Project: Deep Learning-Based Symbolic Processing in Prolog

Peiyu Liang
Email: tuh38526@temple.edu

Hanzi Xu
Email: tun47067@temple.edu

Mina Gabriel
Email: tun17109@temple.edu

Index Terms

DeepRank,logic-based dialog engine, query-driven dialog engine, graph-based NLP, dependency graphs,Prolog

I. Introduction

Natural language processing tasks have successfully used logic programming languages for inference and planning, the motivation from this project is to narrow the gap between neural and symbolic interaction.

Our work is an extended version of the recent research by DeepRank [1], this system is a combination of Symbolic Artificial Intelligence and Natural Language Processing that already emerged from Deep Neural Networks, the system uses python for text graph implementation that is used for ranking nodes and edges of the parsed text, python is also used in user interaction, i.e. loading documents and questions into the system, on the backend the system is connected as python client to the Stanford CoreNLP Java server [3] to use it for low-level text graph initialization, which results in a dependency graph with directed edges represents the relationship between the part-of-speech (POS) of a sentence.

The relationship of the graph with other keyphrases are then used to enact the logical rules of the documents to be later processed by the reasoning engine.

II. Overview of the System Architecture

Figure[1] summarizes the system architecture, the system expects a collection of one or more documents where every document is a collection of sentences start at a new line and ends with a period, some documents require some text preprocessing for sentence preparation and removing unwanted notations like HTML tags in case of Web scraping.

The Stanford CoreNLP server will receive the documents and the user query to initialize the text dependencies, after figuring the node ranking of the documents and the query using Personalized PageRank a set of rules and facts will be generated for the dialog engine which is responsible for handling the user query.

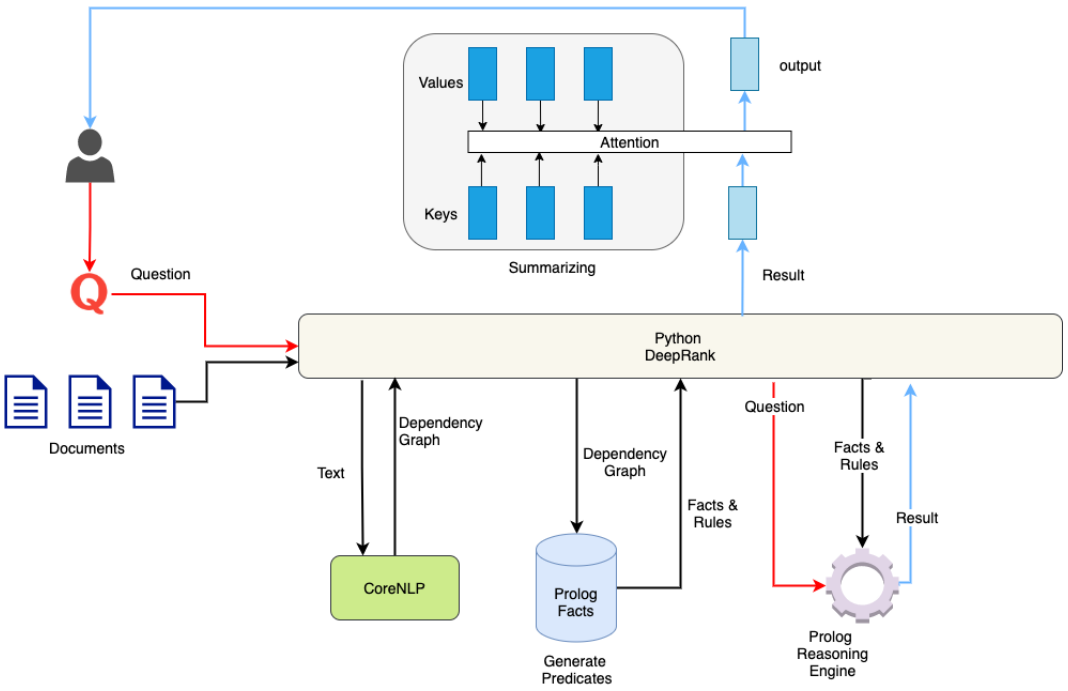

Figure 1. Neural to Symbolic NLP system architecture

Finally, if the system fails to determine good reasoning about the user query, the default behavior of the DeepRank system is to return the best matching sentence based on the highest graph rank previously defined using Personalized PageRank (PPR), we put some effort into summarizing the sentence before reply back to the end-user to give a better answer, for this last step we used Long Short Term Memory (LSTM) [2], the training process depends on the context of the document, thus the chance that the returned summary matches the user query answer is high, for implementation and testing we used the Attention Keras library [4], our implementation and testing result can be found in this google colab link

