Neural CRFs for Constituency Parsing

Murali Mohana Krishna Dandu
UC San Diego (PID: A59004607)
mdandu@ucsd.edu

Abstract

This documents details the implementation results of a Bi-LSTM CRF model for Constituency Parsing using the WSJ Penn TreeBank dataset. The inside algorithm for global normalization and loss calculation was implemented which works in tandem with the CKY decoding algorithm. The results are analyzed using different metrics across different sentence types.

1 Introduction

Before Deep Learning era, graph based parsing relied on many hand-crafted features for maximizing the scoring function. Neural mechanisms like Bi-LSTM layers and Attention scoring functions have been proven to be able to learn these features automatically in the network using the word, character and POS embedding representations. However, independent classification layers on top of RNN features are locally optimized i.e., they don't learn the dependency tree structure between the label tags. Hence, the Neural + Tree CRF combines the best of both worlds, using a globalized structural training loss, maximizing the conditional probability of entire tree-y given x.

2 Approach

This section briefly discusses the network architecture of the Bi-LSTM Tree CRF along with the inside algorithm for structural training loss calculation.

2.1 Architecture

The architecture contains Bi-LSTM, MLPs and Biaffine layer to calculate the scoring function ϕ for the learning task. This scoring function is used by the Tree CRF layer to learn the constituency structure among words. The Bi-LSTM layer takes the word and tag embeddings as input (note that this problem statement takes the tags also as input along with the words) to provide the bi-directional hidden states. To represent the boundary features of the words, we use two MLPs (left and right) which takes the concatenated hidden states as inputs. Now we create the Biaffine scoring function for each candidate span and label combination by using the left/right span representations and weight matrix for labels. We will use this scoring function to define our conditional log-likelihood for our tree CRF model.

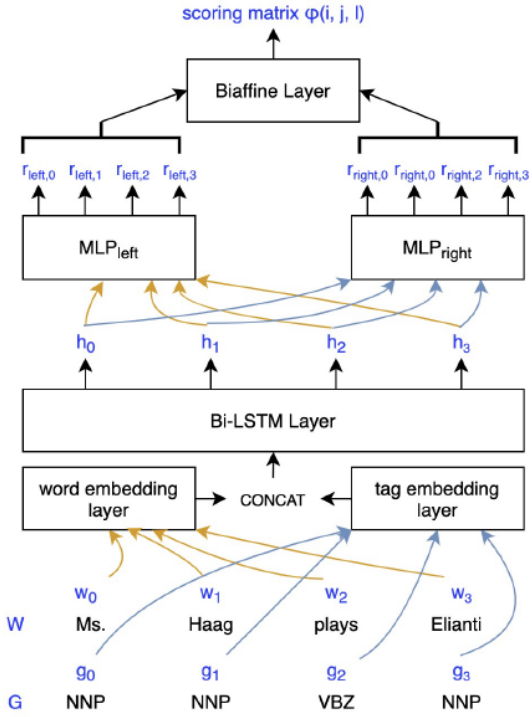


Figure 1: BiLSTM CRF Architecture

$$logP(Y|X) = log\frac{\Phi(X,Y)}{\Sigma_{Y' \in \bigcirc}\Phi(X,Y)}$$

2.2 Inside Algorithm

During learning, we estimate the network parameters θ in such a way that it minimizes the negative log likelihood over the entire sequences in the dataset. The right side of the equation is called the partition function which calculates all the scores for all possible output combinations. We use the inside algorithm based on dynamic programming in log space to calculate this.

$$NLL = \Sigma - [log\Phi(X, Y^*) - log\Sigma_{Y' \in \bigcirc}\Phi(X, Y')]$$

- We start with the scores which is a seq_len size square matrix extended for all possible labels. Since we are calculating the score for batch of sentences, we will have batch as the fourth dimension
- We initialize matrix s with negative infinity and update them with cumulative logsumexp values as we move from the main diagonal to the corner.
- We start with offset=1 which represents sequence length 2. We calculate label scores by logsumexp across seq_len * seq_len dimensions diagonal to create s_label
- The stripe function is used to create diagonal stripe of s across different span lengths
- Finally, we add the s and s_label according to the pseudo-code logic and take the offset diagonal

3 Results and Discussion

3.1 Overview

The above BiLSTM-CRF is trained for 20 epochs using the following hyper-parameters: 128 batch size, 200 lstm hidden dimensions with 2 lstm layers. Figure 2 shows that the NLL loss consistently decreases for the training data but the evaluation loss starts increasing after around 8 epochs. This may suggest overfitting as the training data is small. However, we can observe that the loss is not completely correlated with the metrics that we want to optimize like Complete Match % and F1.

