CSE 291E: Project 3

Keerthana Purushotham

I. Abstract

In this report I present the CKY algorithm for constituency parsing on the PTB_LE10 and PTB_first_2000 subsets (which were provided in the starter) as well as the complete PTB dataset. To compute model features , we need to calculate the partition function in the NLL score function, since minimizing this score would be the goal of the learning process. This computation takes place in the inside function and implementing it with the rest of the code is the specific goal of the implementation task. All implementations were run on Google Colab using a GPU.

II. Introduction

Constituency parsing is the task of assigning a constituency structure to a sentence based on a context-free grammar generated from a training set that is assumed to follow the same grammar as the sentence. Entire languages that are as complex as or comparable to English, tend to have incredibly large grammars which make it a computationally difficult task. Grammar can additionally be distinctive to different contexts and domains which can make prediction challenging in terms of deciding what is right and wrong. To represent the constituency structure, we choose to use constituency trees which are within the context of the algorithm, further converted into Chomsky normal form and processed to handle unaries. To introduce the solution to this task abstractly, we generate a set of all possible and valid constituency trees for a given sentence and pick the constituency parse derived from the highest scoring tree.

III. Approach

The PTB dataset is actually a tree bank of constituent trees for a set of sentences. We use these trees to train our model. First the words in each sentence and their POS tags are passed through an embedded layer to get the word and tag embeddings. These embeddings are concatenated and fed into a Bi-LSTM Layer consisting of a forward layer and backward layer to gain context of each word with respect to the rest of the sentence before and after this word. This is how we obtain the learned features of the given sequence or the specific context of each sentence which gives us the independent tag predictions.

These hidden states are then concatenated and fed into the left and right Multi Layer Perceptron (MLP). This will create feature representations of both left and right span endpoints. One could otherwise say that this is how we generate a set of potential trees to score and consider. The vectors generated are then fed into a Biaffine scoring layer. An Affine function is of the form $WX + b$ where W is a weight matrix, X is the matrix upon which we are applying the weight and b is some bias if needed. A Biaffine function is a similar function wherein one applies a transformation function on an input twice and is of the form, $W(W(x) + b) + b$. In our project, given the right and left span representations, we apply a Biaffine function where a Weight matrix is obtained based on the label 'l' which becomes the first transformation and is then applied on the two span representations which becomes the second transformation. The Biaffine function chosen [1] applies W as $X^T W(Y/d^s)$ (where d is vector dimension and s is scaling factor respectively), after which an exponent of this score is taken. This part of score calculation tallies the parent-child unit scores.

The CRF_Constituency function is the part of score calculation that links up the final tree based on parent-child unit scores and is where we implement the inside function. As mentioned earlier, the BiLSTM predicts tags while the CRF scores a combination of tags. So the CRF score is the log of products of individual tag scores calculated from the Biaffine function. In order to train this model, we take this score into negative log space and the learning process is implemented by minimizing this negative log likelihood value (NLL) across training.

$$NLL = \Sigma^{x,y^*} - \Big[\log \Phi(X, Y^*) - \log_{Y' \in O(X)} \Phi(X, Y') \Big] \quad (1)$$

The partition function which is the term on the right side of the bracket, is what is calculated using the inside function in our code where O(X) refers to a set of all valid trees. We use an approach similar to the CKY algorithm which is further mentioned below. It is a dynamic recursive algorithm which calculates the score at each level of the constituency tree till the root node as,

```
S[i, j] := logSumExp(S[i, j], logSumExp
(Score[i, j, *]) + S[i, k] + S[k+1, j])
```

Where, **logSumExp(Score[i, j, *])** becomes the base case when $j - i = 1$ for a potential parent tag '*', given the POS tags for the words at position i and j. I use a vectorized approach and pretty much follow the provided pseudocode except I skip using the k loop by instead using the stripe function provided similar to how the CKY algorithm was implemented in the starter.

The final part is where we predict the best scoring sentence. We decode the score using the CKY algorithm where the best scoring children at each node of the constituency tree are predicted and their values are tallied (which should ideally give you the maximum score). The CKY algorithm is implemented as a dynamic recursive algorithm which calculates the score at each level of the constituency tree till the root node just like in the inside function mentioned above as,

```
Result[i, j] := max(Result[i, j], max(Score
```

```
[i, j, *]) + Result[i, k] + Result[k+1, j])
```

Where, **max(Score[i, j, *])** becomes the base case when $j - i = 1$ for the decoded best parent tag '*', given the span scores for its potential children. The loss for this final decoded constituency tree is calculated against the true tree in the training data and further minimized. The loss graphs for training are presented in the Results section.

