PROJECT TEAM: Geeks

TEAM MEMBERS:

Prachi Vats: PXV180021

Shivam Gupta: SXG190040

CS 6320 – Natural Language Processing Fall 2020
Dr. Mithun Balakrishna

PROBLEM DESCRIPTION:

- Design and implement models for extracting relations between two named entities in a sentence using NLP features and techniques.

- The project requirement is to extract relation between e1 and e2 entities of the sentence and even determine the direction of the relation.

  For example: Input Sentence is:

  "The system as described above has its greatest application in an arrayed <e1>configuration</e1> of antenna <e2>elements</e2>."

  Component-Whole (e2, e1)

  So, in the above input sentence e1 and e2 have relationships as Component and Whole and direction is from e2 to e1.

- The various types of relationship that are possible between e1 and e2 are:

  Component-Whole
  Cause-Effect
  Entity-Destination
  Member-Collection

- We need to implement a machine learning model that would predict the relationship between entities. Every sentence has two tagged entities.

PROPOSED SOLUTION:

To accomplish the objective, the problem statement is divided into several tasks.

Task 1: Pre-Processing

In Task 1 we do following steps:

- Read the input file
- Parse every sentence and extract the entities and sentences without entity tagging<e1>/<e2>.
- From the input file relations, entities and sentence information were stored separately and even the index information for the e1 and e2 words was gathered.
- Edited the tokens by expanding the Abbreviations. For instance there's becomes there is, what's becomes what is.
- Return the output of the preprocessing so that it can be utilized further for feature extraction.

Task 2: Feature Extraction

In Task 2 we implemented a NLP based feature extraction to pass to the machine learning model. Following steps were done as a part of task 2.

- **Tokenize** – Parse the input sentence and tokenize the sentence using NLTK. We made use of nltk.word_tokenize(), it takes sentences as input parameters and returns tokens.

- **Lemmatize** – Parse the input sentence and find lemmas corresponding to it using NLTK. We made use of the WordNetLemmatizer() of NLTK. This has a function lemmatize which takes a token as input and returns a corresponding lemma for it.

- **Part-of-Speech (POS) Tags** - Find the tags for every token in the sentence using NLTK by using nltk.pos_tag function. nltk.pos_tag function takes a list of tokens as input and returns a POS Tag.

- **Hypernyms, hyponyms, meronyms –** Extract the hypernyms, hyponyms and meronyms from the given sentence. Spacy is used to create tokens and we pass a token in Wordnet Synset to retrieve the synset for the token. From there we get Hypernyms, Hyponyms and meronyms.

- **NER Tags** – Named Entity Recognition is found for the 2 named entities present in the sentence. Spacy is used to find NER Tags. To access entity annotations we have used doc.ents. It gives entities for the tokens.

- **Dependency Tree**- This is found for syntactic parsing and we are able to find the root of the sentence and shortest path that could be possible between two entities. Spacy is used to find the dependency relations using dep_, which gives syntactic relations. Displacy is used to render the dependency tree. Inorder to display, IPython core display and HTML library is used

Task 3: Machine Learning Model Implementation

- To accomplish task 3 machine learning based approach is used to learn rules that can determine the relation and its direction.

- In Task 3 we have trained an RNN- LSTM based model. We have trained the model using tokens, posTags and shortest dependency path between two entities of the sentence. We utilized the features extracted in task 2 to train our LSTM Model. We are using Dependency parsing to extract the dependency trees and to extract the Lowest Common Ancestor(**LCA**) between the 2 Entities e1 and e2.

- We used the Shortest dependency path(SDP) between the e1 and e2. We achieved this by utilizing the path from the LCA node to e1 and from LCA node ro e2 because Words along the SDP forms a trimmed phrase and the LCA nodes gives the maximum relationship between the 2 tagged entities.

- The final features were basically all the words utilized in the shortest path between LCA and e1 and e2 respectively, their path lengths, their POS tags and their corresponding dependency relations which we obtained in the graphs.

- Each of the above features contains data in textual format.So we developed a mapping to convert those textual information into some number or IDs. The features are mapped to vectors using word embeddings which we created using the mapping that are used to be fed in our SDP-LSTM Model. We decided the maximum value of the embedding for every feature. We defined Dimension of the embedding layer for words, POS Tags, Dependency Types to 100, 25, 25 respectively which was sufficient for our data.

- RNN LSTM model is used for training. During training the parameters including weight, bias and embeddings are updated. Adam optimizer with L2 regularization is used to update the parameters.

- As part of training data, a set of sentences were given, where each sentence had tagged entities and their relationship.

IMPLEMENTATION DETAILS:

Programming Tools –

TOOLS USED:

Spacy:

NER Tags for entities for the sentence, dependency parsing of the sentence and creating a parse tree.

It is also used to find hypernyms, meronyms and hyponyms of tokens.

NLTK:

Sentence segmentation, tokenization, lemmatization, pos tagging.

Stanford NLP:

We used stanford NLP for dependency parsing and calculation of LCA and SDP for feature extraction of the dependency embedding.

TensorFlow: 1.14 and Keras: 2.3.1

Tensorflow and keras were used for our model training (SDP-LSTM). We trained our model on CPU and GPU(NVIDIA GeForce RTX 2060 6GB) as well.

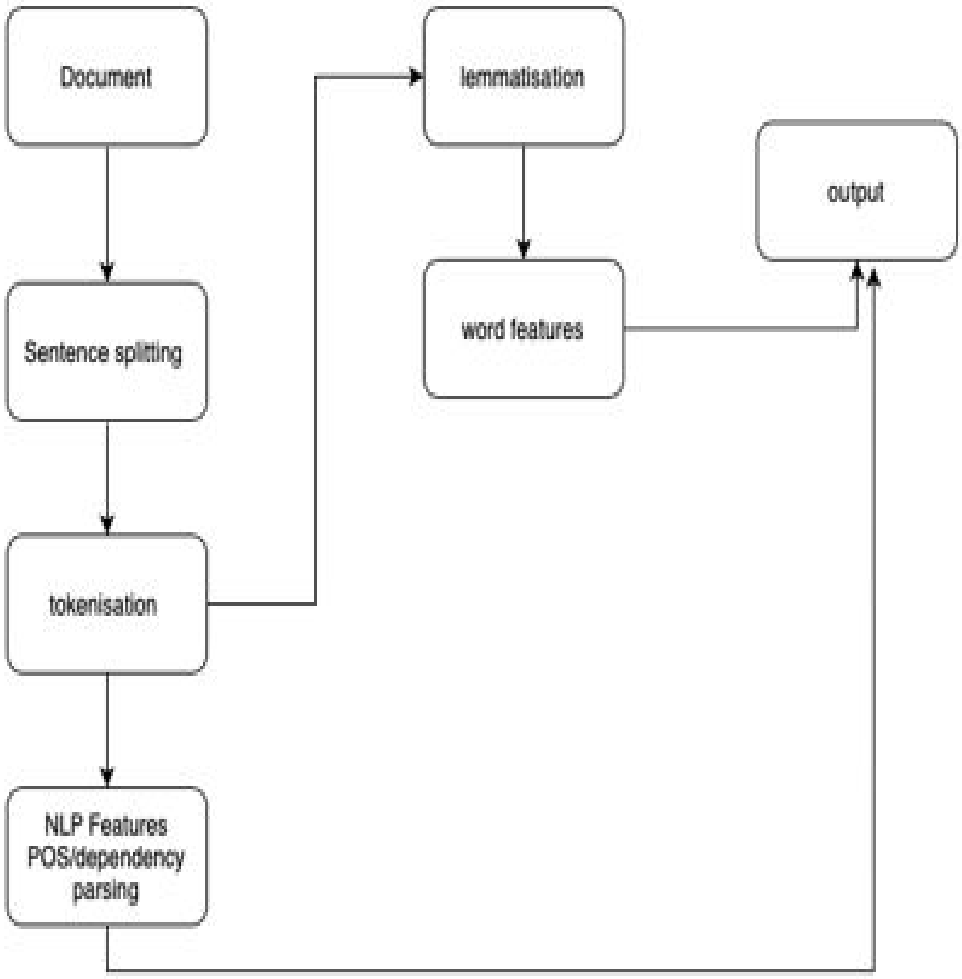
Scikit-Learn

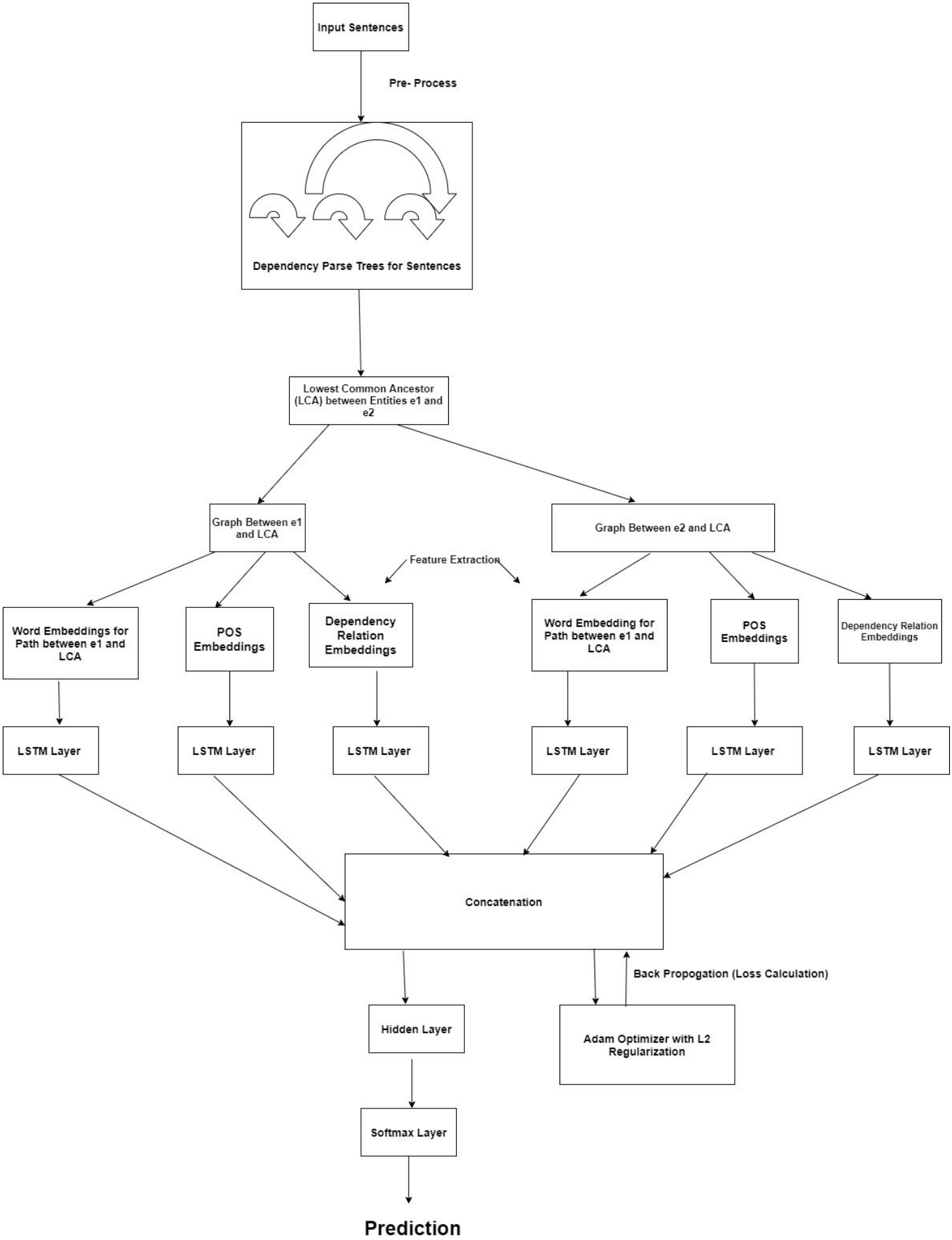
We used sk-learn for the classification report to calculated the precision, accuracy(weighted, macro) and F1 score, etc

Architectural Diagram -

Part 1 and Part 2-

Part 1 Architecture diagram




Input Sentences
Pre- Process
Dependency Parse Trees for Sentences
Lowest Common Ancestor (LCA) between Entities e1 and e2
Graph Between e1 and LCA
Graph Between e2 and LCA
Feature Extraction
Word Embeddings for Path between e1 and LCA
POS Embeddings
Dependency Relation Embeddings
Word Embedding for Path between e1 and LCA
POS Embeddings
Dependency Relation Embeddings
LSTM Layer
LSTM Layer
LSTM Layer
LSTM Layer
LSTM Layer
LSTM Layer
Concatenation
Back Propogation (Loss Calculation)
Hidden Layer
Adam Optimizer with L2 Regularization
Softmax Layer
Prediction

RESULTS and ANALYSIS:

We trained our SDP-LSTM model with the learning rate of 0.001, batch size of 10 for 100 epochs. We got our training accuracy to be pretty good 98.5 and the testing accuracy was 16.5% on the testing dataset (2717 Sentences). We saved our checkpoints after every 3 epochs and did validation as well for those checkpoints. After the validation we decided to choose the best checkpoint weight file.

The classification report for accuracy and F1 score is as well.

```
test accuracy 15.719557195571957
C:\Users\sadam\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
                          precision    recall  f1-score   support

                   Other       0.17      0.81      0.28       454
Entity-Destination(e1,e2)       0.11      0.21      0.15       291
      Cause-Effect(e2,e1)       0.09      0.10      0.12       194
 Member-Collection(e2,e1)       0.07      0.08      0.10       201
     Entity-Origin(e1,e2)       1.00      0.01      0.02       211
     Message-Topic(e1,e2)       0.02      0.16      0.21       210
   Component-Whole(e2,e1)       1.00      0.01      0.01       150
   Component-Whole(e1,e2)       0.11      0.21      0.06       162
 Instrument-Agency(e2,e1)       1.00      0.01      0.01       134
  Product-Producer(e2,e1)       0.07      0.13      0.09       123
 Content-Container(e1,e2)       0.09      0.05      0.17       153
      Cause-Effect(e1,e2)       0.02      0.08      0.25       134
  Product-Producer(e1,e2)       1.00      0.02      0.04       108
 Content-Container(e2,e1)       0.03      0.01      0.11        39
     Entity-Origin(e2,e1)       0.01      0.11      0.08        47
     Message-Topic(e2,e1)       0.05      0.01      0.03        51
 Instrument-Agency(e1,e2)       0.13      0.14      0.19        22
 Member-Collection(e1,e2)       0.17      0.19      0.29        32
Entity-Destination(e2,e1)       1.00      1.00      1.00         1

               accuracy                           0.17      2717
              macro avg       0.28      0.14      0.14      2717
           weighted avg       0.26      0.19      0.13      2717
```

fig 3 Classification Report(Assuming both the relation and direction are classified correctly.)

```
test accuracy 26.74956775383687897
C:\Users\sadam\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
                          precision    recall  f1-score   support

                   Other       0.17      0.81      0.28       454
Entity-Destination(e1,e2)       0.15      0.29      0.32       292
      Cause-Effect(e2,e1)       0.12      0.18      0.23       328
 Member-Collection(e2,e1)       0.19      0.21      0.26       233
     Entity-Origin(e1,e2)       0.32      0.09      0.11       258
     Message-Topic(e1,e2)       0.06      0.17      0.22       261
   Component-Whole(e2,e1)       0.26      0.24      0.10       312
 Instrument-Agency(e2,e1)       0.24      0.16      0.21       156
  Product-Producer(e2,e1)       0.37      0.16      0.12       231
 Content-Container(e1,e2)       0.11      0.09      0.19       192


               accuracy                           0.26      2717
              macro avg       0.31      0.27      0.24      2717
           weighted avg       0.33      0.21      0.23      2717
```

fig 4 Classification Report(Assuming that the relation is classified correctly (but not the direction))

Epoch Number	Loss	Training Accuracy	Testing Accuracy
60	0.285	0.86	0.12
80	0.198	0.91	0.14
100	0.097	0.96	0.17

Performance time:

Prediction Script for a single sentence was taking around 15 Seconds.

Training Time was 3.5 Minutes per Epoch - CPU

Training Time 1.1 Minutes per Epoch - GPU(CUDA)

SUMMARY OF PROBLEMS AND HOW WE SOLVED THEM:

- Lack of relevant information on the internet: A lot of initial research was done for the project implementation like how to create the feature vector, embedding and deciding which model to use and how to feed our embedding to them.
- The wide range of models and the issue of what to use in which situation: tried and tested on lots of data and formed assumptions based on the results.
- There was a version compatibility issue with TensorFlow and Keras. Posted questions on stack overflow and other forums. As we used the RNN layer from keras so we handled compatibility issues.
- It was difficult to come up with the idea of translating features to vectors. Did lots of research and went through research papers, to get the idea of word embeddings.
- Testing accuracy was not coming out to be pretty good so we tried to train on different parameters like changes embedding size, learning rate, number of channels, maximum word embedding length. So we tried to twig our model to improve the prediction.
- As our model was also complex so we solved a lot of problems in the shapes incompatibility issues. So we debugged each and every LSTM cell and printed their shapes to rectify issues.

- We also faced a lot of issues in training and Testing/Prediction like default session, graph issues in the tensorflow. So we created a defaugh graph in tensorflow and did everything in that.

PENDING ISSUES:

- We are using spacy to find NER Tags of the entities, but some of the NER tags were not coming using spacy. So we could look into this issue and find some other appropriate library for NER Tags.
- While preprocessing, dependency paths for the entities were created, which took a considerable amount of time. A code optimization could be done for creating dependency paths for the entities.

POTENTIAL IMPROVEMENTS:

- We were thinking of adding the drop-out to remove the over-fitting from our model but we ran out of time. So this could be done to improve the model Accuracy.
- We can also exponential learning rate decay in order to optimize weights so that we can improve our predictions.
- We can create this project as an end-to-end application to use it as a web application.
- We can use the feedback of the prediction to improve our model as well and use validation after every certain epoch to provide the feedback to our model.
- We can improve the prediction time as well because we are standforNLP for dependency parsing and it is quite a heavy library so building dependency parsing takes time.

CONCLUSION:

In the end we would like to conclude that Relationship extraction between entities is a very important sub-field of Natural language processing. We built our Deep neural network by using the Dependency parsed based features, word embeddings using tensorflow and keras. We could extend this project by using the type of the entities which will involve training the named entity model as well. if we could have the type of both entities as one of our features that would have added more textual-relationship information in the sentences. Also if we could add more data like these and use the state-of-the-Art models like **Bidirectional Encoder**

Representations from Transformers(BERT) which was recently developed by Google as the pretrained base model. We can create our own wrapper Model to achieve the best accuracy.