Graph-to-Graph Transformer for Transition-based Dependency Parsing

Alireza Mohammadshahi
Idiap Research Institute and EPFL / Switzerland

James Henderson
Idiap Research Institute / Switzerland
{alireza.mohammadshahi, james.henderson}@idiap.ch

Abstract

We propose the Graph2Graph Transformer architecture for conditioning on and predicting arbitrary graphs, and apply it to the challenging task of transition-based dependency parsing. After proposing two novel Transformer models of transition-based dependency parsing as strong baselines, we show that adding the proposed mechanisms for conditioning on and predicting graphs of Graph2Graph Transformer results in significant improvements, both with and without BERT pre-training. The novel baselines and their integration with Graph2Graph Transformer significantly outperform the state-of-the-art in traditional transition-based dependency parsing on both English Penn Treebank, and 13 languages of Universal Dependencies Treebanks. Graph2Graph Transformer can be integrated with many previous structured prediction methods, making it easy to apply to a wide range of NLP tasks.

1 Introduction

In recent years, there has been a huge amount of research on applying self-attention models to NLP tasks. Transformer (Vaswani et al., 2017) is the most common architecture, which can capture long-range dependencies by using a self-attention mechanism over a set of vectors. To encode the sequential structure of sentences, typically absolute position embeddings are input to each vector in the set, but recently a mechanism has been proposed for inputting relative positions (Shaw et al., 2018). For each pair of vectors, an embedding for their relative position is input to the self-attention function. This mechanism can be generalised to input arbitrary graphs of relations.

We propose a version of the Transformer architecture which combines this attention-based mechanism for conditioning on graphs with an attention-like mechanism for predicting graphs and demonstrate its effectiveness on syntactic dependency parsing. We call this architecture Graph2Graph Transformer. This mechanism for conditioning on graphs differs from previous proposals in that it inputs graph relations as continuous embeddings, instead of discrete model structure (e.g. (Henderson, 2003; Henderson et al., 2013; Dyer et al., 2015)) or predefined discrete attention heads (e.g. (Ji et al., 2019; Strubell et al., 2018)). An explicit representation of binary relations is supported by inputting these relation embeddings to the attention functions, which are applied to every pair of tokens. In this way, each attention head can easily learn to attend only to tokens in a given relation, but it can also learn other structures in combination with other inputs. This gives a bias towards attention weights which respect locality in the input graph but does not hard-code any specific attention weights.

We focus our investigation on this novel graph input method and therefore limit our investigation to models which predict the output graph one edge at a time, in an auto-regressive fashion. In auto-regressive structured prediction, after each edge of the graph has been predicted, the model must condition on the partially specified graph to predict the next edge of the graph. Thus, our proposed Graph2Graph Transformer parser is a transition-based dependency parser. At each step, the model predicts the next parsing decision, and thereby the next dependency relation, by conditioning on the partial parse structure specified by the previous decisions. It inputs embeddings for the previously specified dependency relations into the Graph2Graph Transformer model via the self-attention mechanism. It predicts the next dependency relation using only the vectors for the tokens involved in that relation.

To evaluate this architecture, we also propose two novel Transformer models of transition-based dependency parsing, called Sentence Transformer, and State Transformer. Sentence Transformer computes contextualised embeddings for each token of the input sentence and then uses the current parser state to

identify which tokens could be involved in the next valid parse transition and uses their contextualised embeddings to choose the best transition. For State Transformer, we directly use the current parser state as the input to the model, along with an encoding of the partially constructed parse graph, and choose the best transition using the embeddings of the tokens involved in that transition. Both baseline models achieve competitive or better results than previous state-of-the-art traditional transition-based models, but we still get substantial improvement by integrating Graph2Graph Transformer with them.

We also demonstrate that, despite the modified input mechanisms, this Graph2Graph Transformer architecture can be effectively initialised with standard pre-trained Transformer models. Initialising the Graph2Graph Transformer parser with pre-trained BERT (Devlin et al., 2018) parameters leads to substantial improvements. The resulting model significantly improves over the state-of-the-art in traditional transition-based dependency parsing.

This success demonstrates the effectiveness of Graph2Graph Transformers for conditioning on and predicting graph relations. This architecture can be easily applied to other NLP tasks that have any graph as the input and need to predict a graph over the same set of nodes as output.

In summary, our contributions are:

- We propose Graph2Graph Transformer for conditioning on and predicting graphs.
- We propose two novel Transformer models of transition-based dependency parsing.
- We successfully integrate pre-trained BERT initialisation in Graph2Graph Transformer.
- We improve state-of-the-art accuracies for traditional transition-based dependency parsing.[1]

2 Transition-based Dependency Parsing

Our transition-based parser uses arc-standard parsing sequences (Nivre, 2004), which makes parsing decisions in bottom-up order. The main data structures for representing the state of an arc-standard parser are a buffer of words and a stack of partially constructed syntactic sub-trees. At each step, the parser chooses between adding a leftward or rightward labelled arc between the top two words on the stack (`LEFT-ARC` (l) or `RIGHT-ARC` (l), where l is a dependency label) or shifting a word from the buffer onto the stack (`SHIFT`). To handle non-projective dependency trees, we allow the `SWAP` action proposed in Nivre (2009), which shifts the second-from-top element of the stack to the front of the buffer, resulting in the reordering of the top two elements of the stack.

[1] Our implementation is available at: https://github.com/alirezamshi/G2GTr

3 Graph2Graph Transformer

We propose a version of the Transformer which is designed for both conditioning on graphs and predicting graphs, which we call Graph2Graph Transformer (G2GTr), and show how it can be applied to transition-based dependency parsing. G2GTr supports arbitrary input graphs and arbitrary edges in the output graph. But since the nodes of both these graphs are the input tokens, the nodes of the output graph are limited to the set of nodes in the input graph.

Inspired by the relative position embeddings of Shaw et al. (2018), we use the attention mechanism of Transformer to input arbitrary graph relations. By inputting the embedding for a relation label into the attention functions for the related tokens, the model can more easily learn to pass information between graph-local tokens, which gives the model an appropriate linguistic bias, without imposing hard constraints.

Given that the attention function is being used to input graph relations, it is natural to assume that graph relations can also be predicted with an attention-like function. We do not go so far as to restrict the form of the prediction function, but we do restrict the vectors used to predict graph relations to only the tokens involved in the relation.

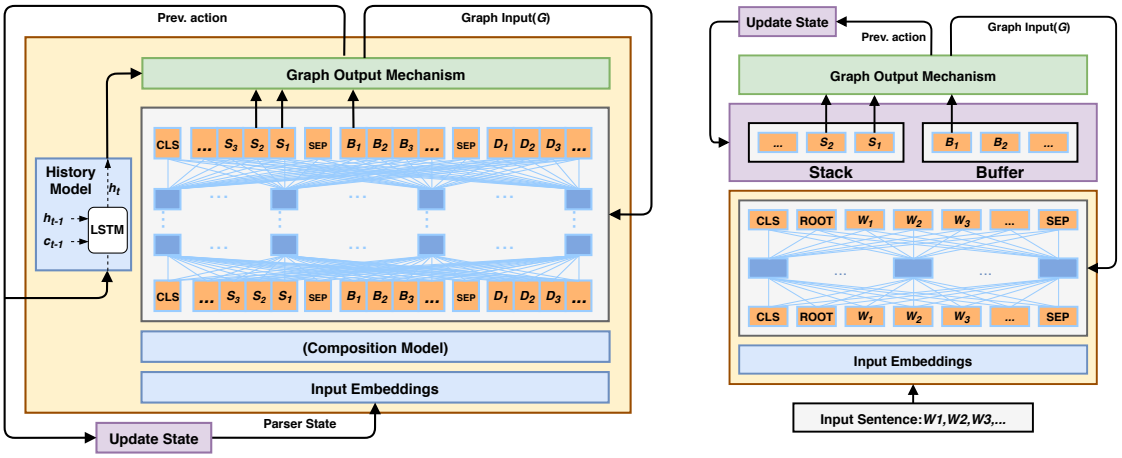
3.1 Original Transformer

Transformer (Vaswani et al., 2017) is an encoder-decoder model, of which we only use the encoder component. A Transformer encoder computes an output embedding for each token in the input sequence through stacked layers of multi-head self-attention. Each attention head takes its input vectors $(x_1,...,x_n)$ and computes its output attention vectors $(z_1,...,z_n)$. Each $z_i \in R^m$ is a weighted sum of transformed input vectors $x_j \in R^m$:

$$z_i = \sum_j \alpha_{ij}(x_j \boldsymbol{W^V}) \quad (1)$$

with the attention weights $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n}\exp(e_{ik})}$ and

$$e_{ij} = \frac{(x_i \boldsymbol{W^Q})(x_j \boldsymbol{W^K})}{\sqrt{d}} \quad (2)$$



(a) StateTr+G2GTr.

