Multi-layer Biaffine Model for Neural Dependency Parsing

Anonymous ACL submission

Abstract

The biaffine model is a strong and efficient model for graph-based dependency parsing. However, previous work only used the biaffine method in single-layer form. In this paper, we propose a multi-layer biaffine model for neural dependency parsing. In this model, we modify the biaffine method so that it can be utilized in multi-layer form. We evaluate our model on PTB and CTB and show our model achieves state-of-the-art results on both datasets. Further experiments show the benefits of introducing multi-layer form into the biaffine method with low efficiency loss.

1 Introduction

Dependency parsing is a fundamental task in NLP. Given a input sequence $\boldsymbol{s} = w_0 w_1 ... w_n$, the output is a dependency tree $\boldsymbol{t} = \{(h, d, l), 0 \leq h \leq n, 1 \leq d \leq n, l \in L\}$, where w_0 is a pseudo-word as root and (h, d, l) is an arc from w_h to w_d with label l in a relation set L. Due to the simplicity and effectiveness of representing syntactic information by using the tree structure, many works involve dependency parsing, such as syntax-enhanced pre-trained model (Xu et al., 2021).

Biaffine first-order graph-based parser (Dozat and Manning, 2017), which utilizes biaffine mechanism and training via local head selection for each token, is frequently used in dependency parsing due to its high performance and efficiency. There are two ways to extend the biaffine model in previous work, one is to extend the biaffine method to triaffine for modeling second-order information (Zhang et al., 2020; Wang and Tu, 2020), the other is to modify the encoder (Li et al., 2019; Mrini et al., 2020). However, no previous work has tried to introduce multi-layer form into the biaffine method.

In this paper, we propose a multi-layer biaffine method for dependency parsing. We modify the biaffine model into a connectable layer form. The output matrices of the biaffine model are used as weight matrices to construct the new token representation. The representation contains the information of head, dependents, and labels provided by the biaffine mechanism, and is used as the input token representation for the next layer. Experiments on PTB and CTB show that our model achieves state-of-the-art results on both datasets and has advantages over the single-layer biaffine model. Our model also has a low efficiency loss compared with the single-layer biaffine model.

2 Model

Our graph-based dependency parser consists of two parts, i.e., Encoder and Multi-layer biaffine model.

2.1 Encoder

Encoder consists of Embedding layer and BiLSTM layer. In Embedding layer, input token w_i with Part-of-speech tag p_i are used to construct input vector e_i:

$$\boldsymbol{e}_i = [emb(w_i); posemb(p_i)] \quad (1)$$

Where emb is word embedding, $posemb$ is learned Part-of-speech tag embedding. We use pre-trained model for word embedding. Following Straka et al. (2019), we use the linear combination of hidden states of the last four layers as the embedding, and a word embedding is the average of its subword embeddings. We project word embedding to a lower dimension. In BiLSTM layer, $\boldsymbol{e}_0 \boldsymbol{e}_1 ... \boldsymbol{e}_n$ is input into a three-layer BiLSTM model. We arrange the output vectors of the last layer $\boldsymbol{h}_0, \boldsymbol{h}_1, ..., \boldsymbol{h}_n$ into the matrix $\boldsymbol{X}^0 \in \mathbb{R}^{n \times 2h}$, as the initial input of Multi-layer biaffine model:

$$\boldsymbol{h}_i = BiLSTM_i(\boldsymbol{e}_0 \boldsymbol{e}_1 ... \boldsymbol{e}_n) \quad (2)$$

$$\boldsymbol{X}^0 = \begin{bmatrix} \boldsymbol{h}_0 \\ \boldsymbol{h}_1 \\ ... \\ \boldsymbol{h}_n \end{bmatrix} \quad (3)$$

