Multi-Layer Pseudo-Siamese Biaffine Model for Dependency Parsing

Ziyao Xu[1], **Houfeng Wang**[1], **Bingdong Wang**[2]
[1]MOE Key Lab of Computational Linguistics, Peking University
[2]Beijing Huilan Technology Co., Ltd.
{xzyxzy,wanghf}@pku.edu.cn, wangbd@huilan.com

Abstract

Biaffine method is a strong and efficient method for graph-based dependency parsing. However, previous work only used the biaffine method at the end of the dependency parser as a scorer, and its application in multi-layer form is ignored. In this paper, we propose a multi-layer pseudo-Siamese biaffine model for neural dependency parsing. In this model, we modify the biaffine method so that it can be utilized in multi-layer form, and use pseudo-Siamese biaffine module to construct arc weight matrix for final prediction. In our proposed multi-layer architecture, the biaffine method plays important roles in both scorer and attention mechanism at the same time in each layer. We evaluate our model on PTB, CTB, and UD. The model achieves state-of-the-art results on these datasets. Further experiments show the benefits of introducing multi-layer form and pseudo-Siamese module 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 $w_i (1 \leq i \leq n)$ is a word, 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).

There are two approaches to dependency parsing: transition-based and graph-based methods. Graph-based dependency parsing scores the components of a sentence and selects the highest scoring tree. Dozat and Manning (2017) for the first time introduce biaffine method into the dependency parsing. This strong and efficient biaffine first-order graph-based parser consists of BiLSTM encoder and biaffine scorer. After the biaffine parser is proposed, many works have focused on further improving the performance of this parser. Since the biaffine parser consists of two parts: BiLSTM encoder and biaffine scorer, there are two main directions of improvement focusing on the two parts respectively.

One direction of improvement is modifying the encoder. Straka (2018) introduce pre-trained model in the embedding stage. Na et al. (2019) fuse the hidden states of different layers of BiLSTM to construct the input of the biaffine scorer. Ji et al. (2019) add graph neural networks before the biaffine scorer to capture high-order information. Li et al. (2019) apply the self-attention mechanism as the replacement of BiLSTM encoder. Mrini et al. (2020) introduce the Label Attention Layer, a new form of self-attention where attention heads represent labels, into the encoder. In these works, although the encoder changes, the biaffine method remains as a scorer at the end of dependency parser.

Another direction is extending the biaffine scorer for second-order parser. Currently, there are relatively few works in this direction due to its difficulty. Zhang et al. (2020) introduce efficient TreeCRF. Wang and Tu (2020) introduce MFVI. Both works extend the biaffine scorer to the triaffine scorer for scoring second-order subtrees. The results show that introducing high-order information is beneficial to the dependency parser in many ways. In these works, the extended biaffine method still plays the role of a scorer for high-order modeling.

We observe that *biaffine method only plays the role of a scorer in almost all graph-based dependency parsers improved by the biaffine parser.* The ways and effects of making the method play more roles remains to be explored. This leads us to a new direction of improvement: *how to make biaffine method play more roles in dependency parser?* We notice that biaffine method is essentially an attention mechanism, so it can be utilized in a multi-

Proceedings of the 29th International Conference on Computational Linguistics, pages 5476–5487
October 12–17, 2022.

layer architecture like Transformers (Vaswani et al., 2017) to capture more information before the biaffine scorer. However, such an application of biaffine method is equivalent to adding Transformers which uses biaffine attention upon BiLSTM encoder and keeping the biaffine scorer at the end of the model unchanged, that is, the biaffine scorer is independent of the multi-layer architecture, so it cannot obtain more information, which limits the performance gains. This raises a question: *how to fuse the biaffine scorer into the multi-layer architecture, which allows the biaffine scorer to make use of more information?*

To address this question, we propose a multi-layer pseudo-Siamese biaffine model for graph-based dependency parsing. Our multi-layer architecture consists of several connectable layers with the same structure, and each layer contains two biaffine modules that are identical but have separate parameters, namely pseudo-Siamese module. One biaffine module is used as an attention mechanism to obtain the attention weight to construct the token representation for the next layer; the other is used as a scorer to obtain the arc weight matrix that contributes to final prediction. Therefore, in our proposed multi-layer architecture, the biaffine method plays the role of a scorer and an attention mechanism at the same time in each layer. The biaffine scorer can obtain information from each layer before, instead of only seeing the output of the last layer, so it can make use of more information captured by the multi-layer architecture.

We conduct experiments on PTB, CTB, and UD to verify the effectiveness of our model. Since the usage of pre-trained models like BERT (Devlin et al., 2019) and XLNet (Yang et al., 2019) in the embedding stage is common for dependency parsing, we evaluate our model with different pre-trained models. We conduct the experiments in three aspects: 1) compare our model with previous state-of-the-art dependency parsers using the same pre-trained model to show that our model achieves state-of-the-art performance; 2) implement the original biaffine model and compare it with our model when using the same pre-trained model to show that our modification based on biaffine model significantly improves the performance; 3) compare our multi-layer architecture with other multi-layer architectures for dependency parsing to justify the specific layers we propose.

