Project - 3: Neural CRFs for Constituency Parsing

Anonymous ACL-IJCNLP submission

Abstract

In this report, I have build a neural conditional random field for the task of consituency parsing. I have implemented vectorized version of inside algorithm as part of my forward pass. I have also completed a number of optional tasks. I have a) analyzed performance by removing the tag embedding layer, b) used one MLP layer to model the boundary representation, c) used Glove embeddings as input to crf, d) made changes to learn a full neural grammar.

1 Model Description

The neural network model (Figure 1) used in this implementation is as follows:

- For each token, we look up word and part-of-speech tag embeddings.
- We then apply a bidirectional recurrent neural network for computing learned feature representations of sequences.
- We apply two separate MLPs to create feature representations of both left and right span endpoints.
- We use a biaffine scoring function to compute the score for every edge in the sentence.
- We use CRF constituency parser to compute loss. Inside algorithm has been implemented at this step.
- CKY algorithm is used to decode the POS of the input sentence.

1.1 Detailed Explanation

Given a sentence with T words and their part-of-speech tags $X = \{W, G\}$. A constituency parse tree is denoted as Y , which is a set of triples, $(i, j, l) \in Y$, that each denote a constituent spanning the words at positions i to j, inclusive, with syntactic label $l \in L$.

1.1.1 BiLSTM features

The structure of RNNs enables it to model feature representation of sequences in a very powerful fashion. We concatenate the representations of words and POS tags and use it as an input to the BiLSTM layer.

$$x_t = \text{CONCAT}(w_t, g_t),$$

passing it to BiLSTM we get representations in form of hidden states,

$$\rightarrow \text{h t} = \tanh(\text{Wihxt} + \text{bih} + \text{Whhh(t1)} + \text{bhh})$$

1.1.2 MLP layer

Applying MLPs to the recurrent output states before the biaffine scoring has the advantage of stripping away information not relevant to the current decision. The ouputs of recurrent neural network has lot more information than what is required to produce the labels. Intuitively, the MLP layer help in pruning off the unnecessary information and only retain the information required to produce correct labels.

1.1.3 Biaffine scoring function

This is used to obtain scores for all labels. It is applied on the boundary representations in the previous step. Both left and right boundary computations are used to come up with a syntactic representation.

1.1.4 Loss computation

We compute sentence-level global CRF loss, trying the maximize the conditional probability

$$\text{P(Y}\|X) = s(x, y) + logZ(x)$$

where logZ(x) is computed using the inside algorithm. Complete loss function is given by

$$\text{NLL} = \sum_{\boldsymbol{X},\boldsymbol{Y}^*}^{\boldsymbol{X},\boldsymbol{Y}^*} -\log p(\boldsymbol{Y}^* \mid \boldsymbol{X};\theta) = \sum_{\boldsymbol{X},\boldsymbol{Y}^*}^{\boldsymbol{X},\boldsymbol{Y}^*} -\Big[\log \Phi(\boldsymbol{X},\boldsymbol{Y}^*) - \log \sum_{\boldsymbol{Y}' \in \mathcal{O}(\boldsymbol{X})} \Phi(\boldsymbol{X},\boldsymbol{Y}')\Big]$$

Here the second term is logZ(x) mentioned above and is the log of probability of all the possible sequences. First term is the probability of obtaining the prediction from the model. Intuitively, we are trying the maximizing the first term, enabling the model to prune out all the incorrect predictions.

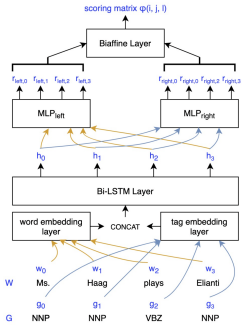

Figure 1: Model architecture

2 Implementation of Inside Algorithm

For inside algorithm, I have implemented **batchified** and **vectorized** version of the inside algorithm. To do this, I have packed the scores for spans of equal width in a tensor. Details of the alogrithm implementation can be found below. The main computation step of the inside algorithm is

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

The above equation can be broken down into two computation tasks:

1 logSumExp(Score[i, j, *])

2 S[i, k] + S[k+1, j]

Computation steps:

- First, I have initialized all the output scores to be negative infinite, since in log space $log(0) = -inf$.
- Next I iterate over each span length from 2 till T.
- I have used stripe function provided in the implementation. Stripe function returns a diagonal rectangular tensor from the input tensor of the specified size.
- Base case is when d is equal to 2. Here I have computed logsumexp of all the diagonal elements of $scores$ at once using , **scores.logsumexp(2).diagonal(offset)**. Since the output scores have been initialized as -inf, their logsumexp will be 0. Therefore we need to compute only $logSumExp(Score[i, j, *])$ for this step. Also, since the inner two loop compute the all the diagonal elements, I have vectorized it using the diagonal function.
- For d $>$ 2, I first compute the second term using the stripe function. I vectorize this by first summing up S[i, k] + S[k+1, j] using the stripe() helper function. This gives results for all k values at once, and we avoid looping over k.
- Since we are computing all the values of S in an iterative fashion, we have information of all the spans of size less than d at each step. We use this information to directly use the value of S[i, k] and S[k+1, j].
- Also, the stripe function helps us compute all values in the inner two loops at once.

3 Results

We observe that for shorter sentences, our model performs exceptionally well. But for longer sentences, the model produces output which are slightly different than the ground truth. This is the reason that we observe low values of Unlabeled and labeled Complete matches, but high F1 score. Via analysing many examples, I have reached the conclusion that at span level, the model performance is decent. But at sentence level, the model makes mistakes in categorizing words in correct phrases. Also there are instances where the parser correctly parses, but the sentence itself had more than one correct parse trees. This has also led in decrease of UCM and LCM. We also observe that the model performance is exceptionally good on sentence length less than or equal to 10. The performance on dev and eval set are given below. Train and Test Loss curves for the experiment can be found in Figure 2.

Dev set Metrics: UCM: 19.82, LCM: 17.71, UP: 84.49, UR: 86.80, UF: 85.63, LP: 82.57, LR: 84.83, LF: 83.68

Eval set Metrics: UCM: 20.99 LCM: 19.33 UP: 85.21 UR: 86.41 UF: 85.80 LP: 83.34 LR: 84.52 LF: 83.92

Eval set Metrics on sentence length less than 10: UCM: 74.07 LCM: 69.63 UP: 93.91 UR: 94.06 UF: 93.98 LP: 90.82 LR: 90.97 LF: 90.89

Results metrics

Figure 2: Loss curve

4 Examples

Given below are set of three example sentences from the experiment. Original & predicted parse trees for these sentences can be found in Table1.

4.1 Example 1

Sentence: Under the agreement signed by the Big Board and Chicago mercetile Exchange trading was temporarily halted in Chicago. Ref Fig a and Fig b. We see that although the sentence is large, the parser does a fairly decent job. It catagorizes the word *temporarily* with *halted* under the same parent phrase VP. However, in the original parse tree the word *temporarily* and *halted* are under separate parent node, also here *temporarily* shares the parent (VP) with the word *was*. Although it is a valid parse, and predicted parse does not change the intended meaning of the sentence, it is catagorized as an incorrect parse.

4.2 Example 2

Sentence: The company currently using about 80% of its North American vehicle capacity has vowed it will run at 100% of its capacity by 1992. Ref Fig c and Fig d. The model makes a mistake in not tagging the word *North American* as an adjective phrase. Also noun phrase in the end of the sentence has been incorrectly identified. Via this example, we can conclude that the model emits sub-optimal parses, larger sentences and also larger spans. To optimize on this aspect, one can try to increase the size of the recurrent layer and also the mlp layer, so that it is able to capture complete context.

4.3 Example 3

Sentence: 28500 common shares via Goldman Sachs. Ref Fig e and Fig f. Here the expected and emitted parses are exactly the same. Since, the sentence is small the grammar used to generate the parse tree is simple and has been correctly learned by the algorithm.

5 Optional Task 1: Removed Tag Embedding layer

As we can see in the metrics below, if we remove tag embeddings from the model, then the model performance drops drastically. This is because tag information is very important in learning the grammar level information of language. Without this the model is just able to learn word mappings but the context information has greatly reduced. This has led to huge decrease in the model performance.

