w^c_t(I) = \exp(\beta_t K(k_t, M_t(i))) / \sum_j \exp(\beta_t K(k_t, M_t(j)))$

$K(u,v) = u \bullet v / ||u|| \, ||v||$

K(k,M0)= $7/\sqrt{6}\sqrt{14}$ = 2~3

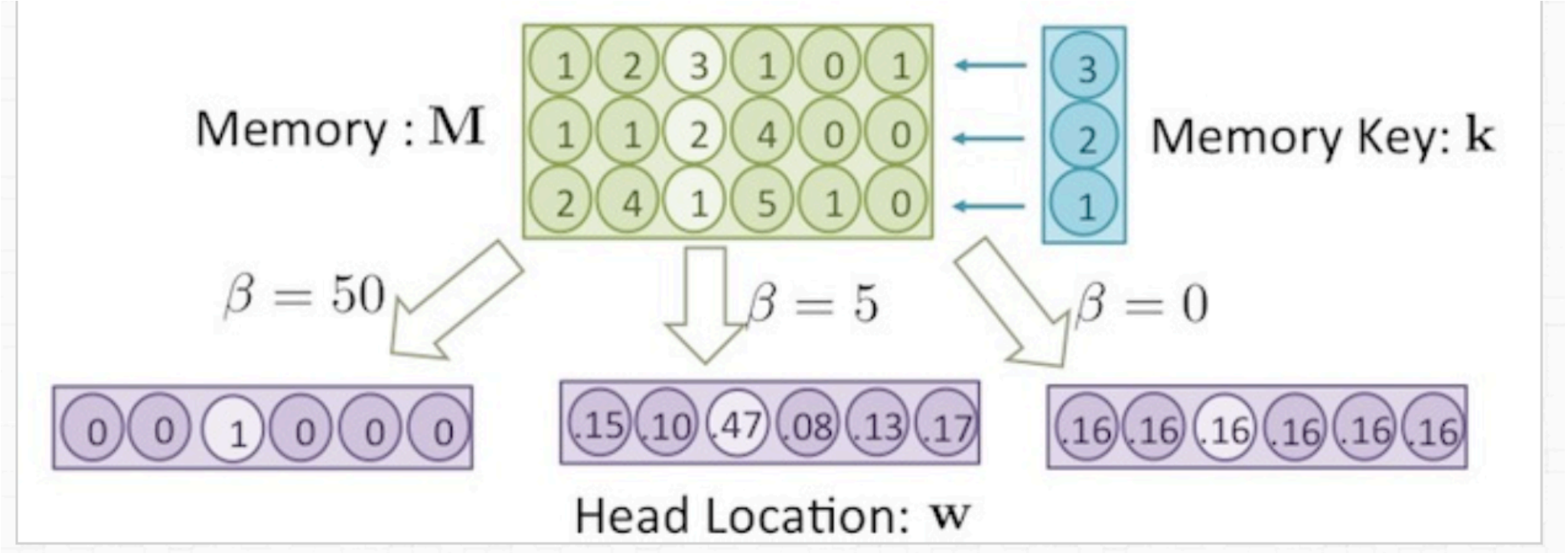
K(k,M1)=$12/\sqrt{21}\sqrt{14}$ =2~3

K(k,M2)=$14/\sqrt{12}\sqrt{14}$=3~4

K(k,M3)=$16/\sqrt{43}\sqrt{14}$=2~3?