We also conduct detailed analysis to illustrate four aspects relevant to our model: 1) the impact of the choice of number of layers and attention function on our model; 2) the importance of fusing the biaffine scorer into the multi-layer architecture; 3) the benefit of introducing multi-layer form and pseudo-Siamese module into the biaffine method in many ways, including the overall performance, the performance on short and long sentences respectively and the performance of label prediction; 4) the low efficiency loss of introducing multi-layer form and pseudo-Siamese module into the biaffine method.

In summary, the major contributions of our work are as follows:

- We introduce a multi-layer architecture using biaffine attention mechanism into the biaffine dependency parser, which makes the biaffine method plays more than a role of a scorer in graph-based dependency parsing.
- We introduce pseudo-Siamese module to fuse the biaffine scorer into the multi-layer architecture. We show that fusing the biaffine scorer into the multi-layer architecture is important for the performance gains.
- We conduct experiments and detailed analysis on PTB, CTB, and UD. The results show the benefits of introducing multi-layer form and pseudo-Siamese module into the biaffine method with low efficiency loss.

2 Model

Our graph-based dependency parser contains three parts, i.e., encoder, multi-layer biaffine model, and pseudo-Siamese module.

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 BERT or XLNet 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

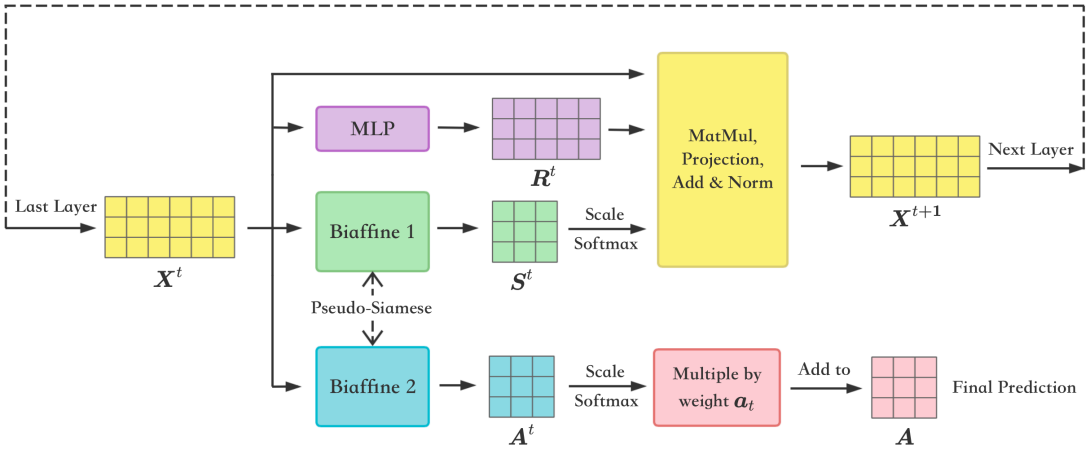

Figure 1: The architecture of the t-th layer biaffine model and its corresponding pseudo-Siamese module.

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 $\boldsymbol{X}^1 \in \mathbb{R}^{n\times 2h}$, as the initial input of multi-layer biaffine model:

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

$$\boldsymbol{X}^1 = \begin{bmatrix} \boldsymbol{h}_0 \\ \boldsymbol{h}_1 \\ ... \\ \boldsymbol{h}_n \end{bmatrix} \quad (3)$$

2.2 Multi-layer Biaffine Model

Multi-layer biaffine model consists of T layers with the same structure. The architecture of the t-th layer is shown in Figure 1. The input matrix of the t-th layer is $\boldsymbol{X}^t$. We use the same biaffine method described in Dozat and Manning (2017) :

$$\boldsymbol{H}^t = MLP^{(head)t}(\boldsymbol{X}^t) \quad (4)$$

$$\boldsymbol{D}^t = MLP^{(dep)t}(\boldsymbol{X}^t) \quad (5)$$

$$\boldsymbol{S}^t_{i,j} = \boldsymbol{H}^t_j \boldsymbol{U}^t (\boldsymbol{D}^t_i)^{\mathrm{T}} + \boldsymbol{H}^t_j \boldsymbol{b}^t \quad (6)$$

Where $MLP^{(head)t}$ and $MLP^{(dep)t}$ are $2h \times d$ (input dimension is $2h$ and output dimension is d; similarly hereinafter); $\boldsymbol{U}^t \in \mathbb{R}^{d\times d}$ and $\boldsymbol{b}^t \in \mathbb{R}^d$ are learned parameters. This biaffine module corresponds to Biaffine 1 in Figure 1.