Parsers are usually evaluated using a) Complete Match: which checks the entire match of all constituents at sentence level. b) F1 score: which checks the constituent level match across all sentences combined. Both the metrics are again evaluated with and without labels for different use cases

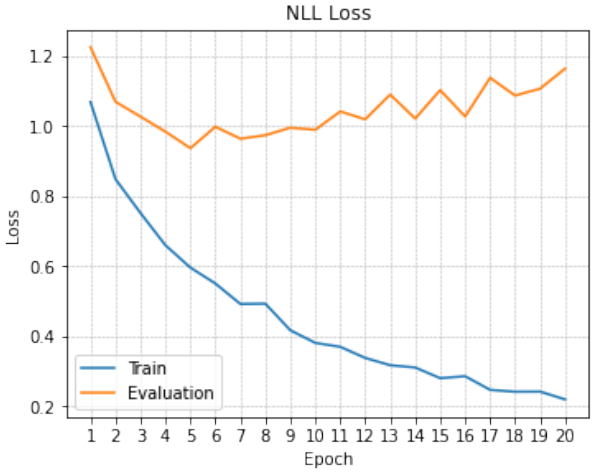


Figure 2: Training and Evaluation Loss (PTB first 2000)

and strictness criteria. Unlabelled match checks whether the span is predicted properly irrespective what name is given to the span and labelled checks the label correctness as well. Figure 3 shows the complete match and F1 metrics (both labelled and unlabelled) at each epoch. We can see that all the evaluation metrics follow a very similar trend and reaches saturation after 12 epochs.

Table 1 shows the metrics for the best epoch that gives the maximum labelled F1 score. We can observe that Complete Match % is around 20% with unlabelled being 2% points higher. The labelled F1 is 84.7% with unlabelled being almost 2% points higher. Precision and Recall are very close implying that model doesn't suggest bias towards false positives vs false negatives. We also observe that the gap in Complete Match of train vs evaluation set is much higher compared to F1 metric. This suggests that the overfitting effects complete match much adversely as it is requires the entire sentence to be correct.

Metric	Train	Evaluation
UCM	43.7%	21.2%
LCM	41.0%	19.2%
UP	94.8%	86.3%
UR	94.9%	86.3%
UF	94.8%	86.3%
LP	94.1%	84.7%
LR	94.2%	84.6%
LF	94.1%	84.7%

Table 1: Evaluation Metrics (PTB first 2000)

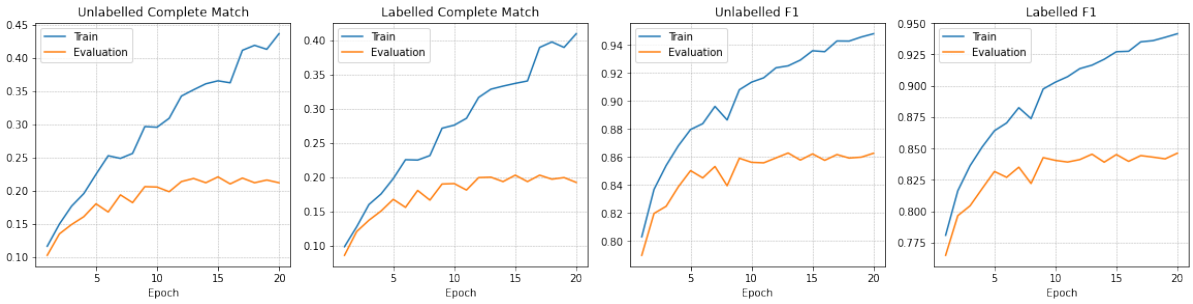


Figure 3: Evaluation Metrics (PTB first 2000)