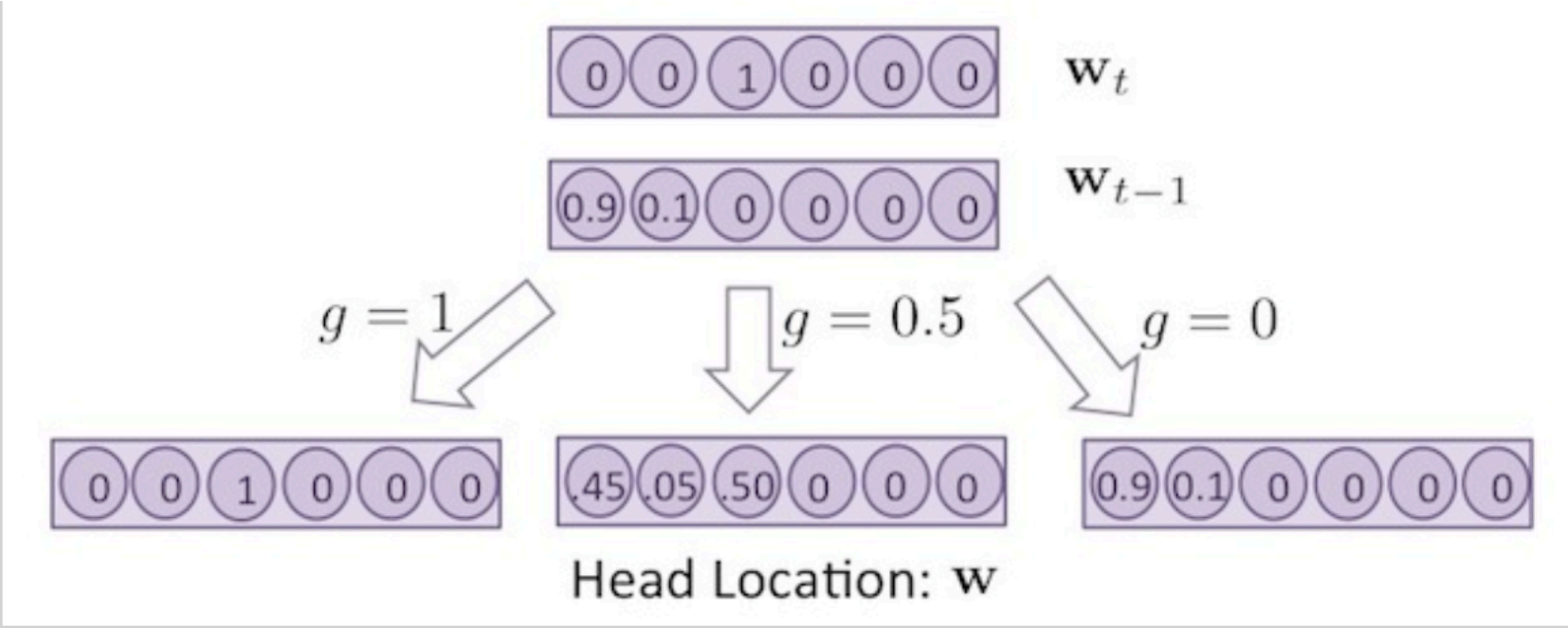
K(k,M4)=$1/1\sqrt{14}$= 1

K(k,M5)=$3/1\sqrt{14}$ = 3

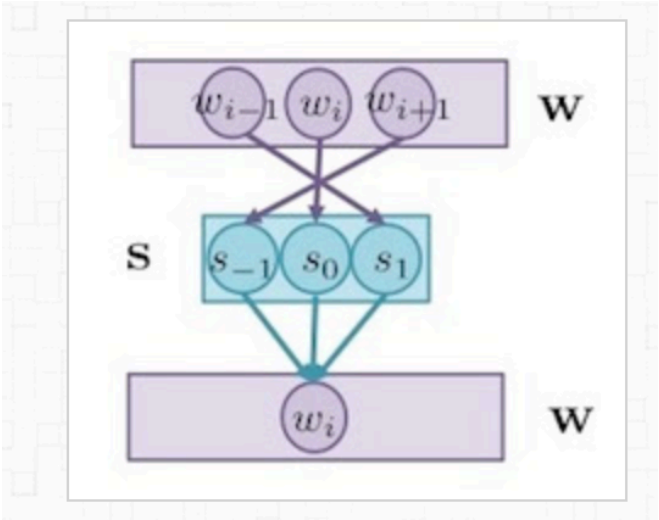


interpolation

$$w_t^g \leftarrow g_t w_t^c + (1 - g_t) w_{t-1}.$$



Convolution shift

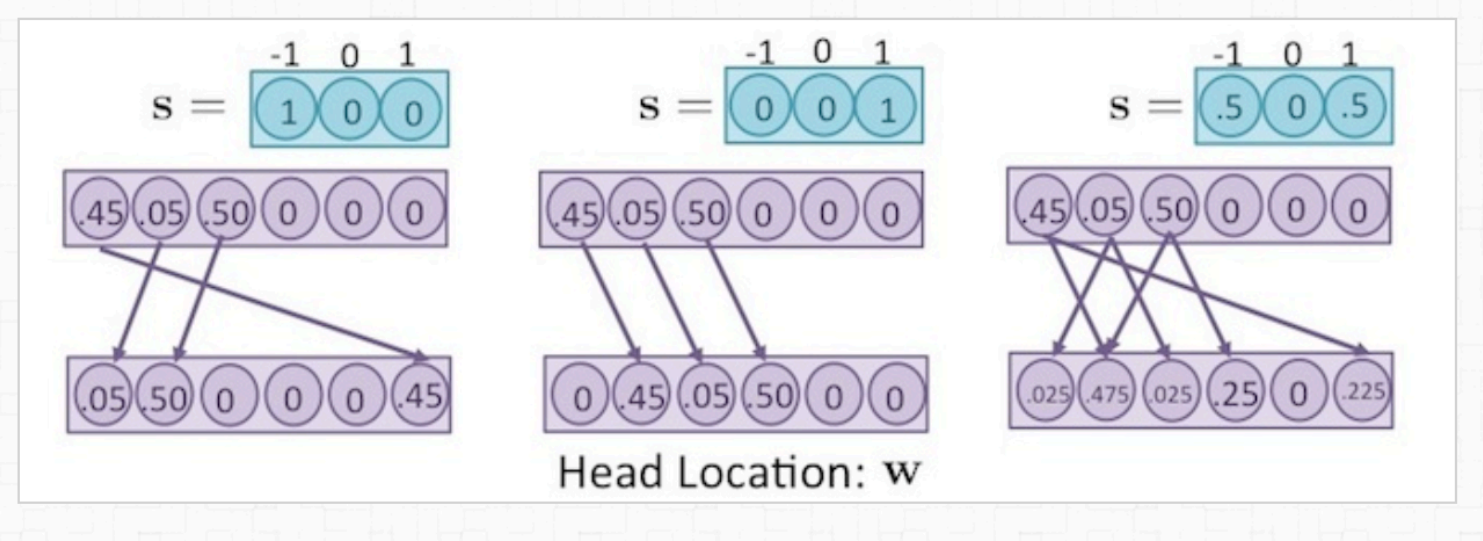


- w(i)←w(i+1)s(−1)+w(i)s(0)+w(i−1)s(1)
- W(-1) ← w(0)s(-1)+w(-1)s(0)+w(-2)s(1)

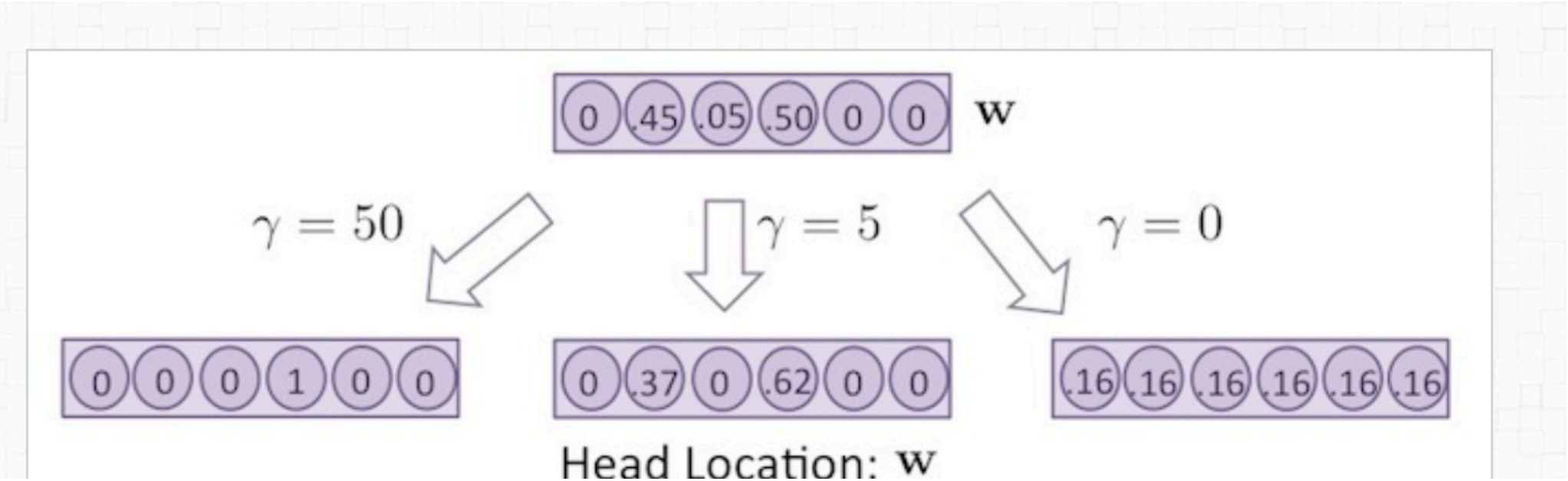
  = 0.45*1+ 0*0+0*0 = 0.45

W(0) ← w(1)s(-1)+ w(0)*s(0) + w(-1)s(1)