(b) SentTr+G2GTr.

Figure 1: The State Transformer and Sentence Transformer parsers with Graph-to-Graph Transformer integrated.

where $\boldsymbol{W^V}, \boldsymbol{W^Q}, \boldsymbol{W^K} \in R^{m \times d}$ are the trained value, query and key matrices, m is the embedding size, and d is the attention head size.

3.2 Graph Inputs

Graph2Graph Transformer extends the architecture of the Transformer to accept any arbitrary graph as input. In particular, we input the dependency tree as its set of dependency relations. Each labelled relation (x_i, x_j, l') is input by modifying Equation 2 as follows:

$$e_{ij} = \frac{(x_i \boldsymbol{W^Q})(x_j \boldsymbol{W^K} + p_{ij} \boldsymbol{W_1^L})}{\sqrt{d}} \quad (3)$$

where $p_{ij} \in \{0, 1\}^k$ is a one-hot vector which specifies the type l' of the relation between x_i and x_j, discussed below, and $\boldsymbol{W_1^L} \in R^{k \times d}$ is a matrix of learned parameters. We also modify Equation 1 to transmit information about relations to the output of the attention layer:

$$z_i = \sum_j \alpha_{ij} (x_j \boldsymbol{W^V} + p_{ij} \boldsymbol{W_2^L}) \quad (4)$$

where $\boldsymbol{W_2^L} \in R^{k \times d}$ are learned parameters.

In this work, we consider graph input for only unlabelled directed dependency relations l', so p_{ij} has only three dimensions (k=3), for `leftward`, `rightward` and `none`. This choice was made mostly to simplify our extension of the Transformer, as well as to limit the computational cost of this extension. The dependency labels are input as label embeddings added to the input token embeddings of the dependent word.

3.3 Graph Outputs

The graph output mechanism of Graph2Graph Transformer predicts each labelled edge of the graph using the output embeddings of the tokens that are connected by that edge. Because in this work we are investigating auto-regressive models, this prediction is done one edge at a time. See (Mohammadshahi and Henderson, 2020) for an investigation of non-autoregressive models using our G2GTr architecture.

In this work, the graph edges are labelled dependency relations, which are predicted as part of the actions of a transition-based dependency parser. In particular, the Relation classifier uses the output embeddings of the top two elements on the stack and predicts the label of their dependency relation, conditioned on its direction. There is also an Exist classifier, which uses the output embeddings of the top two elements on the stack and the front of the buffer to predict the type of parser action, `SHIFT`, `SWAP`, `RIGHT-ARC`, or `LEFT-ARC`.

$$\begin{aligned} a^t &= \text{Exist}([g^t_{s_2}, g^t_{s_1}, g^t_{b_1}]) \\ l^t &= \text{Relation}([g^t_{s_2}, g^t_{s_1}] | a^t) \end{aligned} \quad (5)$$

where $g^t_{s_2}$, $g^t_{s_1}$, and $g^t_{b_1}$ are the output embeddings of top two tokens in the stack and the front of buffer, respectively. The Exist and Relation classifiers are MLPs with one hidden layer.

For the transition-based dependency parsing task, the chosen parser action and dependency label are used both to update the current partial dependency structure and to update the parser state.

4 Parsing Models

In this section, we define two Transformer-based models for transition-based dependency parsing, and integrate the Graph2Graph Transformer architecture with them, as illustrated in Figure 1.

4.1 State Transformer

We propose a novel attention-based architecture, called State Transformer (StateTr), which computes a comprehensive representation for the parser state. Inspired by Dyer et al. (2015), we directly use the parser state, meaning both the stack and buffer elements, as the input to the Transformer model. We additionally incorporate components that have proved successful in Dyer et al. (2015). In the remaining paragraphs, we describe each component in more detail.

4.1.1 Input Embeddings

The Transformer architecture takes a sequence of input tokens and converts them into a sequence of input embedding vectors, before computing its context-dependent token embeddings. For the State Transformer model, the sequence of input tokens represents the current parser state, as illustrated in Figure 1a.

Input Sequence: The input symbols include the words of the sentence $\Omega = (w_1, w_2, ..., w_n)$ with their associated part-of-speech tags (PoS) $(\alpha_1, \alpha_2, ..., \alpha_n)$. Each of these words can appear in the stack or buffer of the parser state. Besides, there is the `ROOT` symbol, for the root of the dependency tree, which is always on the bottom of the stack. Inspired by the input representation of BERT (Devlin et al., 2018), we also use two special symbols, `CLS` and `SEP`, which indicate the different parts of the parser state.

The sequence of input tokens starts with the `CLS` symbol, then includes the tokens on the stack from bottom to top. Then it has a `SEP` symbol, followed by the tokens on the buffer from front to back so that they are in the same order in which they appeared in the sentence. Given this input sequence, the model computes a sequence of vectors which are input to the Transformer network. Each vector is the sum of several embeddings, which are defined below.

Input Token Embeddings: The embedding of each token (w_i) is calculated as:

$$T_{w_i} = \text{Emb}(w_i) + \text{Emb}(\alpha_i) \quad (6)$$

where $\text{Emb}(w_i), \text{Emb}(\alpha_i) \in R^m$ are the word and PoS embeddings respectively. For the word embeddings, we use the pre-trained word vectors of the BERT model. During training and evaluation, we use the pre-trained embedding of first sub-word as the token representation of each word and discard embeddings of non-first sub-words due to training efficiency.[2] The PoS embeddings are trained parameters.

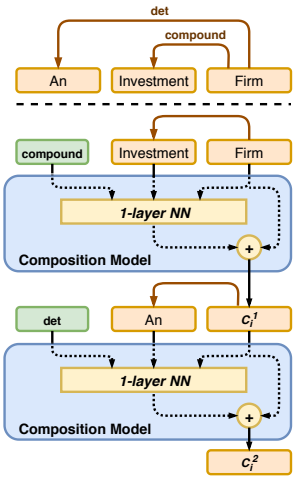


Figure 2: An Example of Composition model.

Composition Model: As an alternative to our proposed graph input method, previous work has shown that complex phrases can be input to a neural network by using recursive neural networks to recursively compose the embeddings of sub-phrases (Socher et al., 2011, 2014, 2013; Hermann and Blunsom, 2013; Tai et al., 2015). We extend the proposed composition model of Dyer et al. (2015) by applying a one-layer feed-forward neural network as a composition model and adding skip connections to each recursive step.[3] Since a syntactic head may contain an arbitrary number of dependents, we compute new token embeddings of head-dependent pairs one at a time as they are specified by the parser, as shown in Figure 2. At each parser step t, we compute each new token embedding C_i^t of token i by inputting to the composition model, its previous token embedding C_j^{t-1} and the embedding of the most recent dependent with its associated dependency label, where j is the position of token i in the previous parser state. At $t = 0$, C_i^0 is set to the initial token embedding T_{w_i}. More mathematical and implementation details are given in Appendix B.

Position and Segment Embeddings: To distinguish the different positions and roles of words in

[2] Using embeddings of first sub-word for each word results in better performance than using the last one or averaging all of them as also shown in previous works (Kondratyuk and Straka, 2019; Kitaev et al., 2019).

