Project 3

Neural Conditional Random Fields for Constituency Parsing

Prerna Trushar Bhavsar PID: A59002324

Abstract

In this project, I have implemented the inside function in the forward algorithm of the CRF constituency parser. I analyse the results of the same using various performance metrics. I also perform analysis for shorter sentences dataset. Further, I visualize some sample trees predicted by the model to compare the results with the ground truth labels of the same samples.

1 Introduction

Constituency Parsing is defined as a problem that deals with assigning a syntactic structure to a sentence. It is a type of syntactic parsing. The task creates a parse tree to explain the structure of the sentence. Parse trees can be used in varied applications like grammar checking, semantic analysis, question answering, etc. One of the major concerns while performing constituency parsing is having to deal with ambiguity. Structural ambiguity is a type of ambiguity where the parser can assign more than one parse to a sentence. There are two common structural ambiguities, viz., attachment and coordination ambiguity. Attachment ambiguity can be defined as a sentence which can have a particular constituent being attached to more than one place in a parse tree. Coordination ambiguity is when phrases can be joined by conjunctions like *and*. This causes the sentence to have multiple interpretations. For example, old men and women can be read as [old [men and women]] or [old men] and [women].

Ambiguities like these make it difficult to work with structural parsing. These problems affect all parsers. The Cocke-Kasami-Younger (CKY) algorithm is a dynamic programming approach that is designed to deal with these structural ambiguities. The CKY algorithm is a parsing algorithm for context free grammars. In this project we use the CKY algorithm to efficiently represent a set of parse tree for a sentence. I then implement the inside function to calculate the partition function value for computing the negative log likelihood. The CKY decoding is used to get best scoring constituency tree. The model build to effectively perform the constituency parsing is a BiLSTM-CRF.

2 Method

2.1 BiLSTM-CRF for Constituency Parsing

ThebBiLSTM-CRF model takes in an input sentence containing T words and returns the best predicted constituency parse tree. The input is in the form of $X = \{W, G\}$, where W are the words and G are the Part-of-Speech (POS) tags. The output constituency parse tree is denoted by Y, where $(i,j,l) \in Y$. Here (i,j,l) refers to each constituent spanning over words from i to j and l is the syntactic label.

The model uses the input sentence's tag embeddings and word embeddings. These are concatenated to get the final input sequence that is used as input to the BiLSTM layer. The hidden states are computed in both the forward and backward directions in the BiLSTM. We then create boundary representations by using to Multi-layer Perceptrons (MLPs) on the hidden states. This gives us feature representations for left and right span endpoints.

$$r_i^{left} = MLP_{left}(h_t) \quad (1)$$

$$r_i^{right} = MLP_{right}(h_t) \quad (2)$$

A biaffine scoring function is used to find the score of each span (i,j). This is done by using the biaffine operation on both the left and right feature representations. The score for a span (i,j) is given as,

$$\phi(i,j,l) = exp(Biaffine(r_i^{left}, r_i^{right}, l)) \quad (3)$$

In the neural CRF implementation, all labels are independent given X. The BiLSTM will compute

the labels for spans independently. Thus for a given sentence X, its score is given as,

$$\Phi(X, Y) = \Pi_{(i,j,l)\in Y}\phi(i, j, l) \quad (4)$$

$$logP(Y\|X) = log\frac{\Phi(X, Y)}{\sum_{Y'\in O(X)}\Phi(X, Y')} \quad (5)$$

Equation (5) refers to the conditional log likelihood for the model. In order to learn the model parameter, we minimize the negative log likelihood.

$$NLL = \sum -logP(Y^*\|X;\theta)$$
$$= \sum -[log\Phi(X,Y^*) - log\sum_{Y'\in O(X)}\Phi(X,Y')] \quad (6)$$

For the decoding we use the CKY function which implements a dynamic program to compute the best scoring parse tree. It takes in the local scoring potentials $log\phi(i, j, l)$, computed through the model and finds the predicted parse tree which has the maximum score.

