Implementation of a relation extraction LSTM based model on a sequence and tree structure

1.0 Goal

The goal is to implement the LSTM-ER model introduced by Eger et al. [1] for end-to-end argument mining from textual data. Miwa and Bansal originally developed this model for end-to-end relation extraction [2].

The original plan was to implement the LSTM-ER model using PyTorch and test it using an existed semi-finished argument mining pipeline. Unfortunately, the pipeline has not been completed; thus, to test my implementation, I built a relation extraction pipeline based on Miwa's and Bansal's original work [2].

2.0 Challenges

1. Understanding the LSTM that works on tree structure: to be able to implement it. Instead of building a tree data structure that can work with PyTorch from scratch, I used a Python library called DGL to build graph neural network models.
2. The paper has missing and vague points. For example, that paper did not discuss how loss is calculated. Also, the method used to predict the relation labels was unclear. For a detailed discussion, see Section 4.3 Changes and pitfalls.
3. Working on a code written by someone else. I learned some practical, non-technical skills, like understanding code written by someone else, coping with code changes, and communicating with others to show how codes work.

3.0 Background

3.1. Introduction

Relation extraction (RE) is a subtask of information extraction (IE) that aims at detecting and classifying semantic relationships present in texts between two or more entities. In the end-to-end extraction type, the task usually consists of entity detection and classification and relation detection and classification for the detected entities. The code with the running instructions can be found here.

3.2. The shortest path hypothesis

In 2005 Mooney and Bunescu introduced a hypothesis related to determining a relationship between two mentioned entities located in the same sentence. The hypothesis suggests that the shortest path between two entities in the undirected dependency graph almost solely encodes the information required to determine the relationship between these two entities [3].

The hypothesis has three assumptions [3]; first, it assumes that the entities are mentioned in the same sentence. Second, the information needed to extract the relationship is independent of the text preceding or following the sentence. Finally, nodes represent the words in the undirected dependency graph while the edges represent the dependency.

4.0 Model

4.1. The authors' motivation

4.1.1. Neural Network Structure

The authors utilize the shortest path hypothesis discussed in Section 3.2. Originally the hypothesis was introduced and tested using a kernel-based model [3]. Other studies successfully use the shortest path hypothesis using neural-based models [2]. In that sense, the proposed model utilizes the dependency graph. However, the authors argued that only relying on dependency information is not enough. As proof of concept and to make the analysis more manageable, I checked the dependency relations between entities where the shortest path is only two nodes (accounts for nearly 30% of the SemEval-2010 Task 8 dataset). Figure 1 shows that the nominal modifiers "nmod" is dominant across all relations, making it more challenging to establish a relation between entities using only the dependency information.

