Transition-based Abstract Meaning Representation
Parsing with Contextual Embeddings

Yichao Liang
Department of Computer Science
University of Oxford
ycliang6@gmail.com

Abstract
The ability to understand and generate languages sets human cognition apart from other known life forms. We study a way of combining two of the most successful routes to meaning of language–statistical language models and symbolic semantics formalisms–in the task of semantic parsing. Building on a transition-based, Abstract Meaning Representation (AMR) parser, AMREAGER, we explore the utility of incorporating pretrained context-aware word embeddings–such as BERT and RoBERTa–in the problem of AMR parsing, contributing a new parser we dub as AMRBERGER. Experiments find these rich lexical features alone are not particularly helpful in improving the parser’s overall performance as measured by the SMATCH score when compared to the non-contextual counterpart, while additional concept information empowers the system to outperform the baselines. Through lesion study, we found the use of contextual embeddings helps to make the system more robust against the removal of explicit syntactical features. These findings expose the strength and weakness of the contextual embeddings and the language models in the current form, and motivate deeper understanding thereof.

1 Introduction
Semantic formalism attempts to, from one perspective represent the meaning of natural language while disregarding its superficial features, and from another precisely represent our knowledge of the state of affairs in the world. Abstract Meaning Representation (AMR) is a sentence-level semantic formalism of languages that attempts to do a practical, replicable amount of canonicalization of sentences, while captures many aspects of meaning in a simple data structure–directed graphs. It encodes information about semantic relations, named entities, co-reference, negation, and modality, which are usually independently addressed in natural language processing (NLP). This representation has been shown to be useful in many downstream NLP tasks, including text summarization, question answering, machine translation, and more.

To use AMR in other NLP tasks, parsers are needed to generate AMR graphs automatically from linguistic sentences. In recent years, many parsers have been developed. Among them, graph-based and transition-based are the two main families; graph-based parsing decomposes the task into concept and relation identification, then determines a full AMR graph with decoding algorithms, e.g., maximum spanning tree, while transition-based systems build up AMR graphs incrementally by applying a sequence of transition actions. The parser of our study, AMREAGER, is an instance of the greedy, transition-based approaches (greedy in the sense that the parser provides a single choice of action at each state, and a single parse is returned in the end, without any exploration and backtracking).

∗Written in 2020 during an internship at Shay Cohen’s lab, The University of Edinburgh.
To predict the transition action at each time step, a parser takes into account lexical information—especially the semantics of words—along with possibly syntactical features from the parts of the sentence it will parse and/or has parsed. To supply this, word embeddings stemmed from the distributional hypothesis [8] are employed. For example, GloVe [21] is a widely-used model for obtaining context-independent word features by computing global word-word co-occurrence statistics of a large corpus, BERT [5] and RoBERTa [14] embeddings are a more modern, contextual approach obtained from bidirectional transformer models [28], which have demonstrated strong signaling ability on a variety of NLP tasks.

AMREAGER was originally developed to use context-independent word embeddings, e.g. GloVe, and performed poorly at, in particular, the Semantics Role Labelling (SRL) metric, a crucial subtask of AMR parsing (as discussed in Damonte et al. [4]). On the other hand, Shi and Lin [24] show a simple BERT model can achieve state-of-the-art in the SRL task, motivating us to experiment with equipping contextual embedding (BERT, RoBERTa) for AMREAGER, which is the main contribution of this paper.

The paper is organized as follows. Section 2 discusses the background of AMR, AMREAGER and contextual embeddings. Section 3 describes the design of our approach, and Section 4 details the experimental design and results.

