Applying Graph Neural Networks for Vietnamese Dependency Parsing

NGUYEN Duc Thien **NGUYEN Thi Thu Trang*** **TRUONG Dang Quang**
Hanoi University of Science and Technology
Hanoi, Vietnam
Óbuda University
Budapest, Hungary
ndthien98@gmail.com *trangntt@soict.hust.edu.vn* *dangquangtruong98@gmail.com*

Abstract

This paper presents a state-of-the-art model to solve the Vietnamese dependency parsing task (HA My Linh, 2020) in VLSP 2020[1] Evaluation Campaign. In this model, the Bidirectional Long Short-Term Memory (BiLSTM) network is used to extract the contextual information, while the graph neural network captures high-order information. Some preprocessing for Vietnamese raw texts are included for the training, such as word segmentation, part-of-speech (POS) tagging for the model.

We modified the network with suitable word embedding mechanisms, i.e., fastText, to represent the semantic information of words more accurately. Therefore, Vietnamese words that are marked as unknown tokens now can have the right embedding; thus, they will be well modeled in dependency parsing.

Experiments on the raw text dataset show that the model achieved an average of 72.85% of unlabeled attachment score (UAS) and 64.35% of labeled attachment score (LAS). With the Segmentation and POS tagging dataset, we achieved a higher average of 81.71% (UAS) and 73.19% (LAS).

1 Introduction

In recent years, dependency parsing is a fascinating research topic and has a large number of applications in natural language processing. This task is to automatically identify the relationship between words in a sentence and label the relationship between the head and the dependency word, and thus, establish the grammatical structure of the sentence. Traditional graph-based dependency parsing only extracts the parent-child relationship and ignores deeper relationships. Hence, we decided to experiment with the idea of extracting deeper relationships of the neighbor nodes, which is extensively covered in the paper Ji et al. (2019).

This state-of-the-art model achieved good performance due to its ability to represent incorrect Out-Of-Vocabulary (OOV) words in the input layer for Vietnamese. Normally, words that are not found in the vocabulary will be marked as unknown tokens before feeding to the embedding layer. This caused the model to embed OOV words incorrectly; therefore, it created the loss of information in calculating attention distribution. In this paper, we modified the pre-trained layer of word embedding for the graph neural networks with a more suitable embedding mechanism for Vietnamese, which solved the issue well.

The rest of this paper is organized as follows: Section 2 presents the architecture and its components of graph neural networks. The experiments are shown in Section 3. Finally, Section 4 concludes the paper and gives some perspectives for the work.

2 Methodology

Normally, Graph-based dependency parser search through the space of possible trees for a given sentence encoded as directed graphs and use methods from graph theory (Maximum Spanning Tree or greedy algorithm) for the optimal solutions. However, in the Graph Neural Network (GNN) model, the dependency parser utilizes the neural network to assign a weight to each edge, then construct a MST from the edge weight (Dozat et al., 2017). For maximum accuracy, we need to analyze the surface form and the deep structure of the graph. There are three main components in the model: Encoder extracts the surface form and the contextual information and turns them into the nodes (words) representations for the next components; The graph attention network (a subset of GNN, using the structure from Veličković et al. (2017)) layers then extract the deep structure and high-order information

*Corresponding author
[1] Vietnamese Language and Speech Processing

to illustrate the head-dependent relationships of the nodes; the final component is the decoder, used to create the dependency tree from the output of the GNN. We will discuss the details in the following sections.

2.1 Pre-processing

First, we used the VNCoreNLP - suggested by Vu et al. (2018) - to segment and perform the POS tagging on the raw text. VNCoreNLP used a transformation rule-based learning model for the segmentation of the Vietnamese document, thus, obtained faster and better accuracy than all previous segmentation tools, as the model accounted for the fact that Vietnamese words are created from syllables including the space character (Nguyen et al., 2017). The VNCoreNLP performed the task of labeling words with POS tag Vu et al. (2018) via MarMot (a CRF framework), state of the art POS and morphological tagger (Müller et al., 2013)

Word embedding is the most popular representation method for words in a document because it captures the context of words, semantic and syntactic similarity, relation with other words, etc. Using word embedding makes it easier to represent words with less memory than using a one-hot vector while also showing the relationship between words.

