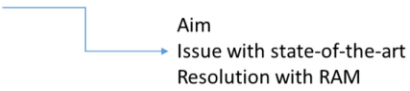
Reasoning, Attention and Memory Based Machine Learning Models

Prepared By:
Dhruv Kohli, Dept. of Mathematics

Content

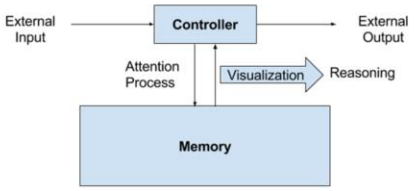
- Introduction
- Neural Turing Machine
- End to End Memory Networks
- Similarity between NTM and E2EMemNN
- Game Playing Agent with RAM



Following is the content of this presentation. First, the introduction, then we'll discuss 4 aspects of Neural Turing Machine and End to End memory Networks, i.e. the tasks in which they are used, elaborating on the models itself, little bit on training the models and finally the results that we obtained. Then we'll see the similarity b/w NTM and E2EMemNN and finally we'll discuss some points on building game playing agent with RAM based machine learning models.

Introduction

- The models discussed comprises of a memory component and a controller, such as an artificial neural network, which
  - Takes inputs from external world
  - Stores a representation or encoding of those inputs into the **memory**
  - Interacts with the memory using a defined mechanism called **attention process**
  - produces outputs for the external world with **reasoning** where the reasoning follows from the graphical visualization of the interaction between controller and memory.



The models discussed here comprises of two components, a memory and a controller. The controller takes input from the external world, stores a representation or encoding of those inputs into the memory, interacts with the memory using a defined mechanism called attention process, and finally produces outputs for the external world with reasoning where the reasoning follows from the graphical visualization of the interaction between the controller and the memory. By interaction we mean reading and writing operations.

Introduction

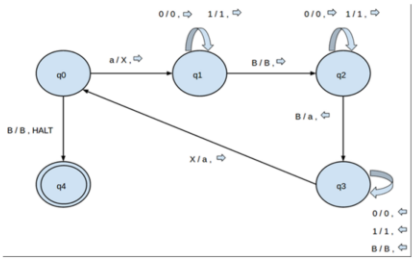
- In this thesis, I focus on applying RAM based machine learning models in the domain of sequence to sequence learning, question answering task and building game playing agents.

In this thesis, I focus on applying RAM based machine learning models in the domain of sequence to sequence learning, question answering task and building game playing agents.

Neural Turing Machine - Task

- NTM is used to learn algorithm mapping an sequence of inputs to a sequence of outputs.
- For ex: in Copy Task, where
  - input is B10110B BBBBB
  - output should be BBB……B 10110

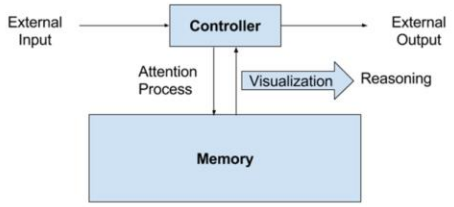
Following is the transition diagram for the copy task:



So, with Neural Turing Machine, the task that we are considering is the task of sequence to sequence learning, where the aim is to find the mapping between a sequence of inputs to a sequence of outputs. The copy task is an example of sequence to sequence learning. For instance the input sequence in copy task can be B10110BB… and the corresponding output sequence will be Blank till the second Blank of the input sequence and the 10110 i.e. the main content of the input sequence. Following is the transition diagram of copy task.

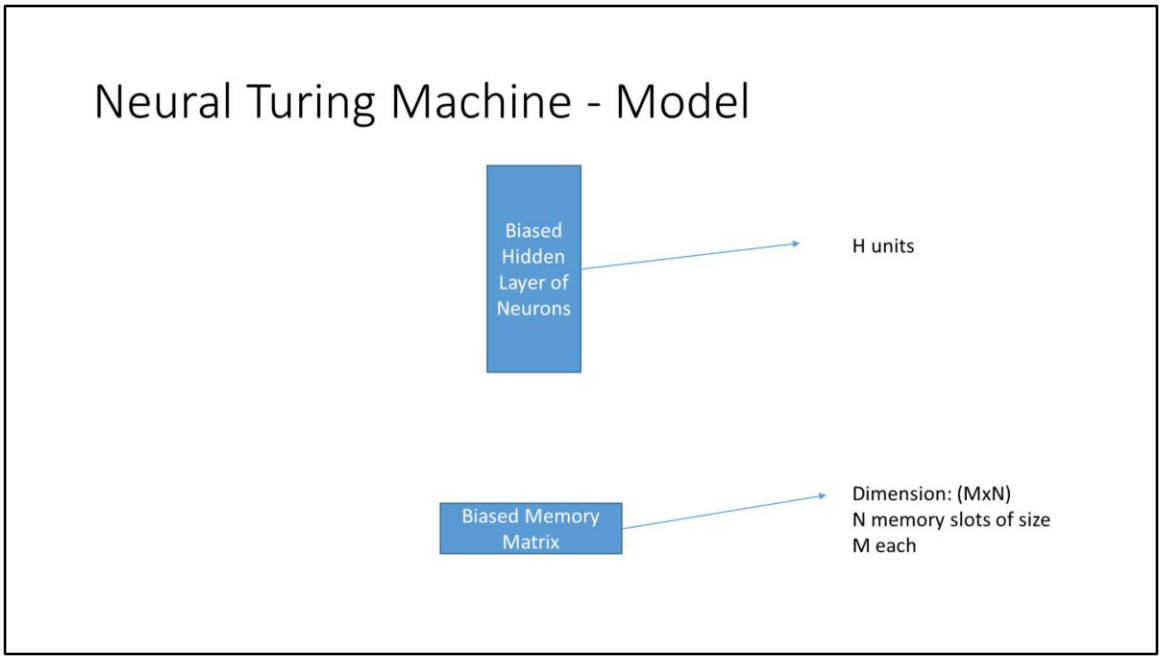
Neural Turing Machine - Model

- Unlike Turing Machine where the controller is defined (in the sense that the controller know the transition function), in NTM, the controller is learnt (in the sense that the transition function is learnt).
- "Learning a function", in statistical terms, is same as "approximating a function", which further reduces to "approximating the parameters of the function" given that the approach being followed for approximating the function is parametric.



