Dynamic Memory Networks & Neural Turing Machines

By Shehjar Sadhu and Mikel Gjergji

Dynamic Memory Networks

For Natural Language Processing

Kumar 2016

Salesforce Inc., CA USA *Authors were interns at MetaMind.

Applications:

Used heavily in question answering.

Question : Where is the football?
Mary got the football there.

Mary got the football there.

John moved to the bedroom.

Sandra went back to the kitchen.

Mary travelled to the hallway.

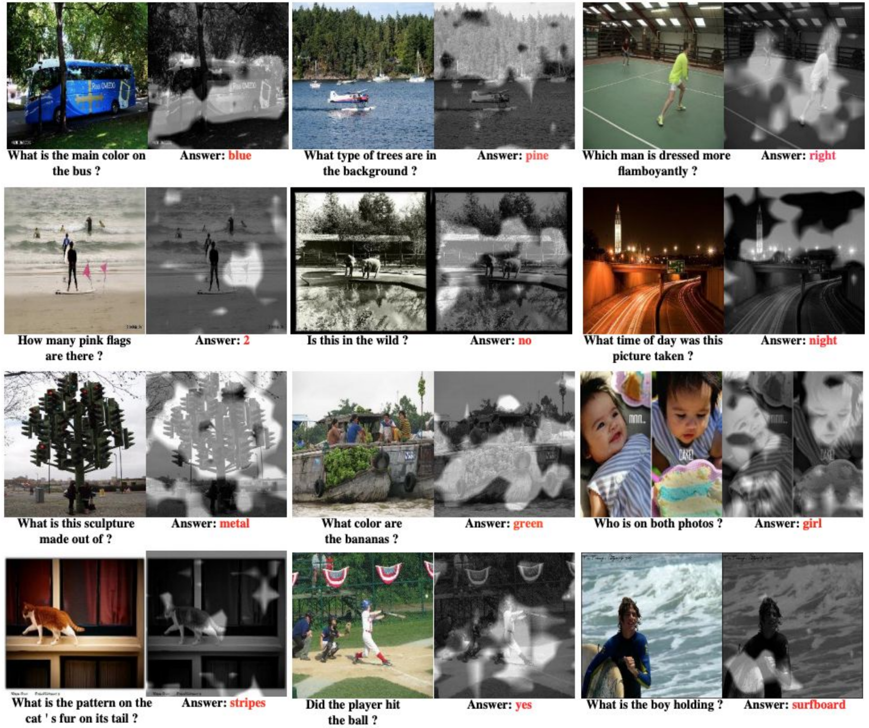
John got the football there.

John went to the hallway.

John put down the football

Mary went to the garden.

Visual question answering (Caiming Xiong*, Stephen Merity*, Richard Soche)



What is the main color on the bus ?
Answer: blue
What type of trees are in the background ?
Answer: pine
Which man is dressed more flamboyantly ?
Answer: right
How many pink flags are there ?
Answer: 2
Is this in the wild ?
Answer: no
What time of day was this picture taken ?
Answer: night
What is this sculpture made out of ?
Answer: metal
What color are the bananas ?
Answer: green
Who is on both photos ?
Answer: girl
What is the pattern on the cat ' s fur on its tail ?
Answer: stripes
Did the player hit the ball ?
Answer: yes
What is the boy holding ?
Answer: surfboard

Figure 6. Examples of qualitative results of attention for VQA. The original images are shown on the left. On the right we show how the attention gate g_i^t activates given one pass over the image and query. White regions are the most active. Answers are given by the DMN+.

Example of a Q & A problem:

Mary got the football there.

John moved to the bedroom.

Sandra went back to the kitchen.

Mary travelled to the hallway.

John got the football there.

John went to the hallway.

John put down the football

Mary went to the garden.

Question : Where is the football?

Question : Where is the football?

Mary got the football there.

John moved to the bedroom.

Sandra went back to the kitchen.

Mary travelled to the hallway.

John got the football there.

John went to the hallway.

John put down the football

Mary went to the garden.

! Intuitively human brain would first pay **attention** to this line since the football was mentioned here.

But how does this actually work ?

Discuss more about the mathematical model next.

Question : Where is the football?

Mary got the football there.	**Episode 1 (e1)**
John moved to the bedroom.	**Episode 2 (e2)**
Sandra went back to the kitchen.	**Episode 3 (e3)**
Mary travelled to the hallway.	**Episode 4 (e4)**
John got the football there.	**Episode 5 (e5)**
John went to the hallway.	**Episode 6 (e6)**
John put down the football	**Episode 7 (e7)**
Mary went to the garden.	**Episode 8 (e8)**

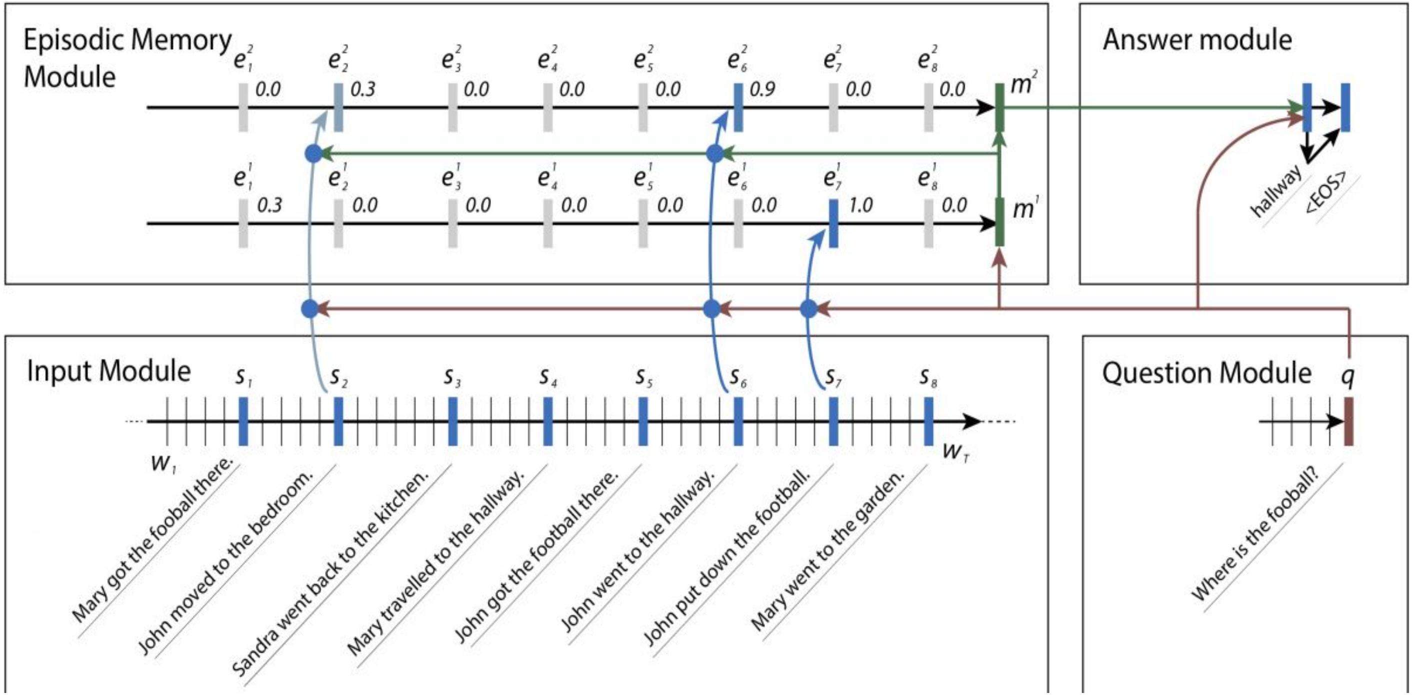
Brief overview of the model

1 . Question module

2. Input module.

3. Episodic Memory.

4. Answer Module



Input Module

First step is to generate a vector representation.

