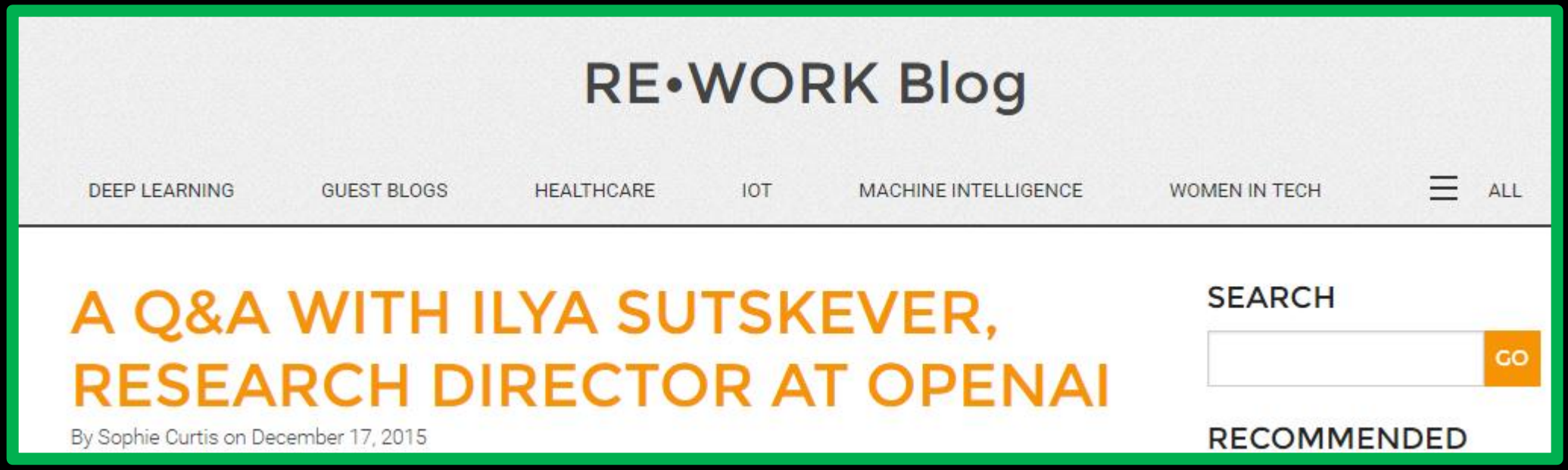
Accessing Memory in Neural Networks

Anantharaman Palacode Narayana Iyer

23 Sep 2016

Attention Networks

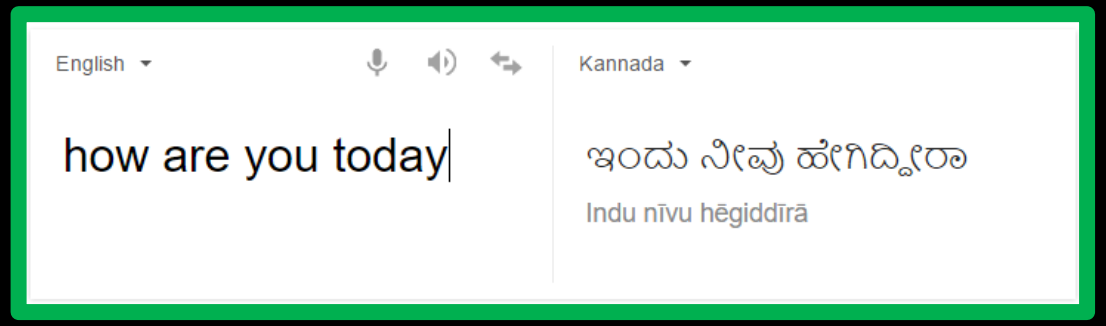


What advancements excite you most in the field?

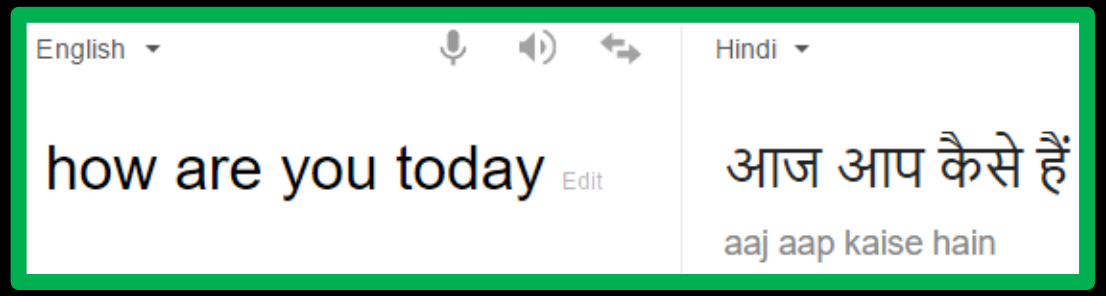
I am very excited by the recently introduced attention models, due to their simplicity and due to the fact that they work so well. Although these models are new, I have no doubt that they are here to stay, and that they will play a very important role in the future of deep learning.