2 Background

AMR In the AMR framework, each sentence is annotated with the PENMAN notation [17], a serialized representation for directed, rooted graphs, similar to Lisp S-Expressions. As an example, Fig. 1 shows an AMR of the sentence “The dog wants to eat” in PENMAN notation and its corresponding graph, respectively. In the graph, each node is labeled by a variable and a concept (e.g., “d / dog”) or a constant (e.g. negation, a quantity), while each edge represents a relation, similar to PropBank arguments [10]. The representation can be roughly expressed as a grammar in Backus–Naur form as in Fig. 2. Notably, there isn’t always a straightforward alignment between words in a sentence and its AMR notes; the mapping from a graph node to a word is neither injective nor surjective (i.e., each word can correspond to zero or more nodes). However, alignments are often still required in training AMR parsers, including for AMREAGER. JAMR [7] is a popular aligner based on a set of heuristics rules and a greedy search process. And it’s used in this work to generate alignments.

Figure 1: (a) shows an AMR of sentence “The dog wants to eat” where “d / dog” means “d is an instance of the concept dog” and “ARG-0” represents a relation. (b) shows the corresponding graph representation of the AMR structure in (a).

(w / want-01
 :ARG0 (d / dog)
 :ARG1 (e / eat-01
 :ARG0 d))

Figure 2: The rough grammar of AMR
Notation  Let \([n]\) represent the set \(\{1, \ldots, n\} \in \mathbb{Z}\). Following the notation of \([4]\), each AMR structure is defined as a 3-tuple \((G, x, \pi)\), where \(x = (x_1, \ldots, x_n)\) is a sentence with \(n\) tokens.

The \textsc{AMREager} Transition System  Taking inspiration from the dependency parsing transition system \textsc{ArcEager} \([20]\), and insight that AMR parsing differs from English dependency parsing mainly in the following three aspects: (i) AMR parsing requires the mapping from word tokens to AMR nodes; (ii) the existence non-projectivity in the graph (in 51% of the sampled sentences); (iii) the existence of reentrant nodes (in 93% of the sampled sentences \([4]\)).

\textsc{AMREager} is proposed by Damonte et al. \([4]\) as a data-driven transition system for generating ARM graphs as sequences of transition actions, with sentence tokens and their dependency graphs as input. A run of the transition system can be completely characterized by a sequence of \textit{configurations} (the state of the parser, the stack, the input buffer of word tokens, and the set of relations representing an AMR graph) and the transition actions among them.

\textsc{AMREager} makes the following three assumptions when perform parsing: (i) AMR graphs are directed acyclic graphs (DAGs). (In practice, AMR graphs with cycles are rare, but do exist.); (ii) Each node in a graph aligns to at most one sentence token, meaning that the node represents a part of the concept expressed by the token. Consequently, co-references of nodes from tokens are not considered; (iii) Co-references between nodes are ignored because reentrancy happens mostly only between siblings of a control verb.

The transition system consists of four components: a \textit{parser} (consisting of an oracle used during training for generating training/validation dataset for all components of the system), a \textit{buffer} for storing input, a \textit{stack} for storing partially completed graph, and a \textit{list} for recording parsing actions.

The parser employs four modules to decide the actions to take:

1. a transition classifier \(\theta(\cdot)\): a feed-forward neural network to decide the transition action to take, among \texttt{Shift}, \texttt{LArc}, \texttt{RArc} and \texttt{Reduce}, based on the current configuration);
2. a concept identifier \(\chi(\cdot)\): a dictionary extracted from the training dataset which decides the concept sub-graph (if any) related to a token to attach, once a \texttt{Shift} action takes place);
3. an edge labelling classifier \(\lambda(\cdot)\): another feed-forward neural net for deciding the relation labels, e.g., “:ARG0”, “:location”, etc, between the top-two nodes of the stack after either a \texttt{LArc} or \texttt{RArc} action takes place;
4. a reentrancy classifier \(\rho(\cdot)\): implemented again by a feed-forward neural net which decides whether to create a reentrant edge after a \texttt{Reduce} action).

For the parser to decide what actions to apply to a configuration, feature templates are designed to be applied to the top nodes of the stack and/or the top tokens in the buffer. These features can be the depth of the subgraph of nodes, the number of parent/child nodes, lexical embeddings for nodes/tokens, POS tags, name-entities for nodes/tokens or dependency relations among the nodes/tokens. The overall algorithm is shown in Algorithm 1.