Encoding a sequence via recurrent neural networks RNN.

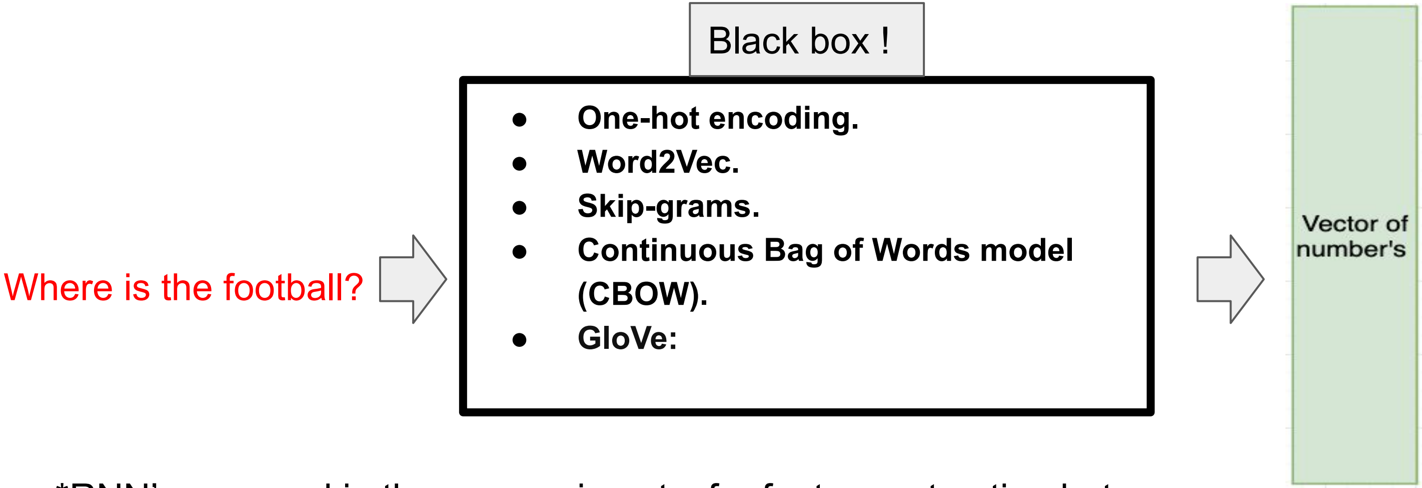
$$\boxed{h_t = RNN(L[w_t], h_{t-1})}$$

Input to RNN: Word embedding matrix(They find meaning) , Here L => Embedding matrix , W(t) => Word index of the t th word of the input sequence.

- In cases where the input sequence is a single sentence, the input module outputs the hidden states of the recurrent network. In cases where the input sequence is a list of sentences, we concatenate the sentences into a long list of word tokens, inserting after each sentence an end-of-sentence token.
- The hidden states at each of the end-of-sentence tokens are then the final representations of the input module.

Question module

- The question module consists of a vector representation of the question.
- For example :- In our example the question is Where is the football?
- **Output**:- Final hidden state of the recurrent network encoder: q = qTQ .



Where is the football?

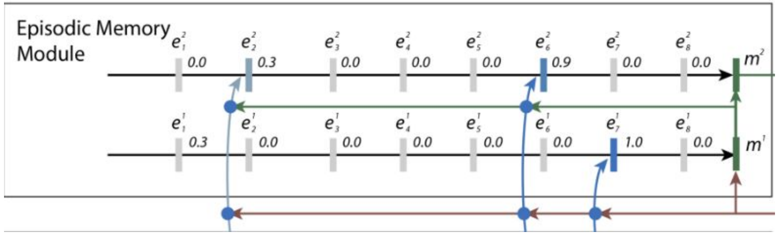
Black box !

- One-hot encoding.
- Word2Vec.
- Skip-grams.
- Continuous Bag of Words model (CBOW).
- GloVe:

Vector of number's

*RNN’s are used in these experiments for feature extraction but we can also use other techniques.

Episodic Memory module



<u>Outline</u>:

1. Internal memory.
2. Attention Mechanism.
3. Recurrent Neural Network (To update it's memory).

This is the most important part of the DMN model. This is what differentiates it with other memory networks.

each iteration :

a. The **attention mechanism** attents over a **fact representation(c)**. (This is done by using a gating function)
b. Also takes into consideration the the **question representation(q)** and **previous memory m(i-1)** to produce and **episode e(i)** in parallel.

Memory update mechanism:

- Uses **GRU model** to update it’s memory and an **attention mechanism**.

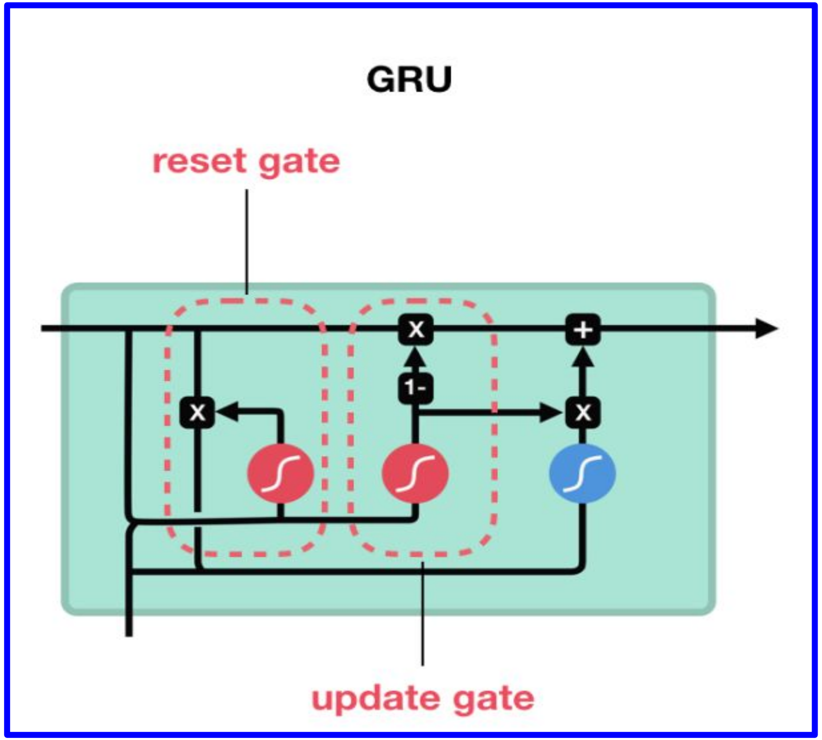
To compute this line: John put down the football. (e7).