[3] These skip connections help address the vanishing gradient problem, and preliminary experiments indicated that they were necessary to integrate pre-trained BERT (Devlin et al., 2018) parameters with the model (discussed in Section 4.4 and Appendix A.A).

the parser state, we add their embeddings to the token embeddings. Position embeddings β_i encode the token's position in the whole sequence.[4] Segment embeddings γ_i encode that the input sequence contains distinct segments (e.g. stack and buffer).

Total Input Embeddings: Finally, at each step t, we sum the outputs of the composition model with the segment and position embeddings and consider them as the sequence of input embeddings for our State Transformer model.

$$x_i^t = C_i^t + \gamma_i + \beta_i \tag{7}$$

4.1.2 History Model

We define a history model similar to Dyer et al. (2015), to capture the information about previously specified transitions. The output h^t of the history model is computed as follows:

$$h^t, c^t = \text{LSTM}((h^{t-1}, c^{t-1}), a^t + l^t) \tag{8}$$

where a^t and l^t are the previous transition and its associated dependency label, and h^{t-1} and c^{t-1} are the previous output vector and cell state of the history model. The output of the history model is input directly to the parser action classifiers in (5).

4.2 Sentence Transformer

We propose another attention-based architecture, called Sentence Transformer (SentTr), to compute a representation for the parser state. This model first uses a Transformer to compute context-dependent embeddings for the tokens in the input sentence. Similarly to Cross and Huang (2016), a separate stack and buffer data structure is used to keep track of the parser state, as shown in Figure 1b, and the context-dependent embeddings of the tokens that are involved in the next parser action are used to predict the next transition. More specifically, the input sentence tokens are computed with the BERT tokeniser (Devlin et al., 2018) and the next transition is predicted from the embeddings of the first sub-words of the top two elements of the stack and the front element of the buffer.[5]

In the baseline version of this model, the Transformer which computes the token embeddings does not see the structure of the parser state nor the partial dependency structure.

In Sentence Transformer, the sequence of input tokens starts with a `CLS` token and ends with a `SEP` token, as in the BERT (Devlin et al., 2018) input representation. It also includes the `ROOT` symbol for the root of the dependency tree. The input embeddings are derived from input tokens as:

$$x_i = \text{Emb}(w_i) + \text{Emb}(\alpha_i) + \beta_i \tag{9}$$

where x_i is the input embedding for token w_i, $\text{Emb}(.)$ is defined as in Equation (6), and β_i is the positional embedding for the element at position i.

4.3 Integrating with G2G Transformer

We use the two proposed attention-based dependency parsers above as baselines, and evaluate the effects of integrating them with the Graph2Graph Transformer architecture. We modify the encoder component of each baseline model by adding the graph input mechanism defined in Section 3.2. Then, we compute the new partially constructed graph as follows:

$$\begin{aligned} Z^t &= \text{Gin}(X, G^t) \\ G^{t+1} &= G^t \cup \text{Gout}(\text{Select}(Z^t, P^t)) \end{aligned} \tag{10}$$

where G^t is the current partially specified graph, Z^t is the encoder's sequence of output token embeddings, P^t is the parser state, and G^{t+1} is the newly predicted partial graph. Gin, and Gout are the graph input and graph output mechanisms defined in Sections 3.2 and 3.3. The Select function selects from Z^t, the token embeddings of the top two elements on the stack and the front of the buffer, based on the parser state P^t. More specifics about each baseline are given in the following paragraphs.[6]

State Tr +G2GTr: To input all the dependency relations in the current partial parse, we add a third segment to the parser state, called the Deleted list D, which includes words that have been removed from the buffer and stack after having both their children and parent specified. The order of words in D is the same as the input sentence. The current partial dependency structure is then input with the graph input mechanism as relations between the words in this extended parser state. To show the effectiveness of the graph input mechanism, we exclude the composition model from the State Transformer model when integrated with the Graph2Graph

[4] Preliminary experiments showed that using position embeddings for the whole sequence achieves better performance than applying separate position embeddings for each segment (More detail in Appendix A.B).

[5] Predicting transitions with the embedding of first sub-word for each word results in better performance than using the last one or all of them as also shown in previous works. (Kondratyuk and Straka, 2019; Kitaev et al., 2019)

[6] A worked example of both baseline models integrated with G2GTr is provided in Appendix C.

Transformer architecture. We will demonstrate the impact of this replacement in Section 6.

Sentence Tr +G2GTr: The current partial dependency structure is input with the graph input mechanism as relations between the first sub-words of the head and dependent words of each dependency relation. For the non-first subwords of each word, we define a new dependency relation with these subwords dependent on their associated first sub-word.

4.4 Pre-Training with BERT

Initialising a Transformer model with the pre-trained parameters of BERT (Devlin et al., 2018), and then fine-tuning on the target task, has demonstrated large improvements in many tasks. But our version of the Transformer has novel inputs that were not present when BERT was trained, namely the graph inputs to the attention mechanism and the composition embeddings (for State Transformer). Also, the input sequence of State Transformer has a novel structure, which is only partially similar to the input sentences which BERT was trained on. So it is not clear that BERT pre-training will even work with this novel architecture. To evaluate whether BERT pre-training works for our proposed architectures, we also initialise the weights of our models with the first n layers of BERT, where n is the number of self-attention layers in the model.

5 Experimental Setup

5.1 Datasets

We evaluate our models on two types of datasets, WSJ Penn Treebank, and Universal Dependency (UD) Treebanks. Following Kulmizev et al. (2019), for evaluation, we include punctuation for UD treebanks and exclude it for the WSJ Penn Treebank (Nilsson and Nivre, 2008).[7]

WSJ Penn Treebank: We train our models on the Stanford dependency version of the English Penn Treebank (Marcus et al., 1993). We use the same setting as defined in Dyer et al. (2015). We additionally add section 24 to our development set to avoid over-fitting. For PoS tags, we use Stanford PoS tagger (Toutanova et al., 2003).

Universal Dependency Treebanks: We also train models on Universal Dependency Treebanks (UD v2.3) (Nivre et al., 2018). We evaluate our models on the list of languages defined in Kulmizev et al. (2019). This set of languages contains different scripts, various morphological complexity and character set sizes, different training sizes, and non-projectivity ratios.

5.2 Models

As strong baselines from previous work, we compare our models to previous traditional transition-based and Seq2Seq models. For a fair comparison with previous models, we consider "traditional" transition-based parsers to be those that predict a fixed set of scores for each decoding step.[8]

To investigate the usefulness of each component of the proposed parsing models, we evaluate several versions. For the State Transformer, we evaluate StateTr and StateTr+G2GTr models both with and without BERT initialisation. To further analyse the impact of Graph2Graph Transformer, we also compare to keeping the composition function of the StateTr model when integrated with G2GTr (StateTr+G2GTr+C). To further demonstrate the impact of the graph output mechanism, we compare to using the output embedding of the `CLS` token as the input to the transition classifiers for both the baseline model (StateCLSTr) and its combined version (StateTr+G2CLSTr). For Sentence Transformer, we evaluate the SentTr and SentTr+G2GTr models with BERT initialisation. We also evaluate the best variations of each baseline on the UD Treebanks.[9]

5.3 Details of Implementation

All hyper-parameter details are given in Appendix F. Unless specified otherwise, all models have 6 self-attention layers. We use the AdamW optimiser provided by Wolf et al. (2019) to fine-tune model parameters. All our models use greedy decoding, meaning that at each step only the highest scoring parser action is considered for continuation. This was done for simplicity, although beam search could also be used. The pseudo-code for computing the elements of the graph input matrix (p_{ij}) for each baseline is provided in Appendix G.

[7] Description of Treebanks are provided in Appendix D.