IV. Evaluation

We evaluate the model across the PTB dataset and two of its subsets- PTB_LE10 and PTB_first_2000. We present the evaluation results and runtime, plot the training loss graph, plot a graph for F1 scores of labelled and unlabelled data and finally compare two sample trees generated against the true tree.

V. Results and Inference

A. English WSJ Penn TreeBank (PTB) [refer PTB link] - PTB_LE10

The PTB_LE10 dataset is a tree bank of size 2000 where each sentence is of lesser length than 10. The model is run for 10 epochs only since the dataset is small with smaller sentences and begins to perform badly after epoch 10. A full training log has been submitted with the code. **Total training lasted 21.201837s with 02.120184s/epoch**. The evaluation results are as follows:

```
Epoch 9 saved
dev:  UCM: 69.94% LCM: 66.87% UP: 93.20%
UR: 94.00% UF: 93.60% LP: 90.66% LR:
91.43% LF: 91.04%
```

The best scores were seen in epoch 9. Two graph plots for training loss/epoch and UF/LF scores/epoch are attached below:

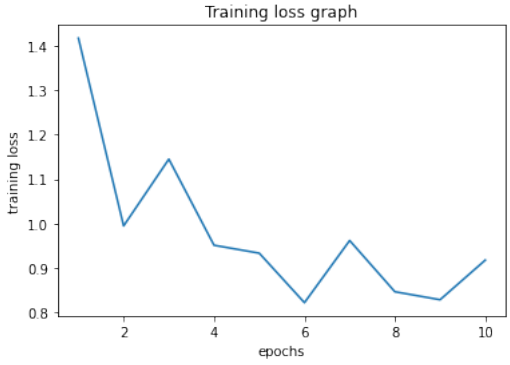

Fig. 1. Training loss per epoch

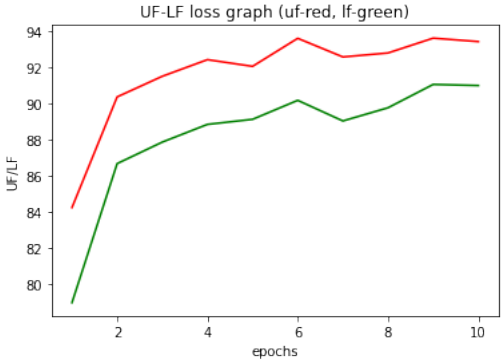

Fig. 2. F1 score for unlabeled and labeled data per epoch

The loss decreases till epoch 10 and increases over 10 epochs indicating overfitting which is reasonable given that the dataset is small. The F1 score also increases per epoch as seen above Lastly, we compare two sample predicted trees against their true trees from the dataset as seen above. The trees on the left are predicted trees and the trees on the right are true trees.

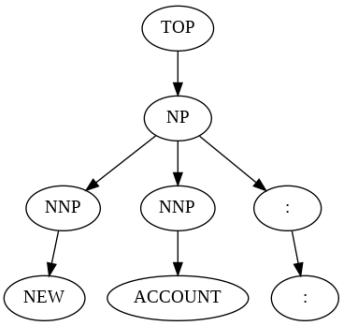

Fig. 3. Predicted Tree for sentence 0

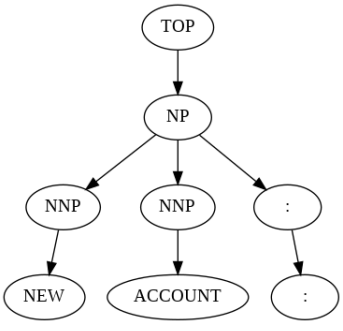

Fig. 4. Ground Truth Tree for sentence 0

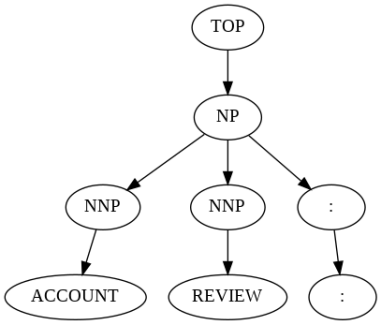

Fig. 5. Predicted Tree for sentence 1

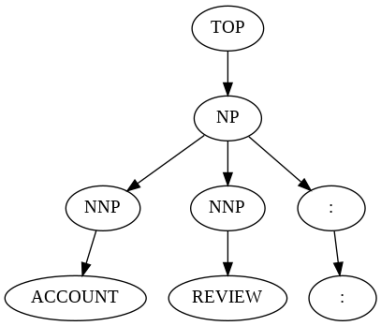

Fig. 6. Ground Truth Tree for sentence 1

As we can see both trees are identical for both samples.