What are attention networks?

- Suppose we have a sequence of input and we are to produce a sequence of outputs for this.
  - E.g. The inputs could be a sequence of words, could also be image segments

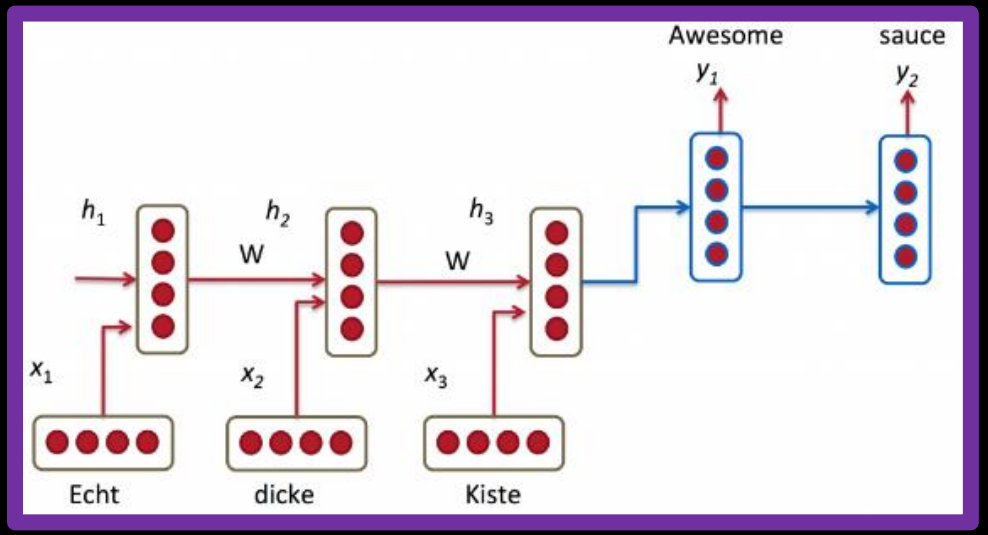


- Attention mechanisms focus on a narrow region of the input when making a classification decision for the output sequence.
  - E.g. In a machine translation, when outputting a word, the attention network might give higher weightage to certain words in the input.



Neural Machine Translation

- Traditional machine translation techniques require domain knowledge to build sophisticated features.
- In NMT, the source sentence is encoded as a single fixed length vector which is then translated in to the target language sentence.
- The encoding process is equivalent to learning a higher level meaning of the sentence, which can't be effectively captured using techniques like n-grams
- However, as the length of input sequence is arbitrary, it is not always effective to encode every one of them with same length vector.
- Attention mechanisms address the issue by varying the focus on input words as the output words are generated one at a time