[8] We do not consider the models of (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019) to be comparable to traditional transition-based models like ours because they make decoding decisions between $O(n)$ alternatives. In this sense, they are in between the $O(1)$ alternatives for transition-based models and the $O(n^2)$ alternatives for graph-based models. Future work will investigate applying Graph2Graph Transformer to these types of parsers as well.

[9] The number of parameters and average running times for each model are provided in Appendix E.

	Dev Set		Test Set	
	UAS	LAS	UAS	LAS
Transition-based:				
Dyer et al. (2015)			93.10	90.90
Weiss et al. (2015)			94.26	91.42
Cross and Huang (2016)			93.42	91.36
Ballesteros et al. (2016)			93.56	92.41
Andor et al. (2016)			94.61	92.79
Kiperwasser and Goldberg (2016)			93.90	91.90
Yang et al. (2017)			94.18	92.26
Seq2Seq-based:				
Zhang et al. (2017)			93.71	91.60
Li et al. (2018)			94.11	92.08
StateTr	91.94	89.07	92.32	89.69
StateTr+G2GTr	92.53	90.16	93.07	91.08
BERT StateTr	94.66	91.94	95.18	92.73
BERT StateCLSTr	93.62	90.95	94.31	91.85
BERT StateTr+G2GTr	**94.96**	**92.88**	**95.58**	**93.74**
BERT StateTr+G2CLSTr	94.29	92.13	94.83	92.96
BERT StateTr+G2GTr+C	94.41	92.25	94.89	92.93
BERT SentTr	95.34	93.29	95.65	93.85
BERT SentTr+G2GTr	**95.66**	**93.60**	**96.06**	**94.26**
BERT SentTr+G2GTr-7 layer	**95.78**	**93.74**	**96.11**	**94.33**

Table 1: Comparisons to SoTA on English WSJ Treebank Stanford dependencies.

Language	Kulmizev et al. (2019)	BERT StateTr+G2GTr	BERT SentTr+G2GTr
Arabic	81.9	82.63	**83.65**
Basque	77.9	74.03	**83.88**
Chinese	83.7	85.91	**87.49**
English	87.8	89.21	**90.35**
Finnish	85.1	80.87	**89.47**
Hebrew	85.5	87.0	**88.75**
Hindi	89.5	93.13	**93.12**
Italian	92.0	92.6	**93.99**
Japanese	92.9	95.25	**95.51**
Korean	83.7	80.13	**87.09**
Russian	91.5	92.34	**93.30**
Swedish	87.6	88.36	**90.40**
Turkish	64.2	56.87	**67.77**
Average	84.87	84.48	**88.06**

Table 2: Labelled attachment score on 13 UD corpora for Kulmizev et al. (2019) with BERT pre-training, BERT StateTr+G2GTr, and BERT SentTr+G2GTr models.

6 Results and Discussion

6.1 English Penn Treebank Result

In Table 1, we show several variations of our models, and previous state-of-the-art transition-based and Seq2Seq parsers on WSJ Penn Treebank.[10] For State Transformer, replacing the composition model (StateTr) with our graph input mechanism (StateTr+G2GTr) results in 9.97% / 11.66% LAS relative error reduction (RER) without / with BERT initialisation, which demonstrates its effectiveness. Comparing to the closest previous model for conditioning of the parse graph, the StateTr+G2GTr model reaches better results than the StackLSTM model (Dyer et al., 2015). Initialising our models with pre-trained BERT achieves 26.25% LAS RER for the StateTr model, and 27.64% LAS RER for the StateTr+G2GTr model, thus confirming the compatibility of our G2GTr architecture with pre-trained Transformer models. The BERT StateTr+G2GTr model outperforms previous state-of-the-art models. Removing the graph output mechanism (StateCLSTr / StateTr+G2CLSTr) results in a 12.28% / 10.53% relative performance drop for the StateTr and StateTr+G2GTr models, respectively, which demonstrates the importance of our graph output mechanism. If we consider both the graph input and output mechanisms together, adding them both (BERT StateTr+G2GTr) to BERT StateCLSTr achieves 21.33% LAS relative error reduction, which shows the synergy of using both mechanisms together. But then adding the composition model (BERT StateTr+G2GTr+C) results in an 8.84% relative drop in performance, which demonstrates again that our proposed graph input method is a more effective way to model the partial parse than recursive composition models.

For Sentence Transformer, the synergy between its encoder and BERT results in excellent performance even for the baseline model (compared to Cross and Huang (2016)). Nonetheless, adding G2GTr achieves significant improvement (4.62% LAS RER), which again demonstrates the effectiveness of the Graph2Graph Transformer architecture. Finally, we also evaluate the BERT SentTr+G2GTr model with 7 self-attention layers instead of 6, resulting in 2.19% LAS RER, which motivates future work on larger Graph2Graph Transformer models.

6.2 UD Treebanks Results

In Table 2, we show LAS scores on 13 UD Treebanks[11]. As the baseline, we use scores of the transition-based model proposed by Kulmizev et al. (2019), which uses the deep contextualized word representations of BERT and ELMo (Peters et al., 2018) as an additional input to their parsing models.

[10] Results are calculated with the official evaluation script provided in https://depparse.uvt.nl/.

[11] Unlabelled attachment scores, and results of development set are provided in the Appendix H. Results are calculated with the official UD evaluation script (https://universaldependencies.org/conll18/evaluation.html).

Our BERT StateTr+G2GTr model outperforms the baseline on 9 languages, again showing the power of the G2GTr architecture. But for morphology-rich languages such as Turkish and Finish, the StateTr parser design choice of only inputting the first sub-word of each word causes too much loss of information, resulting in lower results for our BERT StateTr+G2GTr model than the baseline. This problem is resolved by our SentTr parser design because all sub-words are input. The BERT SentTr+G2GTr model performs substantially better than the baseline on all languages, which confirms the effectiveness of our Graph2Graph Transformer architecture to capture a diversity of types of structure from a variety of corpus sizes.

6.3 Error Analysis

To analyse the effectiveness of the proposed graph input and output mechanisms in variations of our StateTr model pre-trained with BERT, we follow McDonald and Nivre (2011) and measure their accuracy as a function of dependency length, distance to root, sentence length, and dependency type, as shown in Figure 3 and Table 3.[12]. These results demonstrate that most of the improvement of the StateTr+G2GTr model over other variations comes from the hard cases which require a more global view of the sentence.

Dependency Length: The leftmost plot shows labelled F-scores on dependencies binned by dependency lengths. The integrated G2GTr models outperform other models on the longer (more difficult) dependencies, which demonstrates the benefit of adding the partial dependency tree to the self-attention model, which provides a global view of the sentence when the model considers long dependencies. Excluding the graph output mechanism also results in a drop in performance particularly in long dependencies. Keeping the composition component in the StateTr+G2GTr model doesn't improve performance at any length.

Distance to Root: The middle plot shows the labelled F-score for dependencies binned by the distance to the root, computed as the number of dependencies in the path from the dependent to the root node. The StateTr+G2GTr models outperform baseline models on nodes that are of middle depths, which tend to be neither near the root nor near the leaves, and thus require more global information, as well as deeper nodes.

[12]We use MaltEval(Nilsson and Nivre, 2008) tool for computing accuracies. Tables of results for the error analysis in Figure 3, and Table 3 are in the Appendix I.

Type	StateTr+G2GTr	StateTr	StateTr+G2CLSTr
rcmod	86.84	76.38 (-79.5%)	83.91 (-22.3%)
nsubjpass	95.49	92.70 (-61.9%)	94.08 (-31.1%)
ccomp	89.49	81.82 (-73.0%)	87.56 (-18.4%)
infmod	87.38	79.19 (-64.9%)	84.93 (-19.4%)
neg	95.75	94.84 (-21.4%)	93.78 (-46.2%)
csubj	76.94	67.93 (-39.0%)	70.83 (-26.5%)
cop	93.08	92.62 (-6.5%)	91.58 (-21.7%)
cc	90.90	90.45 (-4.9%)	88.80 (-23.1%)

Table 3: F-scores (and RER) of our full BERT model (StateTr+G2GTr), without graph inputs (StateTr), and without graph outputs (StateTr+G2CLSTr) for some dependency types on the development set of WSJ Treebank, ranked by total negative RER. Relative error reduction is computed w.r.t. the StateTr+G2GTr scores.