Dev set Metrics: UCM: 8.29 LCM: 7.12 UP: 76.82 UR: 72.27 UF: 74.48 LP: 71.82 LR: 67.55 LF: 69.62

Eval set Metrics: 13.91 LCM: 10.55 UP: 80.55 UR: 79.71 UF: 80.13 LP: 76.65 LR: 75.85 LF: 76.25

Optional task 1 metrics

6 Optional Task 2: Used single MLP layer for boundary representation

We see a decrease in the performance of the model, but the drop is only marginal. As discussed above, the utility of the MLP layers is to trim down all the unnecessary information from the BiLSTM layer and to only keep the information required to learn the grammar. In this case, I have used only a single MLP layer and fed both the forward and backwards hidden weights into a single linear layer. We observe that the drop in performance is not significant, which signifies that a single MLP layer is still able to learn the required context. It signifies that the

Table 1: Generated parse trees of Examples listed in section 4

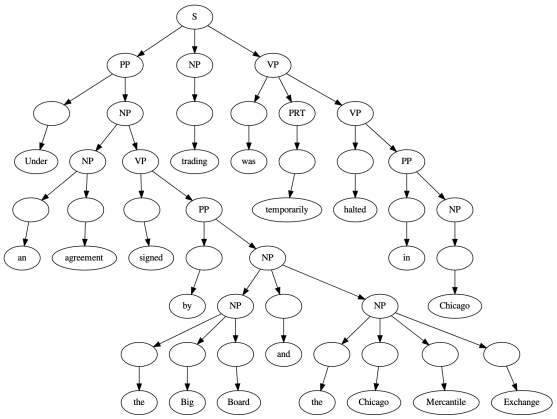

Fig a. Example 1 Original tree

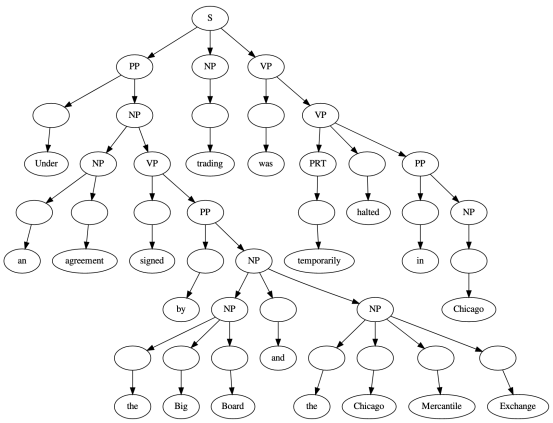

Fig b. Example 1 Predicted Tree

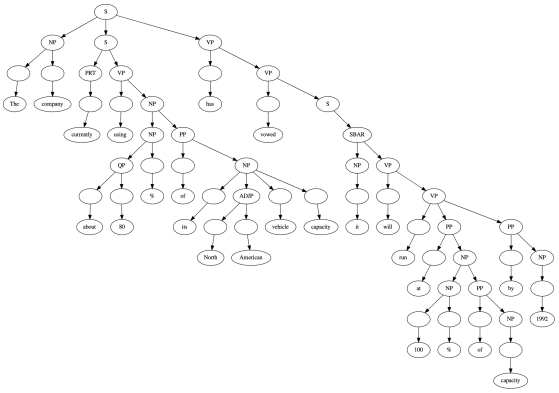

Fig c. Example 2 Original tree

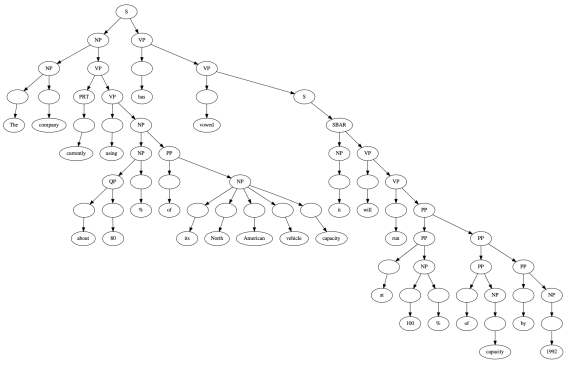

Fig d. Example 2 Predicted Tree

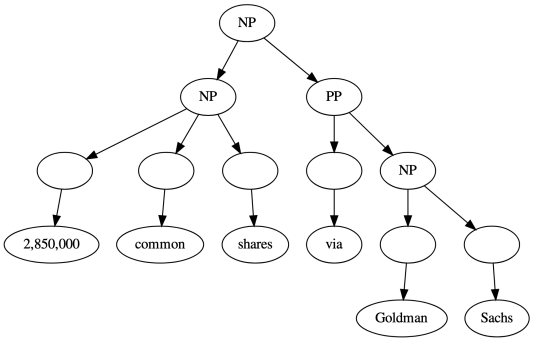

Fig e. Example 3 Original tree

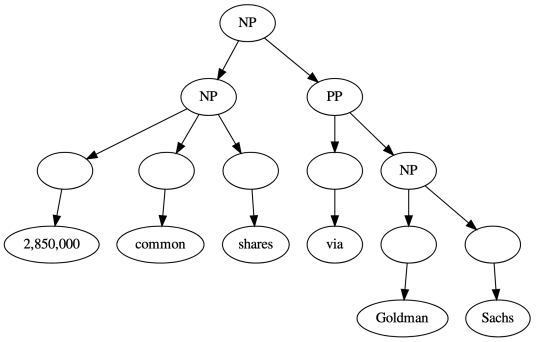

Fig f. Example 3 Predicted Tree

the hidden layers in both the directions have similar information encoded.

Dev set Metrics: UCM: 20.00 LCM: 18.18 UP: 84.01 UR: 86.49 UF: 85.23 LP: 82.31 LR: 84.73 LF: 83.50

Eval set Metrics: UCM: 20.41 LCM: 18.79 UP: 83.76 UR: 85.94 UF: 84.83 LP: 81.89 LR: 84.02 LF: 82.94

Optional task 2 metrics

7 Optional Task 3: Glove Embeddings

As, I have already experimented with BERT embeddings in the last assignment, I wanted to experiment with Glove embedding for this project. I have used 50 dimensional embeddings. The performance with these embeddings are slightly better than the original implementation. This is because Glove embeddings are trained on large dataset and have larger context built in them.