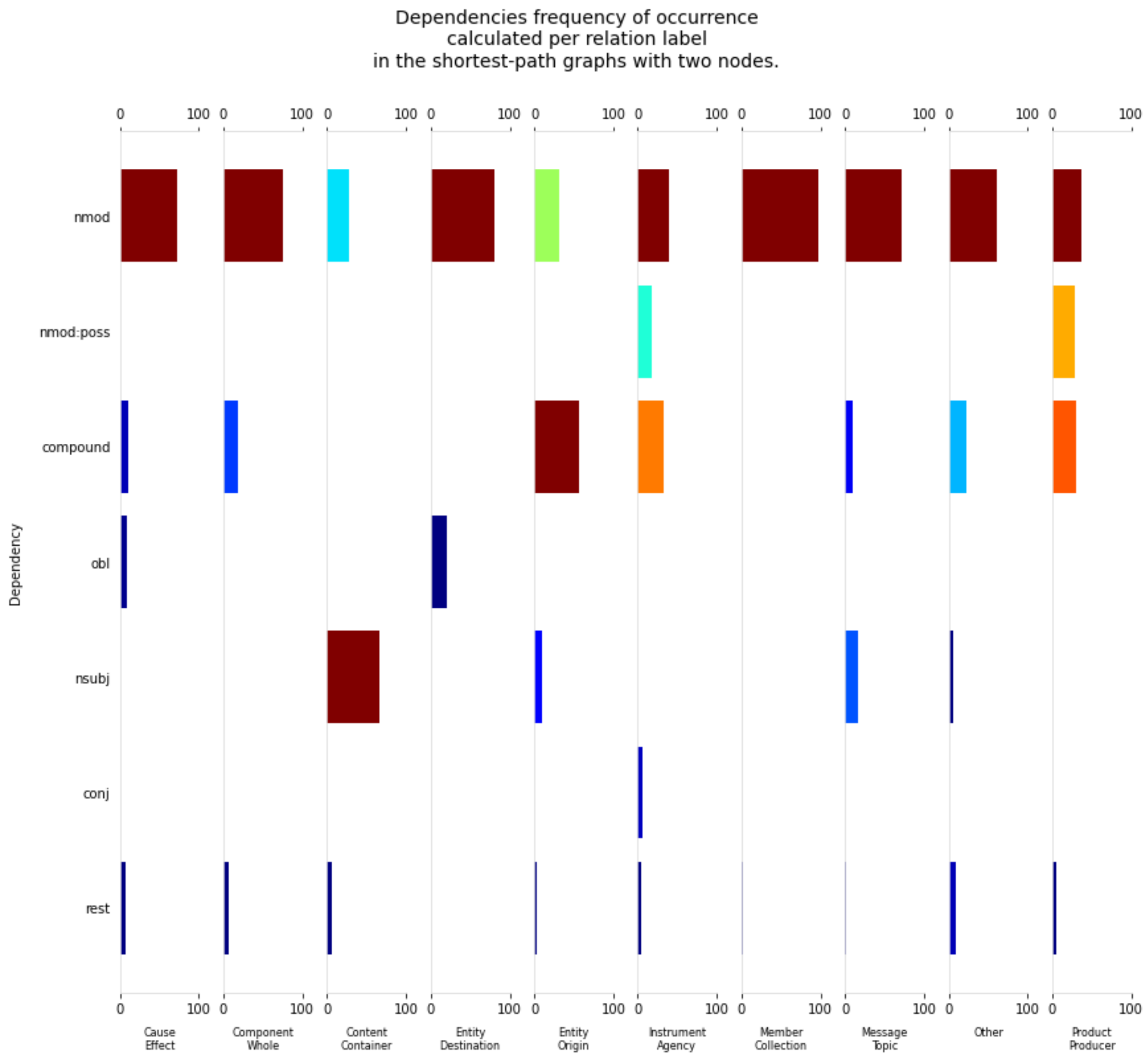


Figure 1:

Moreover, studies preceding Miwa's and Bansal's work [4]–[6] show performance limitations when extracting relation between two entities using LSTM neural networks [2]. The authors suggest that having such low performance is due to focusing only on utilizing one linguistic structure (a tree structure in this case) to extract the relation. The authors were able to push this low performance to exceed state-of-the-art (when publishing their study) by jointly modeling both entities and relations utilizing both sequence and tree LSTM.

4.1.2. Tree structure

Unlike sequential LSTMs, the tree-LSTMs do not process the words in sequential order. Instead, they process them according to their location in the tree data structure representing the complex linguistic unit, like the dependency tree in our example. Tai et al. introduce two types of tree-LSTMs: the child-sum and N-ary LSTMs [7]. N-ary tree-LSTM needs a fixed number of children for each node; thus, it is ideal for processing binarized constituency trees. The child-sum can deal with different numbers of children, where the state for each child has its weight in the forget gate. Thus, the child-sum LSTM can selectively demolish or include the states of each child node. When operating on the dependency tree, the child-sum LSTM can attend to certain dependency relations more than others.

However, the goal is to attend to all nodes that belong to the shortest path. This goal is achieved by sharing weights for the nodes that belong to the shortest path and assigning different weights for all other nodes.