Sentence Length: The rightmost plot shows labelled attachment scores (LAS) for sentences with different lengths. The relative stability of the StateTr+G2GTr model across different sentence lengths again shows the effectiveness of the Graph2Graph Transformer model on the harder cases. Not using the graph output method shows particularly bad performance on long sentences, as does keeping the composition model.

Dependency Type: Table 3 shows F-scores of different dependency types. Excluding the graph input (StateTr) or graph output (StateTr+G2CLSTr) mechanisms results in a substantial drop for many dependency types, especially hard cases where accuracies are relatively low, and cases such as ccomp which require a more global view of the sentence.

7 Conclusion

We proposed the Graph2Graph Transformer architecture, which inputs and outputs arbitrary graphs through its attention mechanisms. Each graph relation is input as a label embedding to each attention function involving the relation's tokens, and each graph relation is predicted from its token's embeddings like an attention function. We demonstrate the effectiveness of this architecture on transition-based dependency parsing, where the input graph is the partial dependency structure specified by the parse history, and the output graph is predicted one dependency at a time by the parser actions.

To establish strong baselines, we also propose two Transformer-based models for this task, called State

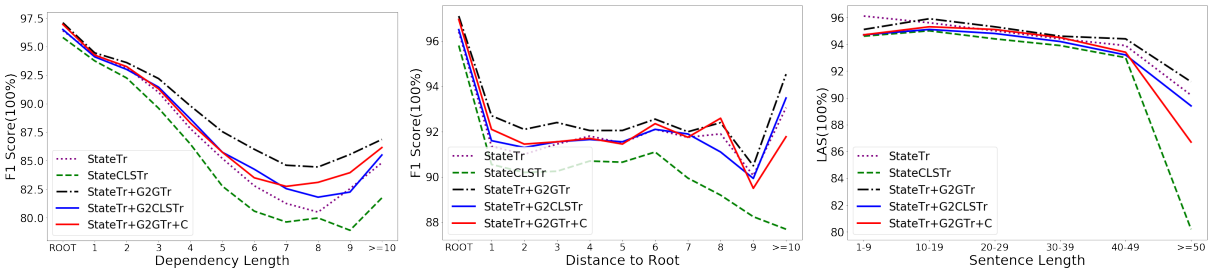


Figure 3: Error analysis of our models on the development set of the WSJ dataset.

Transformer and Sentence Transformer. The former model incorporates history and composition models, as proposed in previous work. Despite the competitive performance of these extended-Transformer parsers, adding our graph input and output mechanisms results in significant improvement. Also, the graph inputs are effective replacements for the composition models. All these results are preserved with the incorporation of BERT pre-training, which results in substantially improving the state-of-the-art in traditional transition-based dependency parsing.

As well as the generality of the graph input mechanism, the generality of the graph output mechanism means that Graph2Graph Transformer can be integrated with a wide variety of decoding algorithms. For example, Mohammadshahi and Henderson (2020) investigate non-autoregressive decoding, which addresses the computational cost of running the G2GTr model once for every dependency edge. Graph2Graph Transformer can also easily be applied to a wide variety of NLP tasks, such as semantic parsing tasks, which we hope to demonstrate in future work.

Acknowledgement

We are grateful to the Swiss NSF, grant CRSII5_180320, for funding this work. We also thank Lesly Miculicich, other members of the IDIAP NLU group, and anonymous reviewers for helpful discussions.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A Smith. 2016. Training with exploration improves a greedy stack-lstm parser. *arXiv preprint arXiv:1603.03793*.

James Cross and Liang Huang. 2016. Incremental parsing with minimal features using bi-directional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 32–37, Berlin, Germany. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

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 (Volume 1: Long Papers)*, pages 334–343, Beijing, China. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019. Left-to-right dependency parsing with pointer networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 710–716, Minneapolis, Minnesota. Association for Computational Linguistics.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 103–110.

James Henderson, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multilingual joint parsing of syntactic and semantic dependencies with a latent variable model. *Computational Linguistics*, 39(4):949–998.

Karl Moritz Hermann and Phil Blunsom. 2013. The role of syntax in vector space models of compositional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 894–904, Sofia, Bulgaria. Association for Computational Linguistics.

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.

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.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Dan Kondratyuk and Milan Straka. 2019. 75 languages, 1 model: Parsing universal dependencies universally. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre. 2019. Deep contextualized word embeddings in transition-based and graph-based dependency parsing – a tale of two parsers revisited.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA. 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 McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

Alireza Mohammadshahi and James Henderson. 2020. Recursive non-autoregressive graph-to-graph transformer for dependency parsing with iterative refinement. *Transactions of the Association for Computational Linguistics*.

Jens Nilsson and Joakim Nivre. 2008. MaltEval: an evaluation and visualization tool for dependency parsing. In *LREC 2008*.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain. Association for Computational Linguistics.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore. Association for Computational Linguistics.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová,

Jesse Kirchner, Kamil Kopacewicz, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phùòng Lê H`ông, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Shinsuke Mori, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lùòng Nguy˜ên Thị, Huy`ên Nguy˜ên Thị Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ̀ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Seyi Williams, Mats Wirén, Tsegay Woldemariam, Tak-sum Wong, Chunxiao Yan, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. 2018. Universal dependencies 2.3. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*, pages 801–809.

Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. 2014. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038, Brussels, Belgium. Association for Computational Linguistics.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks.

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.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Liner Yang, Meishan Zhang, Yang Liu, Nan Yu, Maosong Sun, and Guohong Fu. 2017. Joint pos tagging and dependency parsing with transition-based neural networks.

Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. 2017. Stack-based multi-layer attention for transition-based dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1677–1682, Copenhagen, Denmark. Association for Computational Linguistics.

Appendix A Preliminary Experiments

A.A Skip Connection

Skip connections of composition model help address the vanishing gradient problem, and following experiments show that they are necessary to integrate pre-trained BERT (Devlin et al., 2018) parameters with the model:

Model	UAS	LAS
BERT StateTr	94.78	92.06
BERT StateTr without skip	93.16	90.51

Table 1: Preliminary experiments on the development set of WSJ Penn Treebank for BERT StateTr model with/without skip connections.

A.B Position Embeddings

Following experiments show that using position embeddings for the whole sequence achieves better performance than applying separate position embeddings for each segment:

Model	UAS	LAS
BERT StateTr	94.78	92.06
BERT StateTr with separate pos	93.10	90.39

Table 2: Preliminary experiments on the development set of WSJ Penn Treebank for BERT StateTr model, and its variation with separate position embeddings for each section.

Appendix B Composition Model

Previous work has shown that recursive neural networks are capable of inducing a representation for complex phrases by recursively embedding sub-phrases (Socher et al., 2011, 2014, 2013; Hermann and Blunsom, 2013). Dyer et al. (2015) showed that this is an effective technique for embedding the partial parse subtrees specified by the parse history in transition-based dependency parsing. Since a word in a dependency tree can have a variable number of dependents, they combined the dependency relations incrementally as they are specified by the parser.

