Multi-hop Reading Comprehension via Deep Reinforcement Learning based Document Traversal

Valerio Lorenti Luca Maurici

July 2021

1 Introduction

This project was centred around the topic of Reading Comprehension (RC) in the context of Question Answering (QA); or rather accurately identifying the relevant text from several context documents to answer a given question. More specifically, this work concerned the reference paper "Multi-hop Reading Comprehension via Deep Reinforcement Learning based Document Traversal" by Long *et al.* (Long, 2019). From said paper, it may be deduced that RC is a sequential process, particularly in the case of multi-hop. Therefore, to reach an answer, a series of independent stages are required. Firstly, starting from a collection of documents, graphs of sentences are constructed. The second stage involves an extractor which, through the graph traversal, identifies the appropriate section of knowledge among the various documents. Finally, the output from the extractor is computed by a reader. More specifically it is the Reinforced Mnemonic Reader, which had been shown to perform well compared to previous attentive readers, as seen by Hu *et al.* (Hu, 2018).
As an alternative to PPO optimisation, two other possible novel approaches are proposed: Shortest Path Policy Optimisation (ShPaPO) and Breath Visit Method.
The implementation is described below in terms of said three stages, the report then moves on to outline the research and experiments carried out along with the respective results.

2 Implementation

2.1 Linker

Sentence level graphs have been constructed to facilitate the extraction of the relevant sentences, within the various documents, to answer the given question.

Algorithm 1: Coherency Graph Construction

```
1  Input: Context documents $D$
2  Output: Coherency graph $G$
3  Initialise empty coreference graph, $R$
4  for $d \in D$ do
5      Initialise empty document coreference graph, $R_d$
6      for $s_i$ and $s_j \in d$ do
7          if coreferent ($s_i$ , $s_j$) then
8              if $i < j$ then
9                  $R_d \leftarrow$ add edge($s_i \rightarrow s_j$ )
10     $R_d \leftarrow$ add edge($s_i \rightarrow s_{i+1}$)
11 $R_d \leftarrow \cup_d R_d$
12 Initialise empty entity link graph, $L$
13 for $s_i$ and $s_j \in$ all sentences in $C = \cup_i D_i$ do
14     E $\leftarrow$ get named entities($s_i$)
15     for $e \in E$ do
16         if $e$ in $s_j$ then
```

```
17          $L \leftarrow$ add edge($s_i \rightarrow$ root node$_R(s_j)$)
18    Return $G=R \cup L$
```

The following explanation references the scheme above (Algorithm 1).
There are the two phases to the construction of a graph.

- **Coreference resolution**: For each document, every pair of phrases is sampled, it is then checked whether these are coreferent (Hobbs, 1979). If they are, two nodes are connected through an oriented *coreferential arc*. In any case a node will always be connected to its sub-sequent sentence through a *sequential arc* $i \rightarrow i+1$.
- **Entity linking**: Each pair of phrases in all documents are analysed. If they share at least one entity (Core NLP share entities), an oriented arc is added from each node to the root node of the other one. The root node represents the start of the document.

The nature of the Coherency graph structure preserves the logical flow. As mentioned above, this property is achieved by connecting all consecutive sentences, coreferent ones, and sentences sharing one or more entities (hereon referred to as entity sentences). Another key aspect of the Coherency graph structure is the question node. Connecting the question and graph via entity linking, most likely allows the navigation to begin in the correct context.

Table 1: Ablation study to test coherency graph effectiveness. Wikihop Accuracy results for various Linker options (Long, 2019).

Linker	Extractor	Reader	Wikihop Accuracy
Coherency Graph	Convolutional Policy	Mnemonic Reader	65.12
Fully Connected Graph	Convolutional Policy	Mnemonic Reader	33.45
Coherency without Coreference	Convolutional Policy	Mnemonic Reader	54.18
Coherency without Entity Linking	Convolutional Policy	Mnemonic Reader	3.23

In fully connected graphs every node is connected to all the others. In this way, the property of logical flow is lost, and the density is at its maximum. This leads to an explosion in the number of possible actions to be chosen by the extractor at each navigation step.
Coherency graphs without coreference resolution maintain arcs between entity sentences and between consecutive sentences; arcs between coreferent sentences are excluded. This results in a slight drop in useful connectivity, mitigated by the presence of entity linking.
Coherency graphs without entity linking maintain arcs between coreferent sentences belonging to the same document, and between consecutive sentences; arcs between entity sentences are excluded. As a consequence, a disconnected graph is obtained with different connected components. In this way, starting from the question node, most of the other nodes are unreachable, leading to a small probability of reaching the answer.
Amongst these options the basic Coherency graph was chosen, as according to the reference paper it led to the most satisfactory results (Table 1). The notation used to represent a sentence in a graph is $“d” + doc\ number + ”s” + sentence\ number$ (e.g., d2s3).