= 0.05 *1+ 0.45*0+ 0*0 = 0.05



Sharpening operation



Shapening: $w_t(i) \leftarrow w'_t(i)^{\gamma t} / \sum_j w'_t(j)^{\gamma t}$

Augmented NTM with LRU memory

- 寫的動作是依記憶體的位址最近是否被使用過超過某threshold (m(w_t^u,n))而定 。
- 最近最少被使用過(lu)的位址就有越高的機率(w_t^w)被寫入關鍵詞k_t。
- The write weights w_t^w, a learnable sigmoid gate parameter is used to compute a convex combination of the previous read weights and previous least-used weight

Read:

$$K(\mathbf{k}_t, \mathbf{M}_t(i)) = \frac{\mathbf{k}_t \cdot \mathbf{M}_t(i)}{\| \mathbf{k}_t \| \| \mathbf{M}_t(i) \|}$$

$$w_t^r(i) \leftarrow \frac{\exp\left(K\left(\mathbf{k}_t, \mathbf{M}_t(i)\right)\right)}{\sum_j \exp\left(K\left(\mathbf{k}_t, \mathbf{M}_t(j)\right)\right)}.$$

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

Where, $\mathbf{k}_t$: the controller produces a key, $\mathbf{M}_t(i)$: the memory block

Write:

$$\mathbf{w}_t^u \leftarrow \gamma \mathbf{w}_{t-1}^u + \mathbf{w}_t^r + \mathbf{w}_t^w$$

$$w_t^{lu}(i) = \begin{cases} 0 & \text{if } w_t^u(i) > m(\mathbf{w}_t^u, n) \\ 1 & \text{if } w_t^u(i) \leq m(\mathbf{w}_t^u, n) \end{cases}$$

$$\mathbf{w}_t^w \leftarrow \sigma(\alpha)\mathbf{w}_{t-1}^r + (1 - \sigma(\alpha))\mathbf{w}_{t-1}^{lu}$$

$$\mathbf{M}_t(i) \leftarrow \mathbf{M}_{t-1}(i) + w_t^w(i)\mathbf{k}_t, \forall i$$

Where, $\mathbf{w}_t^u$: usage weights, $w_t^{lu}(i)$: The least-used weights, $\mathbf{w}_t^w$ write weights

How NTM learn general programming

- Copy task:
- if a network that had been trained to copy sequences of length up to 20 could copy a sequence of length 100 with no further training.
- Evaluate if an NTM is able to bridge longer time delays than LSTM.
- Repeat copy task
- Associative memory recall
- N-gram
- Sort task

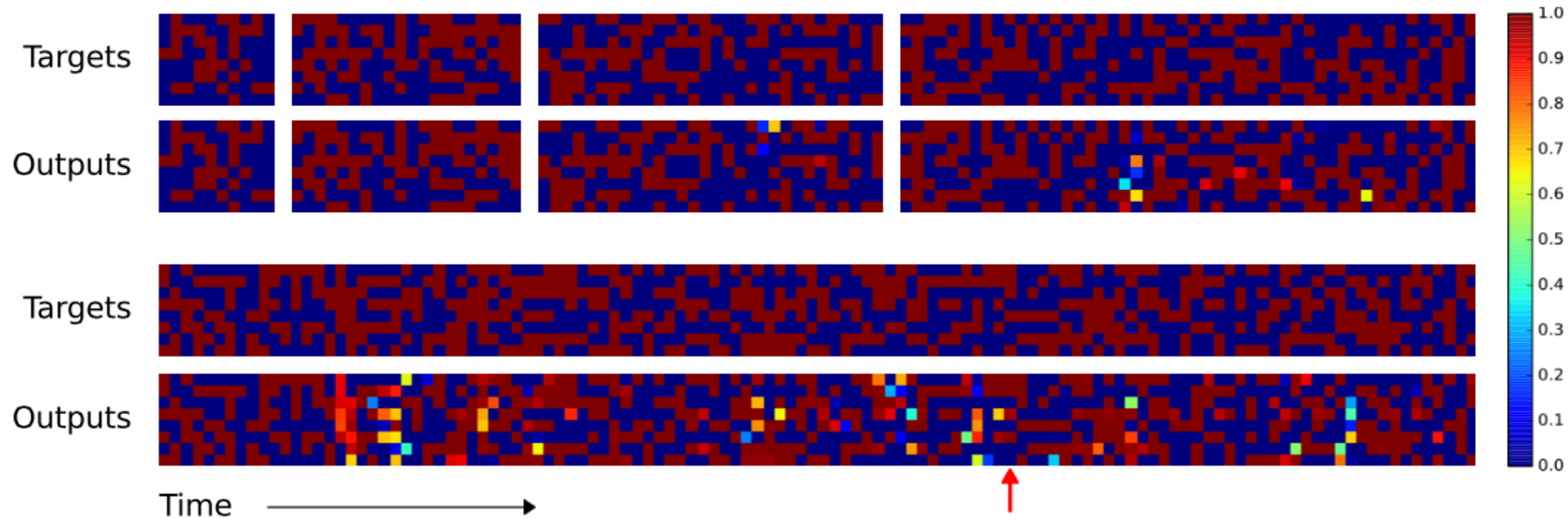


Figure 4: NTM Generalisation on the Copy Task. The four pairs of plots in the top row depict network outputs and corresponding copy targets for test sequences of length 10, 20, 30, and 50, respectively. The plots in the bottom row are for a length 120 sequence. The network was only trained on sequences of up to length 20. The first four sequences are reproduced with high confidence and very few mistakes. The longest one has a few more local errors and one global error: at the point indicated by the red arrow at the bottom, a single vector is duplicated, pushing all subsequent vectors one step back. Despite being subjectively close to a correct copy, this leads to a high loss.

