Introduction of NLP

Spring 2023, IIIT Hyderabad

Project Report

Team No. : 47

Project Title : Neural Dependency Parser

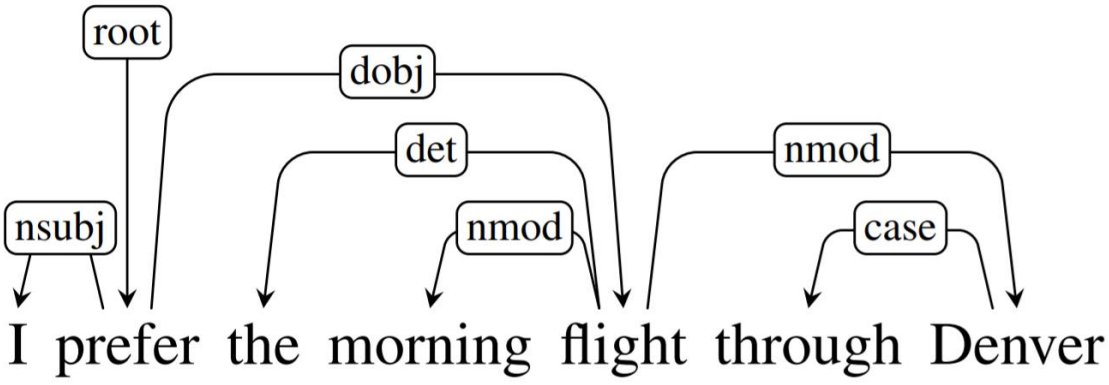
Team Members : Aditya Sharma (2021201016)

Rishu Toshniwal (2021201043)

Rahul Katyal (2021201083)

Introduction:

We have constructed the graph- based dependency parser which will take sentence as input and output the dependency tree of the sentence. We have implemented neural graph-based dependency parser paper link . We have train and test our dependency parser on the English treebank.



Literature Review:

The paper begins by introducing the task of dependency parsing, which involves identifying the syntactic relationships between words in a sentence. The authors then describe their parser, which is based on a graph-based approach that uses a neural network to score the likelihood of each dependency between words in the sentence. The parser is trained on a combination of automatically and manually annotated data, and uses a variety of features, including word embeddings, part-of-speech tags, and morphological features. Finally, the authors perform an error analysis to investigate the strengths and weaknesses of their parser. They find that the parser performs particularly well on languages with rich morphology and complex sentence structures, but struggles with languages with simpler structures.

Detailed Overview:

Dataset:

We have used the given dataset:

ud-treebanks-v2.11/UD_English-Atis/en_atis-ud-{train,dev,test}.conllu.

Pre-processing:

Words that occur in test data but not in training data are replaced by '<unk>' words since our training would unlikely to handle such words. Rest everything is already stored in given dataset that we don't have to do much in the pre-processing.

Word Embeddings:

Each word is first embedded in the training corpus. Each word is represented by a 50-dimensional vector. We additionally encode each POS tag as a 10-dimensional vector because it makes intuitive sense that a word's POS tag is important in figuring out its dependency relations (and the testing data provides the POS tags). The 60-dimensional concatenation of these two vectors is then used to represent a pair of words and POS tags. A sequence (of length n) of these concatenated vectors is used to represent a given sentence of length n.

Due to the fact that they are parameters in our model, the two embeddings change during training. In addition, we choose to initialize the (word, POS tag) embeddings using Gensim's Word2Vec method. First of all, training time is reduced by beginning with an accurate word embedding. Second, the Gensim word embeddings are trained independently of the task to which we apply them, despite being derived from our training corpus. By using these precomputed word embeddings, the risk of overfitting the word embeddings is therefore decreased (of course, this beneficial effect decreases as we increase the number of training rounds).

Bi-directional LSTM:

We therefore have a sequence (of length n) of 60-dimensional vectors given a sentence S of length n. The Bi-directional Long Short Term Memory Network (BiLSTM) is then fed this sequence. An LSTM has the advantage of being able to capture long-range dependencies over a more traditional RNN (for instance, it is less prone to the problem of exploding/vanishing gradient). Because the former is better equipped to capture long-range dependencies, we have chosen to use graph-based dependency parsers rather than transition-based ones. To ensure that this gain is realized, an LSTM is therefore chosen over a simple RNN.

Training:

Our network outputs two outputs for each sentence in our training corpus on the forward pass. The first thing it does is provide a n*n adjacency matrix, where the entry at (j, i) reflects the weight of an arc, with the head denoted by the letter "j" and the dependent by the letter "i." We can think of the columns of this adjacency matrix as equating to a probability distribution over potential heads because each dependent has exactly one head. It then produces a n*l matrix, where 'l' is the total number of labels in the training corpus. The word indexed by each row will be the dependent in each probability distribution over labels to gold arcs in this matrix. The weight that the ith label gives to the gold arc, whose dependent is the jth word, will thus be the entry at (j, i).

We determine the cross entropy loss (with respect to the columns) between the predicted n*n adjacency matrix and the gold n*n adjacency matrix in order to determine the loss. We also compute the cross-entropy loss between the anticipated n*l matrix and the gold n*l matrix (with regard to the rows). The overall loss for a certain forward pass of the model on this sentence is then calculated by adding these two losses. Using this loss, the model may then backpropagate appropriately. Because we are using the same network to predict both arc and labels, it should be noted that by combining the two losses in this manner, we are essentially performing multi-task learning.

The connected graph described in the first part is represented by the n*n adjacency matrix. Therefore, we employ an MST algorithm to extract a dependency tree from this adjacency matrix. We additionally assign the highest weighted label from the dependent's row in the n*l labels matrix to the dependent's incoming arc for each dependent in this dependency tree. We take the adjacency matrix and the MST algorithm to extract the dependency tree. This MST algorithm has the advantage of returning both projective and non-projective MSTs. This is important because languages with less rigid word orders (or grammar) will have a higher percentage of sentences with dependency trees that aren't projective. Without this, graph-based dependency parsing won't be as adaptable.

Result and Analysis:

Following is the graph of training versus validation loss for the 20 epochs:

After training the model for 20 epochs, we tested its accuracy on the test data for labeled dependency prediction and unlabeled dependency prediction:

Unlabeled Dependency Prediction represents:

(word : parent word)

Labeled Dependency Prediction represents:

(word : parent word : dependency relation between the two)

Unlabeled Dependency Prediction	**88.67579908675799**

Labeled Dependency Prediction	**83.33333333333334**

Note: Unlabeled Dependency Prediction Accuracy will always be greater than or equal to the Labeled Dependency Prediction Accuracy because if the parent word prediction is wrong, then both labeled and unlabeled prediction are wrong. But if parent word prediction is correct, then unlabeled prediction is correct, but the dependency prediction may be either correct or wrong.

Example of unlabeled dependency parse tree on a sentence:

Note: Word which is the parent of itself is the root of the sentence

```
***************  unlabeled **********************
what --> what
is --> what
the --> cost
cost --> what
of --> round
a --> round
round --> trip
trip --> cost
flight --> cost
from --> pittsburgh
pittsburgh --> flight
to --> atlanta
atlanta --> flight
beginning --> flight
on --> april
april --> fifth
twenty --> fifth
fifth --> pittsburgh
and --> returning
returning --> beginning
on --> may
may --> returning
sixth --> may
************************************************
```

Example of labeled dependency parse tree on a sentence:

```
*****************  labeled **********************
what --> what : root
is --> what : cop
the --> cost : det
cost --> what : nsubj
of --> round : case
a --> round : det
round --> trip : compound
trip --> cost : compound
flight --> cost : nmod
from --> pittsburgh : case
pittsburgh --> flight : nmod
to --> atlanta : case
atlanta --> flight : nmod
beginning --> flight : acl
on --> april : case
april --> fifth : obl
twenty --> fifth : nummod
fifth --> pittsburgh : acl
and --> returning : cc
returning --> beginning : conj
on --> may : case
may --> returning : obl
sixth --> may : amod
************************************************
```

Conclusion:

The dataset was well-balanced and could be efficiently divided into classes. We got better accuracy above 80% in both the cases.

References:

https://aclanthology.org/K17-3002/

https://towardsdatascience.com/natural-language-processing-dependency-parsing-cf094bbbe3f7

https://nlp.stanford.edu/software/nndep.html