4.2. The Original Model

The model consists of the following modules:

1. **A sequence layer**: a bidirectional one-layer LSTM. The layer creates learning embeddings for words and POS tags. The word embeddings are initialized by the pre-trained word2vec embeddings trained on Wikipedia.
2. **An entity detection layer**: two fully connected layers stacked above the sequence layer. The first layer has a tanh activation function. The layer detects the entity labels for each token.
3. **A dependency layer**: A bidirectional tree-LSTM creates a learning representation for a tree structure containing the two target words. The tree-LSTM concatenates the dependency label embedding, the corresponding Bi-LSTM hidden states, and the related word label prediction from the entity detection layer. The tree structure could be as follows:
    a. The shortest path between the target words in the dependency tree.
    b. The dependency subtree under the lowest common ancestor of the target word pair.
    c. The full dependency tree of the whole sentence.
4. **A relation label detection layer**: two fully connected layers stacked above the dependency layer. The first layer has a tanh activation function. The layer detects the relation label of the two target words.

When the model detects the relation label in an end-to-end fashion, the dependency layer processes the tree structures for each possible combination of the last words of the detected entities.

4.3. Changes and pitfalls

As discussed in Section 1.0 Goal, I built a pipeline to test my model implementation —mainly reimplementing Miwa's and Bansal's work [2]. Three datasets were used: ACE 2004, ACE 2005, and SemEval-2010 Task 8. Unfortunately, ACE 2004 and ACE 2005 are proprietary datasets, so I used SemEval-2010 Task 8 only.

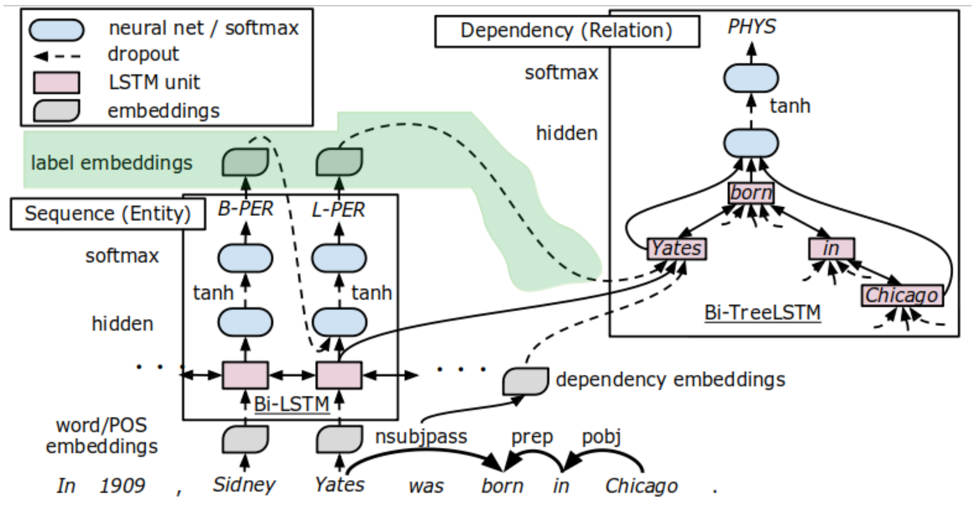


Figure 2: LSTM-ER Model, edited from [2]

As SemEval-2010 Task 8 does not have entity labeling, the implementation pipeline does not support the end-to-end operation. The dependency layer expects to have the target word indices instead of processing the detected entities combinations. Also, I omit the entity detection layer. Figure 2 shows the model architecture.