We extend this idea by using a feed-forward neural network with `Tanh` as the activation function and skip connections. For every token in position i on the stack or buffer, after deciding on step t, the composition model computes a vector $C_{a,i}^{t+1}$ which is added to the input embedding for that token:

$$\begin{aligned} &C_{a,i}^{t+1} = \mathrm{Comp}((\psi_{a,i}^{t}, \omega_{a,i}^{t}, l_{a,i}^{t})) + \psi_{a,i}^{t} \\ &\text{where:} a \in \{S, B\} \end{aligned} \tag{11}$$

where the Comp function is a one-layer feed forward neural network, and $(\psi_{a,i}^{t}, \omega_{a,i}^{t}, l_{a,i}^{t})$ represents the most recent dependency relation with head $\psi_{a,i}^{t}$ specified by the decision at step t for element in position i in the stack or buffer. In arc-standard parsing, the only word which might have received a new dependent by the previous decision is the word on the top of the stack, $i{=}1$. This gives us the following definition

of $(\psi^t_{a,i}, \omega^t_{a,i}, l^t_{a,i})$:

$$
\begin{cases}
\texttt{RIGHT}-\texttt{ARC}(l): \\
\quad \psi^t_{S,1}=C^t_{S,2},\ \omega^t_{S,1}=C^t_{S,1},\ l^t_{S,1}=l, \\
\quad \psi^t_{S,i\neq 1}=C^t_{S,i+1},\ \omega^t_{S,i\neq 1}=\omega^t_{S,i+1},\ l^t_{S,i\neq 1}=l^t_{S,i+1} \\
\quad \psi^t_{B,i}=C^t_{B,i} \\
\texttt{LEFT}-\texttt{ARC}(l): \\
\quad \psi^t_{S,1}=C^t_{S,1},\ \omega^t_{S,1}=C^t_{S,2},\ l^t_{S,1}=l, \\
\quad \psi^t_{S,i\neq 1}=C^t_{S,i+1},\ \omega^t_{S,i\neq 1}=\omega^t_{S,i+1},\ l^t_{S,i\neq 1}=l^t_{S,i+1} \\
\quad \psi^t_{B,i}=C^t_{B,i} \\
\texttt{SHIFT}: \\
\quad \psi^t_{S,1}=C^t_{B,1},\ \omega^t_{S,1}=\omega^t_{B,1},\ l^t_{S,1}=l^t_{B,1} \\
\quad \psi^t_{S,i\neq 1}=C^t_{S,i-1},\ \omega^t_{S,i\neq 1}=\omega^t_{S,i-1},\ l^t_{S,i\neq 1}=l^t_{S,i-1} \\
\quad \psi^t_{B,i}=C^t_{B,i+1},\ \omega^t_{B,i}=\omega^t_{B,i+1},\ l^t_{B,i}=l^t_{B,i+1} \\
\texttt{SWAP}: \\
\quad \psi^t_{S,1}=C^t_{S,1} \\
\quad \psi^t_{S,i\neq 1}=C^t_{S,i+1},\ \omega^t_{S,i\neq 1}=\omega^t_{S,i+1},\ l^t_{S,i\neq 1}=l^t_{S,i+1} \\
\quad \psi^t_{B,1}=C^t_{S,2},\ \omega^t_{B,1}=\omega^t_{S,2},\ l^t_{B,1}=l^t_{S,2} \\
\quad \psi^t_{B,i\neq 1}=C^t_{B,i-1},\ \omega^t_{B,i\neq 1}=\omega^t_{B,i-1},\ l^t_{B,i\neq 1}=l^t_{B,i-1}
\end{cases}
\tag{12}
$$

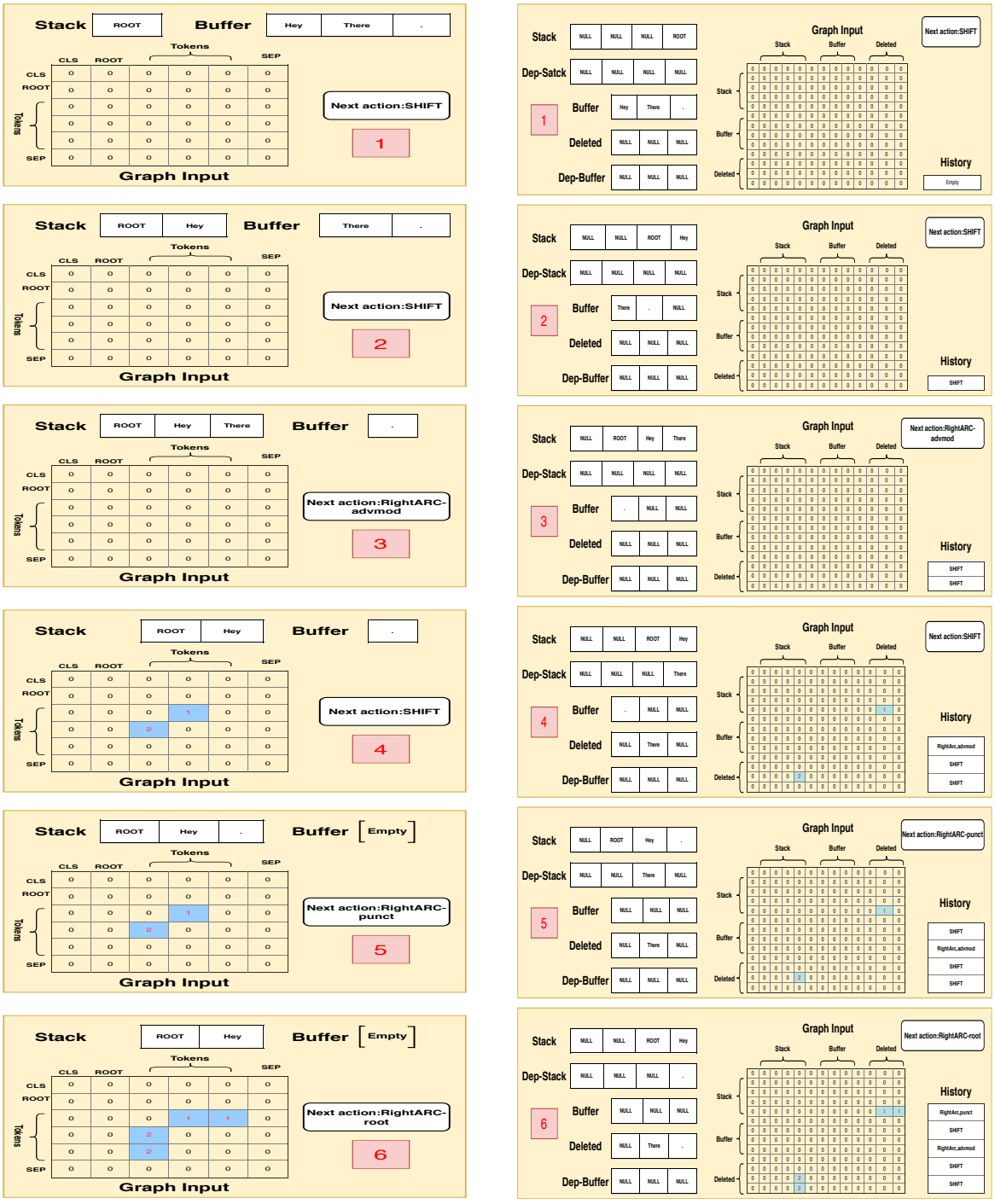
where $C^t_{S,1}$ and $C^t_{S,2}$ are the embeddings of the top two elements of the stack at time step t, and $C^t_{B,1}$ is the embedding of the word on the front of the buffer at time t. $l^t_{a,i} \in R^m$ is the label embedding of the specified relation, including its direction. For all words which have not received a new dependent, the composition is computed anyway with the most recent dependent and label (with a `[NULL]` dependent and label of that position `[L-NULL]` if there is no dependency relation with element i as the head).[13]

At $t=0$, $C^t_{a,i}$ is set to the initial token embedding T_{w_i}. The model then computes Equation 11 iteratively at each step t for each token on the stack or buffer.

There is a skip connection in Equation 11 to address the vanishing gradient problem. Also, preliminary experiments showed that without this skip connection to bias the composition model towards the initial token embeddings T_{w_i}, integrating pre-trained BERT (Devlin et al., 2018) parameters into the model did not work.

[13] Preliminary experiments indicated that not updating the composition embedding for these cases resulted in worse performance.

Appendix C Example of the Graph-to-Graph Transformer parsing



(a) SentTr+G2G-Tr. (b) StateTr+G2G-Tr.

Figure 4: Example of Graph-to-Graph Transformer model integrated with SentTr and StateTr on UD English Treebank (initial sentence: "Hey There .").