For training these classifiers, an oracle similar to Chen and Manning \([3]\)’s is employed to compute transition sequences given AMR labeled sentences. Given any configuration of the parse, the oracle uses a fixed set of rules to determine the action to take (check \([4]\) for more detail).

Contextual Word Embeddings  Pre-trained BERT \([5]\) and RoBERTa \([14]\)\(^2\) are used to extract word embeddings as part of the input feature for parsing. Both BERT and RoBERTa use a multi-layer bidirectional Transformer encoder architecture almost identical to Vaswani et al. \([28]\)’s implementation, where each encoder layer includes two sub-layers; one multi-head self-attention layer followed by a fully connected feed-forward layer. BERT is trained on a mixture of masked language modeling and next sentence prediction objectives; while RoBERTa is trained with dynamic masking during training, without the next sentence prediction objective, but with some other minor differences. Although they are designed chiefly for the fine-tuning approach of pre-training general language representations, many have demonstrated their utility in the feature-extraction approach due to their ability to extract contextual information from the whole sentence, which is another motivation for using them with \textsc{AMREager}.

\(^2\)The models implementations are obtained from: https://huggingface.co/transformers/pretrained_models.html
3 AMR parsing with contextual word embeddings

We build on AMREAGER to be able to incorporate contextual word features, and we dub the new system as AMRBERGER.

These contextual word-features are prepared for training/testing in the preprocessing stage alone with the named-entities, dependency trees, POS tags for the whole dataset. To generate the particular training/validation dataset for the neural-classifiers, the complete 768-dimensional (or 1024-dimensional in case of BERTLARGE and RoBERTaLARGE) word embeddings are recorded along with the other input features, and the target label. This is in contrast to only having to record the word embedding index in a lookup table in the case of non-contextual embeddings.

Alignment method. Due to both BERT and RoBERTa require splitting word-level tokens in sentences to combinations of sub-word units in their vocabulary—WordPiece [29] and Byte-Pair Encoding (BPE) [23] in BERT and RoBERTa respectively. We have to find an alignment between these sub-word units and the word units used in our AMR aligner, dependency parser, etc. The simple alignment algorithm we employ is: feed the BERT/RoBERTa tokenizer the tokens generated from the syntactic annotation pipeline one-by-one, and calculate, according to this output, the span of hidden vectors in the transformer output corresponding to each token. After than, the spans of hidden vectors corresponding to word-level units can be canonicalized by methods such as averaging or picking the head or last vectors, etc. Different methods are experimented in Section 4.

In summary, AMRBERGER differs from AMREAGER mainly in that:

1. In preprocessing, an extra step is required to prepare the contextual token embeddings, for the tokens output from the CoreNLP tokenization pipeline [16];
2. The training/validation dataset generation procedure for the neural classifiers are modified to record the contextual embeddings directly, instead of storing the word indices in the embedding lookup table, which does not contain any information about the context. As a drawback of this approach, each entry in the training/validation dataset is \((d-1) \times n\) times larger than the original ones, with \(d\) denotes the number of dimensions for each embedding, \(n\) the number of tokens/nodes in each entry;
3. For the classifiers, the original table lookup module at the input layer is replaced with an identity module to pass on the pre-computed embeddings.

We note that for AMREAGER the lexical features for the neural classifiers’ input not only consist of embeddings of 1. certain tokens on the buffer; 2. the tokens that certain nodes on the stack are aligned to, but also embedding of the concepts of the selected stack nodes. These concept embeddings (e.g., “United States” might have a concept label “country”) are, as we will see in Section 4, critical to the performance of the parsers. On one hand, we hypothesize this phenomenon is due to imperfect alignment between word tokens and AMR nodes, or abstraction artifact when generating the concept nodes for a sentence. On the other, this illustrates an inherited limitation of using solely contextual embeddings; there is no straightforward way of obtaining the embeddings for these concept words. So one way to mitigate is to use the non-contextual embeddings for concepts, in conjunction with contextual embeddings for sentence words, which is experimented in Section 4.