With a huge training corpus (e.g., a total of 100 billion words with a 3-million-word vocab in Google News), the pre-trained model can cover much more context for word embedding than the auto-updating mechanism of the word embedding in the end-to-end abstractive summarization model with its training corpus (e.g., a total of 240 million words with a 50k-word vocab in Daily Mail/CNN) (Anh and Trang, 2019).

In this paper, we adopted a suitable pre-trained model for Vietnamese with 300-dimensional word embeddings, i.e., fastText from Facebook (Joulin et al., 2016), for the word embedding layer. The fastText trained on the Wikipedia dataset with character n-grams of length 5 by CBOW[2] method. fastText is more suitable in our case as when the GNN model meets unknown vocab, the fastText generates an embedding of the vocab with value 0, resulting in error reductions; meanwhile, the Word2Vec and the GloVe does not do that. This method enables fastText to handle OOV[3] words by constructing the vector for OOV words from its characters.

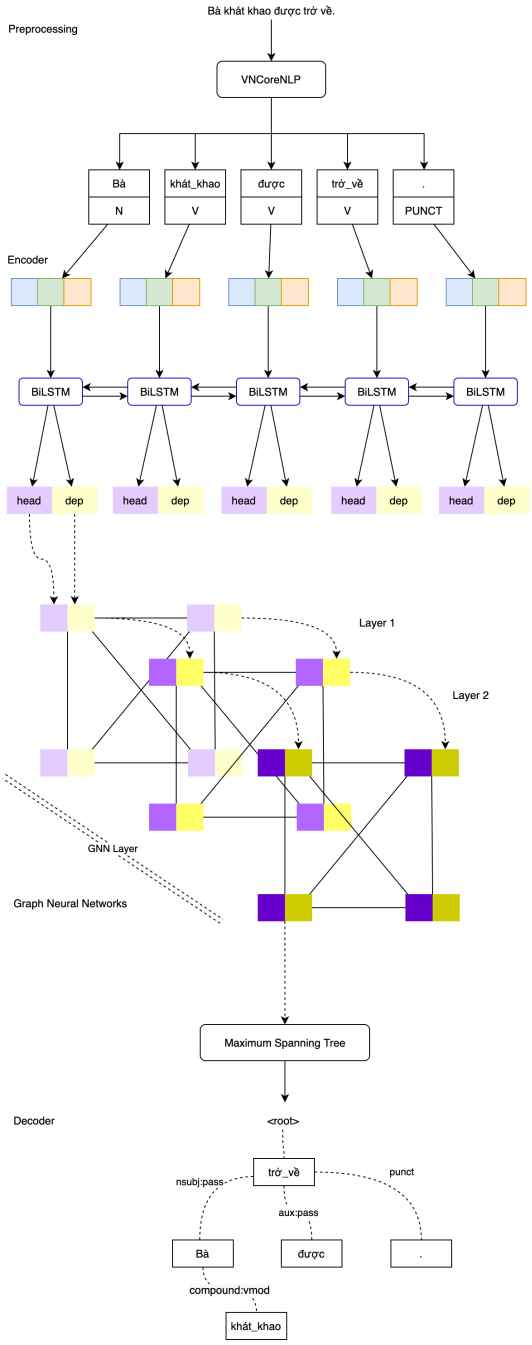

Figure 1: The architecture of Graph Neural Networks

[2]Continuous Bag of Words
[3]Out-of-vocabulary

Both GloVe and Word2Vec are unable to do so.