$$\boxed{h_t^i = g_t^i GRU(c_t, h_{t-1}^i) + (1 - g_t^i) h_{t-1}^i}$$

$$\boxed{e^i = h_{T_C}^i}$$

- This episode vector is given as input to the **answer module.**

Gated Recurrent Units (GRU) formulation:-



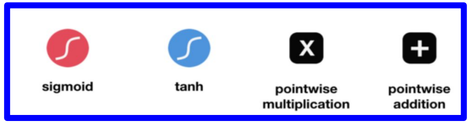


Image credit - Medium post by Michael Nguyen

Update Gate:-

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1} + b^{(z)}\right)$$

Reset gate:-

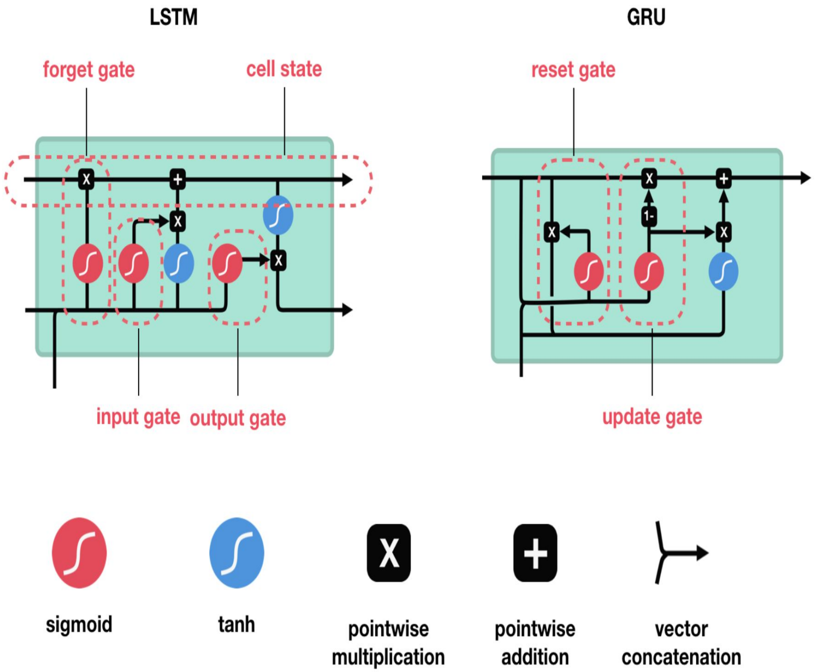
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1} + b^{(r)}\right)$$

Hidden state for the next hidden state:-

$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1} + b^{(h)}\right)$$

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

LSTM’s v/s GRU’s



- LSTM’s have cell state, forget gate, input gate,utput gate.
- GRU’s have reset and update gates.
- GRU’s are claimed to have fewer tensor operations than LSTM’s because they have fewer parameters.
- There is no clear winner GRU’s and LSTM’s are both used and compared against each other for a particular research problem.
- In this example we will use GRU’s.

Image Credit: Image credit - Medium post by Michael Nguyen.

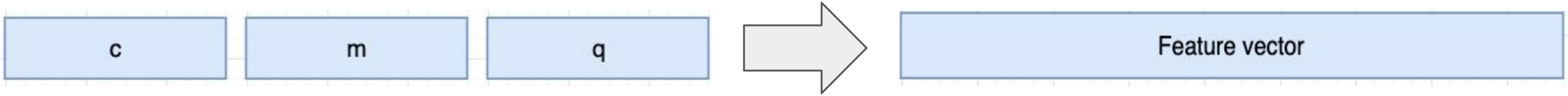
Attention Mechanism :

$$\boxed{g_t^i = G(c_t, m^{i-1}, q).}$$

This equation produces weights. These weights are then used by the model to figure out which episode to pay attention to.

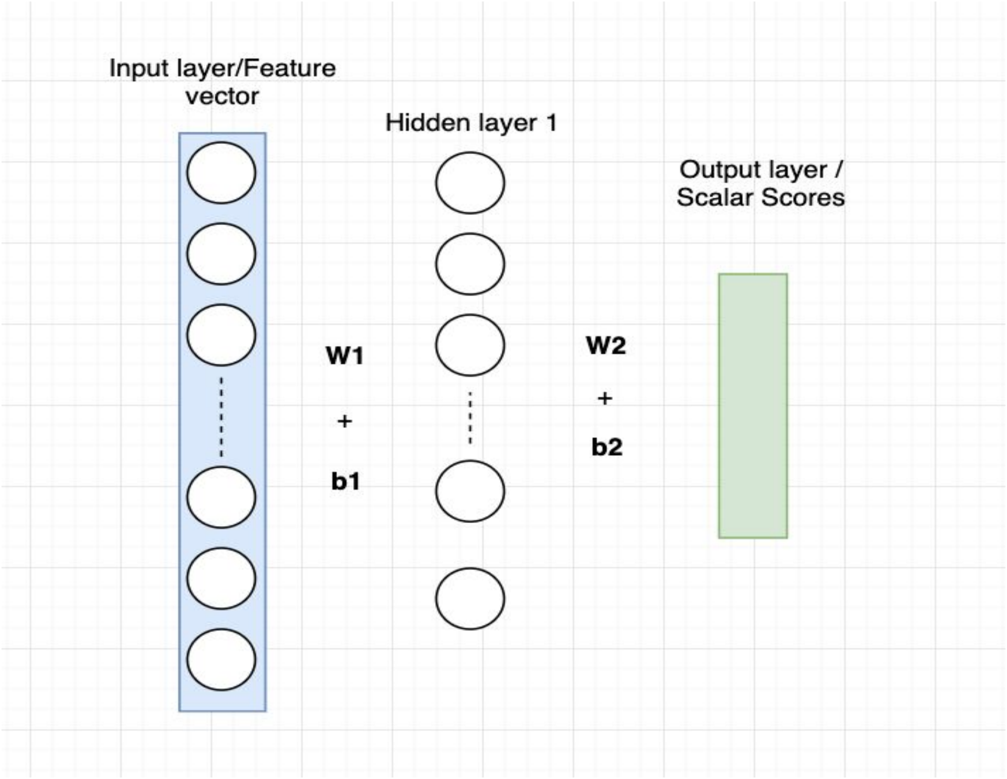
- G is a scoring function here this will give us a scalar score.
- In order to calculate this we need to first calculate a feature vector. As the scoring function G will take this feature vector as input.

z(c,m,q) = $\left[c, m, q, c \circ q, c \circ m, |c - q|, |c - m|, c^T W^{(b)} q, c^T W^{(b)} m\right]$