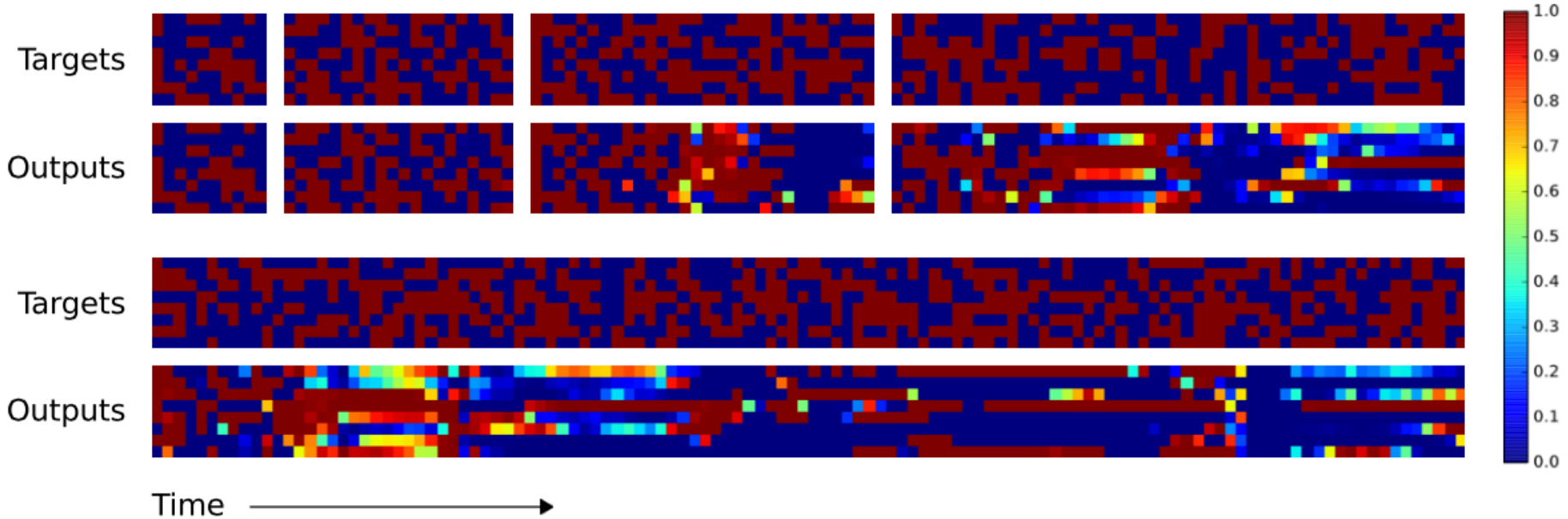


Figure 5: LSTM Generalisation on the Copy Task. The plots show inputs and outputs for the same sequence lengths as Figure 4. Like NTM, LSTM learns to reproduce sequences of up to length 20 almost perfectly. However it clearly fails to generalise to longer sequences. Also note that the length of the accurate prefix decreases as the sequence length increases, suggesting that the network has trouble retaining information for long periods.

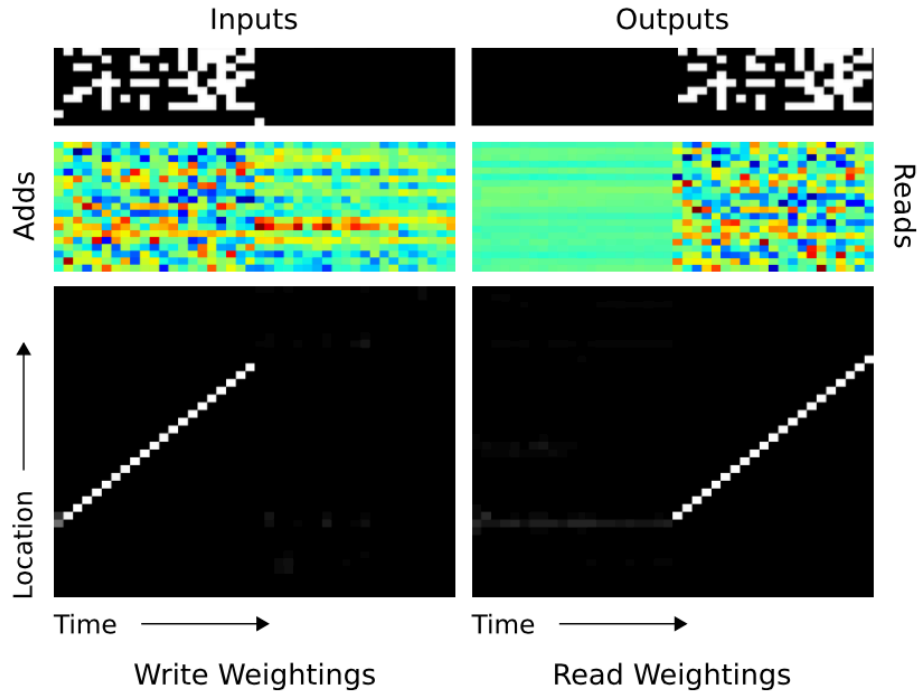


Figure 6: NTM Memory Use During the Copy Task. The plots in the left column depict the inputs to the network (top), the vectors added to memory (middle) and the corresponding write weightings (bottom) during a single test sequence for the copy task. The plots on the right show the outputs from the network (top), the vectors read from memory (middle) and the read weightings (bottom). Only a subset of memory locations are shown. Notice the sharp focus of all the weightings on a single location in memory (black is weight zero, white is weight one). Also note the translation of the focal point over time, reflects the network's use of iterative shifts for location-based addressing, as described in Section 3.3.2. Lastly, observe that the read locations exactly match the write locations, and the read vectors match the add vectors. This suggests that the network writes each input vector in turn to a specific memory location during the input phase, then reads from the same location sequence during the output phase.