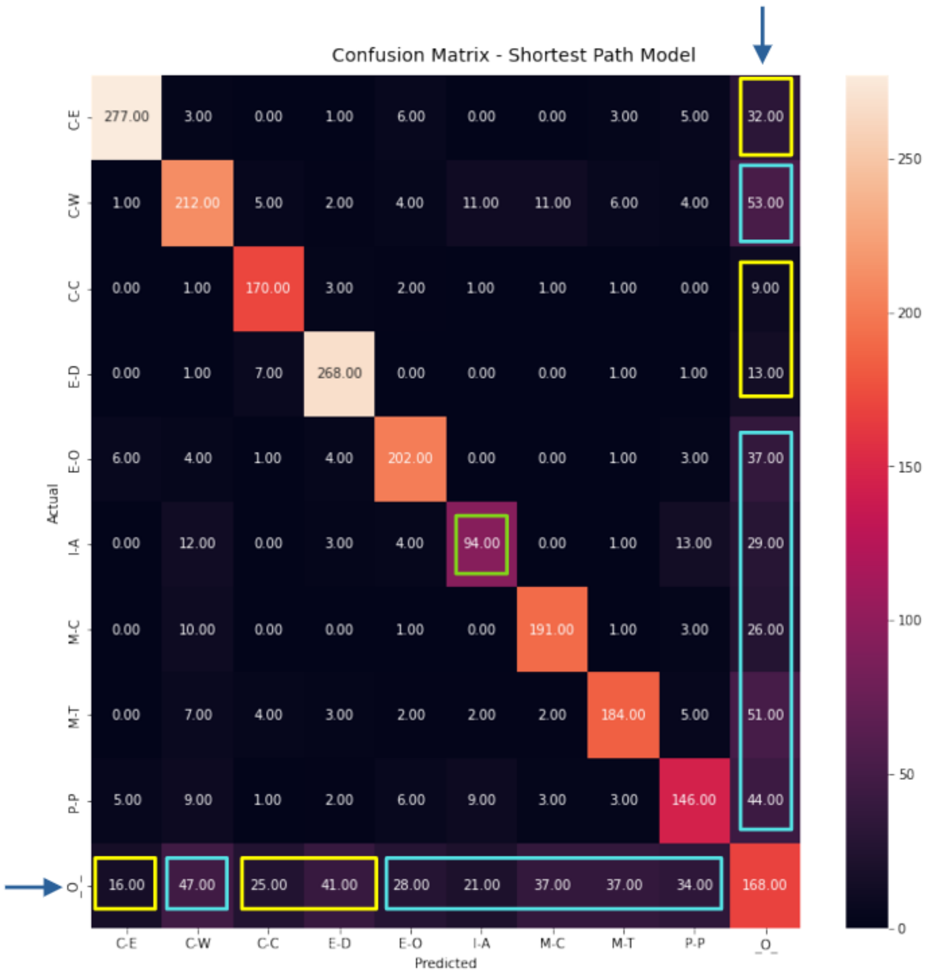
One of the pitfalls of using a "non-end-to-end" pipeline is that the model designed by the authors depends on pre-training of the entity detection module, and skipping the pretraining phase could negatively affect the performance. Also, the dependency layer will not take the detected entity labels as inputs.

I used negative sampling in my implementation. The paper states the following:

> *"we assign two labels to each word pair in prediction since we consider both left-to-right and right-to-left directions."*

The above statement is hard to interpret for me. Suppose we have detected/have two entities e_{right} and e_{left}, so we will construct two pairs: the first pair (p_1) is $\left(e_{right}, e_{left}\right)$ and the second pair (p_2) is $\left(e_{left}, e_{right}\right)$. The truth relation label we have is $R\left(e_{left}, e_{right}\right)$. According to the statement above, I am unsure how to assign the label to the pairs p_1 and p_2. In my implementation, I assigned the relation label R to p_2 (according to the direction) and assigned the relation label $Other$ to the other pair, which is p_1.

Also, I misunderstood the shortest path hypothesis. I thought I should find the path between the two entities in a directed dependency graph. However, the hypothesis originally stated that the path is extracted from an undirected version of the dependency graph[3]. This affects the implementation because you cannot find a path for a pair, say (w_1, w_2), and the w_2 is directly connected to w_1 in the