2.2 Encoder

According to Kiperwasser and Goldberg (2016), we can apply BiLSTM model to create the dependency tree as illustrated in Figure [1] Firstly, each word is embedded using a vector combined from three different vectors: randomly initialized word embedding, pre-trained word embedding, and part-of-speech embedding.

$$x_i = e(w_i) \oplus e'(w_i) \oplus e(pos_i) \quad (1)$$

As a result, the x_i illustrated the sentence of the word i in [2]. Given the position i of the word, the BiLSTM model can compute state vectors $\overrightarrow{c_i}$ and $\overleftarrow{c_i}$ where the $\overrightarrow{c_i}$ is draw from the start of the sentence to the position i and $\overleftarrow{c_i}$ is from the end of the sentence to i.

$$\overrightarrow{c_i} = \overrightarrow{LSTM(x_i)} \oplus \overleftarrow{LSTM(x_i)} \quad (2)$$

The two vectors $\overrightarrow{c_i}$ and $\overleftarrow{c_i}$ then concatenate to become the context-dependent representation of the word i. Thus we can use multilayers perceptron (MLP) to define two-node representations of the word i the probability of being the head role vector and probability of being the dependent role vector (Dozat et al., 2017):

$$\mathbf{h}_i = MLP_h(c_i), \mathbf{d}_i = MLP_d(c_i) \quad (3)$$

The score function is a SoftMax function, where the representations of the word i and j is the input, therefore complementing the analysis of the surface form of the segmented sentence. As a result, the output of the BiLSTM component is a complete weight graph model. (Dozat et al., 2017)

$$\sigma(i,j) = \text{Softmax}_i(h_j^T A d_j + b_1^T h_j + b_2^T h_j) \quad (4)$$

2.3 GNN Layers

In the implementation, the GNN component can utilize at most three layers, each layer consists of 4 graph neural network units as illustrated in Figure [1] - where the representation of the vectors is calculated from the same representation in the previous layer using this formula where g is the *LeakyReLU* function, t is the layer, v_i is the vector representation of i, and a_{ij} is the edge weight of

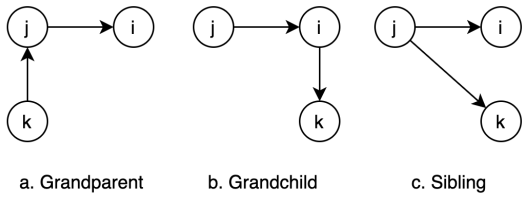

Figure 2: Relations between nodes

v_i and v_j (i and j are forming the neighborhood) (Wang and Chang, 2016):

$$v_i^t = g\left(W \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^t v_j^{t-1} + B v_i^{t-1}\right) \quad (5)$$

We can apply the formula [5] to analyze the high order information of the nodes which is represented in three ways: grandparents, grandchildren, and siblings (Figure [2]) (Eisner, 1997).

Specifically, the head representation of node i should attend to the neighbors' representation as they are the parents of the i. Therefore the model can calculate h_i from the h_j of the previous layer $t-1$ using the formula [5]:

$$\begin{cases} h_i^t = g\left(W_1 \sum_{j \in \mathcal{N}(i)} \alpha_{ji}^t h_j^{t-1} + B_1 h_i^{t-1}\right) \\ d_i^t = g\left(W_2 \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^t d_j^{t-1} + B_2 d_i^{t-1}\right) \end{cases} \quad (6)$$

The dependent node d_i's computation operation is the same as the head node's one i. Thus the equation [6] can assist to analyse the order of the relationship of grandparents and grandchild.