2.2 Extractor

The role of the extractor is to identify the suitable section of text amongst the documents. This is done through a traversal of the graph built during the first stage. Starting from the question node, at each traversal step, the extractor chooses the next node (the next sentence to extract) among the adjacent ones. This choice is based on the current state: the question q, *l* options (the nodes that may be visited), *n* accepted (the nodes that have already been visited).

The reference work displays the various types of policies employed, Convolutional Policy, Dense Policy, and Random Walk. The Convolutional Policy was used for best performance, and Random Walk was instead used to check that the graphs and reader were functioning correctly (namely in the case on Mnemonic Reader). Random Walk Policy samples actions from a uniform action distribution.
Dense Policy is described by Long *et al.* (Long, 2019) as "we flatten the state, and apply two dense layers of 64 tanh units.". Convolutional Policy may be outlined as follows. The input is the current state, composed as mentioned above. The state is initially in text format and is then encoded; or rather an index is assigned to each word of each sentence. This process is done according to the dictionary created based on the training set. Then, the embedding layer transforms these indexes in tensors with embedding dimensions of 50. The filters width chosen is then the same size as the embedding dimension, and the height will vary depending on the word groups chosen (1, 2, 5, 10, 20). A 2D Convolution will then be performed amongst these filters and the embedded state. The output from the 2D Convolution will be features maps with dimensions such that: the width is 1 since the filters and embedding dimensions are equivalent, the height will vary depending on the filters' height, the number of them will be $5 \cdot (l + n + q)$.
Moreover, a max over time pooling is carried out before moving on to the dense layers. In this final part of the net there is a dense layer, a dropout (0.5), and a final dense layer followed by a softmax, which will output the actions' probability distribution. ReLU activation function is placed after the convolutional and the first dense layer.
Formally, the Convolution-based Policy Network, is described by Long *et al.* (Long, 2019) as "our policy π selects an option sentence o provided the current state, x_t. Given $\pi(x_t) = o$, we define feature matrices of size $c_i \in R^{g\times 50}$, for $g = 1, 2, 5, 10, 20$. Let $f_i^{(k)}$ indicate the i^{th} feature of channel k, and r a vector of Bernoulli random variables with $P(r_j = 1) = 0.5$.".

Model 1: Definition of the Convolution-based Policy Network

$$f_i^k = relu[c_i^k \cdot x_{i:i+g}^k + b_i^k \ \boldsymbol{for} \ i = 0,1,\dots,l$$
$$\hat{f}^{(k)} = max(f_0^k, f_1^k, \dots, f_l^k)$$
$$d = [\hat{f}^{(0)}, \hat{f}^{(1)}, \dots, \hat{f}^{(n+l+1)}]$$
$$o = relu[w_{(dense)} \cdot (d \circ r) + z)]$$

Training was carried out using PPO as the RL algorithm (Schulman, 2017), with $\gamma = 0.95$,
Adam as optimiser, dynamic step size and momentum, 30 steps per iteration and 2500 timesteps per batch. The PPO implementation has been adapted from Philtabor (Philtabor, 2020). In order to train the models two phases were carried out: graphs generation and PPO optimisation. Sampling by the Wikihop dataset, 3500 graphs were generated, saved, and filtered by the following process. Firstly, identification of sentences that are most likely to contain the answer. Subsequently, all graphs that do not have a unique estimated answer's position, are discarded.
The identification algorithm we designed consists of checking whether all words in the answer are also present in the considered sentence, for all sentences in each document.

2.3 Reinforced Mnemonic Reader

The final stage in the QA process is to employ a reader to get the predicted answer. The Reinforced Mnemonic Reader (Hu, 2018) was chosen as it outperforms the other two readers considered in the reference paper: R-Net (Wang, 2017) and Document Reader (Chen D., 2017). The two above mentioned stages are intended to select the most useful information to help the reader in its work. Thus, the input given to the reader is the question and the summary produced by the extractor. This input is more easily read, yet has a probability value of < 1 of containing the correct answer.

The reader used, was trained for 2 epochs on Wikihop dataset converted in SQuAD-style format through the concatenation of supporting documents into a single context.

3 Research

3.1 Graph connectivity

In order to ensure that the generated graphs were constructed correctly, some experiments were carried out to produce metrics.

Table 2: Metrics about graphs. Original is referred to the Long *et al.* (Long, 2019). Post-changes are referred to the changes described below.