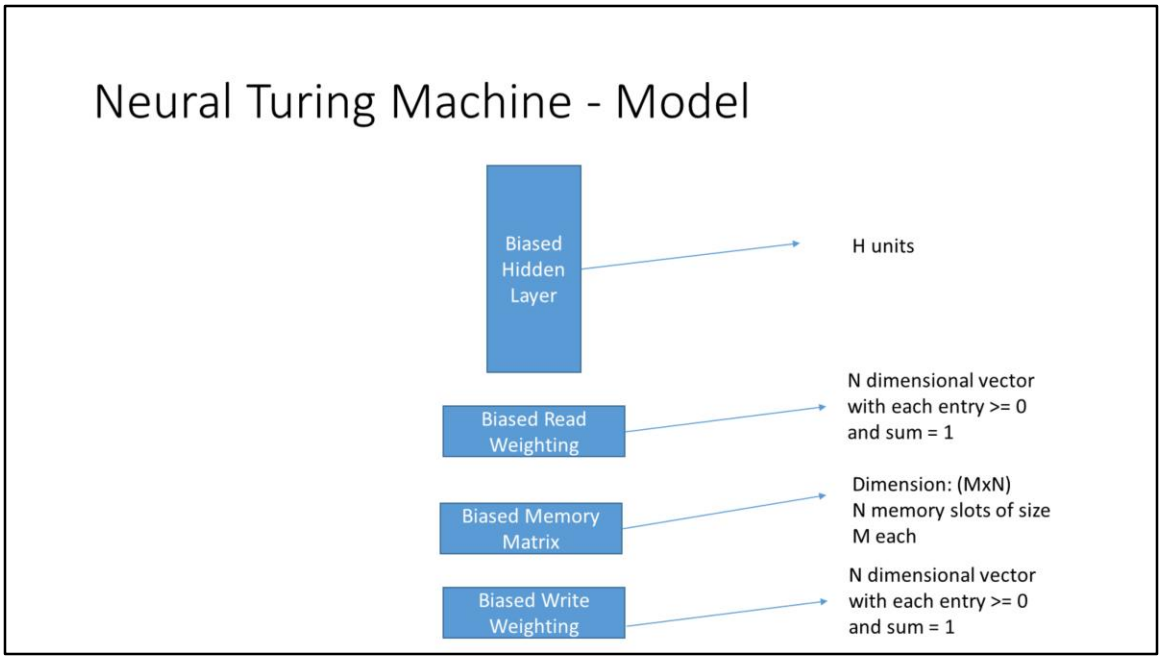
Architecture of NTM.

Unlike Turing Machine where the controller is defined in the sense that the controller knows the transition function, in NTM, the controller is learnt (in the sense that the transition function is learnt). And learning a function in statistical terms is same as approximating a function which further reduces to "approximating the parameters of the function" given that the approach being followed for approximating the function is parametric.

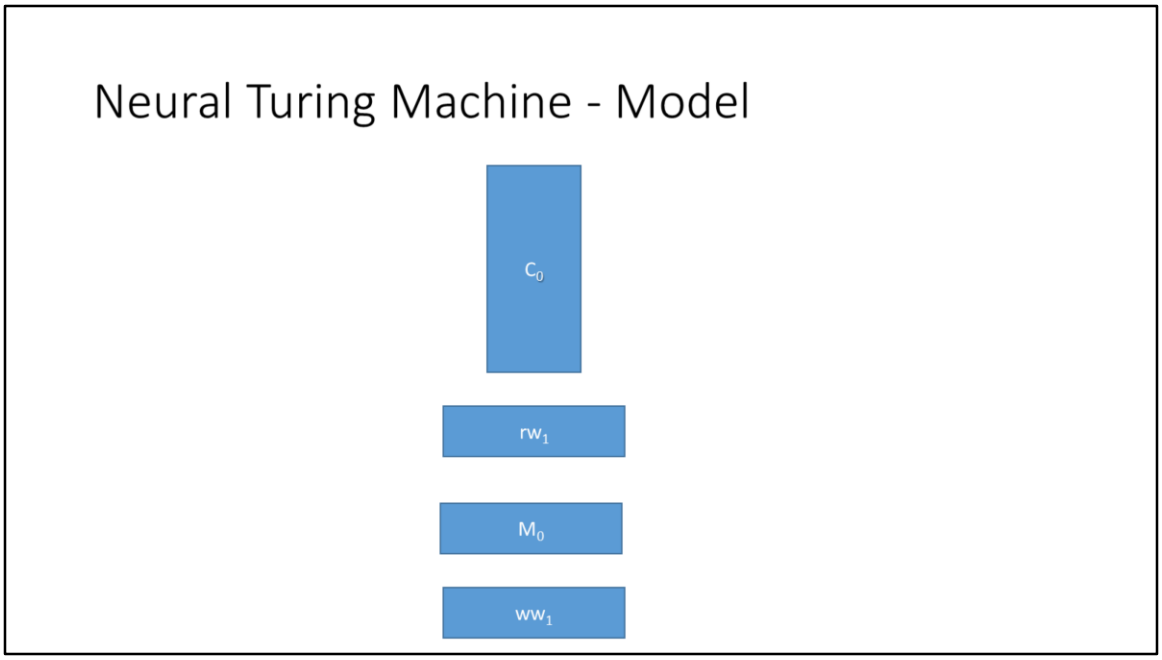


Now, I'll elaborate on the model.

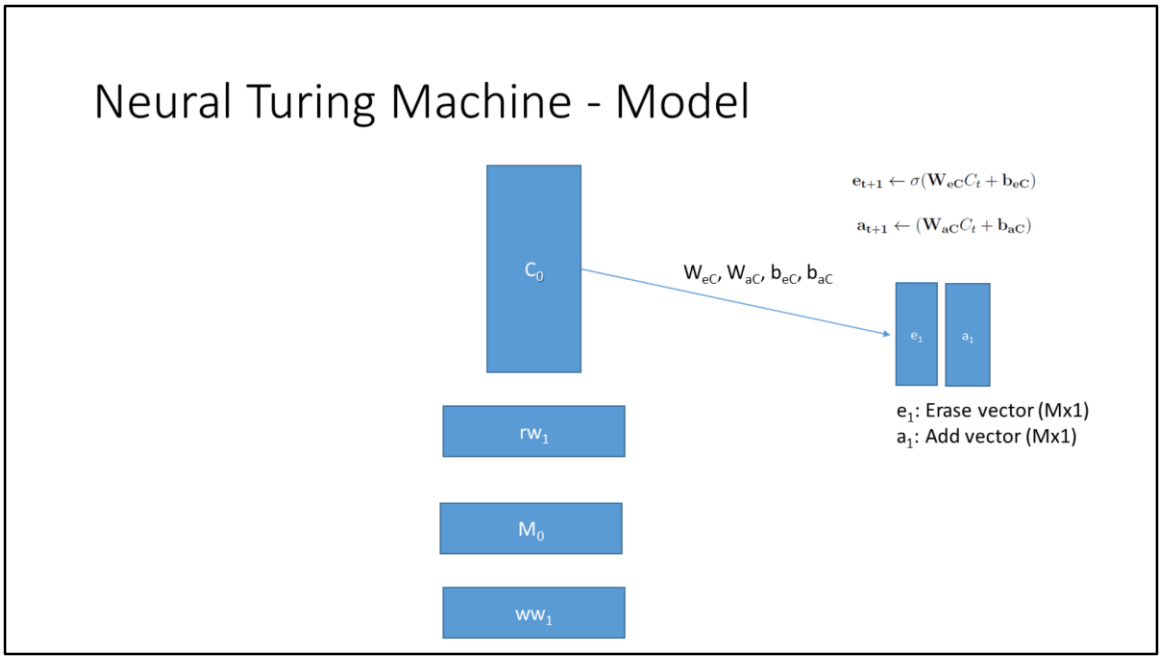
The controller in NTM is a neural network with a hidden layer of H neurons which are initialized with random values. The memory is a real matrix of dimension MxN where N is the number of memory slots or vectors and M is the width of each slot. The memory is also initialized with random values.