To examine the sibling relationships, the head representation of the node i check the neighborhood where they are dependent on node i. Thus the formula will update the h_i in the following way:

$$\begin{cases} h_i^t = g\left(W_1 \sum_{j \in \mathcal{N}(i)} \alpha_{ji}^t d_j^{t-1} + B_1 h_i^{t-1}\right) \\ d_i^t = g\left(W_2 \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^t h_j^{t-1} + B_2 d_i^{t-1}\right) \end{cases} \quad (7)$$

Finally we combine the two equations [6] and [7] above to update the grandparents, grandchild

Param	Ji et al. (2019)	Our paper
Word Embedding	300 dim	500 dim
POS Embedding	100 dim	100 dim
arc MLP size	500 dim	500 dim
rel MLP size	100 dim	100 dim
Dropout	0.33	0.33
Optimizer	Adam	Adam
Learning rate	0.002	0.002
Graph layers	2	2

Table 1: Hyper-parameter.

and siblings.

$$\begin{cases} h_i^t = & g(W_1 \sum\limits_{j \in \mathcal{N}(i)} (\alpha_{ji}^t h_j^{t-1} + \alpha_{ji}^t d_j^{t-1}) \\ & + B_1 h_i^{t-1}) \\ d_i^t = & g(W_2 \sum\limits_{j \in \mathcal{N}(i)} (\alpha_{ij}^t h_i^{t-1} + \alpha_{ji}^t d_j^{t-1}) \\ & + B_2 d_i^{t-1}) \end{cases} \tag{8}$$

As the equations [8] illustrated, the edge weight a_{ij} is the decisive element responsible for the update of the relationship information. The edge weight is figured with the following formula:

$$\alpha_{ij}^t = \begin{cases} \text{Softmax}_i & (h_i^T A d_j + b_1^T h_i + b_2^T d_j) \\ & i \in \mathcal{N}_k^t(j) \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

2.4 Decoder

After the high-order information is extracted from the GNN and enhanced the nodes representations, the node representation will be used to build the dependency tree via Biaffine parser (the setting is identical to Dozat et al. (2017))

3 Experiments

3.1 Dataset

The VLSP provided the datasets and separated them into training datasets and raw text datasets. The data for training was further divided into two packages: the first package consists of 5070 sentences, with a large domain from the social media comments on restaurants and hotels (100 sentences), to the story of the Little Prince (1570 sentences) and the VietTreeBank - VTB (3400 sentences); the second package includes 3000 sentences with diverse origins.

The raw text data for prediction includes the two packages above, and 20 raw text files crawled from VnExpress news articles. The VTB files and the files with index 1,3,4,7,8,10,14 were accurately tokenized and labeled.

The graph-based dependency parsing neural network model has one important characteristic: the raw text dataset's sentences have to be tokenized for the training to be carried out successfully. Therefore the VNCoreNLP - an NLP pipeline used for POS tagging, named entity recognition and dependency parsing is useful here in this case [4]. This tool is capable of providing highly accurate annotation for the input sentences, therefore improving the score of the training model.

3.2 Training

The training operation consists of two methods: First, we have to decode the output of the final layer of the GNN component (denoted by)

$$\alpha_{ij}^t = \sigma^t(i,j) = P^t(i|j) \tag{10}$$

which are the tree structures (computed by $P(i|j)$) and the dependency edge labels (measured by $P(r|i,j)$, which indicated the probability a tree (i,j) holds a dependency relation r, using another MLP from biaffine parser (Dozat et al., 2017), the loss function of the classifier is computed with the equation:

$$\mathcal{L}_0 = -\frac{1}{n} \sum_{(i,j,r) \in T} (\log P^\tau(i|j) + \log P(r|i,j)) \tag{11}$$

Second, the model can supervise on $P^t(i|j)$ from each layer of the GNN component, therefore the layer-wise loss will be computed with the equation:

$$\mathcal{L}' = \sum_{t=1}^{\tau} \mathcal{L}_t = \sum_{t=1}^{\tau} -\frac{1}{n} \sum_{(i,j,r) \in T} \log P(r|i,j) \tag{12}$$

The main objective is to minimize the loss of combination of them:

$$L = \lambda_1 \mathcal{L}_0 + \lambda_2 \mathcal{L}' \tag{13}$$

3.3 Results

We have implemented and operated the model on the AWS Server (AWS Deep Learning AMI

Dataset	UAS	LAS
Test from VTB	81.89	73.34
VNExpress 1	75.12	66.15
VNExpress 3	84.36	75.38
VNExpress 7	76.67	67.22
VNExpress 8	79.25	71.98
VNExpress 10	80.47	72.54
VNExpress 14	80.55	73.57
Total	**81.71**	**73.19**

Table 2: Test on labeled datasets.