B. PTB_first_2000

The PTB_first_2000 dataset is a tree bank of the first 2000 trees in the whole dataset and is the same dataset that is used to calculate the given baseline score in the assignment. The model is run for 20 epochs. A full training log has been submitted with the code. Total training lasted 06 min and 51.212138s with 20.560607s/epoch. The evaluation results are as follows:

```
Epoch 19 saved
dev:  UCM: 22.47% LCM: 20.59% UP: 85.89%
UR: 86.93% UF: 86.41% LP: 84.19% LR:
85.20% LF: 84.69%
```

The best scores were seen in epoch 19. Two graph plots for training loss/epoch and UF/LF scores/epoch are attached below:

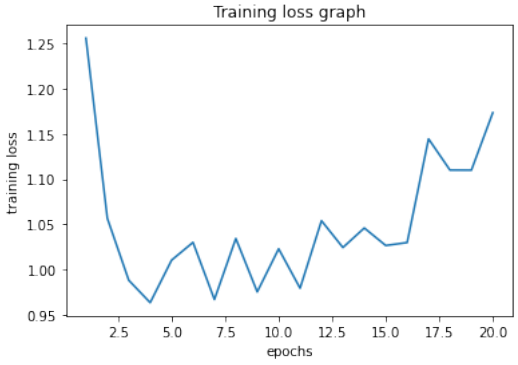

Fig. 7. Training loss per epoch

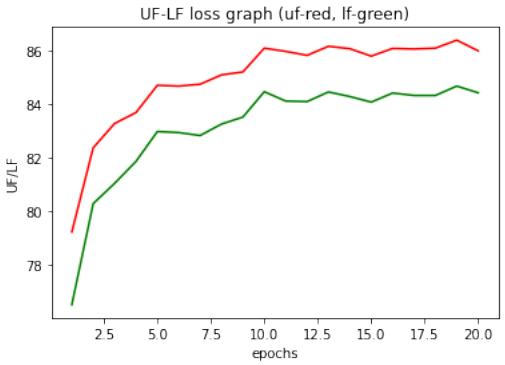

Fig. 8. F1 score for unlabeled and labeled data per epoch

The loss decreases till epoch 4, meanders for a while and then starts to increase again which indicates overfitting after epoch 11. As seen above, both UF and LF scores which are F1 scores for unlabelled and labelled data respectively increase per epoch, peak at epoch 19 with UF=86.41%. This value is higher than the 85.40% seen in the assignment.

Lastly, we compare two sample predicted trees against their true trees from the dataset as seen above. The trees on the left are predicted trees and the trees on the right are true trees.

