Reproduction of "Tracking the World State with Recurrent Entity Networks"

Mehrnaz Mofakhami [*]
Department of Mathematical Sciences
Sharif University of Technology
Tehran, Azadi avenue
`m.mofakhami.77@gmail.com`

Amirhossein Yavari [†]
Department of Mathematical Sciences
Sharif University of Technology
Tehran, Azadi avenue
`poria.y@protonmail.com`

Abstract

Here we reproduce the results of Henaff et al., (2017) proposed model (EntNet) to track semantic entities across a textual medium which represent the "state of the world". We measure and report the performance of our implemented EntNet on bAbi data-set and compare our results with those reported in the original paper. Our implementation is available at `https://github.com/mhrnz/entnet_bAbI_reproduction/`.

1 Summary and Quick Review of EntNet

EntNet is a memory-augmented network that learns to track the state of the world through a set of higher representations, referred to as the dynamic memory. It is claimed that each element of this set could be used to represent a single entity or concept. EntNet consists of three main parts, an input encoder, the dynamic memory and an output layer.

The input encoder summarizes an element of the input sequence with a vector of fixed length, the specific architecture used in the paper is a learned sequential multiplicative mask followed by a summation:

$$s_t = \sum_i f_i \odot e_i \tag{1}$$

where e_i is a sequence of word embeddings and f_i are trainable vectors. The authors suggest that the type of the encoder used is not an rudimentary part of the architecture and other architectural choices might as well work, namely recurrent neural networks.

The dynamic memory is a gated recurrent network with a (partially) block structured weight tying scheme. The hidden states of the network are divided into blocks $h_1, ..., h_m$ and the full hidden state isthe concatenation of these blocks. In the original paper m is set between 5 to 20, and each block is a vector of 20 to 100 dimensions. The gating mechanism works as follows: At each time step t, the content of the hidden states $\{h_j\}$ (which we will call the jth memory) are updated using a set of key vectors $\{w_j\}$ and the encoded input s_t. In its most general form, the first time an entity appears, it is written to a memory slot. Every time that an event occurs in the story that changes the state of an entity, the change in the state of that entity is mixed with the entity's previous state using a varient of GRU update equation and then rewritten to the same slot. To make the job of model in identifying meaningful entities easier, it is suggested that the keys be initialized with vectors that are near to the embedding of words representing important objects or concepts.

[*]Contributions include implementation of EntNet code and writing this extended abstract.

[†]Contributions include ideas regarding implementation and technical details, debugging and writing this extended abstract.

Table 1: Comparison of Results

Task Number	1	2	3	4	5	6	7
Henaff et al., (2017)	**0**	**3**	**9.6**	33.8	**1.7**	**0**	**0.5**
Our Implementation	**0**	19.6	79.6	**0.3**	15.3	15.2	3.5

Output layer Whenever the model is required to produce an output, it is presented with a query vector q, which for example in the question answering setting is a representation of the question. The representation is probably encoded by the same encoder that is used to update internal states to ensure the internal consistency of representations. The method of the querying is to use a form of standard single-head attention to read relevant information from the stored entity slots based on the query. Then the result of the query is passed through a classifier and assessed.

2 Reproduction Settings and Implementation Choices

We preserved training detail mentioned in the paper in our reproduction. Models were trained with ADAM optimizer for at most 200 epochs with mini-batch size equal to 32. Learning rate was set initially to 0.01, but was divided by 2 every 25 epochs. The maximum number of hidden entities, m, was set to 20, and the dimension of each of them was set to 100. All model weights were initially drawn from a Truncated Normal distribution with mean 0 and standard deviation 0.1. We also used gradient clipping with clip value of at most 1. For all tasks except for task 3, the capacity of the memory was limited to the most recent 70 sentences, and for task 3 it was limited to the most recent 130 sentences. Moreover, our input encoder follows equation 1.

The paper have not mentioned how to assign key values specifically. Therefore, we chose to tie key vectors to entity embeddings and our method is as follows. For each paragraph, we derived words with specific tags (e.g. nsubj and pobj) and chose at most 20 of them to act as keys. In this way, keys are updated alongside hidden states during the training, and are always tied to specific words of the paragraph.

3 Results

Validation error rates on bAbI tasks where key vectors are tied with entity embeddings are reported in Table 1. Since we used keys tied to entities, we compare our results with Table 5 of the paper.

4 Difficulties

The most important challenge we faced was regards to the key vectors, as the paper have not discussed it in detail. Henaff et al., (2017) mentioned that tying keys did not significantly change performance, and even hurt it in some cases, but did not mention the baseline it is comparing this method to. Our observation was that tying keys in the way we described in section 2 performed much better than randomized fixed keys, which is intuitive, since important entities in each paragraph are being tracked.

References

[1] Mikael Henaff. & Jason Weston. & Arthur Szlam & Antoine Bordes & Yann LeCun (1995) Tracking the World State with Recurrent Entity Networks., *ICLR 2017*,