Dev set Metrics: UCM: 19.41 LCM: 18.29 UP: 85.12 UR: 85.57 UF: 85.34 LP: 83.18 LR: 83.63 LF: 83.40

Eval set Metrics: UCM: 21.19 LCM: 19.45 UP: 84.91 UR: 85.58 UF: 85.24 LP: 83.09 LR: 83.74 LF: 83.41

Optional task 3 metrics

8 Optional Task 4: Neural Grammar

I have used the deep biaffine parser based on the approach described in the paper **Deep Biaffine Attention for Neural Dependency Parsing**[3]. The metrics for this are slightly better than our base algorithm as it is able to capture more dependencies as compare the biaffine function provided in the implementation.

Dev set Metrics: UCM: 21.12 LCM: 19.65 UP: 85.72 UR: 87.13 UF: 86.42 LP: 83.94 LR: 85.32 LF: 84.62

Eval set Metrics: UCM: 22.27 LCM: 20.57 UP: 85.85 UR: 86.53 UF: 86.19 LP: 84.09 LR: 84.75 LF: 84.42

Optional task 4 metrics

9 Conclusion

I have presented implementations of Neural conditional randon fields for constituency parsing using batchified and vectorized version for Inside algorithm. I have analysed and identified the trade-offs between sentence structure, length, and model performance. I have experimented with simplifying the neural architecture by removing tag embedding later and using one MLP layer to model boundary representation. Without tag embedding layer, there was huge decrease in model performance. Using only single MLP layer did not significantly effect model performance. With Glove, a pretrained language model, I observed a slight improvement of the model performance. Lastly, a deep biaffine parser based approach also resulted in a small improvement of the overall performance.

10 References

[1] Greg Durrett and Dan Klein. Neural CRF parsing. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 302–312, Beijing, China, July 2015. Association for Computational Linguistics.

[2] Yu Zhang, Houquan Zhou, and Zhenghua Li. Fast and accurate neural CRF constituency parsing. In Proceedings of IJCAI, pages 4046–4053, 2020.

[3] Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017.