Dataset	UAS	LAS
Test from VTB	73.18	64.66
VNExpress 1	68.77	58.75
VNExpress 3	74.10	65.81
VNExpress 7	61.67	55.56
VNExpress 8	68.96	61.43
VNExpress 10	73.19	64.13
VNExpress 14	68.4	60.72
Total	**72.85**	**64.35**

Table 3: Test on raw-text datasets.

(Ubuntu 18.04) Version 34.0 installed in the EC2 Instance p3.2xlarge - GPU NVIDIA Tesla v100 16 GB, Memory 61 GB, SSD 100 GB, CPU 8 Virtual Cores) successfully. The hyperparameters configuration in Table [1] has slight modifications. For the word embedding, we used fastText (Bojanowski et al., 2016) with Vietnamese data as the primary pre-trained model, which has 300 dimensions instead of 100 dimensions of GloVe that Ji et al. (2019) used. Then, we concatenate the pre-trained word embedding with 200-dimension randomly initialize word embedding and 100-dimension part-of-speech embedding. Randomly embedding vectors obtained from binomial distribution. The training operation took approximately one hour.

The main evaluators for the dependency parsing problem are LAS and UAS. The results are coming from the script evaluator 2018. For the labeled data, the highest UAS is 81.89% from the VTB package, meanwhile the package Test VNExpress 14 achieved the highest LAS 73.57%.

Table [2] shows results from VLSP 2020 private tests for dependency parsing on labeled datasets, meanwhile raw-text datasets' results are shown on Table [3].

4 Conclusion

To conclude, our experiment on using the graph neural network for graph-based dependency parsing suggests that understanding the deep structure of the representations of words via nodes' message passing improved a slightly better accuracy and efficiency than other traditional graph-based dependency parsers. In future works, we are planning to improve the performance of the model by applying Conditional Random Fields in the labeling process for the nodes before extracting the high-order information via graph neural network.

Acknowledgments

The authors wish to thank VLSP organizers for their reviews and encouragement.

References

Dang Trung Anh and Nguyen Thi Thu Trang. 2019. Abstractive text summarization using pointer-generator networks with pre-trained word embedding. In *Proceedings of the Tenth International Symposium on Information and Communication Technology*, pages 473–478.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada. Association for Computational Linguistics.

Jason Eisner. 1997. Three new probabilistic models for dependency parsing: An exploration. *arXiv preprint cmp-lg/9706003*.

VU Xuan Luong NGUYEN Thi Luong PHAN Thi Hue LE Van Cuong HA My Linh, NGUYEN Thi Minh Huyen. 2020. Vlsp 2020 shared task: Universal dependency parsing for vietnamese. In *Proceedings of The seventh international workshop on Vietnamese Language and Speech Processing (VLSP 2020)*, Hanoi, Vietnam.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Thomas Müller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order crfs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332.

Dat Quoc Nguyen, Dai Quoc Nguyen, Thanh Vu, Mark Dras, and Mark Johnson. 2017. A fast and accurate vietnamese word segmenter. *CoRR*, abs/1709.06307.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Thanh Vu, Dat Quoc Nguyen, Dai Quoc Nguyen, Mark Dras, and Mark Johnson. 2018. VnCoreNLP: A Vietnamese natural language processing toolkit. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 56–60, New Orleans, Louisiana. Association for Computational Linguistics.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315, Berlin, Germany. Association for Computational Linguistics.