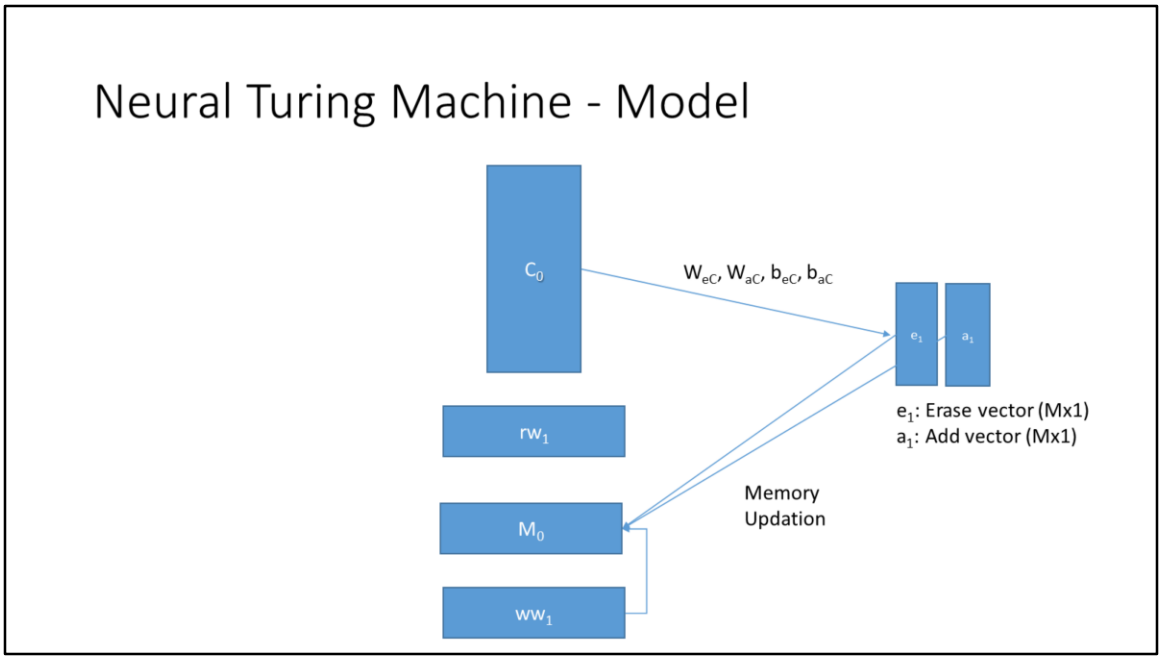
Then the controller read and write into the memory using read and write heads which are weighting vectors over the memory slots of dimension N where each entry of the vector is greater than equal to zero and the sum of entries equal 1. These weighting vectors are used to either strongly focus on single slot of the memory (when the weighting vector has a 1 corresponding to that slot and zeros corresponding to all other slots) or weakly focus on multiple slots when the weighting vector is non zero for multiple slots.



Following is the naming convention for the initial state of NTM.



Now, the hidden layer of neurons are used to produce an erase vector and an add vector of dimension M using the formula shown. Here, W_{eC}, W_{aC}, b_{eC}, b_{aC} are the parameters of appropriate dimension.



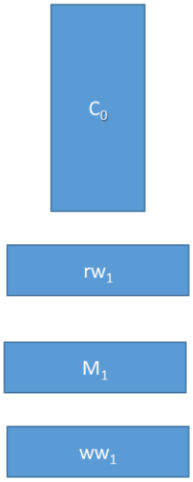
Then, these erase and add vectors combined with the write weighting vector and the previous state of the memory matrix are used to produce new state of the memory matrix. One can imagine that the write weighting vector first chooses the memory slots to write in and then the erase and add vectors are used to erase some content and add some content to these memory slots. These erase and add operations together form a write operation.

Neural Turing Machine – Model (Memory Updation)

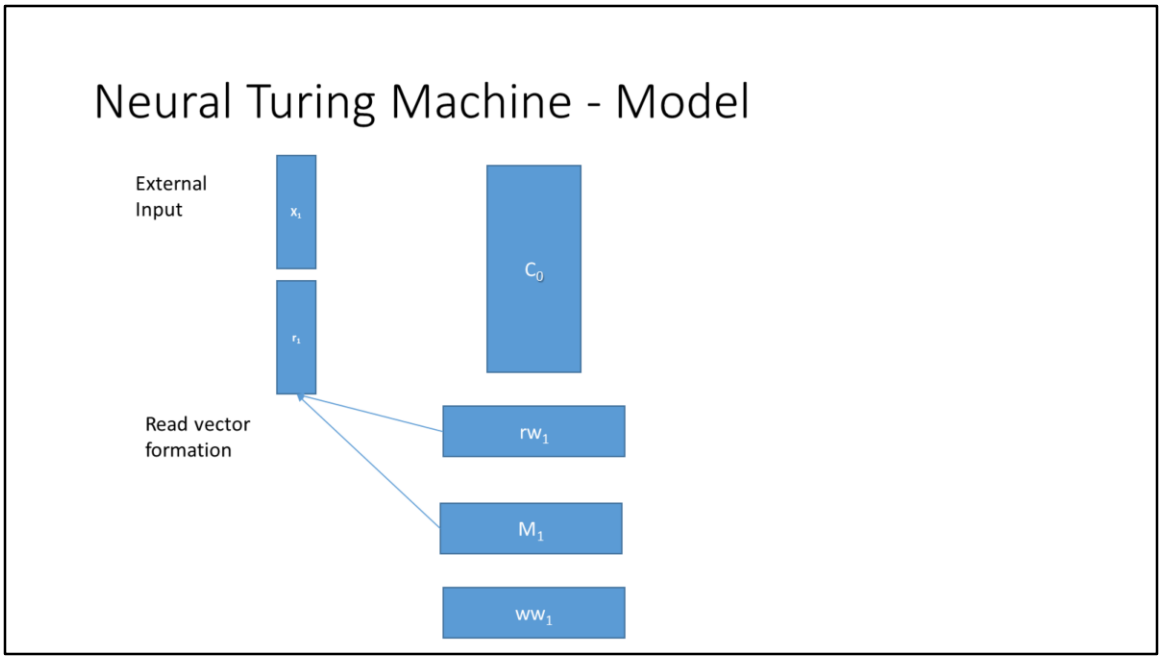
$$\widetilde{\mathbf{M}}_t(i) \leftarrow \mathbf{M}_{t-1}(i)[1 - ww_t(i)\mathbf{e}_t]$$
$$\mathbf{M}_t(i) \leftarrow \widetilde{\mathbf{M}}_t(i) + ww_t(i)\mathbf{a}_t$$

Here, is the mathematics behind the updation of memory. it's quite simple and I've already told you the intuition behind these equations.

Neural Turing Machine - Model



So, now, we have our updated memory M_1.