Generates a large feature vector

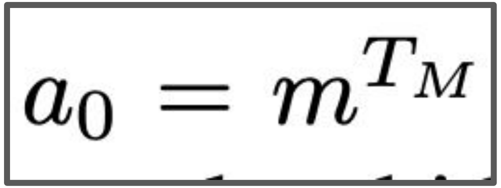
G(c, m, q) = $\boxed{\sigma\left(W^{(2)}\tanh\left(W^{(1)}z(c,m,q)+b^{(1)}\right)+b^{(2)}\right)}$



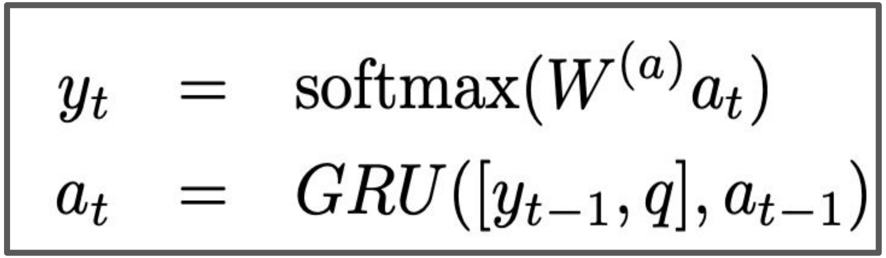
It’s a simple 2 layer feed forward neural network.

- The feature vector computed above is fed as inputs through the network.
- Then a scalar score is calculated as output of that network.
- This network comprises of a tanh and a sigmoid activation function.
- Above is an equation for of this network.

Answer module:



$$a_0 = m^{T_M}$$



$$\begin{aligned} y_t &= \mathrm{softmax}(W^{(a)} a_t) \\ a_t &= GRU([y_{t-1}, q], a_{t-1}) \end{aligned}$$

- They employ another GRU model.
- **Initial state:** Initialized to the last memory
- Initial state of this model is initialized to the last memory.
- At each state it takes the input question (q), a last hidden state a(n-1) as we as previously predicted output.
- Triggered either once or at each time step.
- **Input** :- we concatenate the last generated word and the question vector as the input at each time step.
- **Output** :- trained with the cross-entropy error classification of the correct sequence appended with a special end-of-sequence token.

Memory networks V/s Dynamic Memory networks.

Memory networks:-

(Weston, Chopra, Bordes 2015)

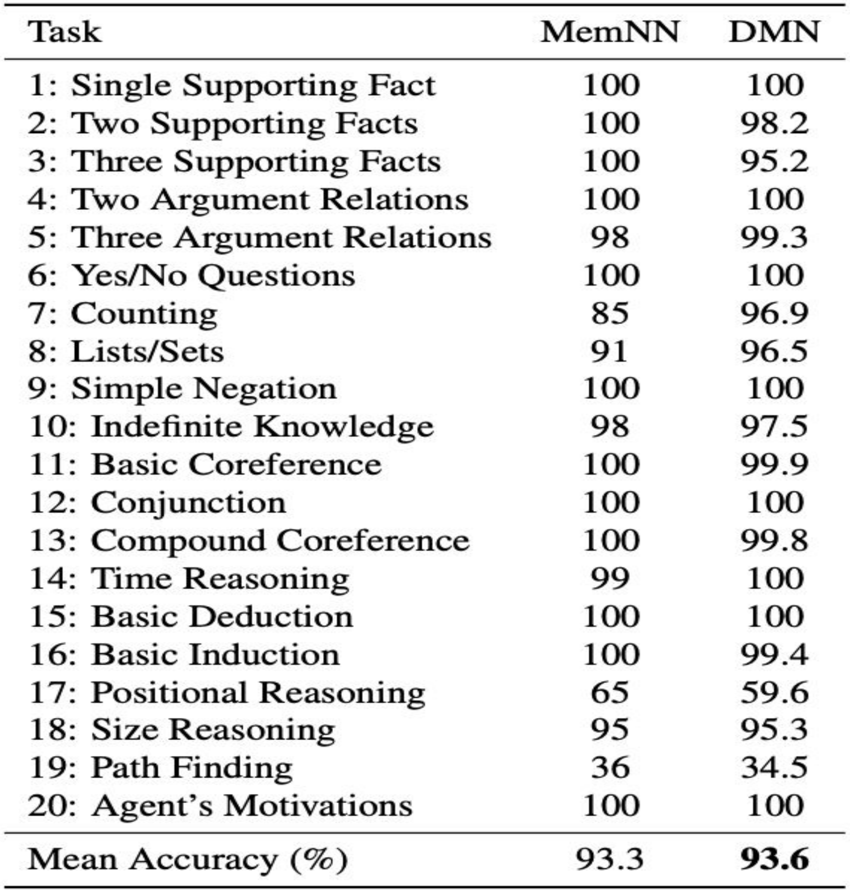
- Memory networks were introduced by facebook AI research lab.
- These networks consists of four (potentially learned) components I((input feature map), G((generalization), O((output feature map) and R(response).

Dynamic Memory networks:-

(Kumar 2016)

- Episodic memory differentiates DMN’s against a regular memory network.
- As it allows for automatic feature extraction.
- Attention and Memory are two important aspects of DMN’s and they help in image captioning, machine translation.
- **Neural Turing Machines** use memory to solve algorithms like list sorting. (Graves 2014)

Results and analysis:-

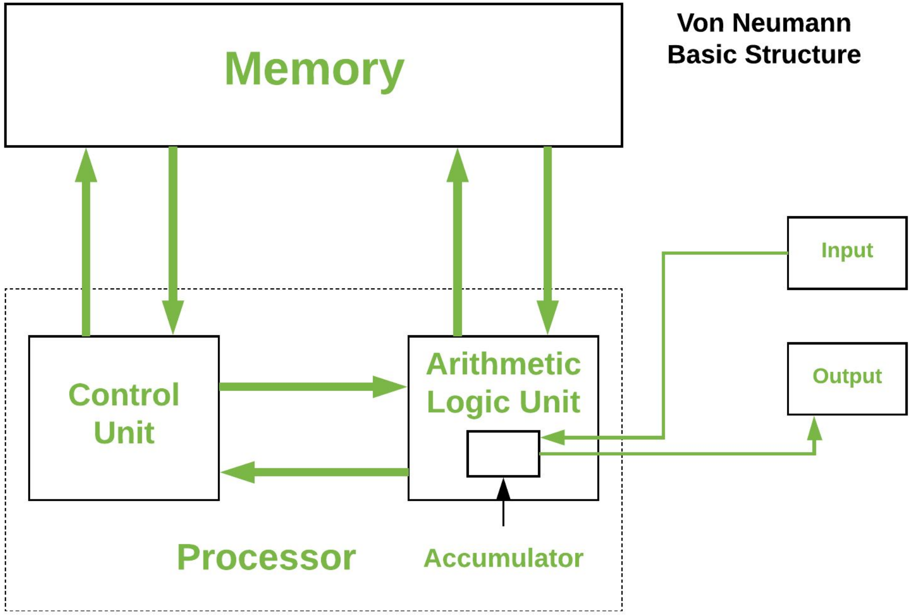


Task	MemNN	DMN
1: Single Supporting Fact	100	100
2: Two Supporting Facts	100	98.2
3: Three Supporting Facts	100	95.2
4: Two Argument Relations	100	100
5: Three Argument Relations	98	99.3
6: Yes/No Questions	100	100
7: Counting	85	96.9
8: Lists/Sets	91	96.5
9: Simple Negation	100	100
10: Indefinite Knowledge	98	97.5
11: Basic Coreference	100	99.9
12: Conjunction	100	100
13: Compound Coreference	100	99.8
14: Time Reasoning	99	100
15: Basic Deduction	100	100
16: Basic Induction	100	99.4
17: Positional Reasoning	65	59.6
18: Size Reasoning	95	95.3
19: Path Finding	36	34.5
20: Agent's Motivations	100	100
Mean Accuracy (%)	93.3	**93.6**

