Transition-based Parsing with Stack-Transformers

Ramón Fernandez Astudillo[†]   **Miguel Ballesteros**[*◇]
Tahira Naseem[†]   **Austin Blodget**[*♡]   **Radu Florian**[†]
[†]IBM Research   [◇]Amazon AI   [♡]Georgetown University
{ramon.astudillo,tnassem,raduf}@ibm.com

Abstract

Modeling the parser state is key to good performance in transition-based parsing. Recurrent Neural Networks considerably improved the performance of transition-based systems by modelling the global state, e.g. stack-LSTM parsers, or local state modeling of contextualized features, e.g. Bi-LSTM parsers. Given the success of Transformer architectures in recent parsing systems, this work explores modifications of the sequence-to-sequence Transformer architecture to model either global or local parser states in transition-based parsing. We show that modifications of the cross attention mechanism of the Transformer considerably strengthen performance both on dependency and Abstract Meaning Representation (AMR) parsing tasks, particularly for smaller models or limited training data.

1 Introduction

Transition-based Parsing transforms the task of predicting a graph from a sentence into predicting an action sequence of a state machine that produces the graph (Nivre, 2003, 2004; Kubler et al., 2009; Henderson et al., 2013). These parsers are attractive for their linear inference time and interpretability, however, their performance hinges on effective modeling of the parser state at every decision step.

Parser states typically comprise two memories, a buffer and a stack, from which tokens can be pushed or popped (Kubler et al., 2009). Traditionally, parser states were modeled using hand selected *local* features pertaining only to the words on the top of the stack or buffer (Nivre et al., 2007; Zhang and Nivre, 2011, inter-alia). With the widespread use of neural networks, *global* models of the parser state such as the stack-LSTM (Dyer et al., 2015) allowed encoding the entire buffer and

stack. It was later shown that *local* features of the stack and buffer extracted from contextual word representations, such as Bi-LSTMs, could outperform global modeling (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2016).

With the rise of the Transformer model (Vaswani et al., 2017), various approaches have been proposed that leverage this architecture for parsing (Kondratyuk, 2019; Kulmizev et al., 2019; Mrini et al., 2019; Ahmad et al., 2019; Cai and Lam, 2020). In this work we revisit the local versus global paradigms of state modeling in the context of sequence-to-sequence Transformers applied to action prediction for transition-based parsing. Similarly to previous works for RNN sequence to sequence (Liu and Zhang, 2017; Zhang et al., 2017), we propose a modification of the cross-attention mechanism of the Transformer to provide global parser state modeling. We analyze the role of local versus global parser state modeling, stack and buffer modeling, effects model size as well as task complexity and amount of training data.

Results show that local and global state modeling of the parser state yield more than 2 percentage points absolute improvement over a strong Transformer baseline, both for dependency and Abstract Meaning Representation (AMR) parsing. Gains are also particularly large for smaller train sets and smaller model sizes, indicating that parser state modeling, can compensate for both. Finally, we improve the AMR transition-based oracle (Ballesteros and Al-Onaizan, 2017a), yielding best results for a transition-based system and second overall.

2 Global versus Local Parser State

Given pair of sentence $\mathbf{w} = w_1, w_2 \cdots w_N$ and graph $\mathbf{g}$, transition-based parsers learn an action sequence $\mathbf{a} = a_1, a_2 \cdots a_T$, that applied to a state machine yields the graph $\mathbf{g} = M(\mathbf{a}, \mathbf{w})$. Actions

---

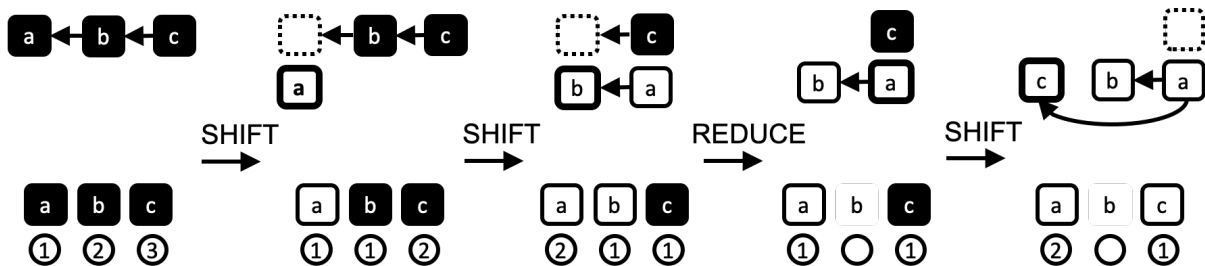[*]Miguel's and Austin's contributions were carried out while at IBM Research.