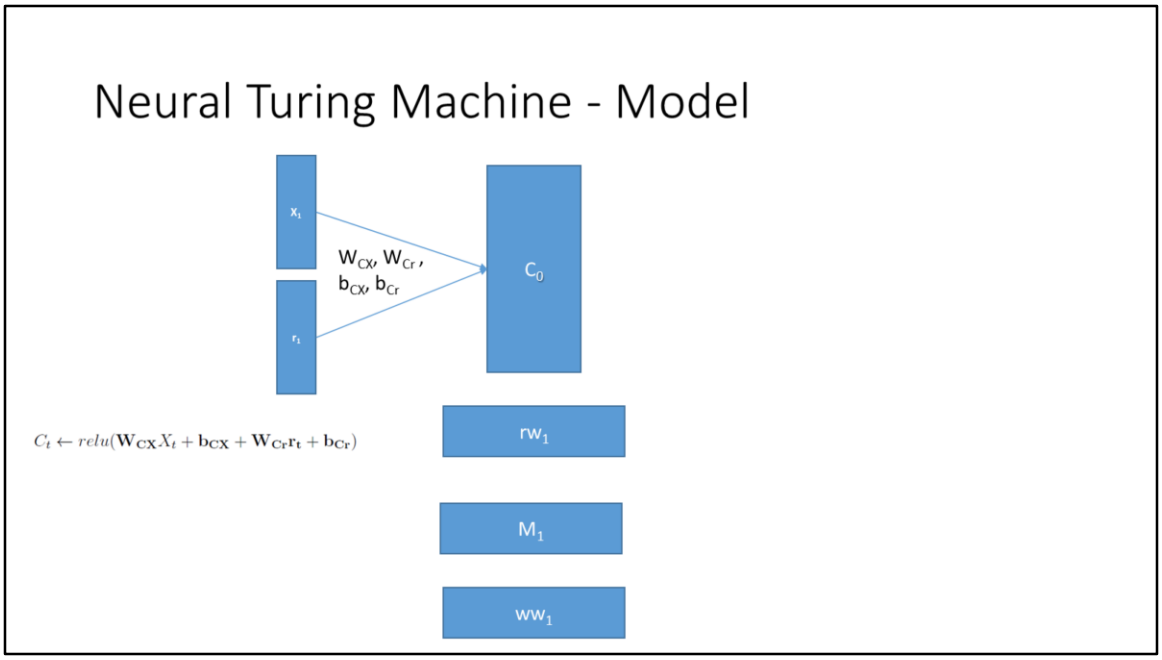


The updated memory M1 and the read weighting vector rw1 are used to form a read vector. And Here, X_1 is the external output.

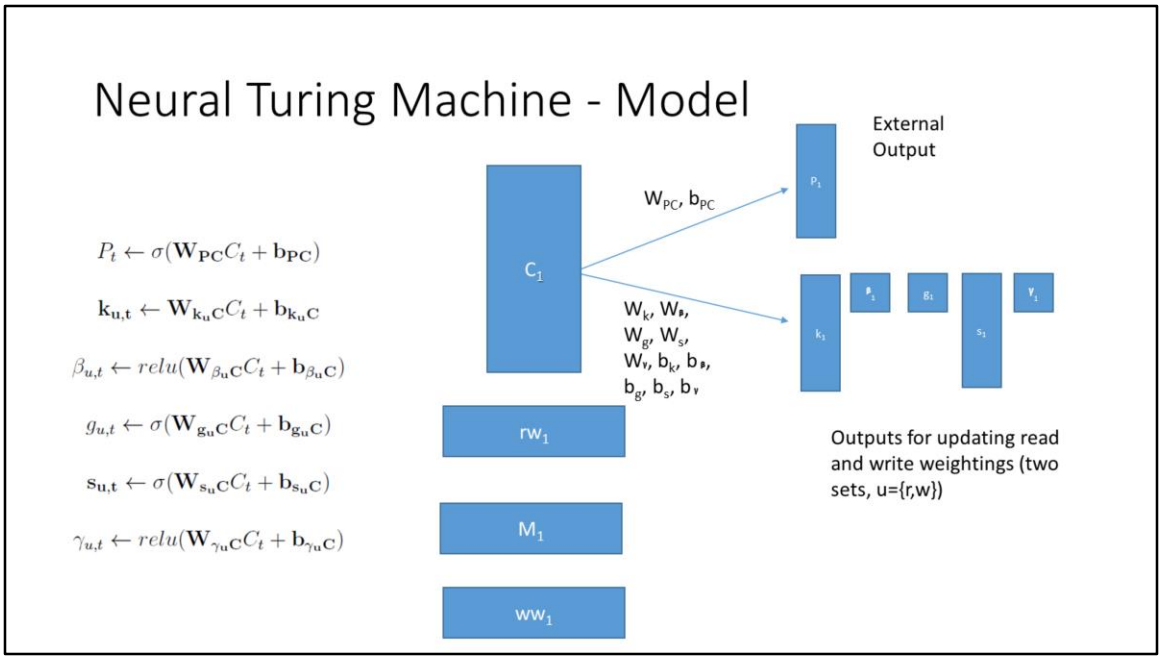
Neural Turing Machine – Model (Read Vector Formation)

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

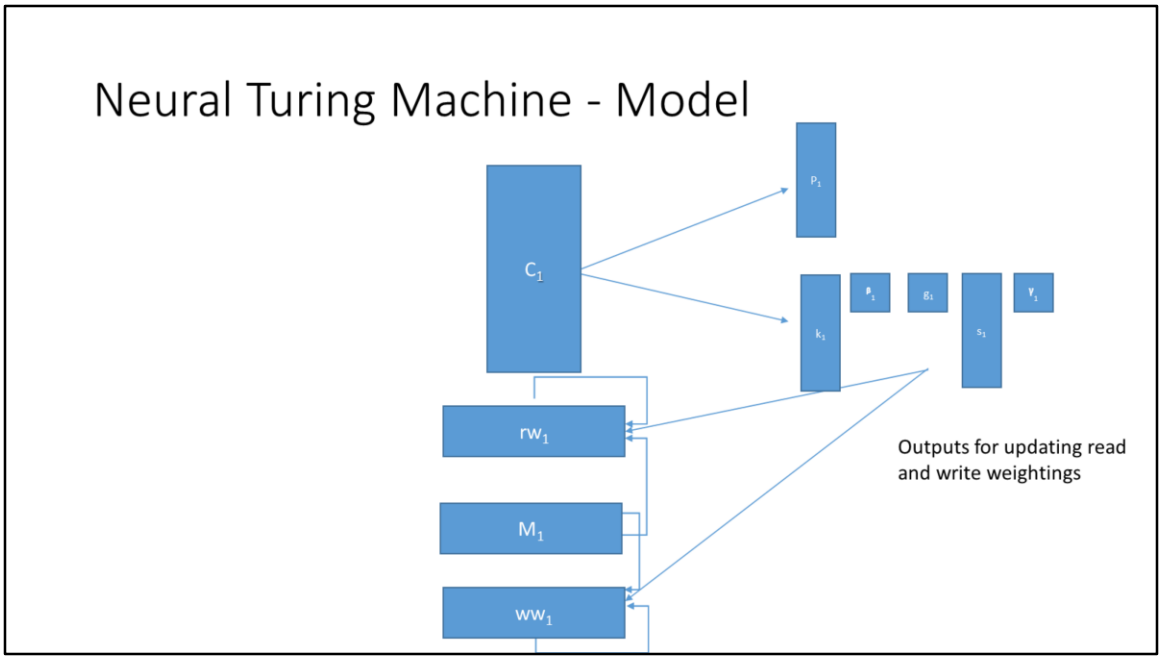
Mathematics behind read vector formation. This just a weighted sum of memory slots weighted by the read weighting vector.



Then, the external input and the read vector are used to compute the new state of hidden layer of neurons (just like forward pass in Neural Network) using the formula shown. Here. Wcx Wcr, bcx and bcr are parameters of the model.



Then, the updated hidden layer is used to produce external output, basically a prediction, P_1 and two sets of outputs, one for updating read weighting vector and other for updating write weighting vector. Again these W's and b's are parameters of the model and the way the outputs are computed are shown,



These sets of outputs with the memory matrix and previous state of weightings are used to produce new weightings.