III. The Graph-Based Natural Language Processing

The first step in this stage is ranking the text graph, Nodes of the graph - as previously mentioned- are generated from the POS and the edges results from the dependencies representation of the grammatical relationship between the keywords.

A. *PageRank*

The **PageRank (PR)** algorithm used by Google Search ranks web pages on their search engine results. PageRank is a measure of the importance of a website, thus manages the priority of these pages when user search, PR works by counting the number and quality of links, the main idea is that the more important websites are more likely to receive external links than the others, the main objectives of this section is as follows:

- Introduce matrix representation for network graph.
- Define transition probability matrix and paths on graph.
- Illustrate PageRank algorithm concepts and it's relation to **TextRank** [5]

For example, Figure[3] represents a directed graph of a series of web-pages and the links from each page to another, an adjacency matrix $\tilde{A}$ of this graph and transition probability matrix A, and they can be represented as follows:

$$\tilde{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1/2 & 0 & 0 & 1/2 \\ 1/4 & 0 & 0 & 0 & 0 \\ 3/4 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 7/8 & 0 & 1/2 \\ 0 & 0 & 1/8 & 1 & 0 \end{bmatrix}$$

In both matrices column represents *from* a web-page - and the probability of columns should sum to 1- and rows represents *to* another web-page, for example if user start at node 1 there the probability that they go to node 2 is $P(2) = 0.25$ and the probability to go to node 3 is $P(3) = 0.75$

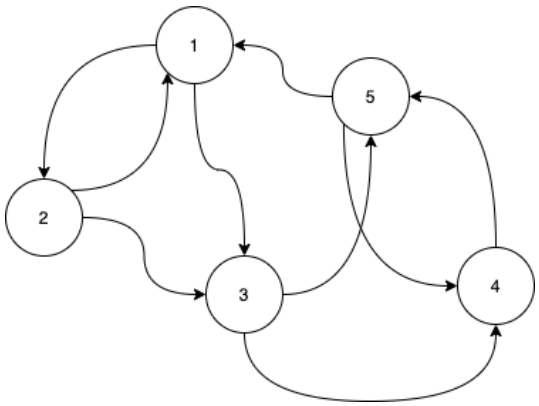

Figure 2. Example of PageRank graph

The following formula can be used to assign weights for web pages after we construct the whole graph.

$$S\left(V_i\right) = (1-d) + d * \sum_{j \in \mathrm{In}(v_i)} \frac{1}{\left|Out\left(V_j\right)\right|} S\left(V_j\right)$$

- $S\left(V_i\right)$ is the weight equation for node i
- d is the demanding factor, in case, of no outgoing links, Various studies have tested different damping factors, but in most cases it is generally assumed that the damping factor will be set around 0.85 [6]
- $In(v_i)$ in bound links of i
- $In(v_j)$ outgoing links of j
- $\left|Out\left(V_j\right)\right|$ the number of the outbound links

B. Nodes Redirection

Redirecting the graph nodes to nouns is one of the main advantages of this process, This can be explained by quoting the original paper, which will be followed by a simple example to illustrate the idea.

"As key phrases are centered around nouns and good summary sentences are likely to talk about important concepts, we will need to reverse some links in the dependency graph provided by the parser, to prioritize nouns and deprioritize verbs, especially auxiliary and modal ones." [1]

We outlined the following steps to summarize the changes.

1) Redirect edges to nouns.
2) Add "about" edge from the sentence node to changed nouns.
3) Verb are recommended as predicate function in graph.