- These were compared to scores against memory networks.
- Memory networks out performed the Dynamic memory network on tasks 2 and 3. The paper suggests it's due to the fact that tasks 2 and 3 have long input sequence and the networks is having trouble modeling really long input sequence.
- Mean Accuracy for all tasks is higher tough.

Neural Turing Machines

Von Neumann Architecture

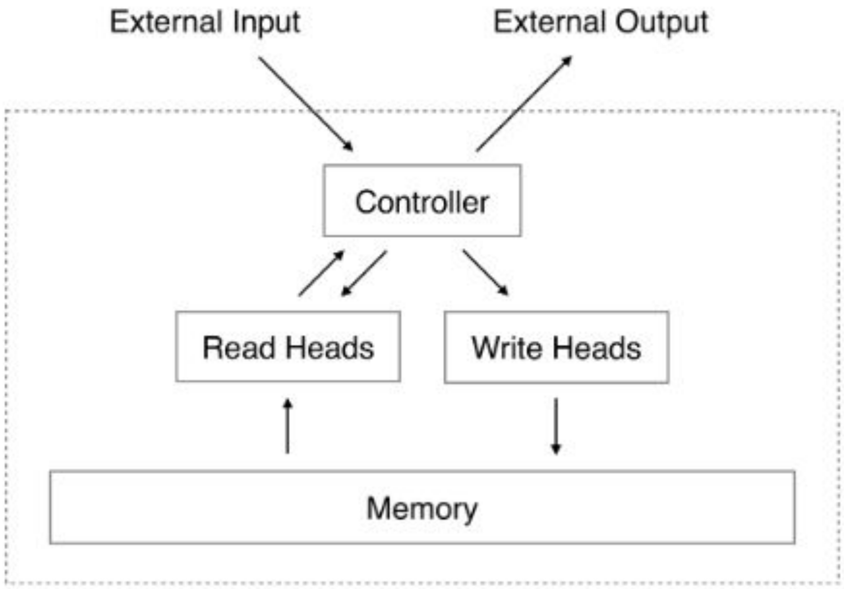


The basic structure of a computer involves a central processing unit and some memory. This takes in inputs and outputs results.

https://www.geeksforgeeks.org/computer-organization-von-neumann-architecture/

Neural Turing Machine

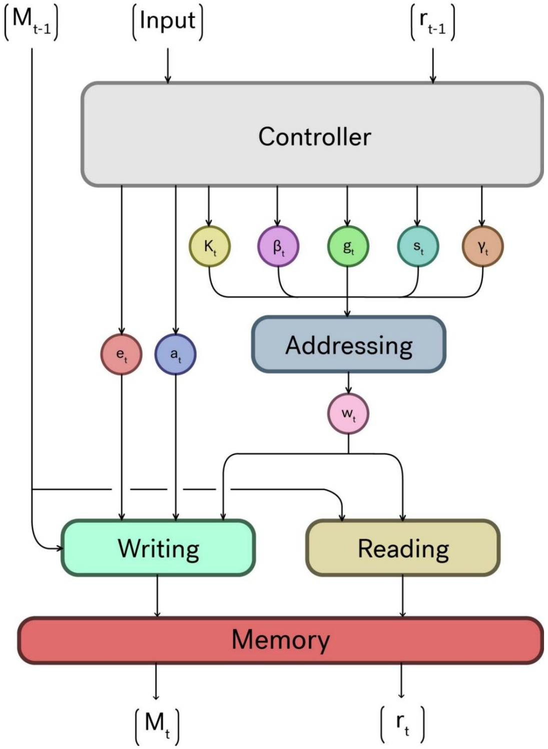
The neural turing machine aims to replicate the Von Neumann model, replacing the cpu with a neural net. This creates a differentiable computer which in theory should be able to learn algorithms through gradient descent.



Origins

Neural Turing machines was originally published in 2014 by Alex Graves et al. at Deepmind.

Overview of a neural turing machine



https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

The Memory Matrix

We define a memory Matrix M_t to be an NxM matrix at a given time t.

N is the number of memory locations.

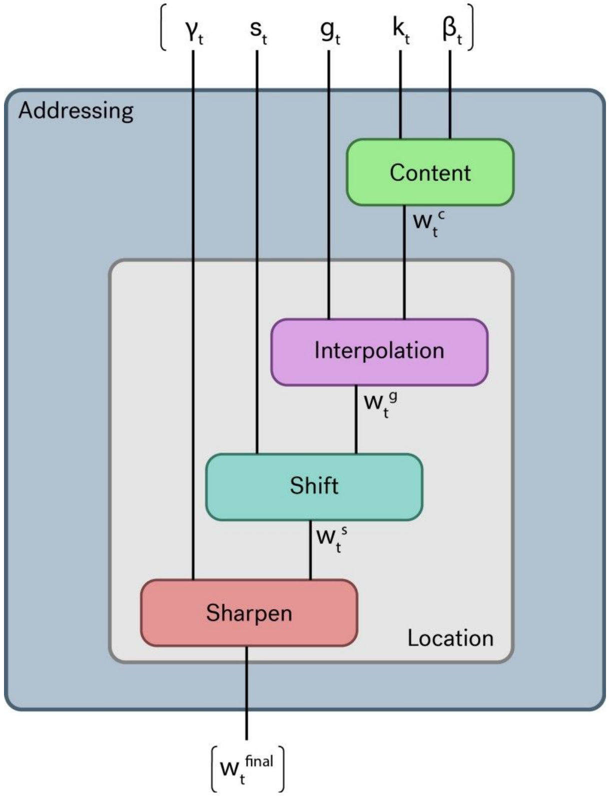
M is the size of the vector at each location.

W_t is a vector of weightings over N locations at a given time t.