Appendix D Description of Treebanks

D.A English Penn Treebank Description

The dataset can be found here under LDC licence. Stanford PoS tagger and constituency converter can be downloaded from here and here, respectively. Here is the detailed information of English Penn Treebank:

Language	Version	Non-projectivity ratio	Train size(2-21)	Development size(22,24)	Test size(23)
English	3	0.1%	39'832	3'046	2'416

Table 3: Description of English Penn Treebank.

D.B UD Treebanks Description

UD Treebanks v2.3 are provided in here. Pre-processing tools can be found here.

Language	Family	Treebank	Order	Train size	Development size	Test size	Non-projectivity ratio
Arabic	non-IE	PADT	VSO	6.1K	0.9K	0.68K	9.2%
Basque	non-IE	BDT	SOV	5.4K	1.8K	1.8K	33.5%
Chinese	non-IE	GSD	SVO	4K	0.5K	0.5K	0.6%
English	IE	EWT	SVO	12.5K	2k	2.1K	5.3%
Finnish	non-IE	TDT	SVO	12.2K	1.3K	1.5K	6.2%
Hebrew	non-IE	HTB	SVO	5.2K	0.48K	0.49K	7.6%
Hindi	IE	HDTB	SOV	13.3K	1.7K	1.7K	13.8%
Italian	IE	ISDT	SVO	13.1K	0.56K	0.48K	1.9%
Japanese	non-IE	GSD	SOV	7.1K	0.51K	0.55K	2.7%
Korean	non-IE	GSD	SOV	4.4K	0.95K	0.99K	16.2%
Russian	IE	SynTagRus	SVO	48.8K	6.5K	6.5K	8.0%
Swedish	IE	Talbanken	SVO	4.3K	0.5K	1.2K	3.3%
Turkish	non-IE	IMST	SOV	3.7K	0.97K	0.97K	11.1%

Table 4: Description of languages chosen from UD Treebanks v2.3.

Appendix E Running Details of Proposed Models

We provide the number of parameters and average run times for each model. For a better understanding, average run time is computed per transition (Second/transition). All experiments are computed with graphics processing unit (GPU), specifically the NVIDIA V100 model. The total number of transitions in the train and development sets are 79664 and 6092, respectively.

Model	No. parameters	Train (sec/transition)	Evaluation (sec/transition)	Evaluation (sent/sec)	Evaluation (token/sec)
StateTr	90.33M	0.098	0.031	16.2	388.13
StateTr+G2GTr	105.78M	0.226	0.071	7.05	168.91
SentTr	63.23M	0.112	0.026	19.27	460.6
SentTr+G2GTr	63.27M	0.138	0.031	16.2	388.13

Table 5: Running details of our models on WSJ Penn Treebank.

Appendix F Hyper-parameters for our parsing models

For hyper-parameter selection, we use manual tuning to find the best numbers. For BERT (Devlin et al., 2018) hyper-parameters, we apply the same optimization strategy as suggested by Wolf et al. (2019). For classifiers and composition model, we use a one-layer feed-forward neural network for simplicity. Then, we pick hyper-parameters based on previous works (Devlin et al., 2018; Dyer et al., 2015). We use two separate optimisers for pre-trained parameters (BERT here) and randomly initialised parameters for better convergence that is shown to be useful in Kondratyuk and Straka (2019). Early stopping (based on LAS) is used during training. The only tuning strategy that has been tried is to use one optimiser for all parameters or two different optimisers for pre-trained parameters and randomly initialised ones. For the latter case, Learning rate for randomly initialised parameters is set to $1e-4$. Results of different variations on the development set of WSJ Penn Treebank are as follows:

Model	UAS	LAS
BERT StateTr with one optimiser	94.66	91.94
BERT StateTr with two optimisers	94.24	91.67
Expected (Average) Performance	94.45	91.81
BERT StateTr+G2GTr with one optimiser	94.96	92.88
BERT StateTr+G2GTr with two optimisers	94.75	92.49
Expected (Average) Performance	94.86	92.69
BERT SentTr with one optimiser	95.34	93.29
BERT SentTr with two optimisers	95.49	93.29
Expected (Average) Performance	95.42	93.29
BERT SentTr+G2GTr with one optimiser	95.27	93.18
BERT SentTr+G2GTr with two optimisers	95.66	93.60
Expected (Average) Performance	95.47	93.40

Table 6: Results on the development set of WSJ Penn Treebank for different optimisation strategy.

Hyper-parameters for training our models are defined as [14]:

Component	Specification
Optimiser	BertAdam
Learning Rate	1e-5
Base Learning Rate	1e-4
Adam Betas(b_1,b_2)	(0.9,0.999)
Adam Epsilon	1e-6
Weight Decay	0.01
Max-Grad-Norm	1
Warm-up	0.01
Self-Attention	
No. Layers(n)	6
No. Heads	12
Embedding size	768
Max Position Embedding	512
BERT model	bert-base-uncased
Classifiers	
No. Layers	2
Hidden size(Exist)	500
Hidden size(Relation)	100
Drop-out	0.05
Activation	$ReLU$
History Model	LSTM
No. Layers	2
Hidden Size	768
Composition Model	
No. Layers	2
Hidden size	768
Epochs	12

Table 7: Hyper-parameters for training our models.

[14]For UD Treebanks, we train our model for 20 epochs, and use "bert-multilingual-cased" for the initialisation. We use pre-trained BERT models of https://github.com/google-research/bert.

Appendix G Pseudo-Code of Graph Input Mechanism

Algorithm 1 Pseudo-code of building graph input matrix for StateTr+G2GTr model.

Graph Sentence(input of attention): P
Graph Input: G
Actions: $A = (a_1,...,a_T)$
Input: (S,B,D)
for $k \leftarrow 1,T$ **do**
    if a_k = `SHIFT` or `SWAP` **then**
        continue
    else
        new relation:$i \xrightarrow{l} j$
        $G_{i,j} = 1$
        $G_{j,i} = 2$
        pop x_j from stack
        change mask of x_j to one
        add l to input embedding of x_j
        P:select G based on Input (S,B,D)
    end if
end for

Algorithm 2 Pseudo-code of building graph input matrix for SentTr+G2GTr model.

Graph Sentence(input of attention): P
Graph Input: G
Actions: $A = (a_1,...,a_T)$
Input: initial tokens
for $k \leftarrow 1,T$ **do**
    if a_k = `SHIFT` or `SWAP` **then**
        continue
    else
        new relation:$i \xrightarrow{l} j$
        $G_{i,j} = 1$
        $G_{j,i} = 2$
        add l to input embedding of j-th word
        $P = G$
    end if
end for

Appendix H UD Treebanks Results

BERT SentTr+G2GTr results:

Language	Test set-UAS	Dev set-UAS	Dev set-LAS
Arabic	87.65	87.01	82.64
Basque	87.17	86.53	83.25
Chinese	89.74	88.36	85.79
English	92.05	93.05	91.3
Finnish	91.46	91.37	89.72
Hebrew	90.85	91.92	89.55
Hindi	95.77	95.86	93.17
Italian	95.15	95.22	93.9
Japanese	96.21	96.68	96.04
Korean	89.42	87.57	84.94
Russian	94.42	94.0	92.70
Swedish	92.49	87.26	85.39
Turkish	74.23	72.52	66.05
Average	90.51	89.80	87.27

Table 8: Dependency scores of BERT SentTr+G2GTr model on the development and test sets of UD Treebanks.

BERT StateTr+G2GTr results:

Language	Test set-UAS	Dev set-UAS	Dev set-LAS
Arabic	86.85	86.41	81.73
Basque	80.91	80.01	73.2
Chinese	87.90	86.64	84.15
English	90.91	91.85	90.11
Finnish	84.35	82.91	78.73
Hebrew	89.51	90.36	87.85
Hindi	95.65	95.92	93.30
Italian	93.5	93.61	92.18
Japanese	95.99	96.18	95.58
Korean	84.35	82.13	77.78
Russian	93.87	93.41	92.09
Swedish	92.49	90.72	88.36
Turkish	65.99	65.92	56.96
Average	87.87	87.39	84.01

Table 9: Dependency scores of BERT StateTr+G2GTr model on the development and test sets of UD Treebanks.