Figure 1: Encoding of buffer and stack for action sequence $a = \{\text{SHIFT}, \text{SHIFT}, \text{REDUCE}, \text{SHIFT}\}$ and sentence $w = \{a, b, c\}$. The stack-LSTM is at the top, with hidden states representation of buffer (black) and stack (white) displayed. The stack-Transformer is at the bottom, with masks for cross-attention heads attending buffer (black) and stack (white) displayed. Circles indicate extra cross-attention positions relative to stack and buffer.

of the state machine generally move words from a buffer, that initially contains the entire sentence, to a stack. Components of the graph, such as edges or nodes, are created by applying transformations to words in the stack. The correct action sequence is given by an oracle $\mathbf{a} = O(\mathbf{w}, \mathbf{g})$, which is generally rule-based. In principle, one could learn the sentence to action mapping $\mathbf{w} \rightarrow \mathbf{a}$ as a sequence to sequence problem

$$p(\mathbf{a} \mid \mathbf{w}) = \prod_{t=1}^{T} p(a_t \mid \mathbf{a}_{<t}, \mathbf{w}),$$

similarly to e.g. Machine Translation. In practice, this approach does not accurately represent the parser state and thus shows limited performance. The parser state at step t is defined implicitly by $(\mathbf{a}_{<t}, \mathbf{w})$. This translates to an explicit state at step t where the stack contains some tokens about to be processed, sometimes along with new composed vector representations, and the buffer contains the remainder of tokens in the sentence. Buffer and stack increase (push) or decrease (pop) their size dynamically with each time step as shown in Fig. 1.

The transition-based formalism relies heavily on the explicit representation of the state i.e. buffer and stack configurations. Prior to widespread use of Neural Networks, *local* features limited to top of the stack and buffer already achieved good performances (Nivre et al., 2007; Zhang and Nivre, 2011, inter-alia). The introduction of stack-LSTMs (Dyer et al., 2015) made possible modeling the *global* state of the parser by separately encoding action history $\mathbf{a}_{<t}$, and the dynamically changing stack and buffer with LSTMs (Hochreiter and Schmidhuber, 1997). In addition to this, stack-LSTMs used the transition-based formalism to recursively build vector representations of sub-graphs, similarly to a graph neural network.

Another well known LSTM model is the Bi-LSTM feature parser (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2016). In this case, a contextual representation of the sentence is first built with a Bi-LSTM $\mathbf{h} = \text{BiLSTM}(\mathbf{w})$. At each time step t the stack configuration determined by $\mathbf{a}_{<t}$ is used to select the elements from $\mathbf{h}$ corresponding to words on the top of the stack and buffer. Although the features utilize local information of the buffer and stack, the use of a strong contextual representation proved to be sufficient and this remains one of the most widely used forms of parsing today.

3 Modeling Parser State in Transformers

3.1 From stack-LSTMs to stack-Transformers

In transition-based parsers, at a given time step t, input tokens $\mathbf{w}$ may be on the buffer, stack or reduced. As displayed in Fig. 1 (top), to encode this state stack-LSTMs unroll LSTMs over the stack and buffer following their respective word order, which can be different from the sentence's token order. If an element is added to the buffer or stack, it is only necessary to unroll one additional LSTM cell. If an element is removed under a pop operation (e.g. REDUCE), stack-LSTMs move back a pointer to reuse previously computed hidden states. This allows efficient encoding of the dynamically changing stack and buffer.