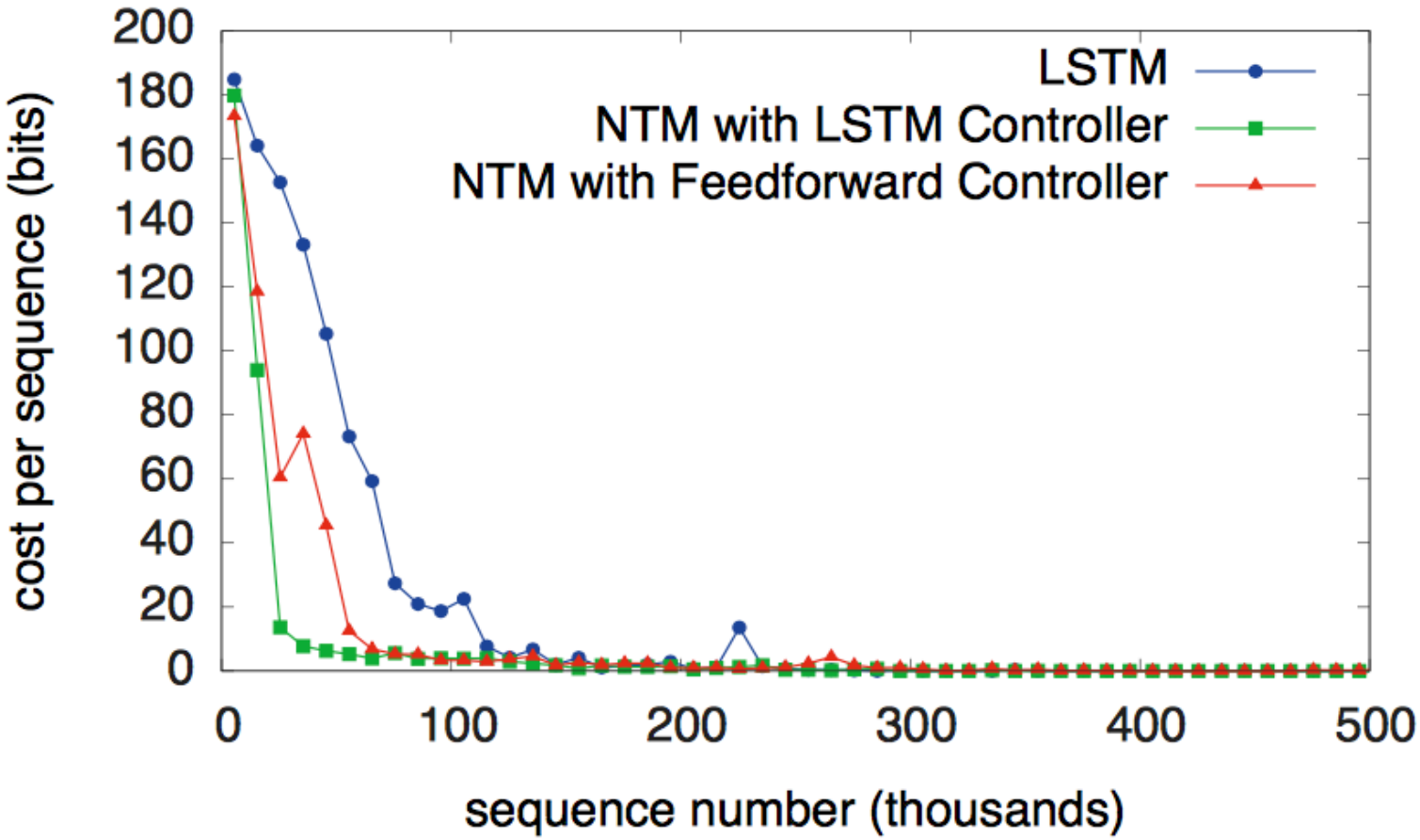


Figure 7: Repeat Copy Learning Curves.

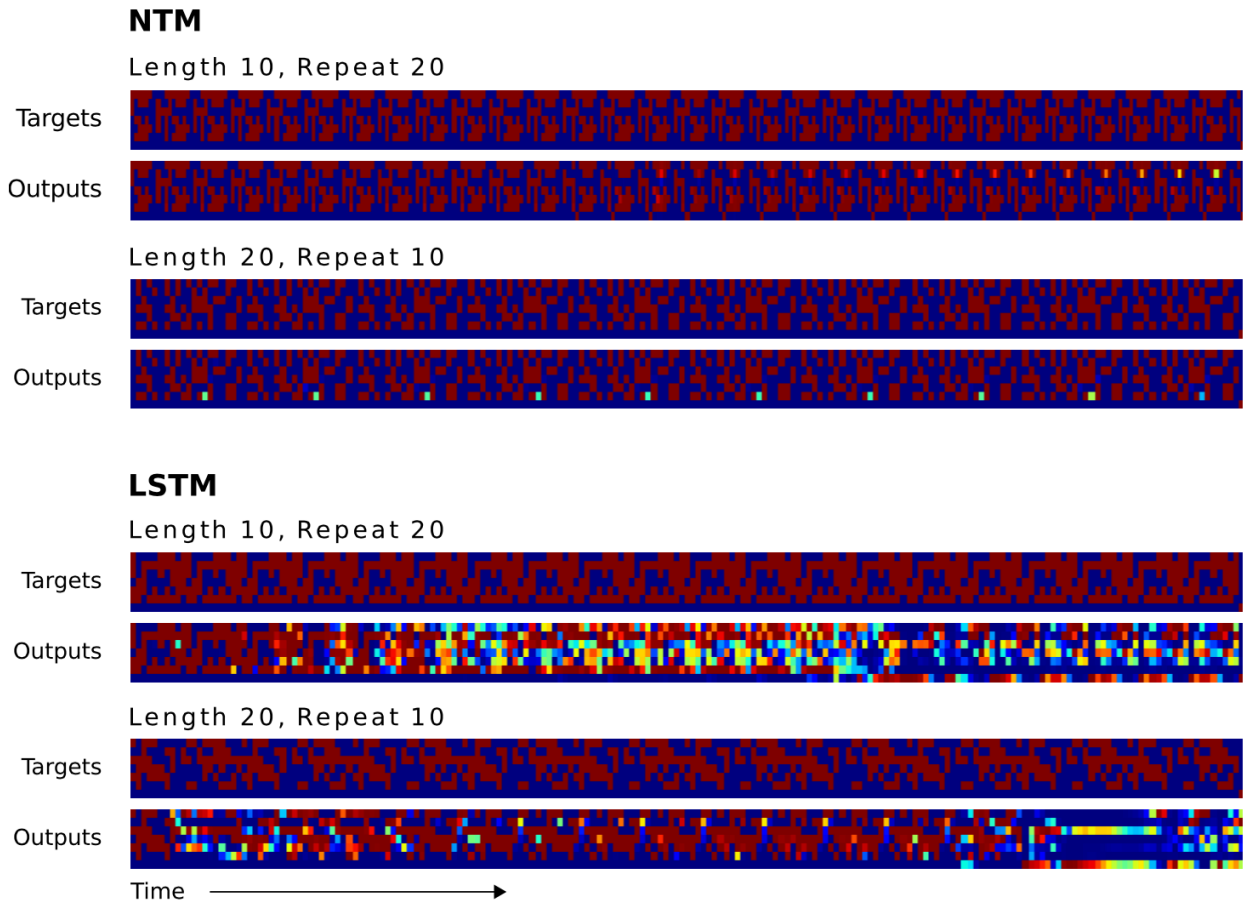


Figure 8: NTM and LSTM Generalisation for the Repeat Copy Task. NTM generalises almost perfectly to longer sequences than seen during training. When the number of repeats is increased it is able to continue duplicating the input sequence fairly accurately; but it is unable to predict when the sequence will end, emitting the end marker after the end of every repetition beyond the eleventh. LSTM struggles with both increased length and number, rapidly diverging from the input sequence in both cases.