Attention Mechanism

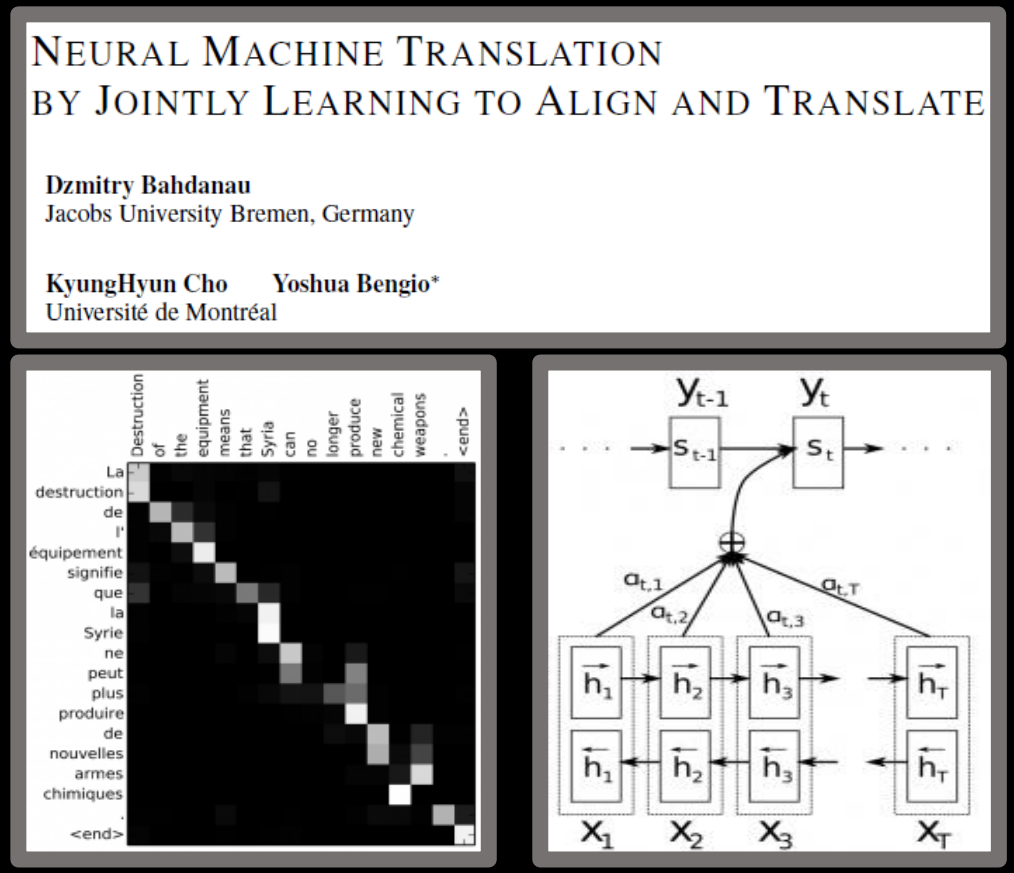
- When using the attention mechanism we make use of the hidden vectors computed at each time step of the encoder and hence the decoding doesn't need to depend on one single hidden vector

- The decoder "attends" to different parts of the source sentence at each step of the output generation.

- The model learns what to attend to based on the input sentence and what it has produced so far.
  - E.g. In the example of English to Kannada translation, the last word "today" translates in to the first word "indu" in Kannada. The attention model learns to focus on "today" when generating "indu"

Attention Mechanism

- X_i are the input words, processed by the encoder. Y_i are the outputs, the translated words
- The output Y_t depends not only on the previous hidden state value but also on the weighted combination of all input states
- α‘s are weights that determine how much of a given input word should be considered in computing the output at a given time step
- α‘s are normalized to 1 using a softmax



Machine Translation – Details

$$p(y_i|y_1, \ldots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

The context vector c_i depends on a sequence of annotations h_j

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij}\, h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

Where: $e_{ij} = a(s_{i-1}, h_j)$, a is the alignment model learnt by a neural network