Below is an example of a 2x3 memory matrix

.05	132.32	.6542
.1391	32	1.2

Addressing by Content



As we have learned prior in class, a dot product produces a similarity metric between two vectors, this can be used to find a memory vector which is most similar to a given “key” vector k. This is how we begin our reading of memory and output a weight vector $w^c{}_t$. A parameter β is used to focus the weight vector

$$w_t^c(i) = softmax(\beta_t \delta(K_t, M_t(i)))$$

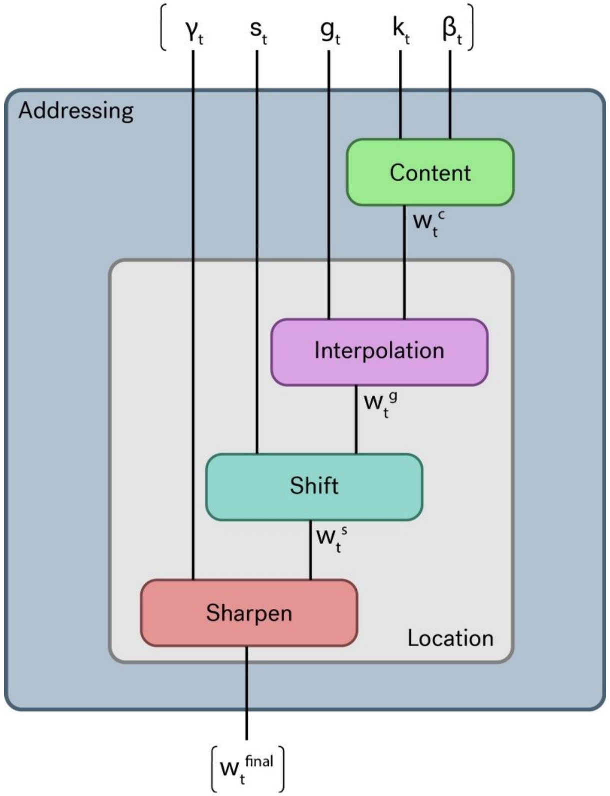
where

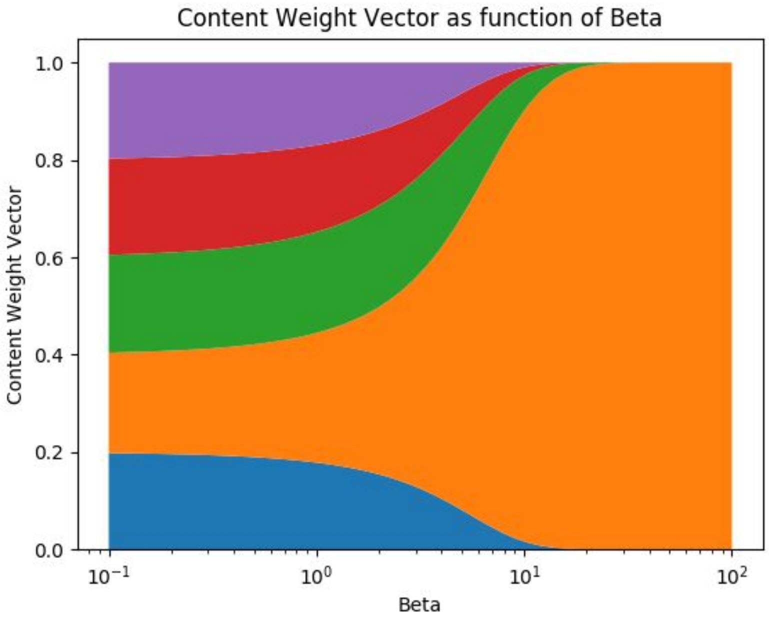
$$\delta(a, b) = \frac{a \cdot b}{|a| \cdot |b|}$$

This produces a normalized weight vector of similarities between each memory vector

https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

Addressing by Content

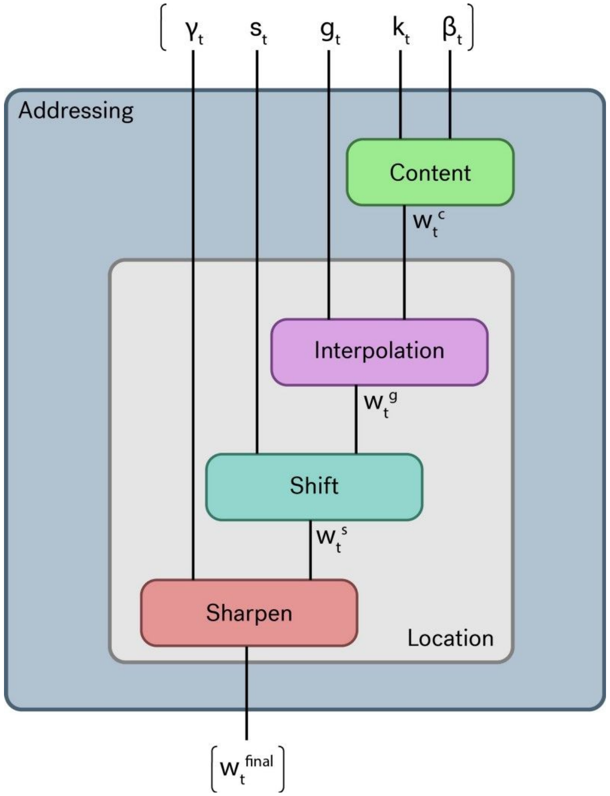




https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

https://rylanschaeffer.github.io/content/research/neural_turing_machine/main.html

Addressing by Location

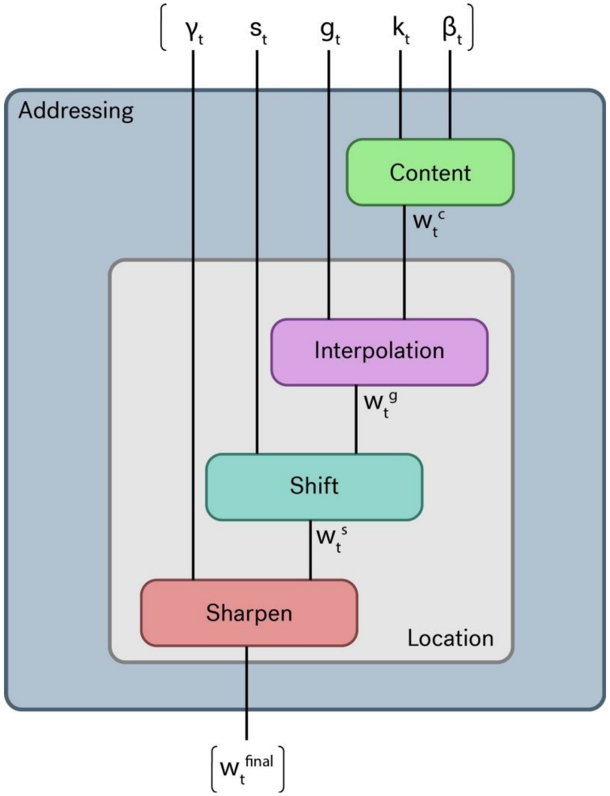


Neural turing machines have a powerful tool, an address by location feature. This is extremely important for generalizing functions to adapt to new values.