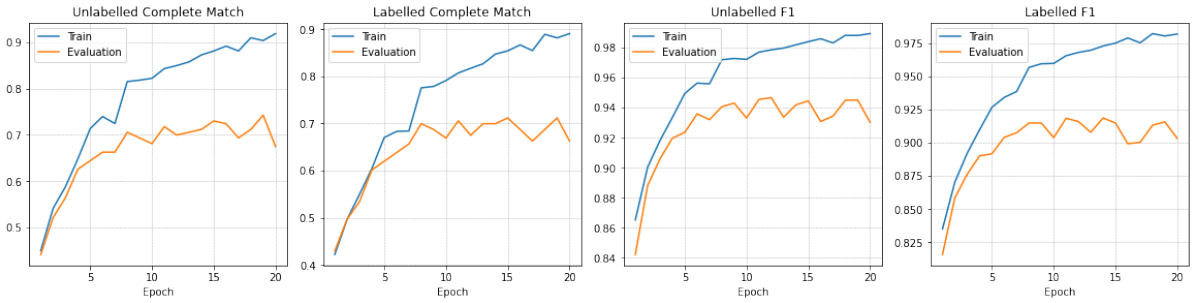
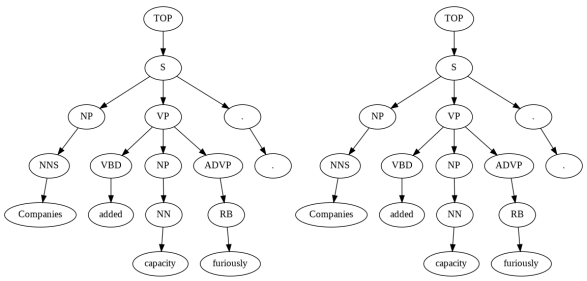


Figure 4: Evaluation Metrics (PTB less than 10)

3.2 Sentence Length Analysis

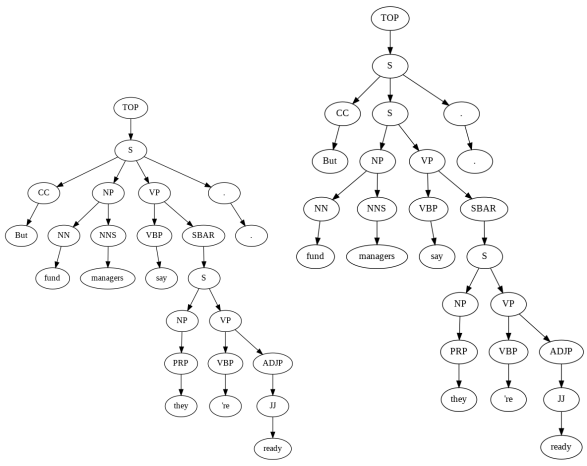
To understand the performance of the model on shorter sentences, I trained and evaluated the model on sentences with less than 10 tokens. Same hyper parameters as above are used for accurate comparison.

Figure 4 shows the training and evaluation metrics for 20 epochs. We can see the metrics are more fluctuating here than the first_2000 sentence dataset mostly due to the very less sample size. Also, the evaluation complete match is 70% compared to the previous 20% and F1 is in the range of 92-94% compared to previous 85%. Given these results, we can say that the model very well with short sentences compared to a generic one as it is difficult to capture long range constituents.



(a) Ground Truth Labels (b) Predicted Labels

Figure 5: Sample 1



(a) Ground Truth Labels (b) Predicted Labels

Figure 6: Sample 2

3.3 Sample Results

Figure 5 and 6 showcase the ground truth vs predicted constituency trees for couple of sentences. For the first example, we can see that the span, depth and constituent are accurately predicted and we observe this trend for most of the smaller length sentences. For the second sample, although the lower parts of the tree are correctly predicted, there is incorrect start symbol added without the first and last token. The tree in itself makes sense without ruining the entire syntax, there is still room for improvement to capture dependencies.

4 Conclusion and Future Work

I have implemented the batched inside dynamic programming algorithm utilizing PyTorch's diagonal and striping operations for our Bi-LSTM Tree CRF model. The results become stabilized after 10 epochs and achieves a labelled F1 score of 85%. We have also seen that the metrics are much better for shorter length sentences. The future work will include understanding which label classes are inaccurately predicted and use that information to develop features in the model. Also, we can pretrained BERT model and char-level LSTM layers to get better feature representations before passing them into MLP and Biaffine functions.