Neural Turing Machine – Model (Weight Updation)

Updation outputs, k_t, β_t, g_t, s_t and γ_t are used.

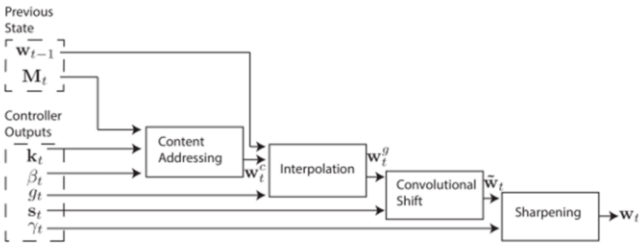
k_t = Key vector (Mx1)

β_t = Key strength scalar

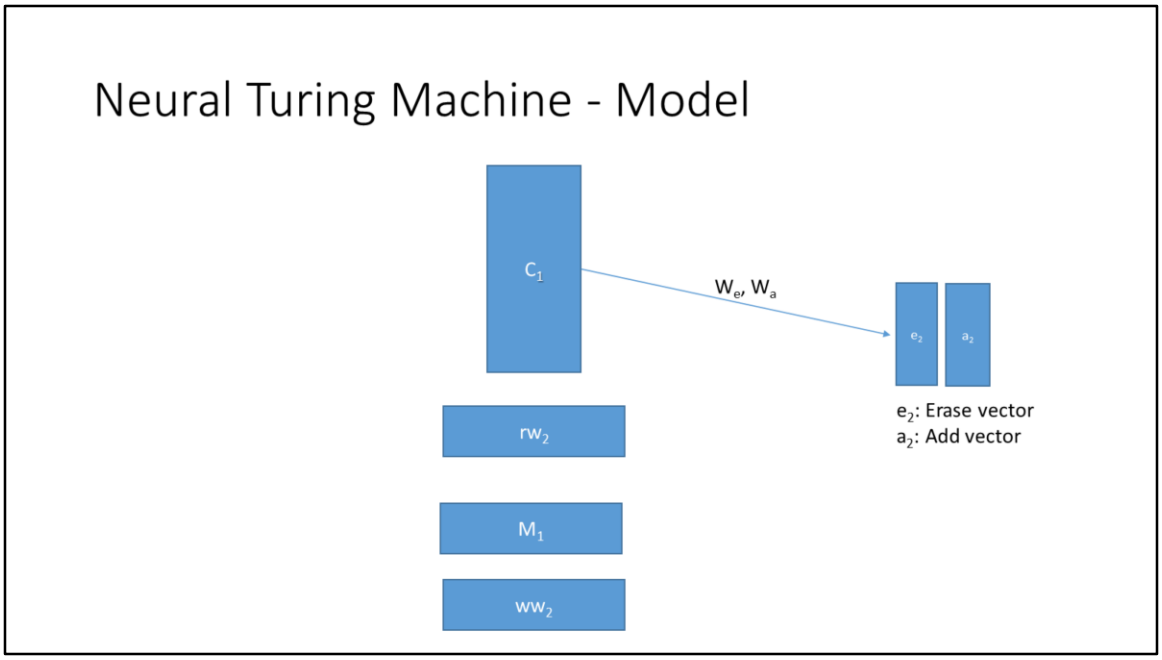
g_t = Gate scalar for interpolation → Hyper-parameter

s_t = Shift weighting vector (number-of-allowed-shiftx1)

γ_t = sharpening scalar



Now, what these set of outputs mean and how these outputs are used to produce the new weighting vector are described in the thesis. And, I'll skip this part as it is quite time consuming.



So, now we have got our updated read and write weighting vectors. Again the hidden layer is used to produce the erase and add vectors.

Neural Turing Machine - Model

And the whole process repeats until the external input is exhausted.

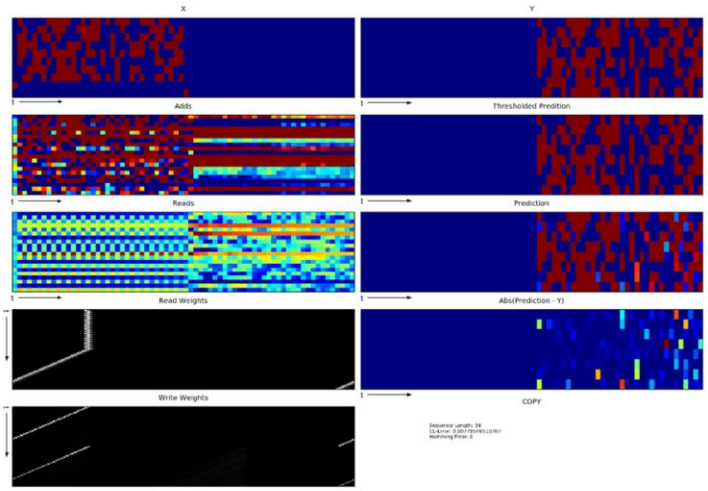
Neural Turing Machine - Model

- Now we have a sequence of predictions and a sequence of external outputs (target sequence).
- The error between the prediction sequence and the target sequence is computed and propagated back.
- Then, using these errors the parameters of the model are updated to a value that reduces the prediction error.

Neural Turing Machine - Training

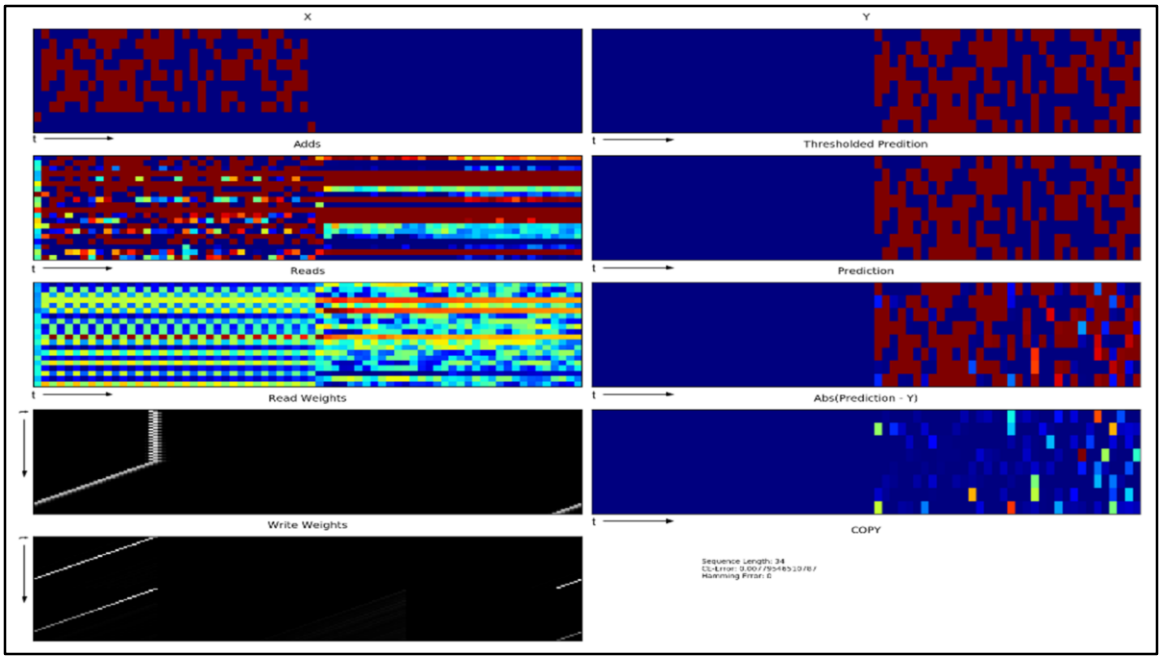
- The model is end to end differentiable with respect to the parameters, so backpropagation algorithm can be used to compute the derivative of loss with respect to the parameters.
- We used RMSProp version of Gradient Descent to update the parameters.
- Previous semester code:
  - CPP, gradient computation from scratch, was not getting good generalization in copy task.
- This semester code:
  - Python-Theano, Theano computes gradients, excellent generalization in copy task.
  - By excellent generalization, we mean that we trained on sequences of length till 20 and got zero hamming distance on sequences of length till 116.