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

$$\boldsymbol{R}^t = MLP^{(arc)t}(\boldsymbol{X}^t) \quad (7)$$

$$\boldsymbol{V}^t = Softmax(\frac{\boldsymbol{S}^t}{\sqrt{2h}})\boldsymbol{R}^t \quad (8)$$

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

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

$$\boldsymbol{X}^{t+1} = LayerNorm(\boldsymbol{X}^t + \boldsymbol{V}^t\boldsymbol{W}^t) \quad (9)$$

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

2.3 Pseudo-Siamese Module

We notice that attention weight matrix in each layer and arc weight matrix for final prediction have the same form. Based on this, we use pseudo-Siamese module to construct the arc weight matrix. In the t-th layer, we use another biaffine module to calculate matrix $\boldsymbol{A}^t \in \mathbb{R}^{n\times n}$. This biaffine module corresponds to Biaffine 2 in Figure 1. Biaffine 1 and Biaffine 2 have the same structure but different parameters. That is, $\boldsymbol{A}^t$ is calculated by Equations $4 \sim 6$ similar to $\boldsymbol{S}^t$ but using another set of parameters. We use the linear combination of $\boldsymbol{A}^t$ as the arc weight matrix $\boldsymbol{A}$ for final prediction:

$$\boldsymbol{A} = \sum_{t=1}^{T} \boldsymbol{a}_t \cdot Softmax(\frac{\boldsymbol{A}^t}{\sqrt{2h}}) \quad (10)$$

Where $\boldsymbol{a} \in \mathbb{R}^T$ is learned weight satisfying that $\sum_{t=1}^{T} \boldsymbol{a}_t = 1$.

2.4 Inference

We use $\boldsymbol{A}$ calculated by Equation 10 as the arc weight matrix and apply the Eisner algorithm (Eisner, 2000) or the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds et al., 1967) to obtain the maximum spanning tree. After obtaining the tree structure, following Dozat and Manning

(2017), we use a biaffine classifier to obtain the label score vector for each word w_i given the predicted head h_i:

$$\boldsymbol{B}_i = \boldsymbol{H}_{h_i}\boldsymbol{U}^{(1)}(\boldsymbol{D}_i)^{\mathrm{T}} + [\boldsymbol{H}_{h_i}; \boldsymbol{D}_i]\boldsymbol{U}^{(2)} + \boldsymbol{b}$$
$$\boldsymbol{L}_i = Softmax(\boldsymbol{B}_i) \quad (11)$$

Where $\boldsymbol{U}^{(1)} \in \mathbb{R}^{d\times k\times d}$, $\boldsymbol{U}^{(2)} \in \mathbb{R}^{2d\times k}$ and $\boldsymbol{b} \in \mathbb{R}^k$ are learned parameters (k is the size of relation set). $\boldsymbol{H}$ and $\boldsymbol{D}$ are calculated by Equations 4 $\sim$ 5 using $\boldsymbol{X}^T$ as input. We select the label with the maximum score for each arc to obtain the final dependency tree.

2.5 Training

We calculate $\boldsymbol{A}$ by Equation 10, and calculate $\boldsymbol{L}$ by Equation 11 using the gold head. We use the cross-entropy loss for arc and label predictions:

$$\mathcal{L}^{(arc)} = -\sum_{i=1}^{n}\log(\boldsymbol{A}_{i,h_i}) \quad (12)$$

$$\mathcal{L}^{(label)} = -\sum_{i=1}^{n}\log(\boldsymbol{L}_{i,l_i}) \quad (13)$$

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:

$$\mathcal{L} = \lambda\mathcal{L}^{(arc)} + (1-\lambda)\mathcal{L}^{(label)} \quad (14)$$

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), CTB 5.1 (Xue et al., 2005), and Universal Dependencies (UD) 2.2. Following Chen and Manning (2014), we use Stanford parser v3.3.0 to convert PTB, and use Penn2Malt tool with the head-finding rules of Zhang and Clark (2008) to convert CTB. Following Fernández-González and Gómez-Rodríguez (2021), we do not use POS tags on PTB, and use gold POS tags on CTB. Following Ma et al. (2018), we evaluate 12 languages selected from UD 2.2 and use POS tags.

3.2 Evaluation

We use UAS and LAS as the metric. During the evaluation, we ignore all punctuation. We use the model after the last epoch of training for evaluation. For all results reported, we run the training process five times with different random seeds and average the results to avoid contingency.

3.3 Implementation Details

We evaluate our model with different pre-trained models, including BERT-uncased and XLNet for PTB, BERT-Chinese for CTB, and BERT-Multilingual-cased for UD. The dimension of word embedding after projection is 300, and the dimension of POS embedding is 50. We set $h = 512$, $\lambda = 0.55$, $d = 512$ for arc and $d = 128$ for label. We set $T = 5$ on PTB and UD, and $T = 6$ on CTB. We apply dropout after embedding and BiLSTM 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 8 epochs on PTB and UD, and 20 epochs on CTB. We decay the learning rate linearly to 0 during training. We batch the sentences of similar length for efficiency. The batch size is 24. During training, we divide the loss by batch size on CTB and UD but not on PTB. During inference, we use the Eisner algorithm on PTB and CTB, and we use the Chu-Liu-Edmonds algorithm on UD. More discussions on inference algorithms are presented in Section 4.6.