2.2 Multi-layer Biaffine Model

Multi-layer biaffine model consists of T layers with the same structure. The input matrix of the t-th layer is $\boldsymbol{X}^{t-1}$. The biaffine method is following Dozat and Manning (2017) for arc prediction:

$$\boldsymbol{R}^{(head)t} = MLP^{(head)t}(\boldsymbol{X}^{t-1}) \quad (4)$$

$$\boldsymbol{R}^{(dep)t} = MLP^{(dep)t}(\boldsymbol{X}^{t-1}) \quad (5)$$

$$\boldsymbol{S}_i^{(arc)t} = \boldsymbol{R}^{(head)t}\boldsymbol{U}^{(1)t}(\boldsymbol{R}_i^{(dep)t})^{\mathrm{T}} + \boldsymbol{R}^{(head)t}\boldsymbol{b}^{(1)t} \quad (6)$$

Where $MLP^{(head)t}, MLP^{(dep)t}$ are $2h \times d_a$ (input dimension is $2h$ and output dimension is d_a; similarly hereinafter) and $\boldsymbol{U}^{(1)t} \in \mathbb{R}^{d_a \times d_a}$, $\boldsymbol{b}^{(1)t} \in \mathbb{R}^{d_a}$ are learned parameters.

For label prediction, we modify the biaffine method in Dozat and Manning (2017). We do not use the predictive head, but calculate the label score vector for each pair of (i, j):

$$\boldsymbol{R}^{(lhead)t} = MLP^{(lhead)t}(\boldsymbol{X}^{t-1}) \quad (7)$$

$$\boldsymbol{R}^{(ldep)t} = MLP^{(ldep)t}(\boldsymbol{X}^{t-1}) \quad (8)$$

$$\boldsymbol{S}_{ij}^{(label)t} = \boldsymbol{R}_j^{(lhead)t}\boldsymbol{U}^{(2)t}(\boldsymbol{R}_i^{(ldep)t})^{\mathrm{T}} \quad (9)$$

Where $MLP^{(lhead)t}, MLP^{(ldep)t}$ are $2h \times d_l$ and $\boldsymbol{U}^{(2)t} \in \mathbb{R}^{d_l \times k \times d_l}$ is learned matrix (k is the size of relation set).

We scale and apply softmax function (Vaswani et al., 2017)on $\boldsymbol{S}^{(arc)t} \in \mathbb{R}^{n \times n}$ to obtain attention weight matrix, using it to construct the arc-related representation:

$$\boldsymbol{R}^{(arc)t} = MLP^{(arc)t}(\boldsymbol{X}^{t-1}) \quad (10)$$

$$\boldsymbol{V}^{(arc)t} = Softmax(\frac{\boldsymbol{S}^{(arc)t}}{\sqrt{2h}})\boldsymbol{R}^{(arc)t} \quad (11)$$

Where $MLP^{(arc)t}$ is $2h \times d_a$.

We apply the same method on $\boldsymbol{S}^{(label)t} \in \mathbb{R}^{n \times n \times k}$ after projection to construct the label-related representation:

$$\boldsymbol{R}^{(label)t} = MLP^{(label)t}(\boldsymbol{X}^{t-1}) \quad (12)$$

$$\boldsymbol{V}^{(label)t} = Softmax(\frac{\boldsymbol{S}^{(label)t}\boldsymbol{b}^{(2)t}}{\sqrt{2h}})\boldsymbol{R}^{(label)t} \quad (13)$$

Where $MLP^{(label)t}$ is $2h \times d_l$ and $\boldsymbol{b}^{(2)t} \in \mathbb{R}^k$ is learned vector.

At the end of the layer, we combine two types of representations, applying projection and Add & Norm (Vaswani et al., 2017) to obtain the input matrix of the next layer:

$$\boldsymbol{Y}^t = [\boldsymbol{V}^{(arc)t}; \boldsymbol{V}^{(label)t}]\boldsymbol{W}^t \quad (14)$$

$$\boldsymbol{X}^t = LayerNorm(\boldsymbol{X}^{t-1} + \boldsymbol{Y}^t) \quad (15)$$

Where $\boldsymbol{W}^t \in \mathbb{R}^{(d_a+d_l)\times 2h}$ is learned matrix. $\boldsymbol{X}^t \in \mathbb{R}^{n \times 2h}$ is used as the input of $(t+1)$-th layer.