Neural Turing Machine – Results (Copy Task)



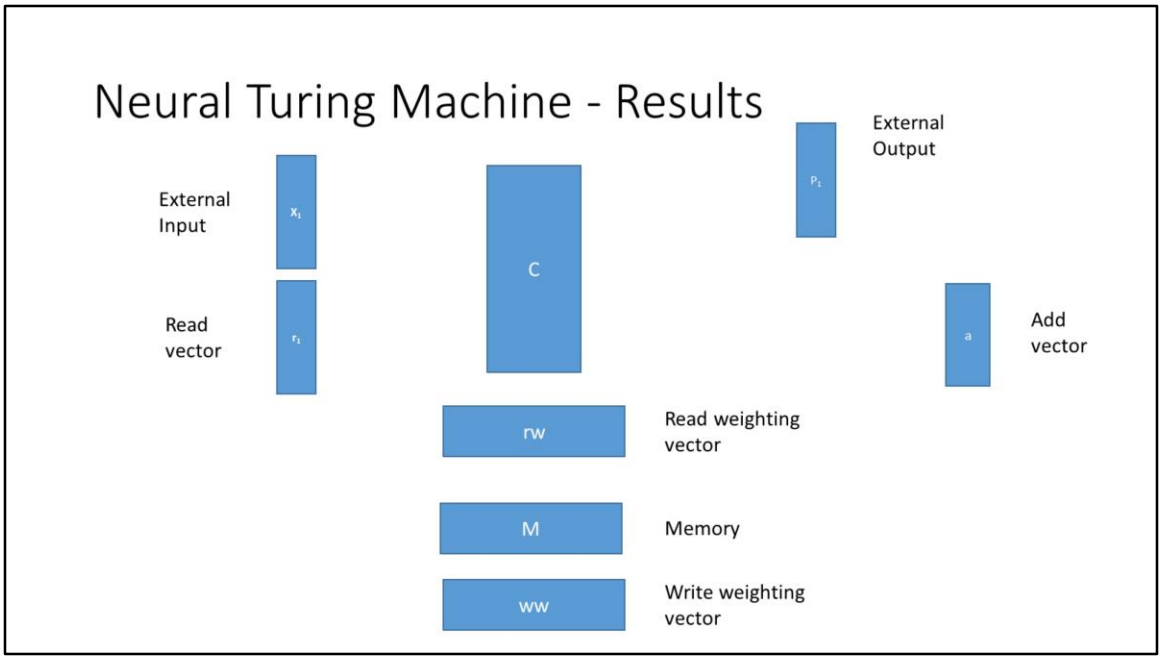
Here is the plot of the simulation of NTM on an input sequence of length 34. Before understanding this plot, let us understand how interpret the results.

Now, let's look at this figure. You can observe that X is a sequence of external inputs which are binary vectors starting from a delimiter and the actual content of the sequence ends with an ending delimiter. Y is the target sequence of external outputs which is blank till the second delimiter of the input sequence and after second delimiter has the actual content of the input sequence. And then we have sequence of add vectors, read vectors, read and write weighting vectors. Now, you can see that till the second delimiter, the add vectors has some non constant pattern and read vectors have constant pattern while after the second delimiter the read vector has non constant pattern while the add vector has constant pattern. Also, the sequences of read and write weighting vectors has the same form as was expected.

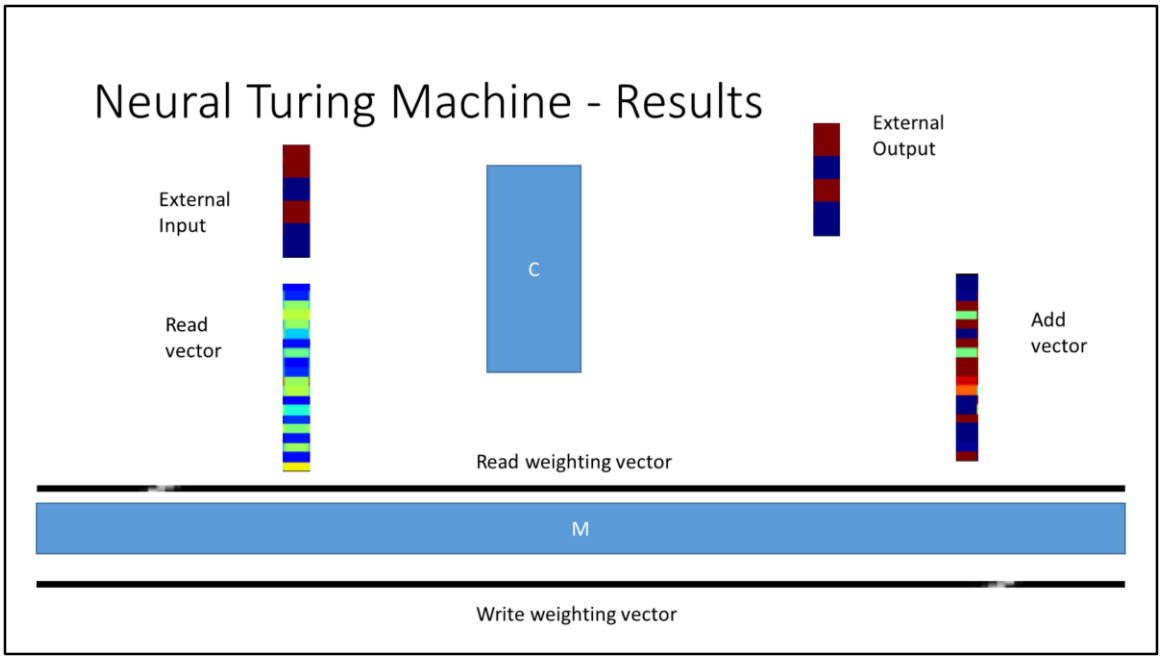


Here is the plot of the simulation of NTM on an input sequence of length 34. Before understanding this plot, let us understand how interpret the results.

Now, let's look at this figure. You can observe that X is a sequence of external inputs which are binary vectors starting from a delimiter and the actual content of the sequence ends with an ending delimiter. Y is the target sequence of external outputs which is blank till the second delimiter of the input sequence and after second delimiter has the actual content of the input sequence. And then we have sequence of add vectors, read vectors, read and write weighting vectors. Now, you can see that till the second delimiter, the add vectors has some non constant pattern and read vectors have constant pattern while after the second delimiter the read vectors have non constant pattern while the add vectors have constant pattern. Also, the sequences of read and write weighting vectors have the same form as was expected.