3.4 Baselines

We use nine strong baseline models for comparison and divide them into five categories. All results of baseline models are from the corresponding papers.

Biaffine. Dozat and Manning (2017) introduces the biaffine method for first-order graph-based dependency parsing.

Second-order. 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.

HPSG. 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.

G2GTr. Mohammadshahi and Henderson (2020) and Mohammadshahi and Henderson (2021) introduces Graph-to-Graph Transformer (G2GTr) for transition-based dependency parsing and graph-based dependency parsing respectively.

Pointer Networks. Ma et al. (2018) introduces stack-pointer networks and Fernández-González and Gómez-Rodríguez (2021) introduces bottom-

Pre-trained	Model	PTB	
		UAS	LAS
w/o	Doz. & Man. (2017)	95.74	94.08
	Zhang et al. (2020)	96.14	94.49
BERT-base	Moh. & Hen. (2020)	96.11	94.33
	Moh. & Hen. (2021)	96.66	95.01
	Ours(BERT-base)	**96.93**	**95.18**
BERT-large	Wang & Tu (2020)	96.91	95.34
	Fer. & Góm. (2021)	97.05	95.48
	Ours(BERT-large)	**97.17**	**95.50**
XLNet-base	Ours(XLNet-base)	97.17	95.49
XLNet-large	Zhou & Zhao (2019)†	97.20	95.72
	Mrini et al. (2020)†	97.42	**96.26**
	Ours(XLNet-large)	**97.44**	95.81

Pre-trained	Model	CTB	
		UAS	LAS
w/o	Doz. & Man. (2017)	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)	93.37	**92.16**

Pre-trained	Model	UD2.2	
		UAS	LAS
w/o	Ma et al. (2018)	93.53	89.75
	Zhang et al. (2020)	-	89.33
BERT-base	Wang & Tu (2020)	-	91.02
	Ours(BERT-base)	94.68	**91.82**

Table 1: Comparison of dependency parsers on PTB, CTB, and UD2.2. **Pre-trained** column indicates pre-trained model used for word embedding. †:These approaches join the constituency parsing and use additional constituency information for training.

up hierarchical pointer networks for transition-based dependency parsing.

3.5 Main Results

Table 1 shows the results of baselines and our model on PTB and CTB. For intuitive comparison, we divide the models according to the pre-trained model used for word embedding. Overall, the ranking of performance of our model with different pre-trained models is XLNet-large > XLNet-base ≈ BERT-large > BERT-base.

On PTB, the previous state-of-the-art model with BERT-base is Mohammadshahi and Henderson (2021) using Graph-to-Graph Transformer for graph-based parsing, which outperforms Mohammadshahi and Henderson (2020) for transition-based parsing. Compared with it, our model with BERT-base improves 0.27 UAS and 0.17 LAS. The previous state-of-the-art model with BERT-large is Fernández-González and Gómez-Rodríguez (2021) using bottom-up hierarchical pointer networks for transition-based parsing, which outperforms second-order graph-based parser Wang and Tu (2020) using MFVI. Compared with it, our model with BERT-large improves 0.12 UAS and performs similarly on LAS. Our model with XLNet-base also outperforms previous models with BERT-large. Our model with XLNet-large achieves 97.44 UAS and 95.81 LAS, which is the state-of-the-art result among dependency parsers without additional constituency information for training. For previous models with XLNet-large, Zhou and Zhao (2019) and Mrini et al. (2020) both use HPSG, which join the constituency parsing and use additional constituency information for training. Our model with XLNet-large outperforms Zhou and Zhao (2019), and has a comparable performance on UAS compared with Mrini et al. (2020).

On CTB, we only evalutate our model with BERT-base-Chinese because it is the only pre-trained model used for CTB in baseline models. Our model with BERT-base-Chinese achieves 93.37 UAS and 92.16 LAS, which is the state-of-the-art result among dependency parsers without additional constituency information for training. Compared with the previous state-of-the-art model on UAS which does not join the constituency parsing (Mohammadshahi and Henderson, 2021), our model improves 0.39 UAS. Compared with the previous state-of-the-art model on LAS (Wang and Tu, 2020), our model improves 0.47 LAS. Compared with Mrini et al. (2020) using additional constituency information for training, our model has a significant performance advantage on LAS.

On UD2.2, our model with BERT-Multilingual-cased achieves 94.68 average UAS and 91.82 average LAS. Compared with the previous state-of-the-art model (Wang and Tu, 2020) using MFVI, our model improves 0.8 average LAS.