2.2 Inside algorithm

In Equation (6), the partition function defined as $log\sum_{Y'\in O(X)}\Phi(X,Y')$ is computed by the inside function. The implementation of inside() is similar to the implementation of CKY function for decoding. It is a dynamic program that computes the log partition function value of all valid binary trees for the particular input sequence *X*. Here $\Phi(X, Y')$ is normalization constant for the CRF parser.

In my implementation, I have coded a vectorized and batched version of the pseudo code. The input to the function is the scores, which is a tensor of the form [batch_size, seq_len, seq_len, n_labels]. It consists of the local log phi scores computed by the model. The output is *s*, which is a tensor of shape[seq_len,seq_len, batch_size]. Initially we initialize *s* with -inf. In order to compute the values of *s* over each span *(i,j)* we use the diagonal function of pytorch and compute the logsumexp for these values. To compute the values for each split joint of span *(i,k)* and *(k+1,j)* we use the stripe function to compute the values for *s[i,k]* and *s[k+1,j]*. The output partition function value is used to minimize the negative loss likelihood for learning.

3 Results and Discussion

3.1 Model Results

In order to evaluate and discuss the results of the model, I trained the BiLSTM-CRF parser for 10 epochs. The code implemented was batched and vectorized. The run-time was around **20-21s** per epoch. The model achieved a validation loss of **1.06**. Figure-1 shows the training and validation loss per epoch. Further, Figure-2 shows the training loss for every 50 iterations (step).

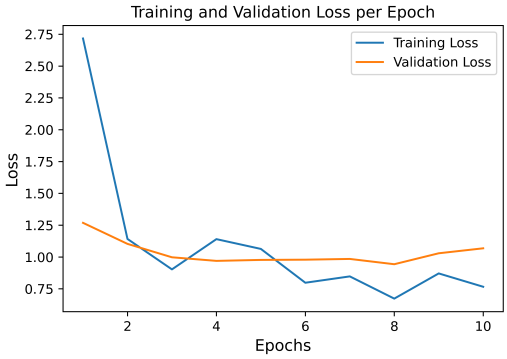


Figure 1: Loss Curves

Figure 2: Training Loss Step Curve

3.2 Evaluation Metrics

Metrics	Complete Match	Precision	Recall	F1 Score
Labeled	18.41	83.83	83.70	83.76
Unlabeled	19.82	85.59	85.46	85.53

Table 1: Model Evaluation Results

The metrics used for evaluation are labeled and unlabeled cases. For the labeled parameter, the predicted parse tree is considered to be correct if the start and end indices as well as the label predicted is correct, i.e. if the ground truth label is NP(2:4) and the model predicts VP(2:4) then this will not be considered as a correctly predicted span in the case of labeled metrics evaluation. But, for unlabeled metrics evaluation we do not see the label assigned to the span and need only the start and end indices to be correct in order to classify that

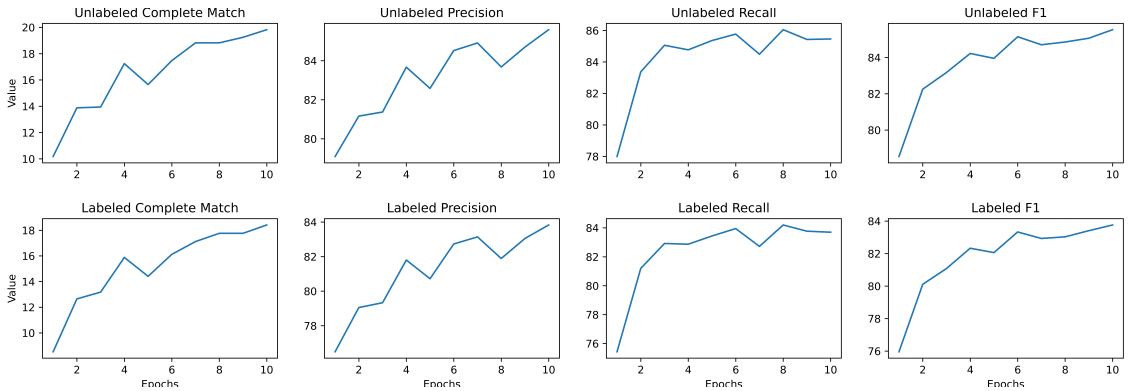


Figure 3: Evaluation Results Per Epoch

predicted output as a match. Thus, for the above example, it will be considered a match in case of unlabeled metrics evaluation.