For the final output dependency tree, we apply softmax function on score matrices of the last layer:

$$\boldsymbol{S}^{(arc)} = Softmax(\boldsymbol{S}^{(arc)T}) \quad (16)$$

$$\boldsymbol{S}^{(label)} = Softmax(\boldsymbol{S}^{(label)T}) \quad (17)$$

We use $\boldsymbol{S}^{(arc)}$ as the edge weight matrix and apply the Eisner algorithm (Eisner, 2000) to obtain the projective maximum spanning tree. After obtaining the tree structure, we use $\boldsymbol{S}^{(label)}$ as the score matrix and select the label with the maximum score for each arc.

2.3 Training

We use the cross-entropy loss for arc and label predictions:

$$\mathcal{L}^{(arc)} = -\sum_{i=1}^{n} \log(\boldsymbol{S}_{i,h_i}^{(arc)}) \quad (18)$$

$$\mathcal{L}^{(label)} = -\sum_{i=1}^{n} \log(\boldsymbol{S}_{i,h_i,l_i}^{(label)}) \quad (19)$$

Where h_i is the gold head of w_i, and l_i is the gold label of arc (h_i, w_i). The final loss is:

$$\mathcal{L} = \lambda\mathcal{L}^{(arc)} + (1-\lambda)\mathcal{L}^{(label)} \quad (20)$$

Where λ is a hyper-parameter between 0 and 1.

3 Experiments

3.1 Datasets

We evaluate our method on PTB 3.0 (Marcus et al., 1993) and CTB 5.1 (Xue et al., 2005). We use the same POS tagger (Toutanova et al., 2003) and data splits as described in Chen and Manning (2014).

3.2 Evaluation

We use UAS and LAS as the metric. During the evaluation, we ignore all punctuation. We select the model based on the sum of UAS and LAS on the dev set. For all results reported, we run the training process five times with different random seeds and average the results to avoid contingency.

Pre-trained	Model	PTB	
		UAS	LAS
w/o	Doz. & Man. (2016)	95.74	94.08
	Zhang et al. (2020)	96.14	94.49
XLNet-large	Zhou & Zhao (2019)†	97.20	95.72
	Mrini et al. (2020)†	97.42	96.26
	Ours(XLNet-large)	**97.46**	**96.45**
XLNet-base	Ours(XLNet-base)	97.32	96.27
BERT-large	Wang & Tu (2020)	96.91	95.34
	Fer. & Góm. (2021)	97.05	95.48
	Ours(BERT-large)	**97.20**	**96.14**
BERT-base	Moh. & Hen. (2020)	96.11	94.33
	Moh. & Hen. (2021)	96.66	95.01
	Ours(BERT-base)	**96.97**	**95.90**

Pre-trained	Model	CTB	
		UAS	LAS
w/o	Doz. & Man. (2016)	89.30	88.23
	Ma et al. (2018)	90.59	89.29
BERT-base	Mrini et al. (2020)†	94.56	89.28
	Wang & Tu (2020)	92.78	**91.69**
	Fer. & Góm. (2021)	92.75	91.62
	Moh. & Hen. (2021)	92.98	91.18
	Ours(BERT-base)	**93.03**	91.21

Table 1: Comparison of dependency parsers on PTB and CTB. **Pre-trained** column indicates pre-trained model used for word embedding. †:These approaches join the constituency parsing and use additional constituency information for training.

3.3 Implementation Details

We evaluate our model with different pre-trained models. All pre-trained models we use for PTB are case-sensitive. The dimension of word embedding after projection is 300, and the dimension of POS embedding is 50. We set $h = 512$, $d_a = 512$, $d_l = 128$, $T = 6$, $\lambda = 0.55$. We apply dropout after embedding, BiLSTM, and MLP layers with dropout rate 0.33. We apply gradient clipping with max 2-norm value 1. We use Adam(Kingma and Ba, 2015) optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$. The learning rate is $1e{-}5$ for pre-trained model and $5e-4$ for other components. We train the model for 10 epochs on PTB and 20 epochs on CTB. We decay the learning rate linearly to 0 during the training process. We batch the sentences of similar length for efficiency. The batch size is 24.

