MEMORY NETWORKS (NOTES)

Weston, Chopra & Bordes

September 18, 2015

Facebook AI Research

Memory

Machine learning algorithms are like computers, but most do so without memory

- Memory is useful
- Memory in RNN/LSTMs is small and not compartmentalized

Joe went to the garden then Fred picked up the milk; Joe moved to the bathroom and Fred dropped the milk, and then Dan moved to the living_room.

Where is Dan? A: living room I believe

Where is Joe? A: the bathroom

Fred moved to the bedroom and Joe went to the kitchen then Joe took the milk there and Dan journeyed to the bedroom; Joe discarded the milk.

Where is the milk now ? A: the milk is in the kitchen

Where is Dan now? A: I think he is in the bedroom

Joe took the milk there, after that Mike travelled to the office, then Joe went to the living_room, next Dan went back to the kitchen and Joe travelled to the office.

Where is Joe now? A: I think Joe is in the office

Architecture

Input x, memory is an array of objects, $m = \{m_1, \ldots, m_N\}$ and 4 layers

- I: convert raw input into internal representation
- G: update old memories
- O: produce output given input and state of memory
- R: convert output into desired output space (e.g textual response)

Generalization (G)

Update memories

$$\forall i, m_i = G(m_i, I(x), m)$$

e.g Store in empty space. Given current n,

$$m_n = I(x), \qquad n = n + 1$$

This paper doesn't overwrite anything. Other memory paper, *Neural Turing Machines*, has a fancy method of (softly) overwriting everything

Details of I, G, O, R

- I converts sentences to integer representation. e.g The cat said yes $= [10, 1, 106, 3]$
- G stores sentence in next free position
- all the fun is in O, R.

Output implementation

Retrieve k most relevant memories.

$$o_1 = O_1(x, m) = \arg\max_{i=1,\dots,N} s_O(x, m_i) \tag{1}$$

where $s_O(a, b)$ scores the match between a and b.

$$o_2 = O_2(x, m) = \arg\max_{i=1,\dots,N} s_O([x, m_{o_1}], m_i) \tag{2}$$

e.g Where is the milk now?

1. Joe left the milk.
2. Joe travelled to the office.

Response

Could easily do RNN/LSTM text generation. Instead, choose a single word.

$$r = \arg\max_{w \in W} s_R([x, m_{o_1}, m_{o_2}], w) \quad (3)$$

Scoring functions

How to compare similarity/relevance?

$$s(x, y) = \Phi_x(x)^\top U^\top U \Phi_y(y). \tag{4}$$

where U is $n \times D$ (word vectors) and $\Phi()$ is $D \times 1$.

Φ_x and Φ_y maps a sentence to the D-dimensional space (bag of words here). Separate U for each scoring function.

Training

Have labelled data, questions and answers, AND supporting memories/statements. Use margin ranking loss

$$\sum_{\bar{f} \neq m_{o_1}} \max(0, \gamma - s_O(x, m_{o_1}) + s_O(x, \bar{f})) + \tag{5}$$

$$\sum_{\bar{f}' \neq m_{o_2}} \max(0, \gamma - s_O([x, m_{o_1}], m_{o_2}]) + s_O([x, m_{o_1}], \bar{f}'])) + \tag{6}$$

$$\sum_{\bar{r} \neq r} \max(0, \gamma - s_R([x, m_{o_1}, m_{o_2}], r) + s_R([x, \_o_1, m_{o_2}], \bar{r}])) \tag{7}$$

Use negative sampling instead of computing full sum.

If LSTM, RNN is used for response, use log likelihood of sequence x, o_1, o_2, r.

Extensions

- Words as input (learn segmenter)
- Efficient memory (hashing or cluster word embeddings)
- Modelling write time (embed concept of absolute time)
- Unseen words (store left and right context (n grams?) in bag of words)