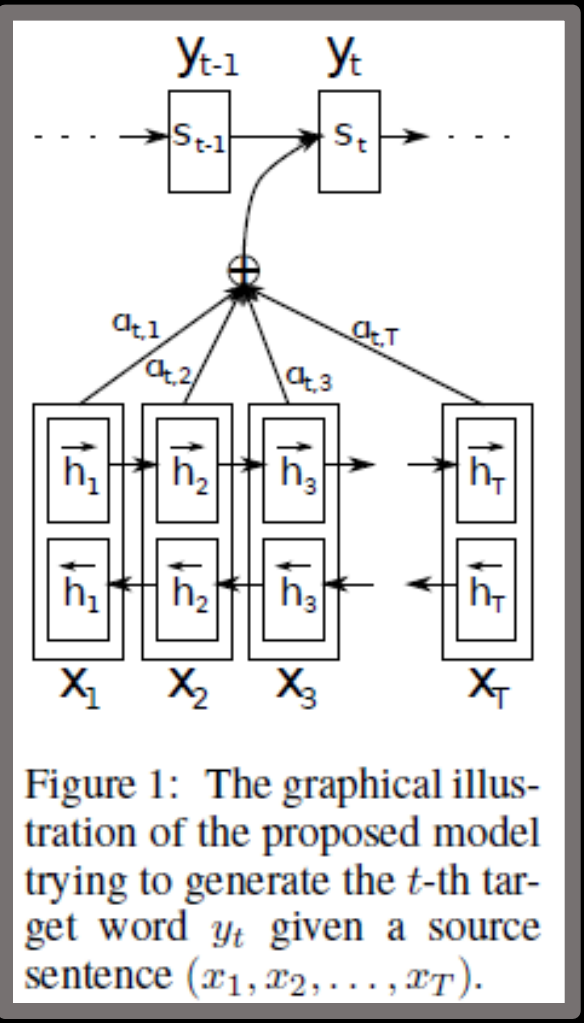
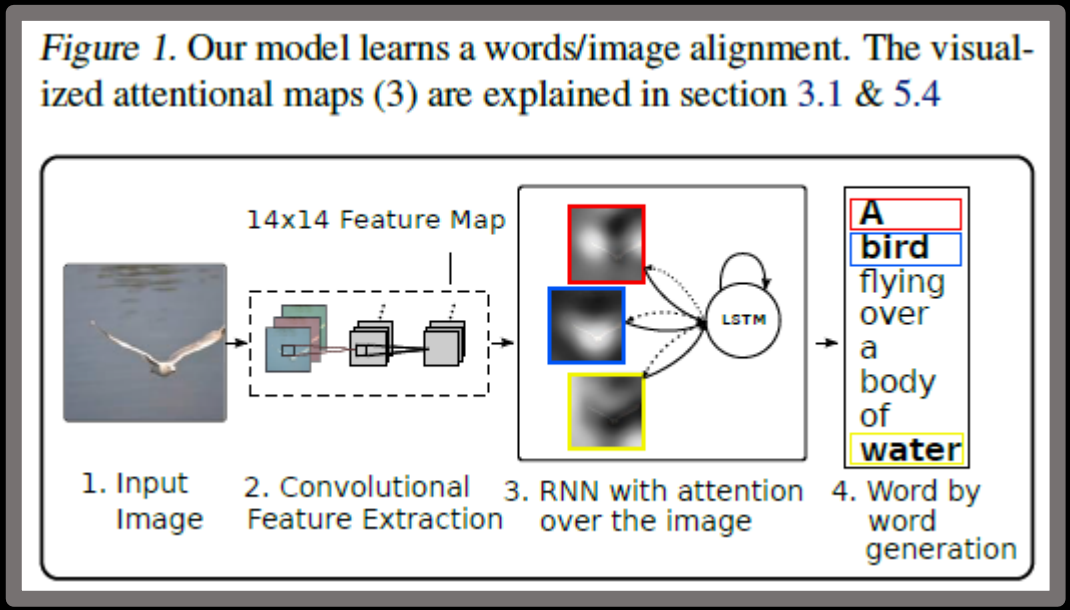
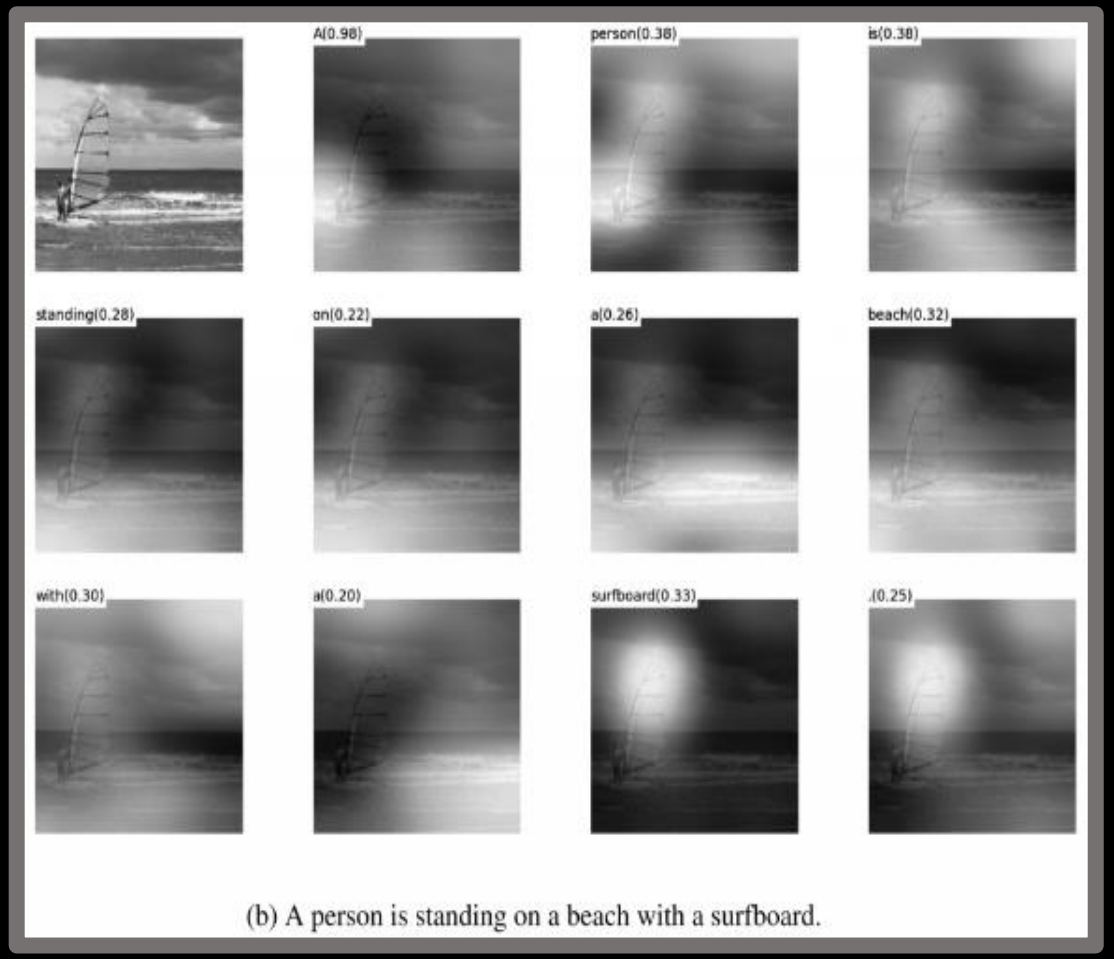