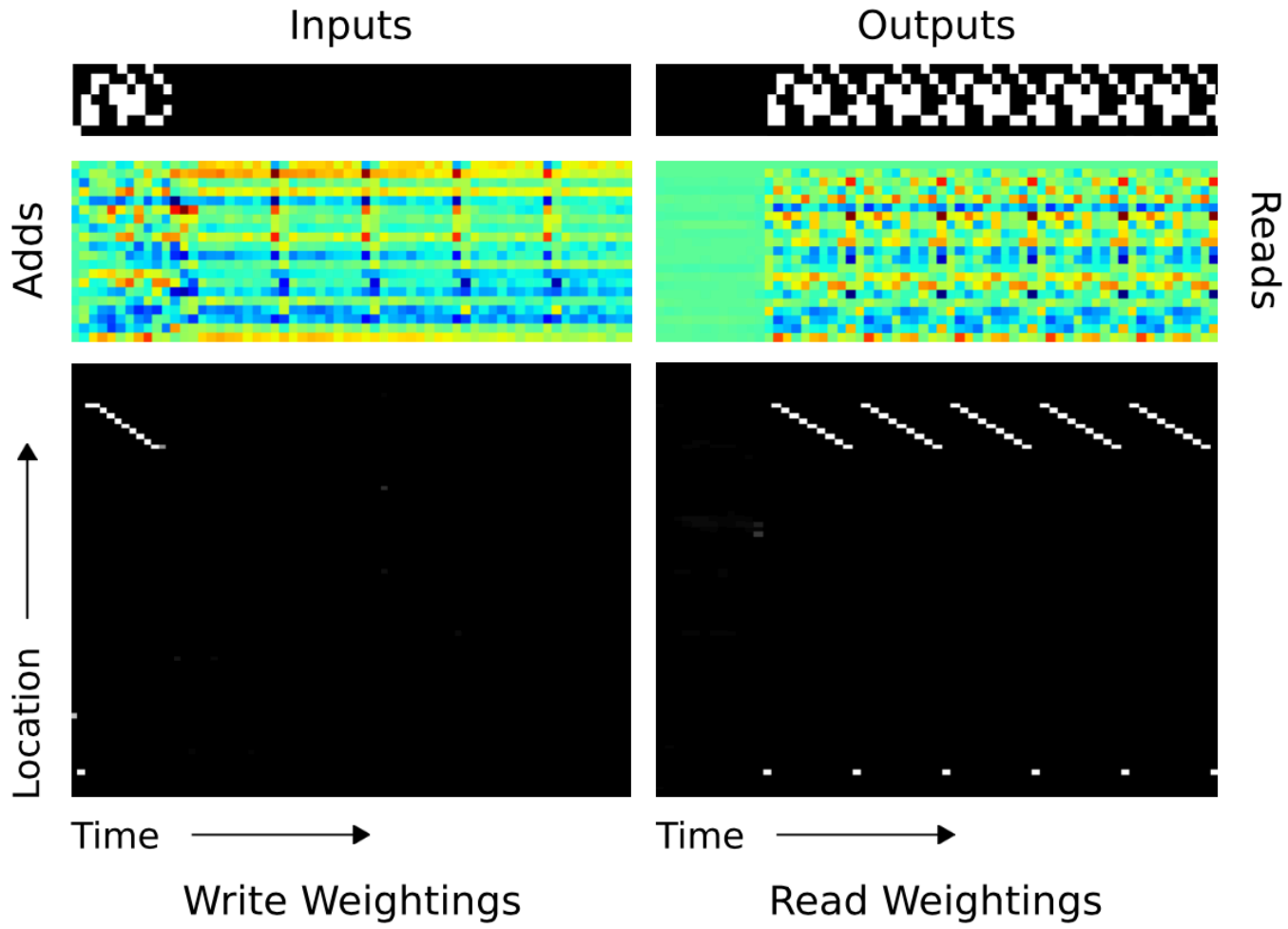


Figure 9: NTM Memory Use During the Repeat Copy Task. As with the copy task the network first writes the input vectors to memory using iterative shifts. It then reads through the sequence to replicate the input as many times as necessary (six in this case). The white dot at the bottom of the read weightings seems to correspond to an intermediate location used to redirect the head to the start of the sequence (The NTM equivalent of a *goto* statement).

Associative memory recall

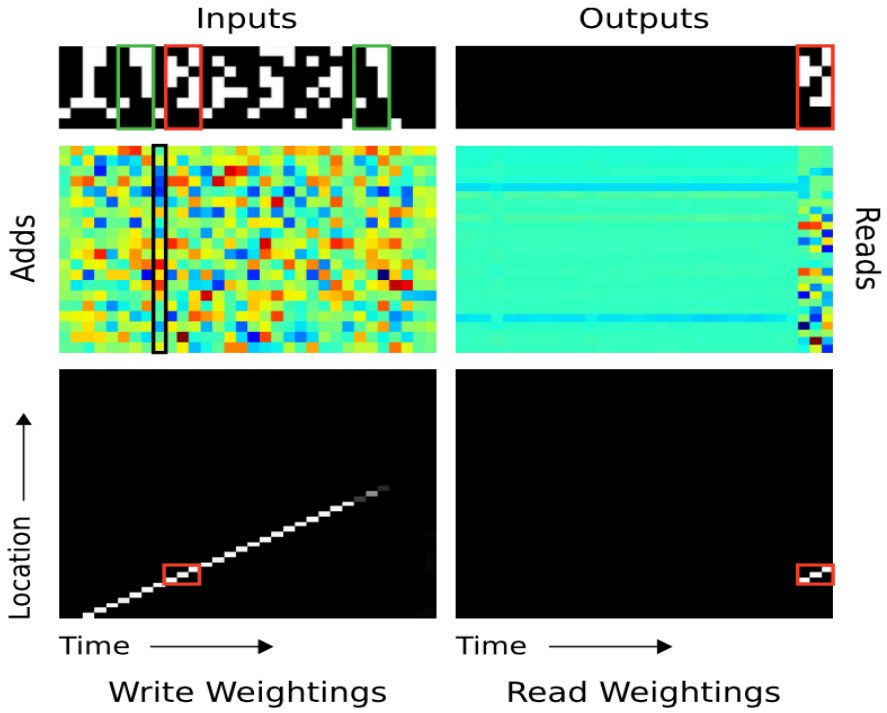


Figure 12: NTM Memory Use During the Associative Recall Task. In "Inputs," a sequence of items, each composed of three consecutive binary random vectors is propagated to the controller. The distinction between items is designated by delimiter symbols (row 7 in "Inputs"). After several items have been presented, a delimiter that designates a query is presented (row 8 in "Inputs"). A single query item is presented (green box), and the network target corresponds to the subsequent item in the sequence (red box). In "Outputs," we see that the network correctly produces the target item. The red boxes in the read and write weightings highlight the three locations where the target item was written and then read. The solution the network finds is to form a compressed representation (black box in "Adds") of each item that it can store in a single location. For further analysis, see the main text.

Dynamic N-Grams

- Test whether NTM could rapidly adapt to new predictive distributions .