Unlike LSTMs, Transformers (Vaswani et al., 2017) encode sequences through an attention mechanism (Bahdanau et al., 2015) as a weighted sum of tokens plus position embeddings. One can take advantage of this mechanism to replace LSTMs with Transformers for stack and buffer encoding. Since Transformers just sum token representations, under a pop operation elements can be masked out and

there is no need for a pointer. Furthermore, since Transformers use multiple heads one can have separate modeling of stack and buffer by specializing two heads of the attention mechanism, see Fig. 1 (bottom), while the other heads remain free.

In practical terms, we modify the cross-attention mechanism of the Transformer decoder. For example, for the head attending the stack, the score function between action history encoding b_t (query) and hidden representation of word h_i (key) is given by

$$e_{ti}^{\text{stack}} = \frac{b_t W^Q \left((h_i + p_{ti}^{\text{stack}}) W^K \right)^T}{\sqrt{d}} + m_{ti}^{\text{stack}},$$

where m_{ti} is a $\{-\infty, 0\}$ mask, p_{ti} are the position embeddings for elements in the stack, $\mathbf{h} = f(\mathbf{w})$ is the output of the Transformer encoder. The attention would be computed from the score function as

$$\alpha_{ti} = \text{softmax}(\mathbf{e}_t)_i$$

Both mask and positions change for each word and time-step as the parser state changes, but they imply little computation overhead and can be precomputed for training. Henceforth this modification will be referred to as stack-Transformer.

3.2 Labeled SHIFT Multi-task

It is common practice for transition-based systems to add an additional Part of Speech (POS) or word prediction task (Bohnet and Nivre, 2012). This is achieved by labeling the SHIFT action, that moves a word from the buffer to the stack, with the word's tag. This decorated actions become part of the action history $\mathbf{a}_{<t}$, which was expected to give better visibility into stack/buffer content and exploit Transformer's attentional encoding of history. In initial experiments, POS tags produced a small improvement while word prediction led to performance decrease. It was observed, however, that prediction of only $100 - 300$ most frequent words, leaving SHIFT undecorated otherwise, led to large performance increases. This is thus the method reported in the experimental setup as alternative parser state modeling.

4 Experiments and Results

To test the proposed approach, different parsing tasks were selected. Dependency parsing in the English-Treebank, is well known and well resourced (40K sentences). The AMR2.0 semantic parsing task is more complex, encompassing named entity recognition, word sense disambiguation and co-reference among other sub-tasks, also well resourced (36K sentences). AMR1.0 has around 10K sentences and can be considered as AMR with limited train data.

The dependency parsing setup followed Dyer et al. (2015), in the setting with no POS tags. This has only SHIFT, LEFT-ARC(label), and RIGHT-ARC(label) base action with a total of 82 different actions. Results were measured in terms of (Un)labeled Attachment Scores (UAS/LAS).

The AMR setup followed Ballesteros and Al-Onaizan (2017a), which introduced new actions to segment text and derive nodes or entity sub-graphs. In addition, we use the alignments and wikification from Naseem et al. (2019). Unlike previous works, we force-aligned the unaligned nodes to neighbouring words and allowed attachment to the leaf nodes of entity sub-graphs, this increased oracle Smatch from 93.7 to 98.1 and notably improved model performance. We therefore provide results for the Naseem et al. (2019) oracle for comparison. Both previous works predict a node creation action and then a node label, or call a lemmatizer if no label is found. Instead, we directly predicted the label and added COPY actions to construct node names from lemmas[1] or surface words, resulting in a maximum of 9K actions. Node label predictions were limited to those seen during training for the word on the top of the stack. Results were measured in Smatch (Cai and Knight, 2013) using the latest version $1.0.4$[2].

Regarding model implementation, all models were implemented on the fairseq toolkit and trained with only minor modifications over the MT model hyper-parameters (Ott et al., 2018). This used cross-entropy training with learning rate $5e-4$, inverse square root scheduling with min. $1e-9$, 4000 warm-up updates with learning rate $1e-7$, and maximum 3584 tokens per batch. Adam parameters 0.9 and 0.98, label smoothing was reduced to 0.01[3]. All models used 6 layers of encoding and decoding with size 256 and 4 attention heads, except the normal Transformers in AMR, which performed better on a 3/8 layer configuration instead of 6/6. To study the effect of model size, *small* versions of all models using a 2/2 configuration were also tested.