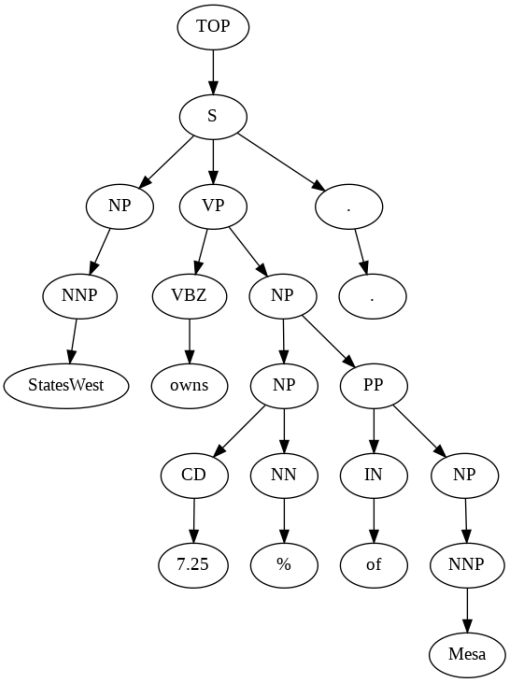

Fig. 9. Predicted Tree for sentence 501

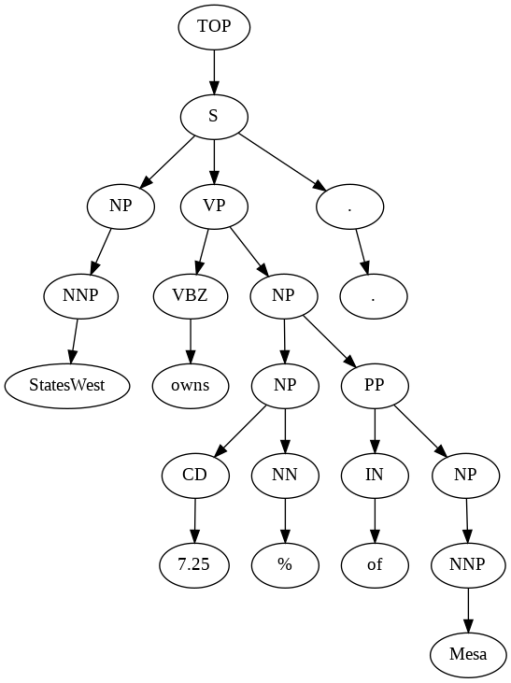

Fig. 10. Ground Truth Tree for sentence 501

In this sample, we see that the trees are identical.

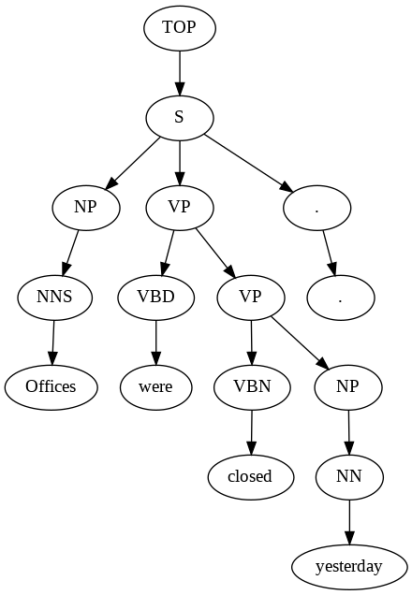

Fig. 11. Predicted Tree for sentence 502

However in the second sample, there is a difference. The true tree (on the right) classifies the word “closed” as an adjective which means to be “not open” or explains that offices were not working the previous day. But in the predicted tree, “closed” is classified as a verb which means to “move or cause to move so as to cover an opening” and would mean that offices were physically closed on the day that was yesterday. Both contexts of the sentence do make sense and is a reasonable ambiguity since even as a human being I would say both parses are linguistically correct without more context.

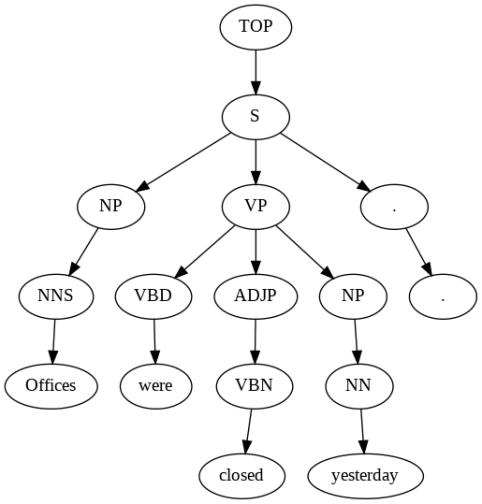

Fig. 12. Ground Truth Tree for sentence 502

C. PTB - Full

In this case, I split the whole PTB treebank into a training set of 27,126 trees, dev set of 7,146 trees and a test set of 5, 560 trees. The model is run for 20 epochs. A full training log has been submitted with the code. Total training lasted 1 hr, 04 min and 14.806166s with 3 min and 12.740308s/epoch. The evaluation results are as follows:

```
Epoch 13 saved
dev:  UCM: 27.60% LCM: 25.75% UP: 89.47%
UR: 88.99% UF: 89.23% LP: 88.09% LR:
87.61% LF: 87.85%
```

The best scores were seen in epoch 13. Two graph plots for training loss/epoch and UF/LF scores/epoch are attached below:

Fig. 13. Training loss per epoch

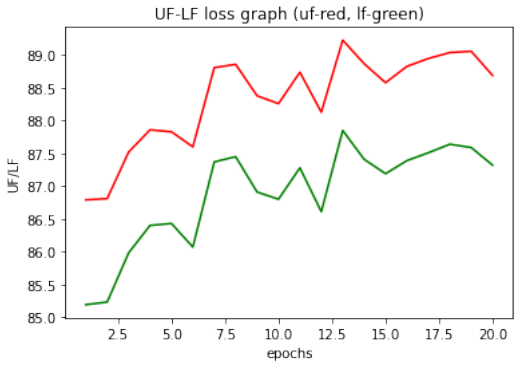

Fig. 14. F1 score for unlabeled and labeled data per epoch

The loss decreases till epoch 20. As seen above, both UF and LF scores which are F1 scores for unlabelled and labelled data respectively increase overall but aren't as steady as the increase in the previous two cases. This could probably mean that perhaps the model hadn't converged yet in the 20 epochs I ran it for, owing to its significantly larger size. The F1 score peaks at epoch 13 with UF=89.23%.