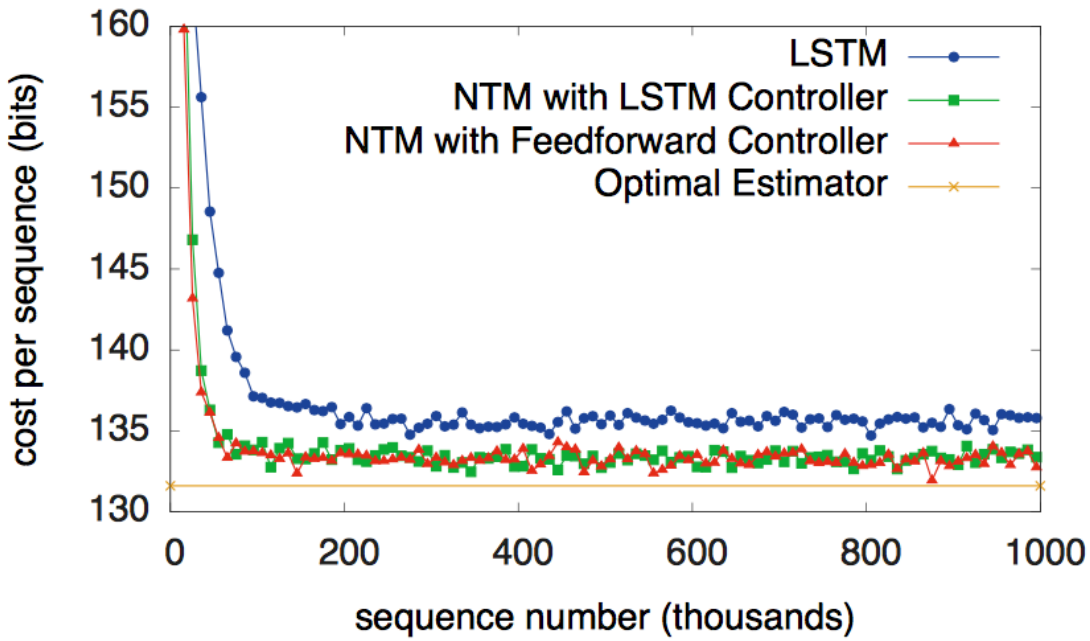


Figure 13: Dynamic N-Gram Learning Curves.

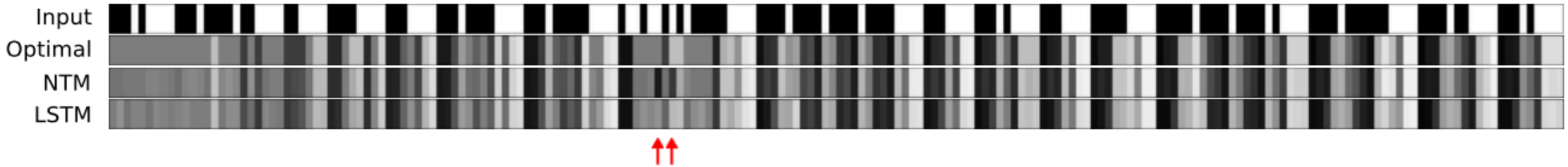


Figure 14: Dynamic N-Gram Inference. The top row shows a test sequence from the N-Gram task, and the rows below show the corresponding predictive distributions emitted by the optimal estimator, NTM, and LSTM. In most places the NTM predictions are almost indistinguishable from the optimal ones. However at the points indicated by the two arrows it makes clear mistakes, one of which is explained in Figure 15. LSTM follows the optimal predictions closely in some places but appears to diverge further as the sequence progresses; we speculate that this is due to LSTM "forgetting" the observations at the start of the sequence.

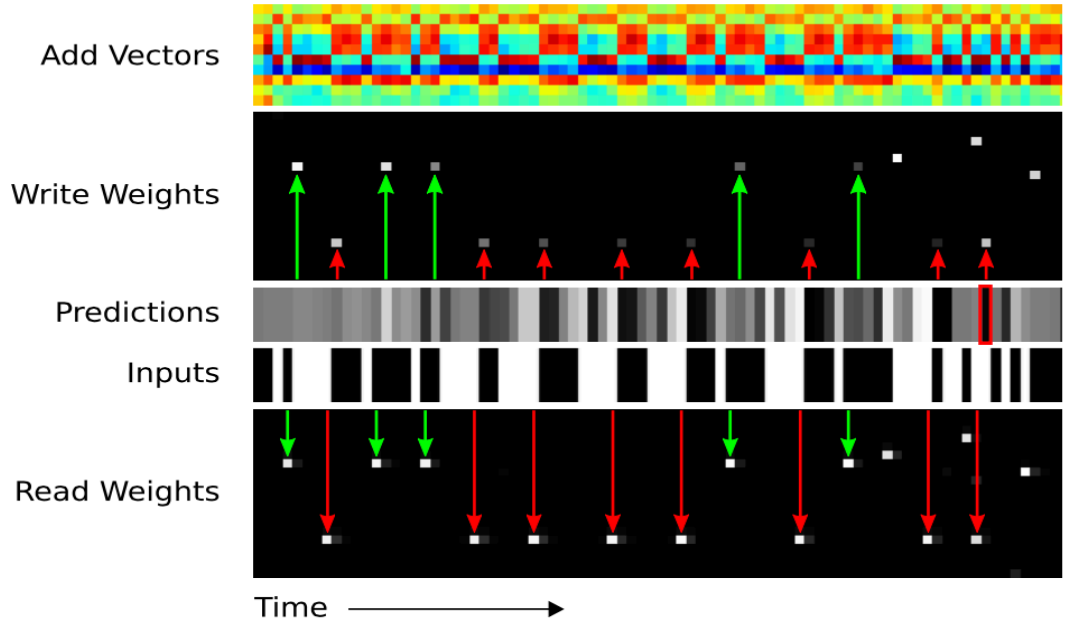


Figure 15: NTM Memory Use During the Dynamic N-Gram Task. The red and green arrows indicate point where the same context is repeatedly observed during the test sequence ("00010" for the green arrows, "01111" for the red arrows). At each such point the same location is accessed by the read head, and then, on the next time-step, accessed by the write head. We postulate that the network uses the writes to keep count of the fraction of ones and zeros following each context in the sequence so far. This is supported by the add vectors, which are clearly anti-correlated at places where the input is one or zero, suggesting a distributed "counter." Note that the write weightings grow fainter as the same context is repeatedly seen; this may be because the memory records a ratio of ones to zeros, rather than absolute counts. The red box in the prediction sequence corresponds to the mistake at the first red arrow in Figure 14; the controller appears to have accessed the wrong memory location, as the previous context was "01101" and not "01111."

Priority Sort

- This task tests whether the NTM can sort data—an important elementary algorithm.
- A sequence of random binary vectors is input to the network along with a scalar priority rating for each vector. The priority is drawn uniformly from the range [-1, 1].