Actual \ Predicted	C-E	C-W	C-C	E-D	E-O	I-A	M-C	M-T	P-P	_O_
C-E	277.00	3.00	0.00	1.00	6.00	0.00	0.00	3.00	5.00	32.00
C-W	1.00	212.00	5.00	2.00	4.00	11.00	11.00	6.00	4.00	53.00
C-C	0.00	1.00	170.00	3.00	2.00	1.00	1.00	1.00	0.00	9.00
E-D	0.00	1.00	7.00	268.00	0.00	0.00	0.00	1.00	1.00	13.00
E-O	6.00	4.00	1.00	4.00	202.00	0.00	0.00	1.00	3.00	37.00
I-A	0.00	12.00	0.00	3.00	4.00	94.00	0.00	1.00	13.00	29.00
M-C	0.00	10.00	0.00	0.00	1.00	0.00	191.00	1.00	3.00	26.00
M-T	0.00	7.00	4.00	3.00	2.00	2.00	2.00	184.00	5.00	51.00
P-P	5.00	9.00	1.00	2.00	6.00	9.00	3.00	3.00	146.00	44.00
_O_	16.00	47.00	25.00	41.00	28.00	21.00	37.00	37.00	34.00	168.00

Figure 3: Confusion Matrix – the Shortest Path Mode

dependency graph. Thus, I built one graph for every two pairs and differentiated between the pairs by swapping the location of the hidden state vectors generated by the top-down treeLSTM when concatenating them in one vector to send them to the relation label detection layer. The implementation should consider building a separate tree (shortest path) for each pair.

5.0 Datasets

The SemEval-2010 Task 8 [8] dataset has nine relation labels constructed between two nominals plus the "Other" label for no relation. It has 8,000 training samples and 2,717 test samples. Eight hundred samples are randomly selected from the training dataset to form the development dataset. The dataset has its official scorer, which produced the following:

- Confusion matrix
- Precision, recall, and F1 score for each label
- F1-score (Macro-F1) on the nine relation types.

6.0 Testing and discussion

The implemented model achieved a Macro-F1 score of 0.763 compared to 0.844 reported in the paper. The difference in performance could be due to the differences in implementation discussed in Section 4.3 Changes and pitfalls.

As shown in Figure 3, the –confusion matrix, the Instrument-Agency relation has the lowest true positive value (marked by green), hence the lowest accuracy. However, from the test notebook, we notice that the Other label has the lowest F1 score. By reviewing the confusion matrix, one can notice that the Other label has the highest confusion; see the raw and the column marked by the blow arrow. Also, Figure 3 shows an easy confusion between the "Other" label and all other labels marked by a blue square. Moreover, it is easier to mistake the Entity-Destination and Content-Container labels for the Other label than the vice versa (marked by a yellow square).

My analysis is that because the Instrument-Agency label has the lowest occurrence in the train data test, it has the lowest true positive value. Also, because we add a lot of negative samples "the Other label", the model hardly can differentiate between the Other label and all other labels. Also, using one tree graph per pair (right-to-left and left-to-right) as discussed in Section 4.3 Changes and pitfalls, may contribute to this confusing issue.

7.0 References

[1] S. Eger, J. Daxenberger, and I. Gurevych, “Neural end-to-end learning for computational argumentation mining,” *arXiv preprint arXiv:1704.06104*, 2017.

[2] M. Miwa and M. Bansal, “End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany, 2016, pp. 1105–1116, doi: 10.18653/v1/P16-1105.

[3] R. C. Bunescu and R. J. Mooney, “A shortest path dependency kernel for relation extraction,” in *Proceedings of the conference on human language technology and empirical methods in natural language processing*, 2005, pp. 724–731.

[4] J. Li, M.-T. Luong, D. Jurafsky, and E. Hovy, “When Are Tree Structures Necessary for Deep Learning of Representations?,” *arXiv:1503.00185 [cs]*, Aug. 2015, Accessed: May 25, 2022. [Online]. Available: http://arxiv.org/abs/1503.00185.

[5] Y. Xu, L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin, “Classifying relations via long short term memory networks along shortest dependency paths,” in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1785–1794.

[6] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng, “Semantic compositionality through recursive matrix-vector spaces,” in *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, 2012, pp. 1201–1211.

[7] K. S. Tai, R. Socher, and C. D. Manning, “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks,” *arXiv:1503.00075 [cs]*, May 2015, Accessed: May 25, 2022. [Online]. Available: http://arxiv.org/abs/1503.00075.

[8] I. Hendrickx *et al.*, “Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals,” *arXiv preprint arXiv:1911.10422*, 2019.