These are some of the elements of our model.

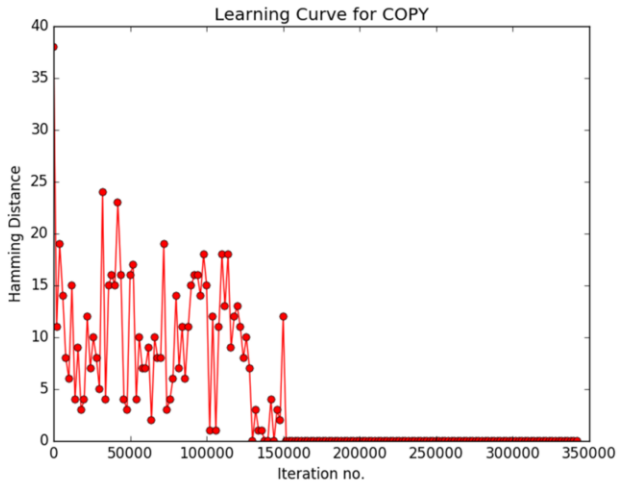


In the plot that I showed you before, the external input is a sequence of binary vectors where a single vector is shown in this figure. Red = 1 and Blue = 0. Similarly, we'll get a sequence of external outputs, add vectors, read vectors, read weighting vector and write weighting vector. An element of each of these sequences are shown in this slide. Note that the black color in the weighting vector represents a value close to 0 and white represents a value close to 1.

Neural Turing Machine - Results

- If the NTM has really learnt the transition function of the copy task then
  - Till the second delimiter, the write operation (into memory) should be dominant and hence plotting the add vectors should show some non constant pattern whereas plotting the read vectors should show some constant pattern. Also the write weighting vector should focus on consecutive slots of the memory and the focus by read weighting vector is of no significance till second delimiter.
  - After the second delimiter, the read operation (from memory) becomes dominant and hence plotting the read vectors should show some non constant pattern whereas plotting the add vectors should show some constant pattern. Also the read weighting vector should focus on consecutive slots of the memory in the same order as the write weighting vector before second delimiter and the focus by write weighting vector is of no significance after second delimiter.

Neural Turing Machine – Results (Copy Task)



This is the learning curve. X axis denotes the iteration number where each iteration comprises of a forward pass to compute prediction and a backward pass to compute gradients and updation of parameters. The Y axis denotes the Hamming distance between the prediction and the target. After 150000 iterations the hamming distance becomes zero.

End to End Memory Networks - Task

- Input:
  - Comprehension:
    1 Mary got the milk there.
    2 John moved to the bedroom.
    3 Sandra went back to the kitchen.
    4 Mary travelled to the hallway.
  - Query:
    5 Where is the milk?
- Output:
  - Answer: hallway
  - Sentences used in computing answer: 1, 4

The task that we consider for end to end memory networks is the task of question answering where the input comprises of a comprehension and a query and the output is a single one word answer to the query and the sentences used in computing the answer.

End to End Memory Networks - Model

- Suppose s_1, s_2, …, s_n be the sentences in our comprehension, q be the query sentence and y be the answer to the query.
- Step 1: Convert s_{ij} (a word) to x_{ij} where x_{ij} = BOW(s_{ij})
- BOW means "Bag Of Words".
- Let V = size of vocabulary.

Suppose s_1, s_2, …, s_n be the sentences in our comprehension, q be the query sentence and y be the answer to the query.
Step 1 is to Convert s_{ij} (a word) to x_{ij} where x_{ij} = BOW(s_{ij})

End to End Memory Networks - Model

- Step 2: Compute memory vectors m_i from x_i using an embedding, A (dxV).

$$m_i = \sum_j l_j \cdot Ax_{ij}$$

where

$$l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$$

where J = total number of words in sentence s_i and d is the dimension of the embedding A.

Note: This was something new that I never saw before. The above representation contains the information regarding context as well as order of words (Positional Encoding).

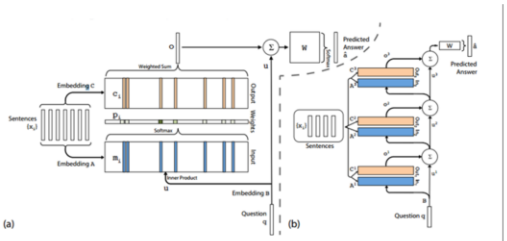


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

Step 2 is to Compute memory vectors m_i from x_i using an embedding, A where l_j is defined in the following manner. The basic intuition behind the formation of the memory vectors is to capture the contextual and the word ordering information from the sentence.

End to End Memory Networks - Model

- Step 3: In the same manner, output vectors c_i are computed from x_i using an embedding, C and internal representation, u, of query q using an embedding B.
  - Both C and B are of dimension dxV.

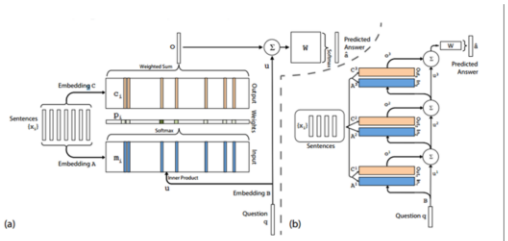


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

End to End Memory Networks - Model

- Step 4: Then a weighting, p, over the memory vectors is computed by computing the similarity between the memory vectors and the internal representation of query by softmax.

$$p_i = \text{Softmax}(u^T m_i).$$

  - One can easily observe that, the higher the value of p_i, more relevant is the memory vector m_i in answering the query.

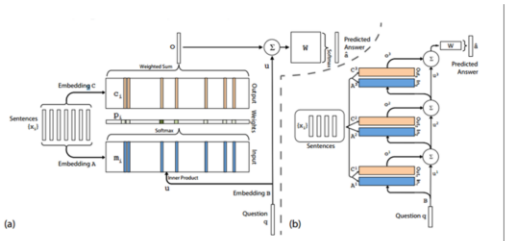


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

End to End Memory Networks - Model

- Step 5: An intermediate response, o, is generated by weighted sum of the output vectors c_i with p_i as weight corresponding to c_i.

$$o = \sum_i p_i c_i.$$

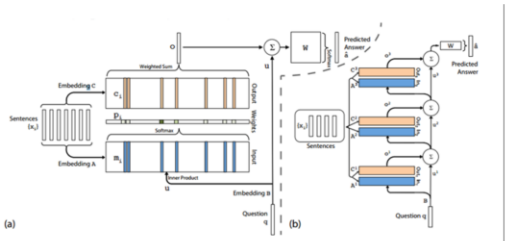


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

End to End Memory Networks - Model

- Step 6: This intermediate response is o, is summed with the internal representation of query u.