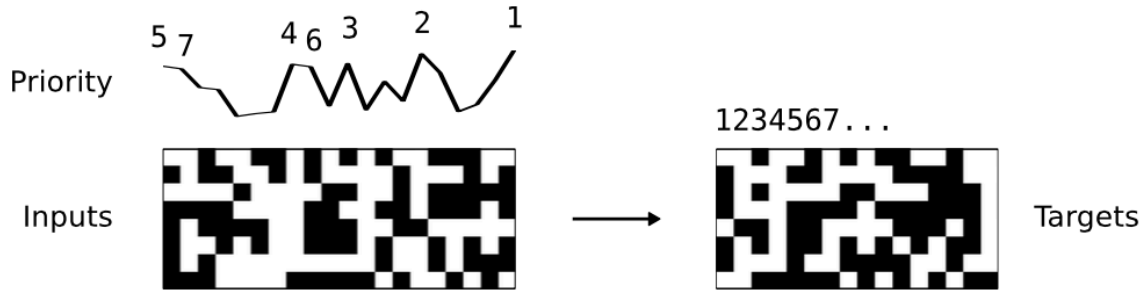


Figure 16: Example Input and Target Sequence for the Priority Sort Task. The input sequence contains random binary vectors and random scalar priorities. The target sequence is a subset of the input vectors sorted by the priorities.

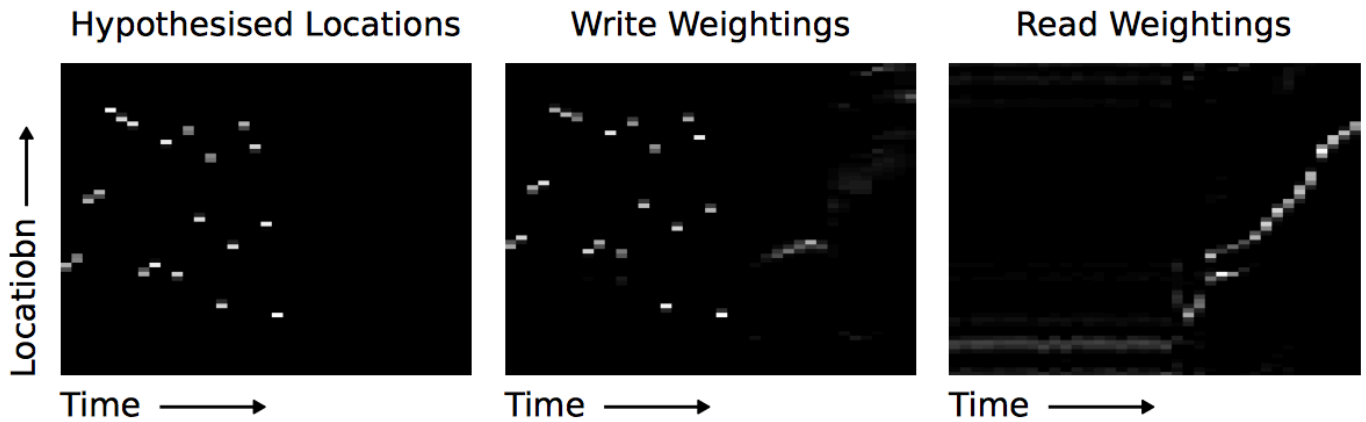


Figure 17: NTM Memory Use During the Priority Sort Task. Left: Write locations returned by fitting a linear function of the priorities to the observed write locations. Middle: Observed write locations. Right: Read locations.

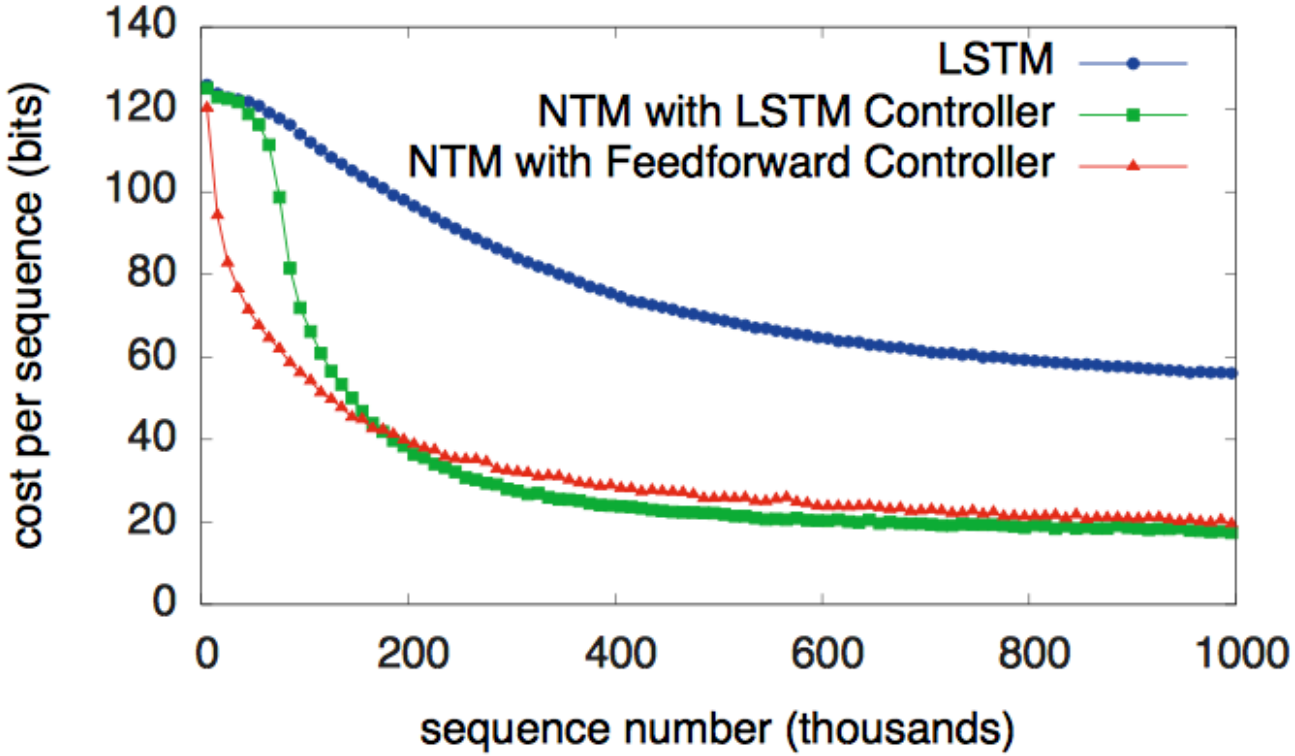


Figure 18: Priority Sort Learning Curves.

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