Linker	Avg number hops	% Reachable answer	Mean Degree	Degree variance	% Reachable answer > 10 hops
Original	3.215	88.8%	5.385	5.718	0
Post-changes	3.129	100%	5.508	5.438	1.480

Starting from the question about 12% of the nodes are not reachable (Table 2), which implies a decrease in the likelihood of the answer being reachable. In fact, observing the plots of some graphs, it was noticed that in some cases the graphs were made up of several connected components. This being said, to increase the quality and performance of the graphs, an improvement was made to the graph construction algorithm. Since each connected component is an entire document, an effective and simple solution was to connect all the final sentences of each document with the first sentence of the next one. E.g., d0s4 $->$ d1s0. In this way, all graphs are connected and consist of only one large, connected component (100% reachable nodes). The Random Walk performance improved from 21.53% to 28.66%.

3.2 Shortest Path Policy Optimisation (ShPaPO)

The idea was to mix RL and Supervised Learning by adding a label (value information) to the nodes of the graph and use them to train a neural network capable of estimating the advantage of being at a certain node in a supervised way.

The concept behind the estimated advantage is as follows. The agent needs to be trained to minimise the path between the question and the answer, resulting in two advantages. Firstly, increasing the likelihood of finding the answer. Secondly, the input to the reader is of reduced size.

The same result would have been obtained by reshaping the reward function, assigning a penalty even in the case of a normal and "legal" step (E.g., -0.05). In this way, the agent would find it convenient to cross the graph in the shortest way. Anyways, this other approach was attempted as follows. The value of being at a certain node may be assigned a priori, instead of using a trained critic network, with the simple algorithm below.

1. Find the shortest path length l between the question and the answer node.
2. Mark all nodes with 0.
3. Find all paths of length $l + c$ between the question and the answer node.
4. Mark with a value < 1 and > 0 all nodes belonging to these paths.
5. $c = c - 1$
6. Repeat from point 3 to point 5 until $c = 0$ increasing the value assigned to nodes.
7. Mark with a value of 1 the nodes belonging to the shortest paths.

At this stage, a network is used simultaneously to traverse graphs with the aim of collecting information about state-value pairs; as well as to be trained with MSE as loss function and Adam as optimiser. Some architectures have been tried; the results are displayed in the experiments section.

3.3 Breadth Visit Method

The method proposed in this paragraph is based on the observation (Table 2) that on average the answer is located at distance 3 from the question node, and only about 1.5% of the answer nodes are more than 10 hops away. This property is due to the nature of the graphs, which, as mentioned above, are constructed in such a way that starting the traversal from the question node, the nodes visited are likely to be in the context of the answer. With these premises, it is possible to infer that a good strategy to find the answer node quite reliably is to explore the graph in breath and collect all sentences at distance < d. This is what the proposed method consists of. Finally, the result of this simple procedure is given to the reader and processed to obtain the expected answer. This simple method has the advantage of finding the answer node with more probability than the reference RL approaches but has the disadvantage of producing a very long summary that might be more difficult to process by the reader. The method proposed in this paragraph was the one that gave the best results (37.48%) and the only one that gave significantly better results than Random Walk (28.66%) without candidate information. Using them, Breath Visit Method obtains an EM-score of 53.86% while Random Walk 45.98%.

4 Experiments and results

Convolutional policies, trained with PPO and ShPaPo respectively, behave strangely in the way that they frequently converge to an output independently from the input, and that respects a common pattern. The probability that an action is chosen decreases as the action index increases. The model learns that it is convenient not to decide, always behaving in an "average" way. Some possible causes of this behaviour include implementation errors, too little network parameters, lack of real patterns in the dataset, and/or poor selection of features to be used. A milder form of this behaviour has also been observed, even after many epochs of dense policy training.

Table 3: EM-scores of different approaches with and without candidate information.

Approach	EM-score no candidates (%)	EM-score with candidates (%)
Random walk (baseline)	28.66	45.98
PPO, Convolutional network	29.92	44.09
PPO, Dense network	30.08	44.57
ShPaPO, Convolutional network	31.50	43.46
ShPaPO, Dense network	29.13	43.78
Breadth Visit Method	37.80	53.86

It can be observed that using trained models there is a slight improvement in performance without using candidate information compared to Random Walk but this can be considered negligible, also considering the slight drop in EM-score compared to random walk with candidate information. It is clear that something has gone wrong. In contrast, using a simpler approach, which does not involve a training process, the results obtained are significantly better than Random Walk both with candidate information and without.