Appendix I Error-Analysis

I.A Dependency Length

Model	ROOT	1	2	3	4	5	6	7	8	9	>=10
BERT StateTr	96.40	94.30	93.15	91.00	87.80	85.19	82.80	81.25	80.49	82.54	84.82
BERT StateCLSTr	95.80	93.75	92.25	89.60	86.45	82.77	80.58	79.62	79.98	78.90	81.74
BERT StateTr+G2GTr	97.10	94.45	93.60	92.20	89.80	87.54	86.00	84.60	84.45	85.55	86.86
BERT StateTr+G2CLSTr	96.50	94.10	93.00	91.45	88.65	85.74	84.25	82.55	81.80	82.25	85.50
BERT StateTr+G2GTr+C	96.95	94.25	93.25	91.30	88.30	85.74	83.50	82.75	83.10	83.95	86.15

Table 10: labelled F-Score vs dependency relation length

Model	ROOT	1	2	3	4	5	6	7	8	9	>=10
BERT StateTr	3046	31245	14478	7565	4153	2461	1681	1185	933	808	5415
BERT StateCLSTr	3046	31409	14473	7553	4131	2406	1641	1153	923	832	5403
BERT StateTr+G2GTr	3047	31240	14457	7572	4171	2465	1688	1188	941	811	5390
BERT StateTr+G2CLSTr	3046	31249	14447	7537	4143	2453	1701	1193	953	814	5434
BERT StateTr+G2GTr+C	3047	31304	14430	7514	4137	2449	1693	1182	951	830	5433
Gold bins	3046	31126	14490	7551	4155	2508	1698	1195	953	821	5427

Table 11: Size of each bin based on dependency length

I.B Distance to Root

Model	ROOT	1	2	3	4	5	6	7	8	9	>=10
BERT StateTr	96.40	91.35	91.00	91.45	91.80	91.50	92.10	91.75	91.90	90.06	93.06
BERT StateCLSTr	95.80	90.55	90.20	90.25	90.70	90.65	91.10	89.95	89.20	88.24	87.69
BERT StateTr+G2GTr	97.10	92.70	92.10	92.40	92.05	92.05	92.55	91.99	92.39	90.49	94.54
BERT StateTr+G2CLSTr	96.50	91.60	91.30	91.55	91.65	91.55	92.10	91.90	91.10	89.93	93.48
BERT StateTr+G2GTr+C	96.95	92.10	91.45	91.55	91.70	91.45	92.35	91.75	92.59	89.50	91.77

Table 12: labelled F-Score vs distance to root

Model	ROOT	1	2	3	4	5	6	7	8	9	>=10
BERT StateTr	3046	16081	15965	12999	9301	6488	3964	2242	1343	740	801
BERT StateCLSTr	3046	15963	15798	12793	9222	6419	3974	2331	1358	827	1239
BERT StateTr+G2GTr	3047	16024	15889	12875	9415	6463	3995	2349	1347	743	823
BERT StateTr+G2CLSTr	3046	16064	15975	12971	9327	6488	3963	2279	1316	708	833
BERT StateTr+G2GTr+C	3047	16079	15947	13020	9419	6481	3930	2259	1274	701	813
Gold bins	3046	16002	16142	13064	9403	6411	3923	2298	1298	702	681

Table 13: Size of each bin based on distance to root

I.C Sentence Length

Model	1-9	10-19	20-29	30-39	40-49	>= 50
BERT StateTr	96.1	95.6	95.0	94.4	93.9	90.2
BERT StateCLSTr	94.6	95.0	94.4	93.9	93.0	80.2
BERT StateTr+G2GTr	95.1	95.9	95.3	94.6	94.4	91.2
BERT StateTr+G2CLSTr	94.7	95.1	94.8	94.2	93.2	89.4
BERT StateTr+G2GTr+C	94.7	95.3	95.1	94.5	93.4	86.7

Table 14: LAS vs. sentence length

I.D Dependency Type Analysis

Type	StateTr+G2GTr	StateTr	StateTr+G2CLSTr
acomp	68.62	58.60 (-31.9%)	66.04 (-8.2%)
advcl	82.75	70.68 (-70.0%)	81.85 (-5.2%)
advmod	83.85	84.40 (3.4%)	84.35 (3.1%)
amod	92.55	92.25 (-4.1%)	92.30 (-3.4%)
appos	87.85	84.94 (-23.9%)	83.25 (-37.8%)
aux	98.55	98.35 (-13.8%)	98.45 (-6.9%)
auxpass	96.65	96.04 (-18.0%)	95.84 (-24.0%)
cc	90.90	90.45 (-4.9%)	88.80 (-23.1%)
ccomp	89.49	81.82 (-73.0%)	87.56 (-18.4%)
conj	86.45	84.70 (-12.9%)	84.15 (-17.0%)
cop	93.08	92.62 (-6.5%)	91.58 (-21.7%)
csubj	76.94	67.93 (-39.0%)	70.83 (-26.5%)
dep	54.66	50.88 (-8.3%)	51.99 (-5.9%)
det	98.25	98.30 (2.9%)	98.00 (-14.3%)
discourse	15.40	15.40 (-0.0%)	28.60 (15.6%)
dobj	94.85	94.10 (-14.6%)	93.95 (-17.5%)
expl	96.39	94.99 (-38.8%)	96.39 (-0.0%)
infmod	87.38	79.19 (-64.9%)	84.93 (-19.4%)
iobj	88.01	90.66 (22.1%)	84.24 (-31.4%)
mark	95.04	95.19 (2.9%)	94.84 (-4.1%)
mwe	86.46	89.71 (24.0%)	88.36 (14.1%)
neg	95.75	94.84 (-21.4%)	93.78 (-46.2%)
nn	94.25	94.10 (-2.6%)	93.65 (-10.5%)
npadvmod	91.89	92.75 (10.5%)	90.64 (-15.5%)
nsubj	96.35	95.55 (-21.9%)	95.60 (-20.6%)
nsubjpass	95.49	92.70 (-61.9%)	94.08 (-31.1%)
num	95.25	94.89 (-7.4%)	95.15 (-2.0%)
number	92.50	90.65 (-24.6%)	92.10 (-5.3%)
parataxis	69.59	62.89 (-22.1%)	72.10 (8.2%)
partmod	82.11	72.82 (-52.0%)	79.74 (-13.3%)
pcomp	88.12	86.49 (-13.7%)	85.77 (-19.8%)
pobj	97.15	96.95 (-7.0%)	96.60 (-19.3%)
poss	97.60	97.15 (-18.7%)	97.70 (4.2%)
possessive	98.29	98.04 (-14.5%)	98.44 (8.9%)
preconj	85.71	84.65 (-7.5%)	84.65 (-7.5%)
predet	79.34	79.34 (-0.0%)	77.44 (-9.2%)
prep	90.25	89.90 (-3.6%)	89.50 (-7.7%)
prt	84.48	83.42 (-6.9%)	83.10 (-8.9%)
punct	88.45	88.05 (-3.5%)	87.65 (-6.9%)
quantmod	86.97	84.40 (-19.7%)	84.49 (-19.0%)
rcmod	86.84	76.38 (-79.5%)	83.91 (-22.3%)
root	97.15	96.40 (-26.3%)	96.50 (-22.8%)
tmod	86.02	86.38 (2.6%)	85.03 (-7.1%)
xcomp	90.75	84.44 (-68.2%)	89.75 (-10.8%)

Table 15: F-score of StateTr, StateTr+G2GTr, and StateTr+G2CLSTr models for the dependency types on the development set of WSJ Treebank. Relative error reduction is computed by considering StateTr+G2GTr scores as the reference.