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Attention Mechanism for Image Captioning



- CNN are used to encode an image and RNN is used with attention mechanism to decode this in to an image caption



Accessing Memory: Motivation

- The cell state in LSTMs can be considered to be some form of memory and the model parameters allow us to perform different operations on this memory

- However, the amount of content it can hold is quite limited compared to what we can do with external RAMs
  - E.g. The simple task of reading in a sequence and outputting the same sequence by itself is a difficult task, particularly when the sequence length is long
  - Imagine a situation where the classifier needs to watch a movie and answer questions on it. Without support for a powerful memory such applications are not possible

Case Study: Facebook (Refer: Weston's Paper)

Figure 1: Example "story" statements, questions and answers generated by a simple simulation. Answering the question about the location of the milk requires comprehension of the actions "picked up" and "left". The questions also require comprehension of the time elements of the story, e.g., to answer "where was Joe before the office?".

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

Components of Memory Networks

- I: (input feature map) – converts the input to the internal feature representation.
- G: (generalization) – updates old memories given the new input. We call this generalization as there is an opportunity for the network to compress and generalize its memories at this stage for some intended future use.
- O: (output feature map) – produces a new output (in the feature representation space), given the new input and the current memory state.
- R: (response) – converts the output into the response format desired. For example, a textual response or an action

Memory Network: Flow of the model

- Suppose the input is a character or word or sentence representation. It can be image or audio MFCC vectors based on the application. Let this be x
- Convert x to an internal feature representation I(x)
- Update memories m_i given the new input: $m_i = G(m_i, I(x), m)\ \forall i$
- Compute output features o given the new input and the memory:
$$o = O(I(x), m)$$
- Decode the output features o to get the final response: $r = R(O)$

Memory Networks – components design

The components I, G, O, R can be designed using any machine learning classifier: SVM, Neural Networks, Decision Trees, etc

I component: Component I can make use of standard pre-processing, e.g., parsing, coreference and entity resolution for text inputs. It could also encode the input into an internal feature representation, e.g., convert from text to a sparse or dense feature vector.

G component: The simplest form of G is to store $I(x)$ in a "slot" in the memory:

$$\mathbf{m}_{H(x)} = I(x), \tag{1}$$

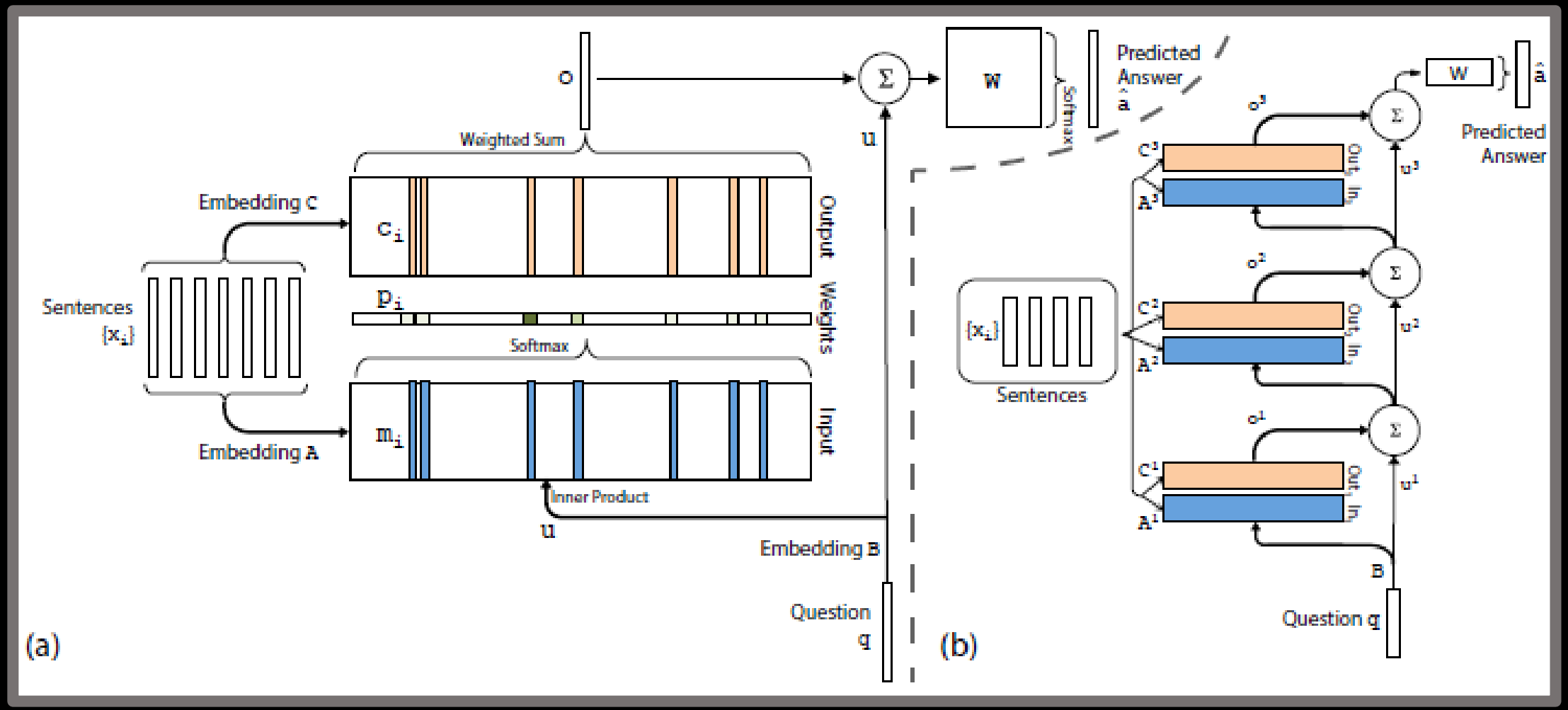
where $H(.)$ is a function selecting the slot. That is, G updates the index $H(x)$ of $\mathbf{m}$, but all other parts of the memory remain untouched. More sophisticated variants of G could go back and update earlier stored memories (potentially, all memories) based on the new evidence from the current input x. If the input is at the character or word level one could group inputs (i.e., by segmenting them into chunks) and store each chunk in a memory slot.

If the memory is huge (e.g., consider all of Freebase or Wikipedia) one needs to organize the memories. This can be achieved with the slot choosing function H just described: for example, it could be designed, or trained, to store memories by entity or topic. Consequently, for efficiency at scale, G (and O) need not operate on all memories: they can operate on only a retrieved subset of candidates (only operating on memories that are on the right topic). We explore a simple variant of this in our experiments.

If the memory becomes full, a procedure for "forgetting" could also be implemented by H as it chooses which memory is replaced, e.g., H could score the utility of each memory, and overwrite the least useful. We have not explored this experimentally yet.