3.6 Comparison with Original Single-layer Biaffine Model

We compare our model with the original biaffine model when using the same pre-trained model. When our model has only one layer, it degenerates to the original biaffine model, so we simply set $T = 1$ and keep other hyperparameters unchanged to implement the original biaffine model with different pre-trained models. The results on PTB and CTB are shown in Table 2. It can be seen that our model with each kind of pre-trained model performs significantly better than the original single-

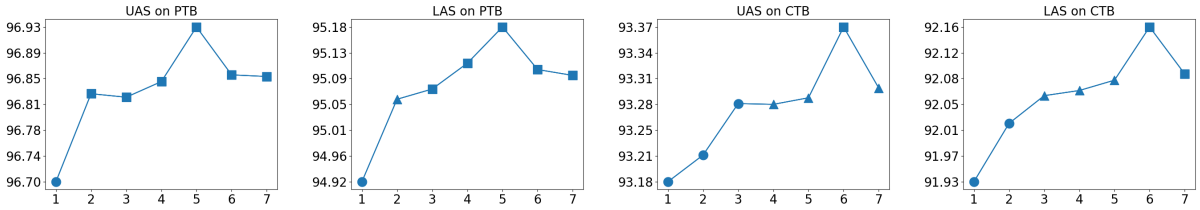

Figure 2: Results of our model with T layers on PTB and CTB ($T = 1$ means the original biaffine model). For $T > 1$, we perform significance test against $T = 1$. Triangle point means $p < 0.05$ and square point means $p < 0.005$.

Pre-trained	Model	PTB	
		UAS	LAS
BERT-base	Original	96.70	94.92
	Ours	**96.93**‡	**95.18**‡
BERT-large	Original	97.00	95.33
	Ours	**97.17**‡	**95.50**‡
XLNet-base	Original	97.01	95.26
	Ours	**97.17**‡	**95.49**‡
XLNet-large	Original	97.28	95.59
	Ours	**97.44**‡	**95.81**‡

Pre-trained	Model	CTB	
		UAS	LAS
BERT-base	Original	93.18	91.93
	Ours	**93.37**‡	**92.16**‡

Table 2: Comparison of original biaffine model and our model with different pre-trained models on PTB and CTB. We perform significance test for each pair of results. ‡ means $p < 0.005$.

layer biaffine model with the same pre-trained model. Compared with the original single-layer biaffine model, our model improves 0.18 UAS, 0.22 LAS on average on PTB, and improves 0.19 UAS, 0.23 LAS on CTB. The results on UD2.2 are shown in Table 3. It can be seen that our model outperforms the original single-layer biaffine model on 12 languages and improves 0.18 average UAS, 0.11 average LAS. The results show that our modification based on the original biaffine model significantly improves the performance. As a supplement, the results without gold POS tags on CTB are shown in Appendix A.

3.7 Comparison with Other Multi-layer Architectures

We compare our multi-layer architecture with other multi-layer architectures to justify the specific layers we propose. We select two strong multi-layer architectures which use BERT-base pre-trained model and have been evaluated on PTB or CTB for comparison: (1) Self-attentive parser proposed by Li et al. (2019). This architecture uses standard multi-head self-attention mechanism in multi-layer form. (2) Syntactic Transformer proposed by Mohammadshahi and Henderson (2021). This architecture uses the same architecture as BERT (Devlin et al., 2019) but changes the functions used by each attention head. For a fair comparison, we remove BiLSTM from our model and use BERT-base pre-trained model, so that the only difference between the models is the multi-layer architecture. The results are shown in Table 4. It can be seen that our multi-layer architecture achieves better results with fewer layers compared with other multi-layer architectures on PTB and CTB, which justifies the specific layers we propose.

4 Analysis

4.1 Number of Layers

We evaluate our model with number of layers $T \in \{1, 2, 3, 4, 5, 6, 7\}$ ($T = 1$ means the original biaffine model). The pre-trained model is BERT-base for PTB and BERT-base-Chinese for CTB, and we use this setting in the whole section 4 unless otherwise specified. The results are shown in Figure 2. It can be seen that: (1) Our model with any $T > 1$ outperforms the original single-layer biaffine model on both PTB and CTB. (2) With the increase in the number of layers, the performance of our model gradually improves and starts to be significantly different from the original single-layer model at $T = 2$ on PTB and $T = 3$ on CTB. (3) From $T = 2$ on PTB and $T = 3$ on CTB, the performance of our model improves with a similar trend on both datasets. This continuous improvement is due to more useful information captured by more layers. (4) After three more layers on both datasets, that is, at $T = 5$ on PTB and $T = 6$ on CTB, our model achieves the optimal performance.

Model	UD2.2 (UAS)												
	bg	ca	cs	de	en	es	fr	it	nl	no	ro	ru	Avg.
Original	95.71	95.49	95.28	90.49	93.05	94.77	94.28	96.16	94.68	95.93	93.13	95.52	94.54
Ours	**95.78**	**95.69**‡	**95.31**†	**90.61**	**93.27**†	**94.83**	**94.42**†	**96.33**	**94.84**	**96.05**†	**93.36**	**95.65**‡	**94.68**‡

Model	UD2.2 (LAS)												
	bg	ca	cs	de	en	es	fr	it	nl	no	ro	ru	Avg.
Original	91.87	93.77	92.30	85.89	90.90	92.69	91.34	94.43	91.88	94.46	86.95	94.03	91.71
Ours	**92.00**†	**94.00**‡	**92.34**†	**85.97**	**91.03**	**92.72**	**91.45**	**94.61**	**92.05**	**94.56**	**87.00**	**94.18**‡	**91.82**‡

Table 3: Comparison of original biaffine model and our model with BERT-base-Multilingual-cased on UD2.2. We perform significance test for each pair of results. † means $p < 0.05$ and ‡ means $p < 0.005$.