3.4 Baselines

We use five types of baselines for comparison. All results of baselines are from the corresponding papers. 1) Dozat and Manning (2017) introduces the biaffine method for first-order graph-based dependency parsing. 2) Zhang et al. (2020) introduces TreeCRF, and Wang and Tu (2020) introduces MFVI. These parsers extend the biaffine method to triaffine for modeling second-order information. 3) Zhou and Zhao (2019) introduces head-driven phrase structure grammar (HPSG) for joint dependency and constituent parsing. Mrini et al. (2020) uses HPSG and introduces label attention layer. These parsers use additional constituency information for training. 4) Mohammadshahi and Henderson (2020) and Mohammadshahi and Henderson (2021) introduces Graph-to-Graph Transformer for transition-based and graph-based dependency parsing respectively. 5) Ma et al. (2018) introduces stack-pointer networks, and Fernández-González and Gómez-Rodríguez (2021) introduces bottom-up hierarchical pointer networks, both for transition-based dependency parsing.

3.5 Main Results

Table 1 shows the results of baselines and our models on PTB and CTB. For intuitive comparison, we divide the models according to the pre-trained model used for word embedding. It can be seen that on PTB, our model with XLNet-large achieves state-of-the-art performance. Compared with previous state-of-the-art model (Mrini et al., 2020) using addtional constituency information for training, our model improves 0.04 UAS and 0.19 LAS without additional constituency information. Additionally, our model with BERT-large and BERT-base respectively outperforms previous state-of-the-art models with the same pre-trained model (Fernández-González and Gómez-Rodríguez, 2021; Mohammadshahi and Henderson, 2021). Our model with BERT-large improves 0.15 UAS and 0.66 LAS, and our model with BERT-base improves 0.31 UAS and 0.89 LAS. On CTB, our model with BERT-base-Chinese achieves state-of-the-art UAS among dependency parsers without additional constituency information for training. Compared with previous state-of-the-art model on UAS (Mohammadshahi and Henderson, 2021), our model improves 0.05 UAS and 0.03 LAS. We also report the results of our model with XLNet-base on PTB for possible future comparison.

4 Analysis

4.1 Number of Layers

We evaluate our model with number of layers $T \in \{1, 2, 5, 6\}$. The pre-trained model is XLNet-base for PTB and BERT-base-Chinese for CTB, and we use this setting in the whole section 4. The results are shown in Table 2. It can be seen that the multi-

Layers	PTB		CTB	
	UAS	LAS	UAS	LAS
$T = 1$	97.27	96.23	92.95	91.05
$T = 2$	**97.32**	**96.28**	92.97	91.17
$T = 5$	97.30	96.27	92.95	91.16
$T = 6$	**97.32**	96.27	**93.03**	**91.21**

Table 2: Results of our model with T layers on PTB and CTB.

A	L	PTB		CTB	
		UAS	LAS	UAS	LAS
Yes	Yes	**97.32**	**96.27**	**93.03**	**91.21**
Yes	No	97.30	96.24	93.00	91.19
No	Yes	97.29	96.23	92.95	91.13

Table 3: Results on PTB and CTB of the ablation study on two types of representation. **A** and **L** mean the use of arc-related and label-related representation respectively.

layer biaffine model with any $T > 1$ outperforms the single-layer biaffine model with $T = 1$ on both PTB and CTB. The optimal multi-layer biaffine model improves 0.05 UAS, 0.05 LAS on PTB and 0.08 UAS, 0.16 LAS on CTB compared with the strong base of single-layer biaffine model. The results show the benefit of introducing multi-layer form into the biaffine method.

4.2 Ablation Study

The construction of arc-related and label-related representation are two main components of the biaffine layer. In the ablation study, we respectively remove one of two types of representations and evaluate our model. The results are shown in Table 3. It can be seen that removing either of two types of representation makes the model perform worse, though the results are still better than the single-layer biaffine model. The results show that both types of representations contribute to the performance improvement of our model.

4.3 Error Analysis

Sentence length. We evaluate head prediction error rate of our model on sentences of length $L < 50$ and $L \geq 50$ on CTB and PTB. We compare our model with number of layers $T = 1$ and $T = 6$. The results are shown in Tabel 4. It can be seen that the multi-layer biaffine model performs better in head prediction than the single-layer biaffine model on both short and long sentences.