To illustrate the idea, we considered the following simple sentence for ranking:

The cat sits on the mat.

the dependency graph following the application of the previous steps gives the following graph formation:

The rank of the keyword cat can be mathematically computed as:

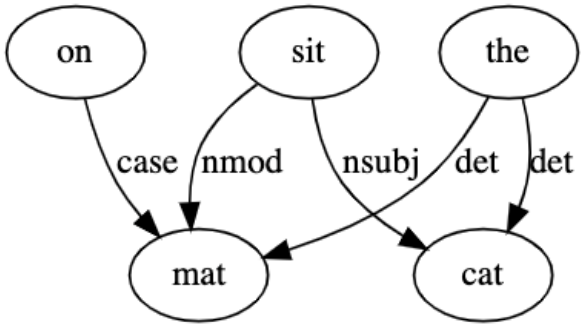

Figure 3. The cat sits on the mat.

$$\begin{aligned}
\operatorname{In}(v_{cat}) &= \{sit, the\}, j \in \{sit, the\} \\
\sum_{j \in \{sit, the\}} \frac{1}{|\operatorname{Out}(V_j)|} S(V_j) &= \frac{1}{|\operatorname{Out}(V_{sit})|} S(V_{sit}) + \frac{1}{|\operatorname{Out}(V_{the})|} S(V_{the}) \\
&= \frac{1}{|\{mat, cat\}|} S(V_a) + \frac{1}{|\{mat, cat\}|} S(V_b) \\
&= \frac{1}{2} S(V_{sit}) + \frac{1}{2} S(V_{the})
\end{aligned}$$

applying the pageRank algorithm until convergence will result in the following rank dictinary:

{0: 0.3748812025081476, 'cat': 0.269746898745926, 'mat': 0.12104622137883939, 'on': 0.025, 'sit': 0.1843256773670866, 'the': 0.025}

IV. Prolog Facts Generation Process

Nowadays, people have been used to use word embeddings to represent text when dealing with NLP problems. However, word embedding is not the only way for text representation. The embedding mechanism might be helpful for semantic analysis but does not contribute to generating logic relations. DeepRank, instead, chooses to use a dependency graph as text representation. In this section, we will briefly explore the details of DeepRank's dependency parsing process and introduce the significant improvement we made on the Prolog facts generated from the graph.

Given a context, we need to extract the linguistic features and generate relations that can represent the context and are also readable by Prolog, so that later we can feed these relations into Prolog as Prolog facts. Our focus is the Prolog facts generation process.

To keep the relations simple and portable to Prolog, DeepRank generates the following facts and rules in the form of Prolog-readable code. These predicates are coming from linguistic knowledge

of words in the sentence, and they are mainly based on two linguistic: part of speech(POS) tag and syntactic dependency tag. A part of speech is a particular grammatical class of words, for example, noun, ‘adjective, or verb. And a syntactic dependency tag is a relation between two words in a sentence with one word being the governor and the other being the dependent.

These tags all Followed Universal Dependencies (UD), which is a framework for reliable annotation of linguistic features across different human languages. However, later we will see that even using the same rules, different NLP APIs have their understanding of sentences.

DeepRank uses Stanford CoreNLP API to provide POS and syntactic dependency tags. We decide to first use another API, Spacy, to do the job for the same job and compare the performance between them. More importantly, we write our prolog fact generation program, and facts generated by our system are richer than what DeepRank has. We will go over these facts one by one and explain our improvement.

Given this sample context:

Socrates did not write any books.

Plato is a student of Socrates.

Plato wrote books and put words into Socrates mouth.

*A. **Keyphrases***

If the POS tag of a word is ’NOUN’ or a proper noun, this word will be kept as an extracted keyword.

```
keyword('Socrates').
keyword('student').
keyword('book').
keyword('Plato').
keyword('word').
keyword('mouth').
```

Figure 4. Keyword facts

*B. **Summary***

Summary fact contains sentence identifiers and a list of words in the sentence.

```
summary(0, ['Socrates', 'did', 'not', 'write', 'any', 'books', '.']).
summary(1, ['Plato', 'is', 'a', 'student', 'of', 'Socrates', '.']).
summary(2, ['Plato', 'wrote', 'books', 'and', 'put', 'words', 'into', 'Socrates', 'mouth', '.']).
```

Figure 5. Summary facts

C. *Sent*

Sent includes the list of sentences in the document.

```
sent(0, ['Socrates', 'did', 'not', 'write', 'any', 'books', '.']).
sent(1, ['Plato', 'is', 'a', 'student', 'of', 'Socrates', '.']).
sent(2, ['Plato', 'wrote', 'books', 'and', 'put', 'words', 'into', 'Socrates', 'mouth', '.']).
```

Figure 6. Sent facts

D. *W2L*

W2l includes the lemma and POS for each word. Lemma is the root form of a word.

```
w2l('Plato', 'Plato', 'NNP').
w2l('any', 'any', 'DT').
w2l('a', 'a', 'DT').
w2l('wrote', 'write', 'VBD').
w2l('put', 'put', 'VB').
w2l('mouth', 'mouth', 'NN').
w2l('books', 'book', 'NNS').
w2l('write', 'write', 'VB').
w2l('Socrates', 'Socrates', 'NNP').
w2l('words', 'word', 'NNS').
w2l('of', 'of', 'IN').
w2l('.', '.', '.').
w2l('student', 'student', 'NN').
w2l('not', 'not', 'RB').
w2l('and', 'and', 'CC').
w2l('into', 'into', 'IN').
w2l('did', 'do', 'VBD').
w2l('is', 'be', 'VBZ').
```

Figure 7. W2L facts

E. *Summary*

Summary fact contains sentence identifiers and a list of words in the sentence.

```
summary(0, ['Socrates', 'did', 'not', 'write', 'any', 'books', '.']).
summary(1, ['Plato', 'is', 'a', 'student', 'of', 'Socrates', '.']).
summary(2, ['Plato', 'wrote', 'books', 'and', 'put', 'words', 'into', 'Socrates', 'mouth', '.']).
```

Figure 8. Summary facts

All the above features that I have mentioned are linguistic facts that do not have much to play with, so we generated similar content in our system. Note that, even though CoreNLP and Spacy both use the UO system as the reference, the way they analyze the sentence structure are not the same, so sometimes they give different tags for some of the words, but that won't affect the result since tags are just terms for prolog.

F. *SVO*

1) ***Fact generation****:* DeepRank cannot generate all SVOs when the sentence is a 'compound sentence with more than one SVO, so we modified this part. For the third sentence "Plato wrote books and put words into Socrates mouth ", DeepRank can only read Plato write book, while we can read both Plato write book and Plato put words.

```
svo('Socrates', 'write', 'book', 0).
svo('Plato', 'write', 'book', 2).
```

Figure 9. SVO facts from DeepRank

```
svo('socrate', 'write', 'book', '0').
svo('Plato', 'wrote', 'book', '2').
svo('Plato', 'put', 'word', '2').
```

Figure 10. SVO facts from us

2) ***Efficient SVO****:* Our work improves the Subject-Verb-Object scheme to catch the negation effect on the verb and somewhat to reason the SVO weights, including the negation.

Negation:In the baseline model, we see that the SVO recording the verb without considering the negation effect may misunderstand the fact. In the example, *SVO ('Socrates', 'write', 'book')* violate the fact in sentence 1.

Our work catches the negation for which the verb has a 'neg' edge with the adverb. Now, the misunderstood one is correct to *SVO('Socrates', 'not write', 'book')*.

Weight:Since we have ranks of each word, there is a reason to rank the SVO as well. The default setting calculates the Rank(SVO) as:

$$Rank(SVO) = Rank(2 * S + O)/3 \tag{1}$$

, which exclude the effect of verb. From the default formula, we see that ignoring the negation will lead a false rank as shown in Fig.11, due to twice the weight of subject and none weight of verb and adverb.

```
(0.1087975198156778, ('Socrates', 'write', 'book', 0))
(0.08656855067870105, ('Plato', 'write', 'book', 2))
```

Figure 11. Rank(SVO) without negation.

```
(0.08656855067870105, ('Plato', 'write', 'book', 2))
(0.006835113727826585, ('Socrates', 'not write', 'book', 0))
```

Figure 12. Rank(SVO) with negation.

Our improvement in negation gives a more reasonable weight by counting the $Rank(verb + negation)$. The rank calculation now improve to:

$$Rank(SVO) = Rank(S + O - (V + N))/4 \tag{2}$$

Now, we see a more desirable rank of SVO shows in Fig.12.

What's more? In the query process, we also recognize 'who' as a potential subject to give the query, such as 'who can do something ?', an SVO predicate. This also efficient the Prolog answering engine.

G. *NER*

NER means Named Entity Recognizer Annotations. The problem we met here is that ‘’Socrates’ cannot be recognized as a person. In different positions, Spacy recognizes this word differently. In the first sentence, instead of a person, Spacy recognizes it as a name of a product. Therefore, we must manually correct that. CoreNLP did a much better job on this one.

```
ner(0, [(0, ('Socrates', 'PERSON'))]).
ner(1, [(0, ('Plato', 'PERSON')), (1, ('student', 'TITLE')), (2, ('Socrates', 'PERSON'))]).
ner(2, [(0, ('Plato', 'PERSON')), (1, ('Socrates', 'PERSON'))]).
```

Figure 13. NER facts

H. *Dependency edges*

1) ***Pre-defined dependency edges****:* Some edges are edges that are pre-defined in the default dependency graph to help with the understanding of the relations.

```
edge(0,'socrate', 'NNS', 'nsubj', 'write', 'VB').
edge(0,'do', 'VBD', 'aux', 'write', 'VB').
edge(0,'not', 'RB', 'neg', 'write', 'VB').
edge(0,'any', 'DT', 'det', 'book', 'NNS').
edge(0,'book', 'NNS', 'dobj', 'write', 'VB').
```

Figure 14. Pre-defined edges

2) ***Other dependency edges****:* Some are the edges that are defined by DeepRank to help with the understanding of the relations.

Recommend edges: The recommend edge comes from a word that is not a verb or a noun to the sentence (index).

```
edge(0, 'do', 'VBD', 'recommends', '0', 'SENT').
edge(0, 'not', 'RB', 'recommends', '0', 'SENT').
edge(0, 'any', 'DT', 'recommends', '0', 'SENT').
edge(1, 'be', 'VBZ', 'recommends', '1', 'SENT').
```

Figure 15. Recommend edges

Predicate edges: The predicate edge comes from the sentence index to the verb. DeepRank failed to extract all predicate edges when it is given a compound sentence, and it also mistakenly recognize noun as verb sometimes. These issues are solved in our generation system(Figure 16, Figure17).

First in edges: The first in edge is the first occurrence of a noun to emphasize sentences where the concepts are likely to be mentioned and defined for the first time. DeepRank sometimes misses some of the first occurrences, while they will all be recognized in our system.

```
edge(0, 0, 'SENT', 'predicate', 'write', 'VB').
edge(1, 1, 'SENT', 'predicate', 'student', 'NN').
edge(2, 2, 'SENT', 'predicate', 'write', 'VBD').
```

Figure 16. Predicate edges from DeepRank

```
edge(0, 0, 'SENT', 'predicate', 'write', 'VB').
edge(2, 2, 'SENT', 'predicate', 'write', 'VBD').
edge(2, 2, 'SENT', 'predicate', 'put', 'VBD').
```

Figure 17. Predicate edges from US

```
edge(0, 'book', 'NNS', 'first_in', 0, 'SENT').
edge(1, 'Plato', 'NNP', 'first_in', 1, 'SENT').
edge(2, 'mouth', 'NN', 'first_in', 2, 'SENT').
edge(2, 'word', 'NNS', 'first_in', 2, 'SENT').
edge(0, 'Socrates', 'NNP', 'first_in', 0, 'SENT').
```

Figure 18. First in edges from DeepRank

```
edge(0, 'socrate', 'NNS', 'first_in', '0', 'SENT').
edge(0, 'book', 'NNS', 'first_in', '0', 'SENT').
edge(1, 'Plato', 'NNP', 'first_in', '1', 'SENT').
edge(1, 'student', 'NN', 'first_in', '1', 'SENT').
edge(2, 'word', 'NNS', 'first_in', '2', 'SENT').
edge(2, 'mouth', 'NN', 'first_in', '2', 'SENT').
```

Figure 19. First in edges from US

About edges: The about edge is from the sentence index to noun. Again, our generation will collect all about edges while DeepRank would miss almost half of them.

```
edge(0, 0, 'SENT', 'about', 'book', 'NNS').
edge(0, 0, 'SENT', 'about', 'Socrates', 'NNP').
edge(2, 2, 'SENT', 'about', 'Plato', 'NNP').
edge(2, 2, 'SENT', 'about', 'book', 'NNS').
edge(2, 2, 'SENT', 'about', 'word', 'NNS').
```

Figure 20. About edges from DeepRank

```
edge(0, 0, 'SENT', 'about', 'socrate', 'NNS').
edge(0, 0, 'SENT', 'about', 'book', 'NNS').
edge(1, 1, 'SENT', 'about', 'Plato', 'NNP').
edge(1, 1, 'SENT', 'about', 'student', 'NN').
edge(1, 1, 'SENT', 'about', 'Socrates', 'NNP').
edge(2, 2, 'SENT', 'about', 'Plato', 'NNP').
edge(2, 2, 'SENT', 'about', 'book', 'NNS').
edge(2, 2, 'SENT', 'about', 'word', 'NNS').
edge(2, 2, 'SENT', 'about', 'Socrates', 'NNP').
edge(2, 2, 'SENT', 'about', 'mouth', 'NN').
```

Figure 21. About edges from US

I. Rankings

The last component in the prolog facts file is the rank computed for each word. The ranking is generated using pagerank function based on the graph we have. Since our dependency graph has more edges and richer information, the ranking generated from our edges is more representative(Figure 22, Figure23).

V. Query Answering with Prolog

In this section, we discuss how to feed the facts to the Prolog and return the most desirable sentences for query answering. We first introduce the general consulting scheme for all purpose questions asking, then we show how an improved SVO could efficient the answering engine.

```
rank('Socrates', 0.1310626248491194).
rank('student', 0.10238177148979814).
rank(0, 0.09341050158946196).
rank('book', 0.09237385442971471).
rank('Plato', 0.0866046865751659).
rank('write', 0.08069097513475772).
rank('Socrates mouth', 0.07110601307663854).
rank(1, 0.05761621064983509).
rank('word', 0.022464486285448743).
rank(2, 0.02176587685439639).
rank('mouth', 0.01722968961126349).
rank('do', 0.010521923321179575).
rank('a', 0.010521923321179575).
rank('any', 0.010521923321179575).
rank('of', 0.010521923321179575).
rank('and', 0.010521923321179575).
rank('put', 0.010521923321179575).
rank('not', 0.010521923321179575).
rank('be', 0.010521923321179575).
rank('into', 0.010521923321179575).
```

Figure 22. Rankings from DeepRank

```
rank('socrate', 0.031664621106719255).
rank('write', 0.1443361684974098).
rank('do', 0.013634397630838672).
rank('not', 0.013634397630838672).
rank('any', 0.013634397630838672).
rank('book', 0.045075918685064165).
rank('Plato', 0.0606080370938254).
rank('be', 0.06290512987142398).
rank('a', 0.013634397630838672).
rank('student', 0.07552767499008473).
rank('of', 0.03939236564968121).
rank('Socrates', 0.0606080370938254).
rank('and', 0.013634397630838672).
rank('put', 0.044568235777200 99).
rank('word', 0.02125099275674815).
rank('into', 0.03361297552439599).
rank('mouth', 0.04700896077559069).
rank('0', 0.06363427408302272).
rank('1', 0.13890969894353).
rank('2', 0.06272492099728424).
```

Figure 23. Rankings from US

A. *General Consulting*

After we have facts from parsers, we warp them as a knowledge base then feed them into Prolog for query answer consultors. The general query answering will return the K-best sentences, where users can decide K. There are four major facts that the engine will consult namely, NER, Rank, Edge and SVO.

We show how the consulting process (framework shows in the Fig.**??**) by giving an example query and the three facts as shown previously.

Example query: *'Who can write books?'*.

Step 1: Consulting the knowledge base for finding all matching items:

- ***NER***: Return None. Name of entity unfound in the query.
- ***Rank***: Return (0,2). From the ranks in Fig.25 and Fig.26, 'word' and 'write' are the two words appear in both predicates, returning the SentID for which these two words included.
- ***Edge***: Return (0,2). From the Fig.27 and Fig.28, matching item 'write -¿ book' share the same relationship and tags.
- ***SVO***: None. In the default setting, 'Who' does not be recognized as a subject. We will

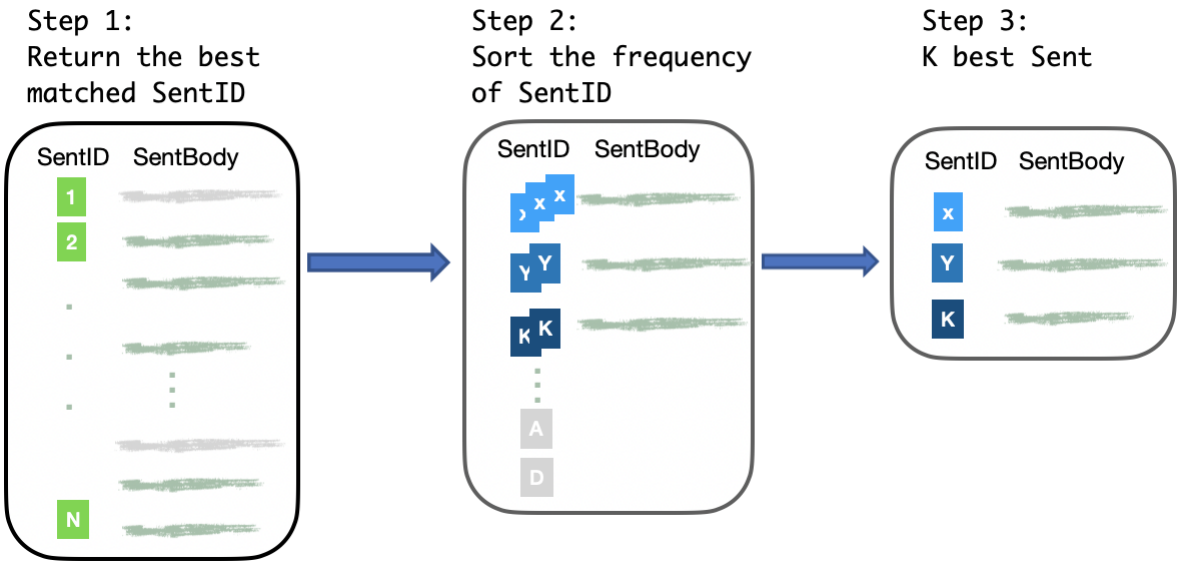

Figure 24. General consulting process.

```
rank('Socrates', 0.1310626248491194).
rank('student', 0.10238177148979816).
rank(0, 0.10140813437792442).
rank('book', 0.0923738544297147).
rank('Plato', 0.0866046865751659).
rank('write', 0.08069097513475772).
rank('Socrates mouth', 0.07110601307663854).
rank(1, 0.06254920306074477).
rank('word', 0.022464486285448743).
```

Figure 25. WordRank of the knowledge base.

```
query_rank(0, 0.36509895015634664).
query_rank('book', 0.36423363913906204).
query_rank('write', 0.21066741070459125).
query_rank('can', 0.030000000000000006).
query_rank('who', 0.030000000000000006).
```

Figure 26. WordRank of the query.

```
edge(0, 'do', 'VBD', 'aux', 'write', 'VB').
edge(0, 'do', 'VBD', 'recommends', 0, 'SENT').
edge(0, 'not', 'RB', 'neg', 'write', 'VB').
edge(0, 'not', 'RB', 'recommends', 0, 'SENT').
edge(0, 'write', 'VB', 'dobj', 'book', 'NNS').
edge(0, 0, 'SENT', 'about', 'book', 'NNS').
edge(2, 'write', 'VBD', 'nsubj', 'Plato', 'NNP').
edge(2, 2, 'SENT', 'about', 'Plato', 'NNP').
edge(2, 'write', 'VBD', 'dobj', 'book', 'NNS').
edge(2, 2, 'SENT', 'about', 'book', 'NNS').
edge(2, 'and', 'CC', 'cc', 'write', 'VBD').
edge(2, 'and', 'CC', 'recommends', 2, 'SENT').
```

Figure 27. Edge of the knowledge base.

```
query_edge(0, 'who', 'WP', 'nsubj', 'write', 'VB').
query_edge(0, 'who', 'WP', 'recommends', 0, 'SENT').
query_edge(0, 'can', 'MD', 'aux', 'write', 'VB').
query_edge(0, 'can', 'MD', 'recommends', 0, 'SENT').
query_edge(0, 'write', 'VB', 'dobj', 'book', 'NN').
query_edge(0, 0, 'SENT', 'about', 'book', 'NN').
query_edge(0, 'book', 'NN', 'first_in', 0, 'SENT').
query_edge(0, 0, 'SENT', 'predicate', 'write', 'VB').
```

Figure 28. Edge of the query.

show the optimization part in the next subsection.

Step 2: From general consulting schemes, we got twice return of the SentID: 0 and 2. Sorting and ranking the frequency of the SentID, the top ranks sentences has a strong reasoning for the query answering. We could see there is no reason for the weak related sentence to pop up and be considered as an candidate answer.

Step 3: Finally, the K-top sentences will return to the user, where K could be configured by users.

Fig.29 shows the general consultation result of the example case. Although the strong related items can be found, we see some of the redundant and useless information exist. For instance, the SentID: 0.

```
Question:  Who can write books?
General consulting:
        0 : Socrates did not write any books .
        2 : Plato wrote books and put words into Socrates mouth .
```

Figure 29. Results from general consultor.

```
Question:  Who can write books?
Consulting SVO:
        Plato
```

Figure 30. Result from SVO consultor.

B. Consulting SVO

SVO is an useful structure to efficient query answering, if queries also include SVO predicates. In our example, we can mask the word 'who' as an potential subject, giving *query_svo ('who', 'write', 'book')*. Then return all matching items from consulting the SVO predicates in the knowledge base. As shown in Fig.30, consulting an optimized SVO target an efficient feedback.

VI. Conclusion

- We successfully build a more advanced Prolog fact generation system to extract relations from the POS and dependency annotations given by the Spacy API. Out system can handle compound sentences while DeepRank can only deal with short sentences including one SVO relation.
- We successfully found a good solution for text summarizing in case if there is not enough rules and facts that can return a good answer to the end user.
- After comparing the performance of Spacy and CoreNLP, we find that CoreNLP does a better job justifying the identity of the word. These two APIs can have a disagreement when annotating the dependency relation for each word as well as understanding the sentence structures. Therefore, the dependency graph would be different, especially when the sentence is a compound sentence. Overall, either API can give enough support for the Prolog fact generation process.

REFERENCES

[1] Paul Tarau and Eduardo Blanco, Interactive Text Graph Mining with a Prolog-based Dialog Engine, July 2020, Intl. Symposium on Practical Aspects of Declarative Languages

[2] Long Short-Term Memory. Wikipedia, Wikimedia Foundation, 25 Mar. 2021, en.wikipedia.org/wiki/Long_short-term_memory.

[3] [ChenManning14] Danqi Chen and Christopher Manning, A Fast and Accurate Dependency Parser using Neural Networks, 2014, In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 740–750.

[4] GitHub. 2021. thushv89/attention_keras. [online] Available at: ¡https://github.com/thushv89/attention_keras¿ [Accessed 5 May 2021].

[5] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in Proceedings of the 2004 conference on empirical methods in natural language processing, 2004, pp. 404–411.

[6] "PageRank." Wikipedia, Wikimedia Foundation, 28 Apr. 2021, en.wikipedia.org/wiki/PageRank.