---

[1] We used https://spacy.io/ as lemmatizer

[2] Note that bug fixes in Smatch seem to yield 0.3 improvements against its 2019 version.

[3] see https://github.com/pytorch/fairseq/tree/master/examples/scaling_nmt

Model	Penn-Treebank		AMR 1.0	AMR 2.0
	UAS	LAS	Smatch	Smatch
a) vanilla sequence-to-sequence Transformer	93.9±0.2	92.0±0.3	70.5 ±0.1	77.7±0.1
b) label top 100 SHIFT multi-task	95.2±0.1	93.5±0.1	74.9±0.3	79.0±0.1
c) 1 head attends stack, 1 head attends buffer	**95.7±0.1**	**94.1±0.2**	**76.3±0.0**	**79.5±0.2**
d) 1 head attends stack, 1 head attends buff. + stack/buff. positions	94.8±0.1	92.7±0.2	70.7±0.2	70.4±2.6
e) 1 head attends entire buffer	**95.8±0.1**	**94.0±0.1**	76.2±0.1	**79.7±0.0**
f) 1 head attends entire stack	95.5±0.1	93.8±0.1	75.9±0.2	79.4±0.2
g) 1 head attends top two words of the buffer	95.7±0.1	93.8±0.1	75.9±0.2	79.4±0.1
h) 1 head attends top two words of the stack	95.4±0.1	93.8±0.1	76.1±0.2	79.4±0.2
+ label top 100 SHIFT multi-task	95.4±0.1	93.8±0.1	**76.5±0.1**	79.4±0.0
small a) sequence-to-sequence Transformer	91.0±0.6	88.5±0.6	66.5±0.1	74.4±0.2
small b) label top 100 SHIFT multi-task	94.6±0.2	92.5±0.2	72.6±0.1	76.7±0.1
small c) 1 head attends stack, 1 head attends buffer	**95.5±0.1**	**93.7±0.1**	**75.8±0.0**	**79.1±0.1**

Table 1: Dev-set performance for PTB (simpler parsing task), AMR2.0 (complex parsing task) and AMR1.0 (one third of AMR2.0 train data). Top: encoding parser state through multi-task or multi-head attention modification. Middle: different encodings of global/local state by multi-head attention modification. Bottom: Effect of small model size (4 layers). All models use fixed RoBERTa-base contextualized embeddings, checkpoint average and beam 10. All results are average of 3 different random seeds with standard deviation indicated with ±.

Model	UAS	LAS
Dozat and Manning (2016)	95.7	94.0
F-Gonz. and G-Rodr. (2019)	96.0	94.4
Moh. and Hen. (2020)[β]	96.7	95.0
Mrini et al. (2019)[X]	**97.3**	**96.3**
a) Transformer	94.4±0.1	92.6±0.2
b) Transformer + (mul.-task)	96.0±0.1	94.4±0.1
e) Stack-Transformer (buff)	96.3±0.0	94.7±0.0
c) Stack-Transformer	96.2±0.1	94.7±0.0

Table 2: Test-set performance for Table 1 selections and prior art on the English Penn-Treebank.

We used RoBERTa-base (Liu et al., 2019) embeddings without fine-tuning as input, averaging wordpieces to obtain word representations. Weight averaging of the best 3 checkpoints (Junczys-Dowmunt et al., 2016) and beam 10 were used in all models. This improves results at most by 0.4/0.8 points for AMR2.0/AMR1.0 with no significant differences across models. Models were trained for a fixed number of epochs, selecting the best model on validation by either LAS or Smatch. A maximum epoch number of 80 − 120 was set to guarantee a margin of 5 epochs from best model to last epoch. No other hyper-parameters were changed across models or tasks. Training took at most 6h on a Nvidia Tesla v100 GPU. It should be noted that this is around 10 times faster than our Pytorch stack-LSTM implementation for the same data. The labeled SHIFT strategy used the 100 most frequent words.