Multi-layer Architecture	PTB	
	UAS	LAS
SelfAtt (8 Layers)	96.67	95.03
SynTr (12 Layers)	96.60	94.94
Ours (5 Layers)	**96.75**	**95.05**

Multi-layer Architecture	CTB	
	UAS	LAS
SynTr (12 Layers)	92.42	90.67
Ours (6 Layers)	**92.91**	**91.80**

Table 4: Comparison of different multi-layer architectures for dependency parsing on PTB and CTB. SelfAtt: Li et al. (2019); SynTr: Mohammadshahi and Henderson (2021).

Our optimal model improves 0.23 UAS, 0.26 LAS on PTB and 0.19 UAS, 0.23 LAS on CTB compared with the original single-layer biaffine model. (5) The performance of our model starts to decline after $T = 5$ on PTB and $T = 6$ on CTB. This indicates that the use of more layers helps the model to capture more information, but also makes the model more prone to overfitting the training data, which is consistent with previous works related to multi-layer architecture for dependency parsing (Li et al., 2019; Mrini et al., 2020).

4.2 Attention Function

We compare the biaffine attention function used in Equation 6 with another three common attention functions described in Luong et al. (2015):

$$\boldsymbol{S}^t_{i,j} = \begin{cases} \boldsymbol{H}^t_j(\boldsymbol{D}^t_i)^{\mathrm{T}} & \text{Dot} \\ \boldsymbol{H}^t_j\boldsymbol{U}^t(\boldsymbol{D}^t_i)^{\mathrm{T}} & \text{General} \\ \tanh([\boldsymbol{D}^t_i;\boldsymbol{H}^t_j]\boldsymbol{P}^t)\boldsymbol{p}^t & \text{Concat} \end{cases} \quad (15)$$

Where $\boldsymbol{U}^t \in \mathbb{R}^{d\times d}$, $\boldsymbol{P}^t \in \mathbb{R}^{2d\times d}$ and $\boldsymbol{p}^t \in \mathbb{R}^d$ are learned parameters. The results are shown in Table 5. It can be seen that biaffine attention function outperforms another three attention functions

Function	PTB		CTB	
	UAS	LAS	UAS	LAS
Biaffine	**96.93**	**95.18**	**93.37**	**92.16**
Dot	96.87†	95.14	93.32	92.11
General	96.89	95.14	93.35	92.12
Concat	96.87†	95.09‡	93.30	92.08

Table 5: Results of our model with four attention functions used in Equation 6 on PTB and CTB. We perform significance test against Biaffine function for another three functions. † means $p < 0.05$ and ‡ means $p < 0.005$.

in our multi-layer architecture, although the difference is not significant in some cases. Our model with any of another three attention functions still significantly outperforms the original single-layer biaffine model. The results show that our model can achieve high performance using any common attention function, and biaffine attention function is the best choice for our model.

4.3 Effect of Pseudo-Siamese Module

In order to fuse the biaffine scorer into the multi-layer architecture, we use pseudo-Siamese module in our model to construct the arc weight matrix for final prediction. To verify the effect of pseudo-Siamese module, we compare it with two other construction methods: (1) True-Siamese. Two biaffine modules in each layer share the parameters, which means $\boldsymbol{A}^t = \boldsymbol{S}^t$. (2) Non-Siamese. The biaffine module for prediction is removed, and the final prediction is based on $\boldsymbol{S}^T$. The results are shown in Table 6. It can be seen that either replacing pseudo-Siamese module with true-Siamese module or removing pseudo-Siamese module, will lead to a significant decrease in the performance of our model. In particular, removing pseudo-Siamese module, which means not fusing the biaffine scorer into the multi-layer architecture, will make the performance of our model not significantly different

Method	PTB		CTB	
	UAS	LAS	UAS	LAS
P-S	**96.93**	**95.18**	**93.37**	**92.16**
T-S	96.81‡	95.06‡	93.20†	91.99†
N-S	96.72‡	94.91‡	93.19†	91.98†

Table 6: Results of our model with three methods of constructing arc weight matrix for final prediction on PTB and CTB. P-S: pseudo-Siamese; T-S: true-Siamese; N-S: non-Siamese. For T-S and N-S, we perform significance test against P-S. † means $p < 0.01$ and ‡ means $p < 0.005$.

