Technical Report for KG2 paper

In this report, I mainly describe some choices I have made in implementation of KG2 paper. This report is written is since the original paper is intuitive only instead of illustrating the technical details.

Generating Hypothesis

This step is to combine the question stem and the choices items. In our code, since in the ARC dataset, there are various kinds of questions, we first classified each question, and according to its type, we chose to attach the item to the end directly or substitute the wh-word with the choice items. Here, we defined the last sentence as the sentence ending with '.' or '?' or just '' . Of course, if there are more than one sentence in one question stem, we used the last sentence of that paragraph. If we cannot find the last sentence and the last symbol is not '.' or '?', which means the last sentence is incomplete at the end, such as "This is an example of", we attached the choice items to the end of it. Like stated in the original paper, we defined the question word as one of the following "___|identify|what|who|where|when|whose|whom|why|how|which of these|which of those|which of the following|which". If any word of those is found in the last sentence, we substitute the question word with the choice item as the hypothesis.

Searching Potential Supports

In this part, we first filtered the noisy sentences in the ARC-Corpus using the specific rules stated here. Then used ElasticSearch to pick up the top-20 relevant sentences from the **cleaned** ARC-Corpus for each hypothesis as supports.

Constructing Knowledge Graphs

Since OpenIE we used cannot iterate each python dictionary structure to get each hypothesis or support, we first store all **lemmatized** sentences in a file for hypothesis and supports, and differentiate each choice and question using different special symbols. Then after using OpenIE to extract the relation triples for **each** sentence, we cleaned and lowerize all pieces in a sentence and construct the knowledge graph for each **hypothesis** or **support**. Since there are several candidate triples for each sentence, one with the highest score will be utilized. There are two special cases: 1) OpenIE can tell the difference between what is context of a sentence and what is not, since the core for a sentence is not laid on context, and OpenIE cannot generate relation triple for the context part, we just ignored it and only get triples for the core part. 2) all candidates have the same score assigned by OpenIE, here is an "visiting" example to illustrate the improvement of OpenIE 5 than 4. In this case, since all the subject and predicate for all candidate are the same, we just thought there is only one sentence with more than one objects. It may be the reason that the original paper denotes the triple for each sentence by $T(s, p, o_i)$. According to the Appendix A of the paper, we added directed edges all from "predicate" node, to "subject", "time", "location", and each "object" nodes (Although I don't think it reasonable for GNN to process it since finally we need to rank the graph using all predicate node features, but there is no information from other nodes except for the predicate node, so predicate node features cannot be a good representative for the whole graph). Networkx is used to construct the knowledge graph for each hypothesis and support because python string can be directly stored in each node and edge. **Please Note** 1) aggregate the node with same string, 2) assign different edge type "sub", "pred", "time", "loc" and "obj" to each corresponding edge, 3) since sometimes OpenIE cannot parse the sentence, thus cannot generate the triple. In this case, we assign a special "Empty" node to stand for the whole sentence.

Learning with Graph Embedding

Actually, the original paper gave the intuitive idea this part. For the whole system, we used two network, LSTM as the feature extractor for each node to get the representation of fixed length, GNN to get the representation of the whole graph. To convert the each word into the dense representation, we used simplest 50d GloVe vector trained on Wikipedia 2014 + Gigaword 5. Then feed those graphs into a vanilla GCN model to aggregation neighbor information for each node. The reduced function is $h_v = W_1\sigma(W_2(\sum_{u\in\mathcal{N}(v)} h_u + h_v))$ which is implemented via two linear layers, where h_v the node representation, and initialized by the sentence feature obtained by LSTM. After T round message passing, we used the stable predicate node representation to rank the graphs. For each hypothesis, we found the max inner product of all hypothesis predict nodes and the corresponding support predict nodes. Then used equation 2 of the paper to get the graph score. For the loss function, since this problem can be categorized as a classification problem, we used NLLLoss. For the batch size, we assumed one sample is a ARC-question, including all the hypothesis and support graphs for one question, while the batch size for LSTM is all nodes of each graph, and the batch size for GNN is composed of all hypothesis and supports graphs.