5   Analysis of Results

Table 1 compares the standard Transformer, with and without multi-task with the stack-Transformer, its components, and smaller versions of all models. Comparing LAS and Smatch, stack-transformer provides around 2 points improvement against Transformer on PTB and AMR2.0, and 0.5 points improvement against its multi-task version (a-c). This improvement becomes sensibly larger for the smaller train set AMR1.0 with 5.8 and 1.4 point gains over the Transformer and its multi-task version respectively. Differences are also larger for the 4 layer version of the models. Under this setting, the stack-Transformer looses only 0.4 points against a 12 layer model in AMR2.0. In this same setting, the Transformer and its multi-task version loose 3.3 and 2.3 points respectively, pointing to the fact that modeling parser state compensates for less training data or smaller models.

Regarding ablation of the stack-Transformer components, the use of stack/buffer positions seems clearly detrimental (d) in all scenarios. This was a consistent pattern across various variants for which we do not report numbers such as sinusoidal versus learnable positions and reducing the position range to top three of the stack and buffer. One possible explanation is that positions varying after each time step may be hard to learn, particularly if injected directly in the decoder. It is also worth noting, than the combination of multi-task and stack-Transformer produced little improvement or was even detrimental pointing to their similar role. Re-

Model	AMR1.0	AMR2.0
Lyu and Titov (2018) (G.R.)	73.7	74.4
Naseem et al. (2019)[B]	-	75.5
Zhang et al. (2019) (G.R.) [B]	71.3	77.0
Cai and Lam (2020) [β]	74.0	78.7
Cai and Lam (2020) (G.R.) [β]	**75.4**	**80.2**
a*) Transformer	68.8±0.1	75.9±0.3
a) Transformer	69.2±0.2	77.2±0.2
b) Transformer (mul.-task)	74.0±0.2	78.0±0.1
e) Stack-Transformer (buff)	75.1±0.3	78.8±0.1
c) Stack-Transformer	**75.4±0.0**	79.0±0.1

Table 3: Test-set performance for Table 1 selections and prior art on the AMR1.0 and AMR2.0 in terms of Smatch.

sults for the weakest of the stack-Transformer variants are provided (h).

Comparing across different attention modifications (e-h), most methods perform similarly although there seems to be some evidence for global (full buffer, full stack) variants being more performant. Modeling of the buffer seems also more important than modeling of the stack. One possible explanation for this is that, since the total number of heads is kept fixed, it may be more useful to gain an additional free head than modeling the stack content. Furthermore without recursive representation building, as in stack-LSTMs, the role of the stack can be expected to be less important.

Tables 2 and 3 compare with prior works. Pretrained embeddings used are indicated as XL-net-large[X] (Yang et al., 2019), BERT base[β] and large[B] (Devlin et al., 2019), Graph Recategorization, which utilizes an external entity recognizer (Lyu and Titov, 2018; Zhang et al., 2019) as (G.R.) and a* indicates the Naseem et al. (2019) oracle.

Overall, the stack-Transformer is competitive against recent works particularly for AMR, likely due to the higher complexity of the task. Compared to prior AMR systems, it is worth noting the large performance increase against stack-LSTM (Naseem et al., 2019), while sharing a similar oracle and embeddings and not using reinforcement learning fine-tuning. The stack-Transformer also matches the best reported AMR system (Cai and Lam, 2020) on AMR1.0 without graph recategorization, but using RoBERTa instead of BERT embeddings and provided the second best reported scores on the higher resourced AMR2.0 [4].

---

[4]Code available under `https://github.com/IBM/transition-amr-parser/`

6 Related Works

While inspired by stack-LSTMs (Dyer et al., 2015), the stack-Transformer lacks their elegant recursive composition, where representations for partial graph components are added to the stack and used in subsequent representations. It allows, however, to model the global parser state in a simple way that is easy to parallelize, and shows large performance gains against stack-LSTMs on AMR. The proposed modified attention mechanism, could also be interpreted as a form of feature-based parser (Kiperwasser and Goldberg, 2016), where the parser state is used to select encoder representations, integrated into a Transformer sequence to sequence model.