Model	PTB			
	UAS		LAS	
	$L < 24$	$L \geq 24$	$L < 24$	$L \geq 24$
Original	96.89	96.61	95.17	94.80
Ours	**97.14**	**96.83**	**95.40**	**95.07**

Model	CTB			
	UAS		LAS	
	$L < 28$	$L \geq 28$	$L < 28$	$L \geq 28$
Original	94.69	92.49	93.43	91.26
Ours	**94.72**	**92.75**	**93.44**	**91.57**

Table 7: Results of original biaffine model and our model on sentences shorter or longer than average length on PTB and CTB.

from the original single-layer biaffine model on both PTB and CTB. The results show that using pseudo-Siamese module to fuse the biaffine scorer into the multi-layer architecture is important for the performance gains of our model.

4.4 Sentence Length

We evaluate our model and original biaffine model on sentences shorter or longer than average length on PTB and CTB. The average length is 23.85 words on PTB and 27.22 words on CTB. The results are shown in Table 7. It can be seen that our model outperforms the original single-layer biaffine model on both short and long sentences on PTB and CTB.

4.5 Label Prediction

We evaluate the accuracy of label prediction of our model and the original biaffine model when the gold head is provided on CTB and PTB. Our model achieves 97.79% on PTB and 98.30% on CTB. The original biaffine model achieves 97.76% on PTB and 98.26% on CTB. It can be seen that when the influence of performance difference in head prediction is excluded, our model still performs better in label prediction than the original single-layer biaffine model.

Pre-trained	Algorithm	PTB	
		UAS	LAS
BERT-base	C-L-E	96.88	95.13
	Eisner	**96.93**	**95.18**
BERT-large	C-L-E	97.13	95.48
	Eisner	**97.17**	**95.50**
XLNet-base	C-L-E	97.13	95.46
	Eisner	**97.17**	**95.49**
XLNet-large	C-L-E	97.41	95.79
	Eisner	**97.44**	**95.81**

Pre-trained	Model	CTB	
		UAS	LAS
BERT-base	C-L-E	93.28	92.07
	Eisner	**93.37**	**92.16**

Table 8: Results of our model using two inference algorithms with different pre-trained models on PTB and CTB. C-L-E: the Chu-Liu-Edmonds algorithm.

4.6 Inference Algorithms

On PTB and CTB, we use the Eisner algorithm for inference. The time complexity of the Eisner algorithm is $O(n^3)$. Based on the code of Kiperwasser and Goldberg (2016), we implement the Eisner algorithm with Pytorch, which can obtain projective maximum spanning trees on the entire PTB test set in about 5 seconds on a single TITAN RTX GPU. We compare the performance of the Eisner algorithm and the Chu-Liu-Edmonds algorithm on PTB and CTB. The results are shown in Table 8. It can be seen that the Eisner algorithm slightly outperforms the Chu-Liu-Edmonds algorithm on PTB and CTB, because trees in PTB (99.9% projective) and CTB (100% projective) are almost all projective, and the Eisner algorithm guarantees that the obtained tree is projective. There are many non-projective trees in UD, which the Eisner algorithm cannot handle, so we use the Chu-Liu-Edmonds algorithm with time complexity $O(n^3)$ on UD for inference.

4.7 Efficiency Loss

We evaluate the parameter size and speed of the original biaffine model and our model with 5 layers. Our speed evaluation is performed on a single TITAN RTX GPU. Excluding the same encoder

part, the original biaffine model has 2.34M parameters, and our model has 17.04M parameters. To run one epoch of training with batch size 24 on the entire PTB training set, the original biaffine model takes 10.48 minutes, and our model takes 11.07 minutes. Compared with the original biaffine model, our model uses 5.63% more time on training. For inference, the scoring process described in Section 2.2 and 2.3 is the only process in which there is an efficiency difference between the two models. When only considering this process, to parse the whole PTB test set, the original biaffine model uses 0.06 seconds, and our model uses 0.51 seconds. Our model uses 0.45 seconds more in this process, which has little effect on the entire parsing time (17.5 seconds). The relatively low increase in parameter size and time consumption on both training and parsing indicates the low efficiency loss of introducing multi-layer form and pseudo-Siamese module into the biaffine method.

5 Related Work

Transition-based method (Nivre, 2003; Yamada and Matsumoto, 2003) and graph-based method (McDonald et al., 2005; McDonald and Pereira, 2006) are two main approaches to dependency parsing. Before the deep learning era, almost all dependency parsers classify based on many sparse indicator features. Titov and Henderson (2007) for the first time introduce neural networks into transition-based dependency parsing. Chen and Manning (2014) propose a fast and accurate transition-based dependency parser using neural networks. Dyer et al. (2015) introduce BiLSTM into transition-based dependency parsing. Wang and Chang (2016) introduce BiLSTM encoder into the graph-based dependency parser. Kiperwasser and Goldberg (2016) also introduce BiLSTM encoder into dependency parsing and test it with both transition-based and graph-based methods. Based on these works, Dozat and Manning (2017) introduce biaffine method as a scorer into the graph-based dependency parsing and achieve significant performance improvement.