Lastly, we compare two sample predicted trees against their true trees from the dataset as seen above. The trees on the left are predicted trees and the trees on the right are true trees.

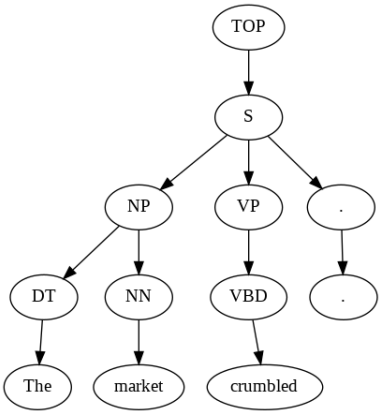

Fig. 15. Predicted Tree for sentence 501

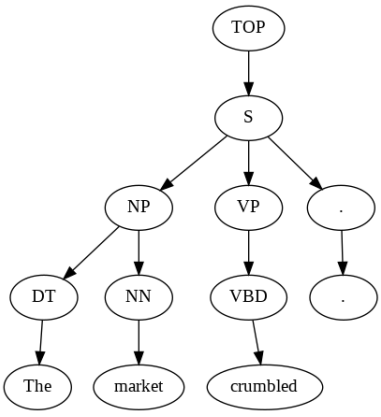

Fig. 16. Ground Truth Tree for sentence 501

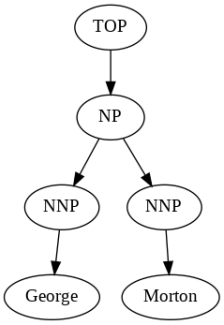

Fig. 17. Predicted Tree for sentence 501

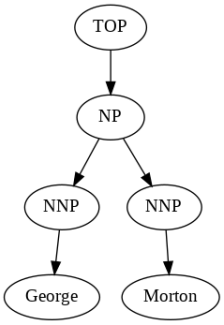

Fig. 18. Ground Truth Tree for sentence 501

As we can see both the predicted as well as the true tree are identical in both cases which indicates that our model is performing well even though it could probably perform better given the above mentioned inference on the training loss plot.

D. Comparison

The table below is a compilation of all the best scoring evaluation metrics for each implementation.
The PTB_LE10 data set is small with only sentences of small size so there is little to learn (which the model does quickly) and it also performs well on the test data since it is also a subset of small sentences. Longer sentences will likely perform badly.
The PTB_first_2000 data set is also just of size 2000 but has no restrictions over size of the sentence. It has comparably good evaluation metrics to the baseline.
The implementation for the whole dataset also performs very well but could likely be trained for longer.
All implementations outperform the baseline.

TABLE I
COMPARISON TABLE

	PTB_LE10	PTB_first_2000	PTB_first_2000 (baseline)	PTB_full
# of epochs	10	20	20	20
time/epoch	02.120184s/epoch	20.560607s/epoch	N/A	3 min, 12.740308s/epoch
UCM	69.94%	22.47%	18.76%	27.60%
LCM	66.87%	20.59%	17.35%	25.75%
UP	93.20%	85.89%	84.74%	89.47%
UR	94.00%	86.93%	86.06%	88.99%
UF	**93.60%**	**86.41%**	**85.40%**	**89.23%**
LP	90.66%	84.19%	82.59%	88.09%
LR	91.43%	85.20%	83.87%	87.61%
LF	**91.04%**	**84.69%**	**83.23%**	**87.85%**

VI. CONCLUSION

I have presented the model for Constituency Parsing using the CKY algorithm, BiLSTM and CRF. I have also presented my implementations of the model with 3 versions of the dataset with their own inferences and have compiled the evaluation scores into a table.

VII. ACKNOWLEDGEMENTS

I sincerely thank the professor and TAs for putting together an incredibly insightful course and I felt like I learned so much more than I anticipated with the research style approach to the course. This course and this project amongst the others were incredibly rewarding.

VIII. REFERENCES

[1] https://arxiv.org/abs/1611.01734

[2] https://web.stanford.edu/ jurafsky/ slp3/13.pdf

[3]https://raw.githubusercontent.com/nikitakit/self-attentive-parser/master/data/02-21.10wayclean

[4] And the remaining references from the assignment.

IX. APPENDIX

1) The runtime logs for all three implementations.
2) Main.py with my edits.
3) The notebook I ran main.py in.
4) Another notebook I used to generate a graph.