A mathematical example of where this is advantageous is the function x * y. We are not attempting to memorize every pairing of x*y but instead should realize to take the value at location of x and add it to the value of location of y.

https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

Addressing by Location



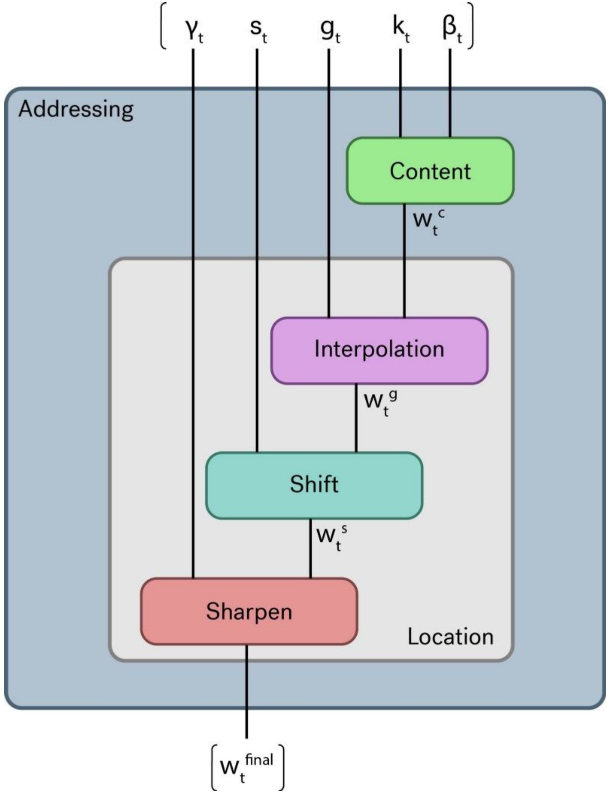
The model moves into its interpolation phase, which receives as inputs the weight vector generated by the content similarity, and a parameter g_t, which decides how much of the current weight vector should be blended with the previous read state.

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

If g_t were 0 we would completely ignore the content similarity weights we generated for the current state, if it were 1 we would completely ignore the previous state. Somewhere in between blends the two.

https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

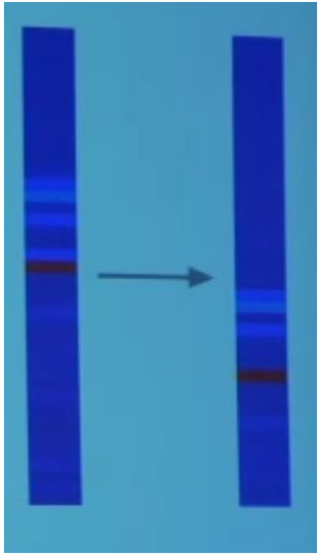
Addressing by Location



We then move onto the shift phase, which actually performs the moving in between memory locations. This takes in the interpolated weight vector and a shift weighting s_t. The shift weighting is (usually) a softmax layer, giving probabilities for each shift.

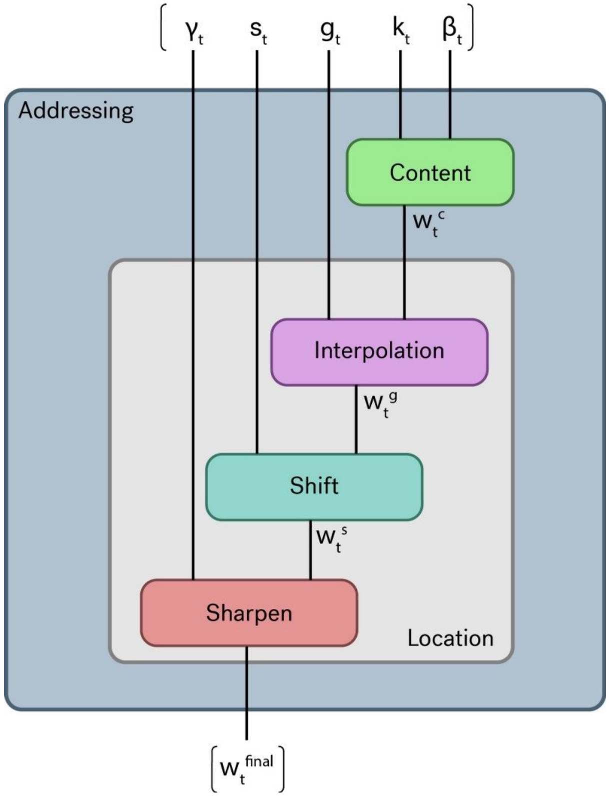
$$\tilde{w}_t(i) \longleftarrow \sum_{j=0}^{N-1} w_t^g(j)\, s_t(i-j)$$

The weight vectors are convolved with this shift weighting to produce the shifted weight vector.



https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

Addressing by Location

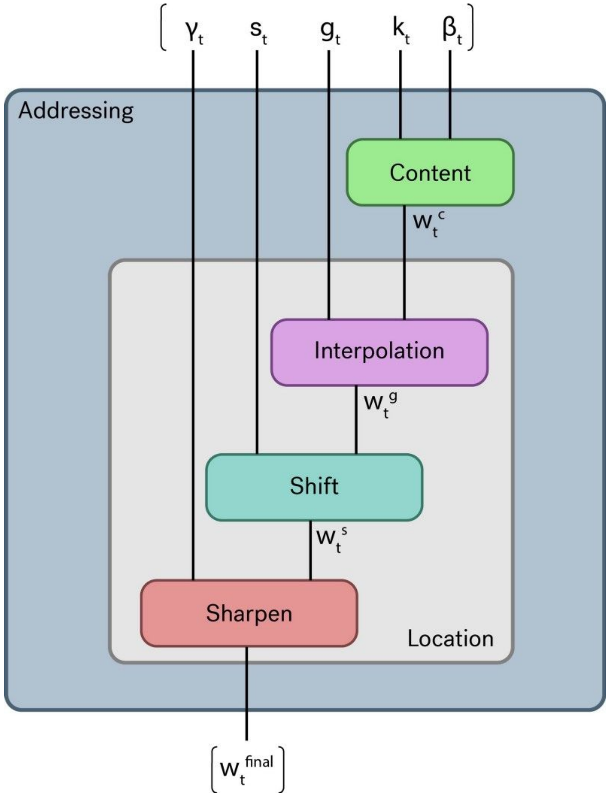


Finally a sharpening process is done to remove the blurred effect caused by the shift mechanism.