Multi-layer architecture is widely used to improve the performance of the original biaffine parser. Since the original biaffine parser uses multi-layer BiLSTM, Na et al. (2019) propose to use hidden states of different layers of BiLSTM to construct role-dependent representations for each layer, which are aggregated as the input of the biaffine scorer. Li et al. (2019) propose to use multi-layer self-attention at the encoder stage as the replacement of BiLSTM. Ji et al. (2019) propose to add multi-layer graph neural networks before the biaffine scorer and apply layer-wise loss to capture high-order information. Mohammadshahi and Henderson (2021) propose a recursive non-autoregressive graph-to-graph transformer for dependency parsing. They introduce a multi-layer architecture similar to BERT but changing the attention function. In these models, the biaffine method is used as a scorer after the multi-layer architecture. Differently, our multi-layer architecture uses the biaffine method in each layer as the role of an attention mechanism and a scorer at the same time.

Pseudo-Siamese network architecture, which means using two separate but identical networks to process a pair of inputs and fusing the information at a later stage, is widely used in the field of computer vision (Hughes et al., 2018; Treible et al., 2019). In our model, we use two separate but identical biaffine modules in each layer to process the same input into two branches, which is a reversed pseudo-Siamese network architecture.

6 Conclusions

In this paper, we propose a multi-layer biaffine pseudo-Siamese model for neural dependency parsing. In our proposed multi-layer architecture, biaffine method plays the role of a scorer and an attention mechanism at the same time in each layer. We conduct experiments and detailed analysis on PTB, CTB, and UD, showing the benefit of introducing multi-layer form and pseudo-Siamese module into the biaffine method. Compared with the original single-layer biaffine model, our model has a significant advantage in performance with low efficiency loss. Our analysis shows that fusing the biaffine scorer into the multi-layer architecture is important for the performance gains of our model. Compared with other multi-layer architectures for dependency parsing, our multi-layer architecture also has a performance advantage. Our model achieves state-of-the-art results on PTB, CTB, and UD. Our code is available at `https://github.com/xzy-xzy/MLPSB-Parser`.

Acknowledgements

The work is supported by National Natural Science Foundation of China under Grant No. 62036001 and PKU-Baidu Fund (No. 2020BD021). The corresponding author of this paper is Houfeng Wang.

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.

Chu and Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. 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.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 334–343. The Association for Computer Linguistics.

Jack Edmonds et al. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240.

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.

Lloyd H. Hughes, Michael Schmitt, Lichao Mou, Yuanyuan Wang, and Xiao Xiang Zhu. 2018. Identifying corresponding patches in SAR and optical images with a pseudo-siamese CNN. *IEEE Geosci. Remote. Sens. Lett.*, 15(5):784–788.

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, Florence, Italy. Association for Computational Linguistics.

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*.

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.

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.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

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.

Ryan T. McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *HLT/EMNLP 2005, Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 6-8 October 2005, Vancouver, British Columbia, Canada*, pages 523–530. The Association for Computational Linguistics.

Ryan T. McDonald and Fernando C. N. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, April 3-7, 2006, Trento, Italy*. The Association for Computer Linguistics.

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.

Seung-Hoon Na, Jinwoon Min, Kwanghyeon Park, Jong-Hun Shin, and Young-Kil Kim. 2019. JBNU at MRP 2019: Multi-level biaffine attention for semantic dependency parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning, CoNLL 2019, Hong Kong, November 3, 2019*, pages 95–103. Association for Computational Linguistics.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies, IWPT 2003, Nancy, France, April 2003*, pages 149–160.

Milan Straka. 2018. Udpipe 2.0 prototype at conll 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Brussels, Belgium, October 31 - November 1, 2018*, pages 197–207. 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.

Ivan Titov and James Henderson. 2007. Fast and robust multilingual dependency parsing with a generative latent variable model. In *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pages 947–951. ACL.

Wayne Treible, Philip Saponaro, and Chandra Kambhamettu. 2019. Wildcat: In-the-wild color-and-thermal patch comparison with deep residual pseudo-siamese networks. In *2019 IEEE International Conference on Image Processing, ICIP 2019, Taipei, Taiwan, September 22-25, 2019*, pages 1307–1311. IEEE.

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.

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.

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.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the Eighth International Conference on Parsing Technologies, IWPT 2003, Nancy, France, April 2003*, pages 195–206.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

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.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii. 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.

A Supplementary Results

Previous works generally use gold POS tags on CTB. To get a more realistic estimate of real performance, we evaluate our model and the original biaffine model without gold POS tags on CTB. Our model achieves 91.45 UAS, 89.31 LAS. The original biaffine model achieves 91.19 UAS, 89.07 LAS. Compared with the original biaffine model, our model improves 0.26 UAS, 0.24 LAS. The significance test shows that $p < 0.005$ on UAS and $p < 0.01$ on LAS. It can be seen that our model still outperforms the original biaffine model without gold POS tags on CTB.