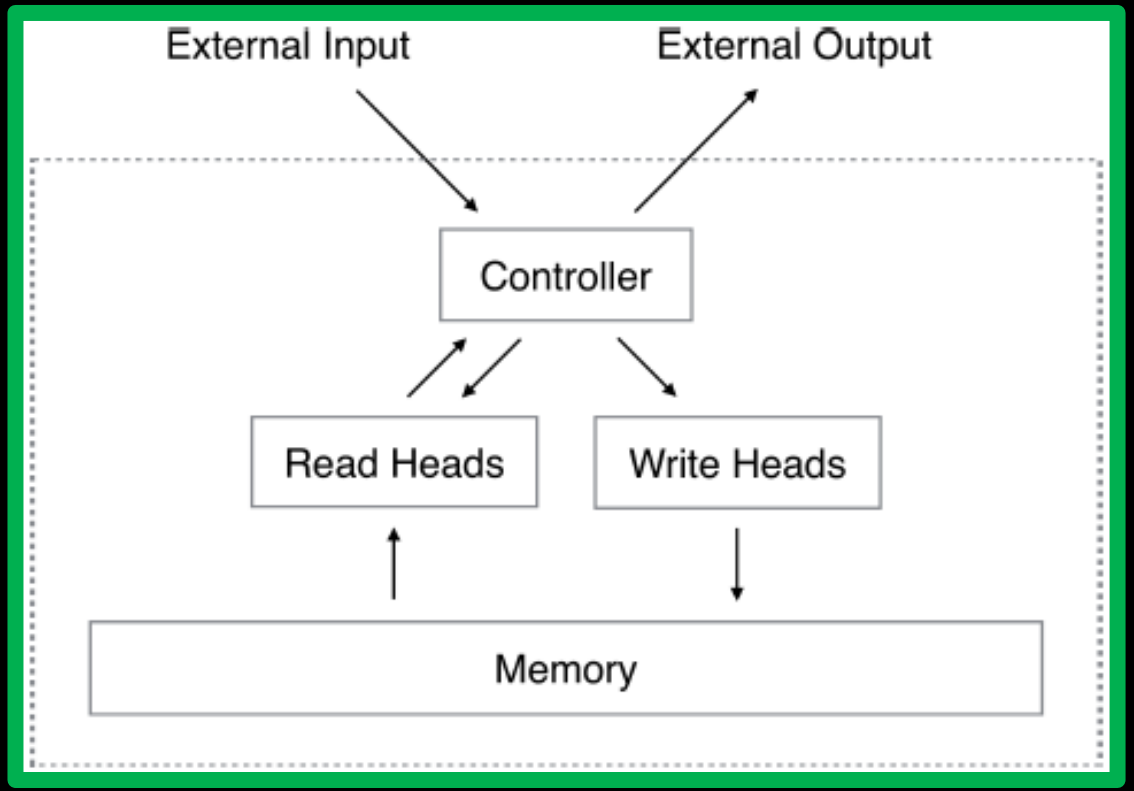
O and R components: The O component is typically responsible for reading from memory and performing inference, e.g., calculating what are the relevant memories to perform a good response. The R component then produces the final response given O. For example in a question answering setup O finds relevant memories, and then R produces the actual wording of the answer, e.g., R could be an RNN that is conditioned on the output of O. Our hypothesis is that without conditioning on such memories, such an RNN will perform poorly.

End To End Memory Networks



Neural Turing Machine (Alex Graves 2014)

- We can understand the working of an NTM as analogous to a Turing Machine
- An NTM has 2 basic building blocks:
  - Controller
  - Memory Bank
- Controller (like a CPU) controls the I/O, performs the computation using the input and memory contents.
- The read/write signals produced by the controller at every time step can be thought of as operating read/write heads of the TM



NTM Basics

- An important property of the NTM is that all its components are differentiable, so that we can train this machine with SGD
- Note that in a Von Neumann computer, the read and write signals operate on a unique memory location. That is, the selection of the memory location is unambiguous.
- In a NTM, the read/write operations are blurry.
- The degree of blurriness is governed by the attention “focus” mechanism
  - constrain each read/write operation to interact with a small portion of memory, ignoring the rest
- The specialized outputs emitted by the heads determine the attentional focus
- These outputs define a normalized weighting over the rows of the memory matrix
- Each weighting, 1 per read or write head, defines the degree to which the head reads or writes to each location. It can attend sharply to a given location or weakly to several

NTM Operations: Read

- Memory is organized as a matrix N x M where N are the number of memory locations and M is the vector size at each location.

- Let M_t be the contents of this matrix at a time t and let w_t be the vector of weightings over the N locations emitted by the read heat at t

- If weightings over all N locations are normalized, we have:

$$\sum_i w_t(i) = 1, 0 \leq w_t(i) \leq 1, \forall i$$

The read vector of dimension M returned is defined as:

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

NTM Operations: Write

- Write operation is modelled as an erase followed by an add on memory locations

- Given the weighting w_t emitted by the write head at time t, along with erase vector e_t whose M elements lie in range (0, 1) the memory vectors M_{t-1} from previous time step are modified as:

$$\widetilde{M_t}(i) \leftarrow M_{t-1}(i)[1 - w_t(i)e_t]$$

$$M_t(i) \leftarrow \widetilde{M_t}(i) + w_t(i)a_t$$

$$where\ a_t\ is\ the\ add\ vector\ produced\ by\ the\ write\ head$$

Addressing Mechanisms

- Content Based Addressing

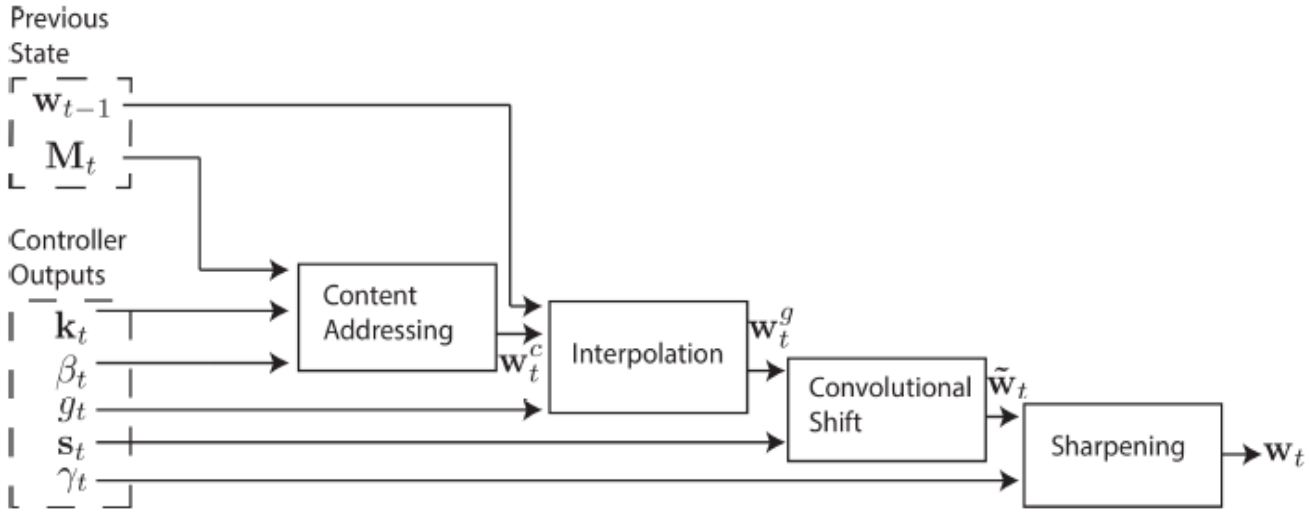


Figure 2: Flow Diagram of the Addressing Mechanism. The *key vector*, $\mathbf{k}_t$, and *key strength*, β_t, are used to perform content-based addressing of the memory matrix, $\mathbf{M}_t$. The resulting content-based weighting is interpolated with the weighting from the previous time step based on the value of the *interpolation gate*, g_t. The *shift weighting*, $\mathbf{s}_t$, determines whether and by how much the weighting is rotated. Finally, depending on γ_t, the weighting is sharpened and used for memory access.

- Location Based Addressing

Focusing By Content

- Content-based addressing focuses attention on locations based on the similarity between their current values and values emitted by the controller.
- The head (read or write, whichever is selected) produces a length M key vector k_t
- This key is compared with $M_t(i)$ using a similarity measure K[., .]
- The content based system produces a normalized weighting w_t^c based on the similarity and a positive key strength β_t which can amplify or attenuate the precision of the focus

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

Focusing By Location: Interpolation by Gated Weighting

- The location-based addressing mechanism is designed to facilitate both simple iteration across the locations of the memory and random-access jumps. It does so by implementing a rotational shift of a weighting

- Prior to rotational shift each head emits a scalar interpolation gate g_t in the range (0, 1)

- Gated Weighting blends the weighting produced at the previous time step with the weighting produced by the content system at the current time step

$$\mathbf{w}_t^g \longleftarrow g_t \mathbf{w}_t^c + (1 - g_t)\mathbf{w}_{t-1}.$$

Focusing By Location: Shift Weighting

- After interpolation, each head emits a shift weighting s_t that defines a normalised distribution over the allowed integer shifts. E.g., if shifts between -1 and 1 are allowed, s_t has three elements corresponding to the degree to which shifts of -1, 0 and 1 are performed.

- The simplest way to define the shift weightings is to use a softmax layer of the appropriate size attached to the controller.

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

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

Example: Copy with NTM

initialise: move head to start location
while input delimiter not seen **do**
    receive input vector
    write input to head location
    increment head location by 1
end while
return head to start location
while true **do**
    read output vector from head location
    emit output
    increment head location by 1
end while