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

https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

Reading



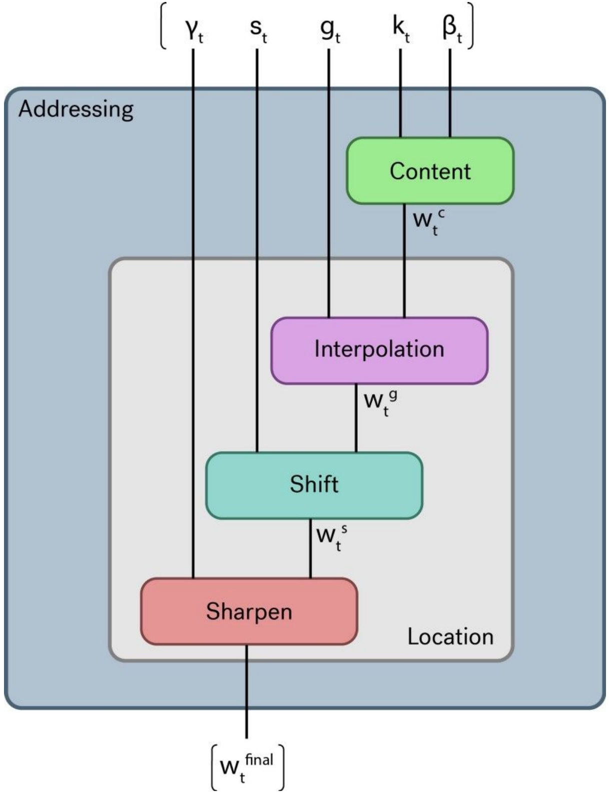
Finally the read vector of length M is produced with the following formula:

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

.05	.13219	1.2331

https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

Types of reads



These addressing schemes shown in the prior slides enable the following:

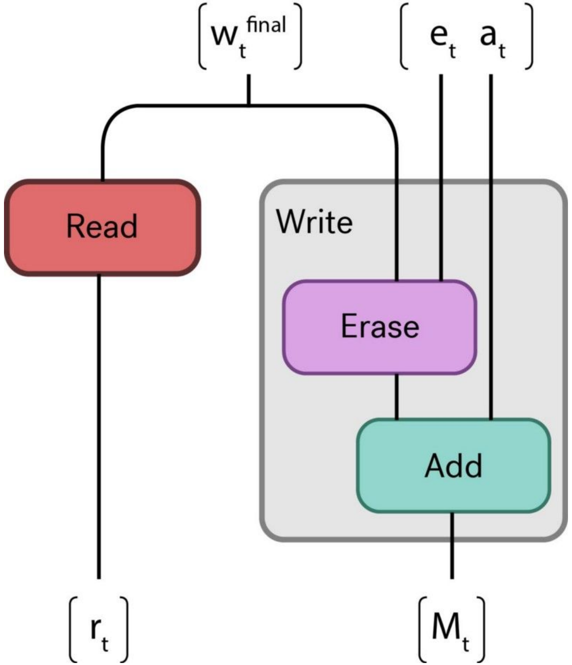
Reading can be done purely by similarity

Reading can be done purely by a shift of location

A combination of the two. Content finds memory location and shifts into it like an array.

https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

Writing



Using inspiration from LSTMs, Neural turing machines use erase(forget in LSTMs) and add(input in LSTMs)

$$\mathbf{M}_t(i) = \mathbf{M}_{t-1}(i)[1 - w_t^{final}(i) \cdot \mathbf{e}_t]$$

$$\mathbf{M}_t(i) = \mathbf{M}_t(i) + w_t^{final}(i) \cdot \mathbf{a}_t$$

https://medium.com/@aidangomez/the-neural-turing-machine-79f6e806c0a1

Experiments

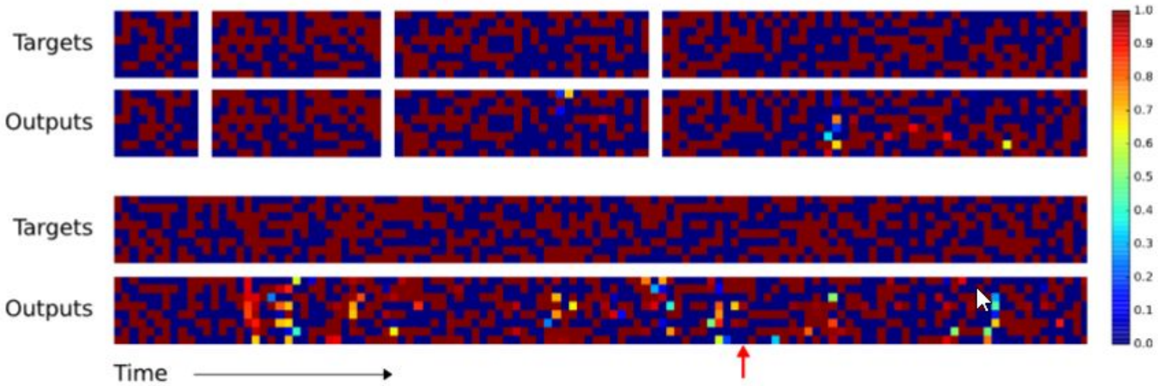


Figure 4: NTM Generalisation on the Copy Task. The four pairs of plots in the top row depict network outputs and corresponding copy targets for test sequences of length 10, 20, 30, and 50, respectively. The plots in the bottom row are for a length 120 sequence. The network was only trained on sequences 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.

Experiments

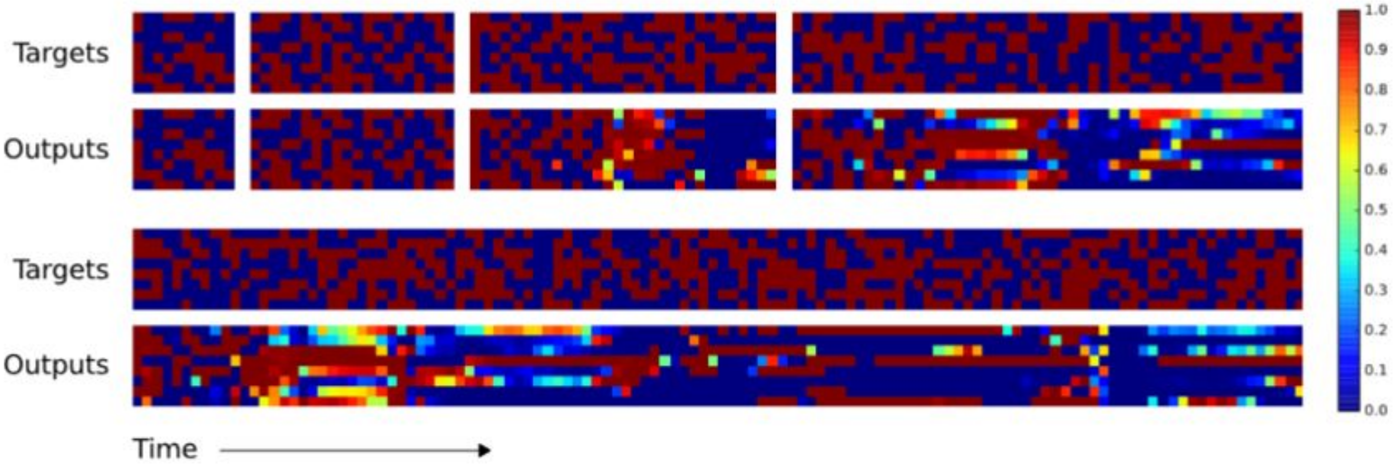


Figure 5: LSTM Generalisation on the Copy Task. The plots show inputs and outputs for the same sequence lengths as Figure 4. Like NTM, LSTM learns to reproduce sequences of up to length 20 almost perfectly. However it clearly fails to generalise to longer sequences. Also note that the length of the accurate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

Conclusions

Neural turing machines are able to learn simple algorithms from training examples and generalize greater than LSTMs.

https://arxiv.org/abs/1410.5401v2 - Neural Turing Machines Paper