Label prediction. We also evaluate label prediction error rate of our model on CTB and PTB when the gold head is provided. On PTB, the error rate is 1.38% when $T = 1$ and $T = 6$. On CTB, the error rate is 2.95% when $T = 1$ and 2.87% when $T = 6$. It can be seen that when the influence of performance difference in head prediction is excluded, the multi-layer biaffine model still performs better in label prediction than the single-layer biaffine model.

Layers	PTB		CTB	
	$L < 50$	$L \geq 50$	$L < 50$	$L \geq 50$
$T = 1$	2.69	3.85	6.52	8.49
$T = 6$	**2.64**	**3.53**	**6.44**	**8.42**

Table 4: Head prediction error rate (%) of our model with $T \in \{1, 6\}$ layers on sentences of length $L < 50$ and $L \geq 50$ on PTB and CTB.

4.4 Efficiency Loss

We evaluate the speed of our model with the number of layers $T = 1$ and $T = 6$ on PTB. We run our model on a single TITAN RTX GPU. To run one epoch of training with batch size 24 on the entire PTB train set, it takes 12.08 minutes when $T = 1$ and 13.07 minutes when $T = 6$. To parse the entire PTB test set, it takes 20.33 seconds when $T = 1$ and 21.07 seconds when $T = 6$. Compared with the single-layer biaffine model, the multi-layer biaffine model with 6 layers uses 8.2% more time on training and 3.6% more time on parsing. The relatively low increase in time consumption on both training and parsing indicates the low efficiency loss of introducing multi-layer form into the biaffine method.

5 Conclusions

In this paper, we propose a multi-layer biaffine model for neural dependency parsing, which uses the modified biaffine method in multi-layer form. Our experiments show that compared with the single-layer biaffine model, our multi-layer biaffine model has advantages in overall performance, head prediction on sentences of different lengths, and label prediction. Our ablation study shows that both types of representations in the biaffine layer contribute to performance improvement. Our speed evaluation shows the low efficiency loss of introducing multi-layer form into the biaffine method. Our model achieves state-of-the-art results on PTB and CTB.

References

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. 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.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2021. Dependency parsing with bottom-up hierarchical pointer networks. *CoRR*, abs/2105.09611.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Ying Li, Zhenghua Li, Min Zhang, Rui Wang, Sheng Li, and Luo Si. 2019. Self-attentive biaffine dependency parsing. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5067–5073. ijcai.org.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Alireza Mohammadshahi and James Henderson. 2020. Graph-to-graph transformer for transition-based dependency parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 3278–3289. Association for Computational Linguistics.

Alireza Mohammadshahi and James Henderson. 2021. Recursive non-autoregressive graph-to-graph transformer for dependency parsing with iterative refinement. *Trans. Assoc. Comput. Linguistics*, 9:120–138.

Khalil Mrini, Franck Dernoncourt, Quan Hung Tran, Trung Bui, Walter Chang, and Ndapa Nakashole. 2020. Rethinking self-attention: Towards interpretability in neural parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 731–742, Online. Association for Computational Linguistics.

Milan Straka, Jana Straková, and Jan Hajic. 2019. Evaluating contextualized embeddings on 54 languages in POS tagging, lemmatization and dependency parsing. *CoRR*, abs/1908.07448.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 252–259.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Xinyu Wang and Kewei Tu. 2020. Second-order neural dependency parsing with message passing and end-to-end training. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 93–99, Suzhou, China. Association for Computational Linguistics.

Zenan Xu, Daya Guo, Duyu Tang, Qinliang Su, Linjun Shou, Ming Gong, Wanjun Zhong, Xiaojun Quan, Daxin Jiang, and Nan Duan. 2021. Syntax-enhanced pre-trained model. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5412–5422, Online. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Martha Palmer. 2005. The penn chinese treebank: Phrase structure annotation of a large corpus. *Nat. Lang. Eng.*, 11(2):207–238.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online. Association for Computational Linguistics.

Junru Zhou and Hai Zhao. 2019. Head-Driven Phrase Structure Grammar parsing on Penn Treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.