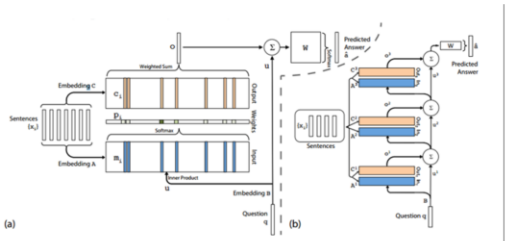


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

End to End Memory Networks - Model

- Step 7: The summed vector is transformed using a decoding matrix W of dimension Vxd and softmax-ed to a V dimensional final response â.

$$\hat{a} = \text{Softmax}(W(o + u))$$

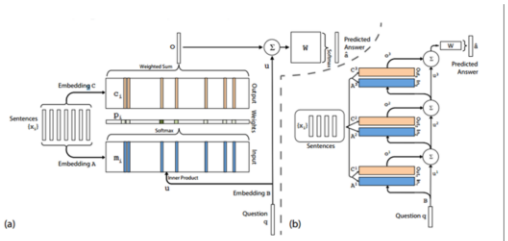


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

End to End Memory Networks - Model

- Loss: Binary cross-entropy error between â and BOW(y) where y is the target one-word answer.

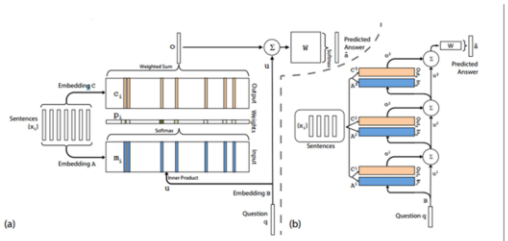


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

End to End Memory Networks - Model

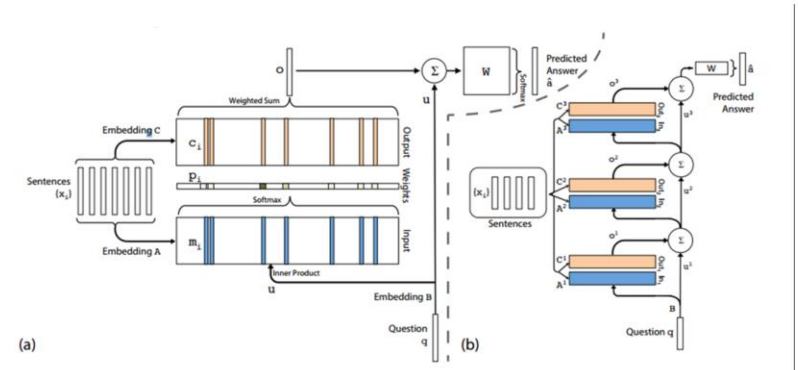


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

On Left hand side, you can see a single layer of memory networks. These layers can be stacked to form a multi layer memory network. One can note this model is end to end differentiable with respect to the parameters of the model.

The above procedure explains a single layer of memory networks. This is incapable of answering questions which require transitive implications. For ex:

Mice are afraid of wolves.
Jerry is a mouse.
What is Jerry afraid of? wolf

To overcome this, concept of Multiple Hops is proposed.
The intermediate response generated, o, and the internal representation of query, u are summed and the summed vector acts as the new internal representation of the query.
THIS SUMMATION is important so that the model doesn't forget the context of the query itself.
Now, with this new internal representation of the query, step 2 to 7 are repeated and for the final hop (layer) step 8 is performed.

End to End Memory Networks - Model

- Observe that the parameter space is too high, B, W, A_i, C_i, i=1..number-of-hops/layers.
- To tackle this, two types of parameter tying is used:
  - RNN like: $A_1 = A_2 = \ldots = A_n = A$ and $C_1 = C_2 = \ldots = C_n$
  - Adjacent: $C_{k+1} = A_k$, $B = A_1$ and $W^T = C_N$ (We used this in our model)

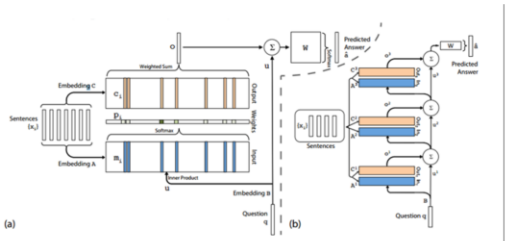


Figure 1: [5](a) Single layer of End to End Memory Networks (b) 3-layered End to End Memory Networks

End to End Memory Networks - Training

- Since the model is end to end differentiable with respect to the parameters, one can easily use backpropagation algorithm for computing the derivative of loss with respect to the parameters.
- Stochastic Gradient Descent is used to update the parameters.

End to End Memory Networks - Results

- We used Babi Project Dataset from FB-AI research.
- To test whether, the network is able to reproduce the results in toy tasks, we tested with 20 toy tasks
- With Single supporting fact toy task we got following results:

	C1	C2	C3	C4	C5	C6
C1	148	0	0	0	0	0
C2	0	168	0	2	0	0
C3	1	0	180	1	1	1
C4	0	0	0	153	0	0
C5	0	0	0	0	156	0
C6	1	0	0	0	0	180

Confusion Matrix

Precision, recall and f1-score calculated using this matrix are consistent with the results shown by FB-AI.

Single Supporting Fact
Comprehension:
1 John travelled to the hallway.
2 Mary journeyed to the bathroom.
Query:
3 Where is John?
Answer:
hallway
Sentence used in inference: 1

One can observe from the confusion matrix that the model predicts the correct class with very high frequency, or probability.

Similarity between Neural Turing Machine and End to End Memory Network

- The content addressing module of NTM is similar to the similarity finding between memory vectors and internal representation of query.
- The location based addressing module of NTM seems to be implementing the multiple hops concept of end to end memory networks.

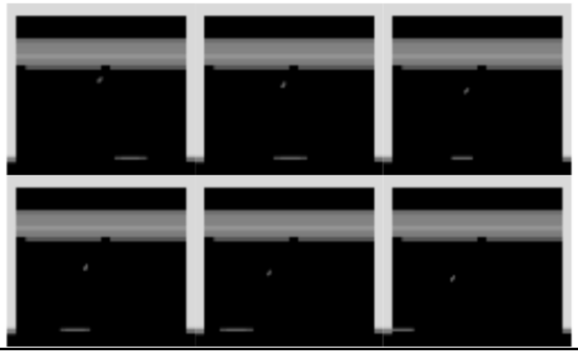
Sorry, I haven't explained the content addressing module and the location based addressing module of NTM.

Game Playing Agent with RAM – Aim

- To build a model
  - that can learn to play any game from the sensory input only
  - in an optimal manner (better than humans)
- Using
  - Reinforcement Learning
    - Markov Decision Process
    - Q-Learning
  - Deep Learning
    - Convolution Neural Networks

Game Playing Agent with RAM – Problem with state-of-the-art

- Problem with current state of the art?
- Consider the game *Breakout* and the following sequence of frames observed by an agent.



Game Playing Agent with RAM – Problem with state-of-the-art

- State-of-the-art agent (based on deep learning) moves to the left.
- But why?
- A human agent might answer that since the ball is moving towards left the action taken is to move left.
- This is the part that the state-of-the-art agent doesn't answer. It has just learnt some set of parameters that once fitted in the neural network predicts optimal action to take.
- But hopefully the RAM-based agent will be able to reason that "since the ball is moving towards left, I should also move towards left to prevent death".

Game Playing Agent with RAM – Resolution with RAM (Ideas)

- Weighting over memory slots as well as over the information (bits) in a slot itself.
  - Hope is that: the ball and the slider in the frame will be focused by modified NTM.
  - That'll provide reasoning to some extent that the decision (choice of action) by modified NTM is taken based on the position of ball and the slider.
- Increasing the number of allowed shifts in NTM to be equal to the number of memory slots so that the focus can shift from one memory slot to arbitrarily any slot.