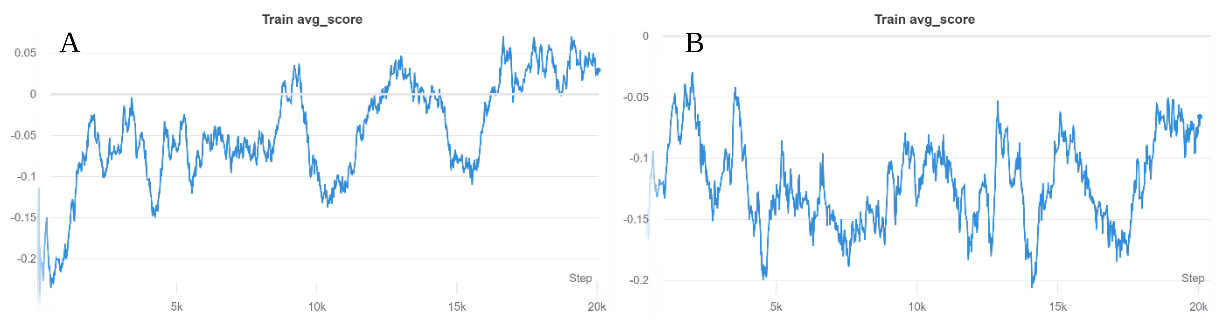

Figure 1: Moving average (500 steps) of train score for Convolutional Policy (A) and Dense Policy (B) both trained with PPO.

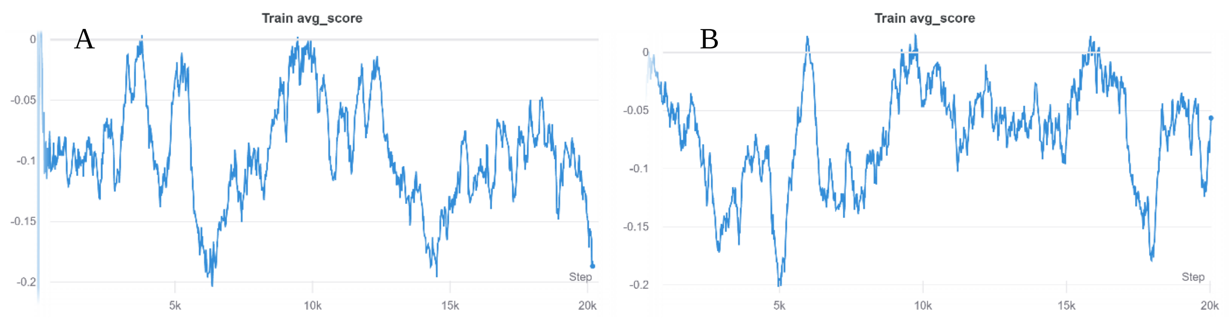

Figure 2: Moving average (500 steps) of train score for Convolutional Policy (A) and Dense Policy (B) both trained with ShPaPo.

Table 4: EM scores by varying the visit depth by applying the Breath Visit Method.

Depth of visit	EM-score no candidates (%)	EM-score with candidates (%)
4	33.23	48.82
5	33.70	50.39
6	34.17	51.65
7	36.85	51.81
8	34.80	51.02
9	36.38	52.76
10	37.48	53.86
11	34.02	51.97
12	37.80	51.81
13	36.54	53.59
14	36.22	51.63

For the Breath Visit Method, in order to choose the best performing depth visit, an experiment was conducted. It turns out that visiting the graph at depths of 10 and 12 gives the best results with and without candidate information respectively. However, in general, since increasing the depth of the visit implies that answer nodes already included in the summaries continue to be included, and considering that only 1.48% of answer nodes are reachable with more than 10 hops, the EM-score remains around 36.5% and 52.5% for candidate and non-candidate cases respectively, when the visit depth $\geq$ 10.

5 Future developments

The Breadth Visit Method could be improved by introducing an estimate of the acceptability of the answer prediction. The Mnemonic Reader provides the prediction of the response with an associated confidence value. This can be exploited to start with a superficial visit, so that, if the response is contained in the visited nodes, the reader's prediction is more reliable. Then, if the answer is judged to be unreliable, the graph visit can proceed deeper. This process can be performed iteratively until the response is reliable or a depth threshold is reached.

References

Chen D., F. A. (2017). Reading Wikipedia to Answer Open-Domain Questions. 1-10.

Hobbs, J. R. (1979). Coherence and coreference. *Cognitive science, 3(1)*, 67-90.

Hu, P. H. (2018). Reinforced Mnemonic Reader for Machine Reading Comprehension. 1-8.

Long, M. B. (2019). Multi-hop Reading Comprehension via Deep Reinforcement Learning based Document Traversal. 1-8.

Philtabor. (2020, Dicembre 24). Retrieved from https://github.com/philtabor/Youtube-Code-Repository/tree/master/ReinforcementLearning/PolicyGradient/PPO/torch

Schulman, J. (2017). Proximal Policy Optimization Algorithms. *OpenAI*, 1-12.

Wang. (2017). R-NET: MACHINE READING COMPREHENSION WITH SELF-MATCHING NETWORKS. Retrieved from https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf