Assignment 3

Anshul Tomar 170070007
Parth Shettiwar 170070021
Anwesh Mohanty 170070009

November 17, 2020

CP to DP: Approach

We did our implementation using the Allen Parser. As described in class, the method was to find the head word in each sub-tree and then propagate it upwards. The head was found using rule basis. We compiled a comprehensive list of rules to determine the head everytime. Finally, if suppose, the tags are not matching any of the current rules, then we used the principle that English is Head initial language and hence propagated the first word of the sub-tree as the head upwards andrest all as modifiers. In this case, we keep the relation of head to modifier as "R".

Rule List

```
Rules["VB CC VB"] = [[0],[[0, 1, 'cc'],[0, 2, 'conj']]]
Rules["VBP NP"] = [[0],[[0, 1, 'dobj']]]
Rules["VBP S SBAR"] = [[0],[[0, 1, 'xcomp'],[0, 2, 'dep']]]
Rules["VBG NP"] = [[0],[[0, 1, 'dobj']]]
Rules["NNP NNP"] = [[0],[[0, 1, 'dobj']]]
Rules["NP JJ"] = [[1],[[1, 0, 'npadvmod']]]
Rules["CD NNS"] = [[1],[[1, 0, 'num']]]
Rules["NP ADJP"] = [[0],[[0, 1, 'amod']]]
Rules["DT NN"] = [[1],[[1, 0, 'det']]]
Rules["DT JJ NN"] = [[2],[[2, 1, 'amod'],[2, 0, 'det']]]
Rules["NNP CD"] = [[0],[[0, 1, 'num']]]
Rules["IN NP"] = [[0],[[0, 1, 'pobj']]]
Rules["VB NP PP NP"] = [[0],[[0, 1, 'pobj'], [0, 2, 'prep'], [0, 3, 'tmod']]]
Rules["MD VP"] = [[1],[[1, 0, 'aux']]]
Rules["VBZ VP"] = [[1],[[1, 0, 'aux']]]
Rules["VBZ NP"] = [[0],[[0, 1, 'dobj']]]
Rules["JJ NNS"] = [[1],[[1, 0, 'amod']]]
Rules["VB NP"] = [[0],[[0, 1, 'dboj']]]
Rules["NP VP"] = [[1],[[1, 0, 'nsubj']]]
Rules["DT JJ NN NN"] = [[3],[[3, 0, 'det'], [3, 1, 'amod'], [3, 2, 'amod']]]
Rules["NP NP"] = [[0],[[0, 1, 'nmod']]]
Rules["PP NP"] = [[0],[[0, 1, 'pobj']]]
Rules["VBD NP"] = [[0],[[0, 1, 'nobj']]]
Rules["TO VP"] = [[1],[[1, 0, 'aux']]]
Rules["VBP S"] = [[0],[[0, 1, 'xcomp']]]
Rules["DT NNS"] = [[1],[[1, 0, 'det']]]
Rules["VBD PP"] = [[0],[[0, 1, 'prep']]]
```

Figure: The rules list

Algorithm

1. Get the constituent Parse tree from Allen Parser
2. Initialise a dependency parse tree with the words as nodes.
3. Get a sub tree of height 3 from bottom of Constituent Parse tree.
4. Find the head of this subtree from the rules as shown in Image.
5. Update the dependency parse tree by making an edge from the head to modifiers found from above step, as well as, updating the edge names, by the relations got by the Rule list.
6. Change the constituent parse tree by replacing the subtree by the head.
7. Go to step 3 and repeat until no subtree of height 3 is found.

Results

Sentence: The boy runs fast

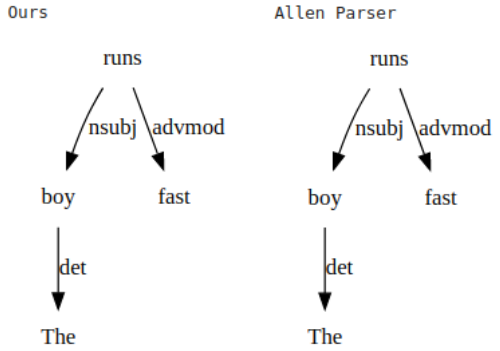
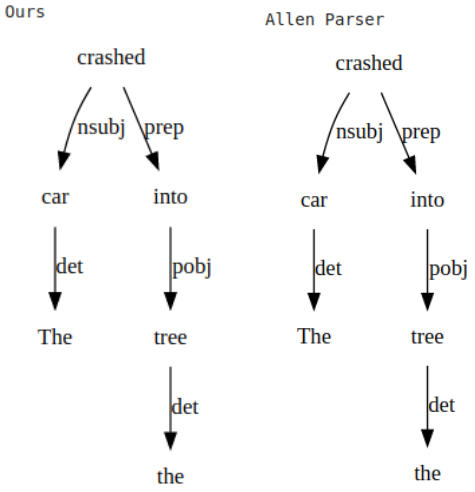


Figure: Left:Ours, Right:Allen Parser

Results

Sentence: The car crashed into the tree



Results: Ours better

Sentence: The quick brown fox jumps over the lazy dog

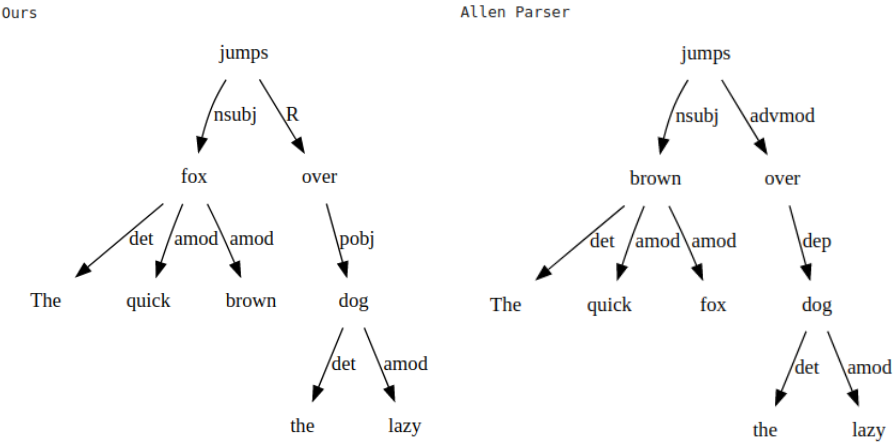


Figure: Left:Ours, Right:Allen Parser

Error Analysis

The Allen Parser has lots of inaccuracies and doesnt give correct dependency parse tree always. Hence we manually checked our accuracy with stanford CoreNLP parse trees for various sentences. We divided the analysis into 2 parts:1) Accuracy on 3-4 word length sentenes, 2)5-9 word length sentences. Following are the number of sentences on which it gave correct parse tress.

Number of words	Number of sentences tested	Number of correct parse tress
3-4	10	9
5-8	10	6

Table: Error analysis of CP to DP conversion

Since we have written a limited number of rules, hence sometimes for higher number of words cases, our algorithm produces an incorrect parse tree.

Strengths and Limitations

Strengths

- Many a times, our parser performs better than Allen parser.
- The time taken for producing the parse tree is minimal.
- With enough time given, our parser can produce accurate results if all rules are listed down.

Limitations

- Since its rule based, we can only type in limited set of rules, and hence cant cover all possibilties at any given day.
- Further, since we consider only a subtree of height 3 at any moment, and not care about the effect of far away words, this limits our performance further.
- Also many a times, allen parser itself gives a wrong constituent parse tree, in that case, our final generated dependency parse tree is also wrong.

DP $\rightarrow$ *CP* : *Approach*

Implemented this using spacy module. Used this because it directly gives each and every word's parent's, children, pos in a dependency parse tree in the form of tokens which makes the recursion part much easier.
Now given a recursion tree we first find out the location of the root as the left part of the root will be a noun phrase and the right part will constitute the verb phrase. Let the root's index be i.
Now that we have separated the sentence into two parts NP(0 - i-1) and VP(i-n), we recursively find out the constituency parse tree.

Algorithm

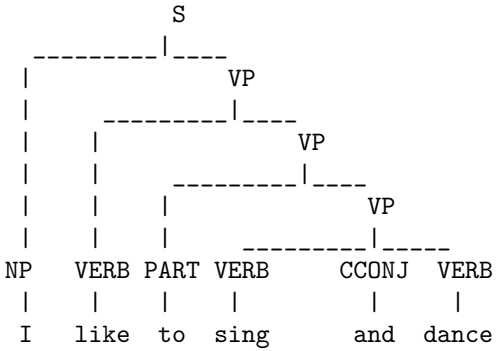
Noun phrase

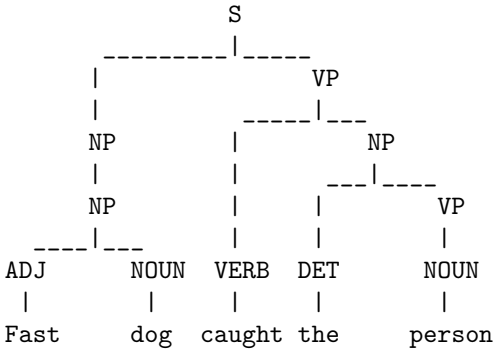
- Select "nsubj" dependency and call NP(nsubj)
- Let's say our function is NP(word).
- The base case is:- No child Return nltk.Tree(word.pos_,[word.text])
- If it is punctuation we ignore it
- Recursion:- For all the children of the word if child.i $>$ word.i :- right_tree.append(NP(child)) else :- left_tree.append(NP(child))
- Return nltk.Tree(L,merge(left_tree,right_tree))
- Where L is decided by label of the first word.

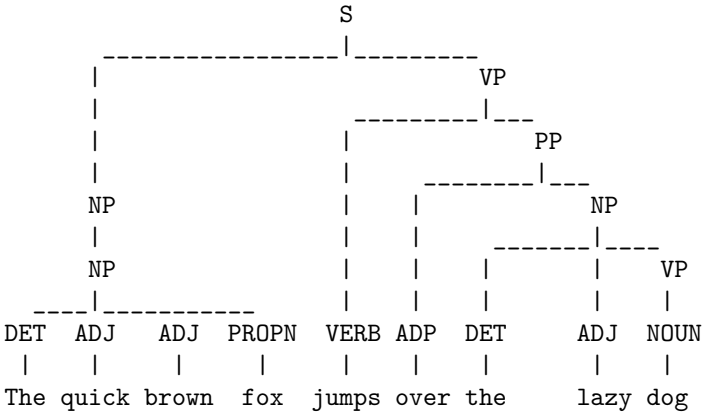
Verb Phrase

- Select "root" dependency and call VP(root)
- Let's say our function is VP(word).
- The base case is:- No child Return nltk.Tree(word.pos_,[word.text])
- If it is punctuation we ignore it
- Recursion:- For all the children of the word if child.i $>$ word.i :- right_tree.append(VP(child)) elif (child.dep_ $==$ 'aux'):- left_tree.append(VP(child))
- Return nltk.Tree(L,merge(left_tree,right_tree))
- Where L is decided by POS of the first word.

Results


S
VP
VP
VP
VP
NP VERB PART VERB CCONJ VERB
I like to sing and dance


S
VP
NP
NP
NP
VP
ADJ NOUN VERB DET NOUN
Fast dog caught the person


S
VP
PP
NP
NP
NP
VP
DET ADJ ADJ PROPN VERB ADP DET ADJ NOUN
The quick brown fox jumps over the lazy dog

Error Analysis

Same sentences that were used for CP to DP have been used here.

Number of words	Number of sentences tested	Number of correct parse tress
3-4	10	8
5-8	10	4

Table: Error analysis of DP to CP conversion

The most common mistakes are when the sentence is in passive voice, the nsubj dependency comes after the verb or the nsubj is completely missing from the sentence.
Limitations Not yet able to convert when nsubj does not come before the verb.

Project-Description

Topic : Machine translation for the dataset English-German WMT 2014 given the condition that we have to minimize the time complexity rather than improving the accuracy.

Idea : We would be using the Adversarial training method as a regulariser to improve our performance, as proposed in the paper "Improving Neural Language Modeling via Adversarial Training" on the RNNSearch model of "Neural Machine Translation by Jointly Learning to Align and Translate".

Slides : `https://docs.google.com/presentation/d/184n3IbMHTDuZHTcpDqberSghMQt2irqPhWTXNA17yvo/edit?usp=sharing`

Project - Progress

We have completed the following things:

- Reviewed the relevant literature required for the project
- Created a baseline model inspired from the RNNSearch Model in PyTorch
- Since Google Colab doesn't have enough resources to run the baseline model on the WMT14 en-de dataset, we are currently using the Multi30K en-de dataset to check the efficacy of the model

Things to be done:-

- Implement the BLEU score function to get an idea of how well the model is performing and conduct an extensive ablation study to get better results while keeping the model complexity as simple as possible.
- We will implement the Adeversarial MLE training method by the end of this week and compare the performance of the model with and without it.
- We are searching for machines where we can run the WMT14 dataset. Once available, we will start testing our model using the WMT14 dataset.

Project - Baseline Model

We have used cross entropy (and perplexity) as a metric for our baseline model. We will incorporate the BLEU score in the model soon. Since the WMT14 de-en dataset has around 4.5M training examples in it, Google Colab is unable to run such a large dataset. Hence we test our baseline on the Multi30K dataset which has only 29000 sentences in training. We achieved the following results on the baseline model after running it for 10 epochs using the Adam optimizer with lr=10^{-3} with batch size = 8, cross entropy as the loss function -

- Best Validation Loss = 3.135; Best Validation Perplexity(PPL) = 22.978
- Test Loss = 3.178; Test Perplexity(PPL) = 24.004 (using model with best validation score)

Problems faced till now

```
Epoch: 01 | Time: 3m 9s
        Train Loss: 3.730 | Train PPL:  41.687
         Val. Loss: 3.395 |  Val. PPL:  29.818
Epoch: 02 | Time: 3m 10s
        Train Loss: 2.639 | Train PPL:  13.993
         Val. Loss: 3.135 |  Val. PPL:  22.978
Epoch: 03 | Time: 3m 10s
        Train Loss: 2.215 | Train PPL:   9.164
         Val. Loss: 3.165 |  Val. PPL:  23.680
Epoch: 04 | Time: 3m 9s
        Train Loss: 1.952 | Train PPL:   7.041
         Val. Loss: 3.216 |  Val. PPL:  24.921
Epoch: 05 | Time: 3m 11s
        Train Loss: 1.788 | Train PPL:   5.979
         Val. Loss: 3.342 |  Val. PPL:  28.267
Epoch: 06 | Time: 3m 10s
        Train Loss: 1.681 | Train PPL:   5.370
         Val. Loss: 3.458 |  Val. PPL:  31.740
Epoch: 07 | Time: 3m 9s
        Train Loss: 1.619 | Train PPL:   5.047
         Val. Loss: 3.561 |  Val. PPL:  35.215
Epoch: 08 | Time: 3m 10s
        Train Loss: 1.577 | Train PPL:   4.841
         Val. Loss: 3.545 |  Val. PPL:  34.624
Epoch: 09 | Time: 3m 9s
        Train Loss: 1.550 | Train PPL:   4.712
         Val. Loss: 3.627 |  Val. PPL:  37.591
Epoch: 10 | Time: 3m 10s
        Train Loss: 1.547 | Train PPL:   4.699
         Val. Loss: 3.743 |  Val. PPL:  42.212
```

Figure: Overfitting in baseline model

As can be observed, after 2-3 epochs the model essentially overfits on the training dataset and hence the validation loss starts increasing again. We are hoping that adding the adversarial mle training will help alleviate this. There is also another method that we found called the "Partial Shuffle" which looks promising and we can look into this also if we check all our boxes in time.