The epoch with the best metrics is saved. This is used for analysis in Table 1. Table 1 shows the results of the precision, recall and F1 score for labeled and unlabeled metrics. Complete match refers to the percent of input sentences, whose predicted parse trees were completely correct with respect to the ground truth labels. In general, a trend is observed where the values for the evaluations are comparatively higher for the unlabeled metrics. For complete match percent we see that the unlabeled metric yields approximately **1.4** points higher percentage. For the rest of the parameters we see an approximate **2** points increased percentage for the unlabeled data. This is because the check for the predicted parse tree is more strict in the case of labeled metrics as discussed above. Figure-3 shows the evaluation results per epoch and the trends observed in it. Overall, I observed that the values for the precision, recall, and F1 scores for both the metrics gave high values thus showing that the model is behaving well on held out data.

3.3 Analysis on Shorter Sentences

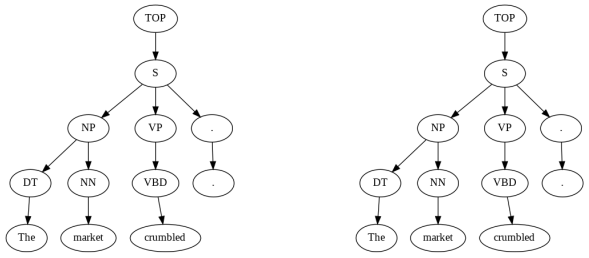
Metrics	Complete Match	Precision	Recall	F1 Score
Labeled	69.33	91.91	91.80	91.86
Unlabeled	71.78	94.61	94.49	94.55

Table 2: Evaluation Results for Shorter sentences

In order to understand the model performance further, I trained the model on the dataset of 2000 input sentences of lengths less than 10. The Table-2 shows the evaluations for the model results on this dataset. We can observe that the model performs much better on this data and precision, recall, and F1 scores for both the labeled and unlabeled metrics have higher percentages. Further, we see that there is a high percentage i.e. **69.33** and **71.28** of complete matches for labeled and unlabeled metrics respectively. These results show that the model performs well on shorter sentences data.

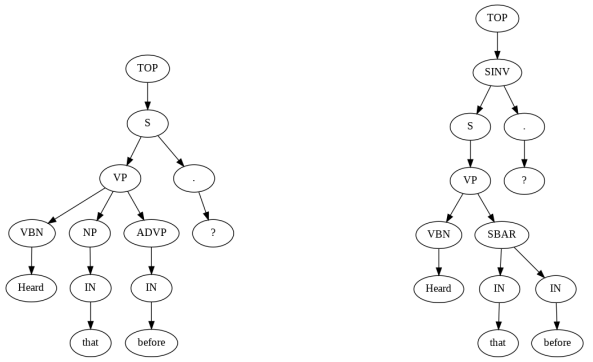
3.4 Sample Analysis

I analyzed some of the predicted parse trees by comparing them to their ground truth labels. Figure 4 and 5 show two of the samples.



(a) Ground Truth Labels (b) Predicted Labels

Figure 4: Sample 1



(a) Ground Truth Labels (b) Predicted Labels

Figure 5: Sample 2

In Figure-4 we see that the model was able to correctly predict the span labels as well as the start and end indices completely, thus giving a complete match. Figure-5 shows an example where the model fails to predict the parse labels correctly. In this we see that the predicted parse tree is only able to get non-terminal to terminal pairs right. This shows that the model struggles to deal with ambiguous statements where the sentence or label can have multiple non-terminals that can be assigned to it.

4 Conclusion and Future Work

In this project, I implemented the vectorized and batched version of inside algorithm for computing the partition function which is used in finding the NLL loss. Further, I evaluated the model results by comparing the precision, recall and F1 scores. I also performed some sample analysis which showed that the BiLSTM-CRF model is performing well on the held out data. The model also showed good results on the sentences with shorter lengths. Further extensions to this project could be to change the architecture of the model and compare the results with the current model. We could also experiment with the Biaffine function to see how these changes may affect the model performance.