End-To-End Memory Networks

Abstract

* Recurrent attention over a large external memory
* Requires less supervision when training over previously introduced memory networks, this makes it applicable to a wider range of tasks

Introduction

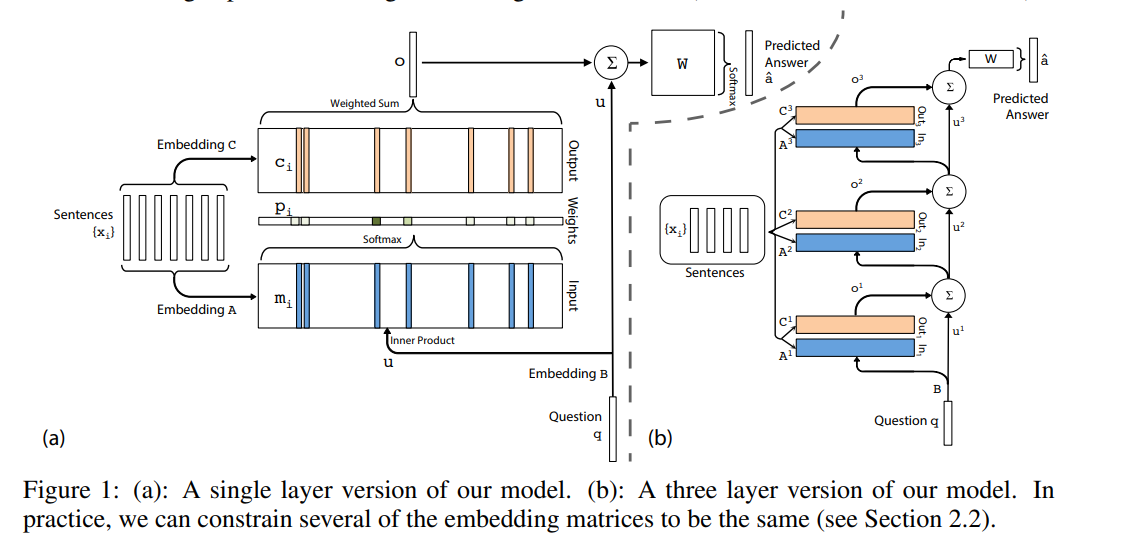
* Introduce an RNN architecture where recurrence reads from a large external memory multiple times before outputting a symbol
* This implementation of a Memory Network can be automated so backpropagation does not require supervision

Approach

* Takes a set of inputs (xi) a query (q), and outputs an answer (a)
* X, q, and a all contain symbols coming from a vocab set V
* Writes x to a memory buffer of a fixed size

Single Layer Model

* Input memory:
  + Suppose we are given the input set x1, x2, …, xn to be stored in memory
  + The entire set x are converted into memory vectors m of dimension d computer by embedding each xi into a continuous space
  + Mi = Axi
  + Q is embedded in a different matrix with the same dimensions obtain an internal state u
  + Then we perform pi = softmax(u­Tmi)
  + P is the probability vector over the inputs
* Output memory
  + Each xi has a corresponding output vector ci which is given by mulitpling x by another embedding matrix C
  + O is the sum of pi\*ci
  + Because we sum everything together, the gradient is smooth
* Final prediction
  + In the single layer case, we just pass the output vector o into a dense layer with weight matrix W and then take the softmax of (W(o+u)) to get our predicted a
  + We learn A, B, C, and W



Task: Question Answering

* Context sentences, asks a question, both are inputs
* Basically, reading comprehension

Implementation

* Xi is a sentence = {x1, x2, …, xn}
* Mi = sum over j (A xij)
* Ci= sum over j (Cxij)
* U = sum over j (Bqj)
* This cannot capture the order of words in a sentence

Positional Encoding

* Mi = sum over j ( lj \* Axij ) where lj is a column vector of lkj of lkj = (1 – j/J) – (k/d)(1-2j/J) assuming we start indexing at 1

Temporal Encoding

* Mi = sum over j (xij + TA(i) where TA(i) is the ith row of a matrix TA that encodes temporal info, I’m assuming we learn these but I don’t think it said in the paper, we have one of these corresponding to the C matrix as well

Take Away

* This method is only 5 years old
* Although recurrence was mentioned a lot in the beginning, it really seems like it just a straight summation of the sequence (with positional and temporal encoding) which would allow for an easy gradient and fast backwards pass
* Outperforms LSTMs which is the current thing I’m doing