The modification of the attention mechanism to reflect the parse state has been applied in the past to RNN sequence-to-sequence models. Liu and Zhang (2017) propose the use of a boundary to separate stack and buffer attentions. While simple, this precludes the use of SWAP actions needed for AMR parsing and non-projective parsing. Zhang et al. (2017) mask out reduced words and add a bias to the attention weights for words in the stack. While being the closest to the proposed technique, this method does not separately model stack and buffer nor retains free attention heads, which we consider a fundamental advantage. We also provide evidence that modeling the parser state still produces gains when using pre-trained Transformer embeddings and provide a detailed analysis of components. Finally, RNN (Ma et al., 2018) and self-attention (Ahmad et al., 2019) Stack-Pointer networks sum encoder representations based on local graph structure, which can be interpreted as masked uniform attention over 3 words and is related to the previous methods.

7 Conclusions

We have explored modifications of sequence-to-sequence Transformers to encode the parser state for transition-based parsing, inspired by stack-LSTM's global modeling of the parser state. While simple, these modifications consistently provide improvements against a normal sequence to sequence Transformer in transition-based parsing, both for dependency parsing and AMR parsing tasks. Results also point to the benefits of modeling the parser state as a way to compensate for limited training resources or limitation in model sizes.

References

Wasi Ahmad, Zhisong Zhang, Xuezhe Ma, Eduard Hovy, Kai-Wei Chang, and Nanyun Peng. 2019. On difficulties of cross-lingual transfer with order differences: A case study on dependency parsing. 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 2440–2452, Minneapolis, Minnesota. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

Miguel Ballesteros and Yaser Al-Onaizan. 2017a. AMR parsing using stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Miguel Ballesteros and Yaser Al-Onaizan. 2017b. AMR parsing using stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275, Copenhagen, Denmark. Association for Computational Linguistics.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465. Association for Computational Linguistics.

Deng Cai and Wai Lam. 2020. Amr parsing via graph-sequence iterative inference. *arXiv preprint arXiv:2004.05572*.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria. Association for Computational Linguistics.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*.

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.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. 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, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multilingual joint parsing of syntactic and semantic dependencies with a latent variable model. *Comput. Linguist.*, 39(4):949–998.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Marcin Junczys-Dowmunt, Tomasz Dwojak, and Rico Sennrich. 2016. The AMU-UEDIN submission to the WMT16 news translation task: Attention-based NMT models as feature functions in phrase-based SMT. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 319–325, Berlin, Germany. 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.

Daniel Kondratyuk. 2019. 75 languages, 1 model: Parsing universal dependencies universally. *arXiv preprint arXiv:1904.02099*.

Sandra Kubler, Ryan McDonald, Joakim Nivre, and Graeme Hirst. 2009. *Dependency Parsing*. Morgan and Claypool Publishers.

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. *arXiv preprint arXiv:1908.07397*.

Jiangming Liu and Yue Zhang. 2017. Encoder-decoder shift-reduce syntactic parsing. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 105–114, Pisa, Italy. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Chunchuan Lyu and Ivan Titov. 2018. AMR parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407, Melbourne, Australia. 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.

Alireza Mohammadshahi and James Henderson. 2020. Recursive non-autoregressive graph-to-graph transformer for dependency parsing with iterative refinement. *arXiv preprint arXiv:2003.13118*.

Khalil Mrini, Franck Dernoncourt, Trung Bui, Walter Chang, and Ndapa Nakashole. 2019. Rethinking self-attention: An interpretable self-attentive encoder-decoder parser. *arXiv preprint arXiv:1911.03875*.

Tahira Naseem, Abhishek Shah, Hui Wan, Radu Florian, Salim Roukos, and Miguel Ballesteros. 2019. Rewarding Smatch: Transition-based AMR parsing with reinforcement learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4586–4592, Florence, Italy. Association for Computational Linguistics.

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

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, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation (WMT)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. Broad-coverage semantic parsing as transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3784–3796, Hong Kong, China. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA. Association for Computational Linguistics.

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.