---

Algorithm 1 AMREAGER Parsing

| Input: sentence \(x\) |
| Return: AMR graph \(G\) |

\[
\text{state} = \{ \text{stack} \leftarrow \{\text{root}\}, \text{buffer} \leftarrow \{x\}, \text{edges} \leftarrow [] \}
\]

\[\text{repeat} \]

\[\text{action} = \theta(\text{state}) \]
\[\text{if} \quad \text{action} == \text{LArC} | \text{RArc} \quad \text{then} \]
\[\quad \text{label} = \lambda(\text{state}) \]
\[\quad \text{state} = \text{APPLY}(\text{action}, \text{label}, \text{state}) \]
\[\text{end if} \]

\[\text{if} \quad \text{action} == \text{Shift} \quad \text{then} \]
\[\quad \text{subgraph} = \chi(\text{state}) \]
\[\quad \text{state} = \text{APPLY}(\text{action}, \text{subgraph}, \text{state}) \]
\[\text{end if} \]

\[\text{if} \quad \text{action} == \text{Reduce} \quad \text{then} \]
\[\quad \text{reentrancy} = \rho(\text{state}) \]
\[\quad \text{state} = \text{APPLY}(\text{action}, \text{reentrancy}, \text{state}) \]
\[\text{end if} \]

\[\text{until} \quad \text{both stack and buffer are empty} \]

\[\text{return} \quad \text{build_graph(state)} \]
4 Experiments

AMRBERGER with different transformer sizes, hidden-layer choices, and embeddings selection methods are experimented. The best performing model is compared with the baseline AMREAGER using the fine-grained Smatch evaluation. We use the labelling classifier’s performance as a heuristic in choosing the embedding configuration due to the following 4 reasons: 1. It requires combinatorially more compute to find a global optimal hyperparameter setting suitable for all three neural nets, e.g., the action classifier alone has more than $1e6$ training samples thus takes a long time to converge; 2. Empirically, we found that under different settings, the performance of the three neural nets are highly positively correlated; 3. The labelling task is the hardest task of the three–having to classify among more than 100 potential labels, and it is where AMREAGER performs the worst–thus an improvement under this task should bring a sizable gain in the overall SMATCH score.

For reference in the following experiments, AMREAGER achieves an accuracy of 79.8% on the labelling-classification development set.

4.1 Transformer configuration

Let $L$ denotes the number of Transformer encoder layers, $H$ the hidden layer dimension, $A$ the number of self-attention heads, $V$ the vocabulary cardinality. The following 5 pre-trained BERT and RoBERTa models are experimented in our system: BERTBASE_UNCASED ($L = 12, H = 768, A = 12, V = 30,000$), BERTBASE_CASED ($L = 12, H = 768, A = 12, V = 30,000$), BERTLARGE_CASED ($L = 24, H = 1024, A = 16, V = 30,000$), RoBERTABASE ($L = 12, H = 768, A = 12, V = 50,000$), and RoBERTALARGE ($L = 24, H = 1024, A = 16, V = 50,000$). At the time of design, the smaller BERT models were mainly designed to compare and contrast with OpenAI’s GPT model [22] with a similar number of parameters, while the bigger ones are developed to assess the architecture’s full potential. The difference between “CASED” and “UNCASED” model is “UNCASED” would treat tokens with capital letters indifferently with them with lower case letters (e.g. treating “Apple”, commonly representing the company’s name, as “apple”, refering to the fruit). On the other hand, since their $V$ equals, “UNCASED” would include more whole-word tokens and less out-of-vocabulary tokens. With each pretrained model, the hyperparameters of each neural classifier are optimized through random search [2]. Across all transformer configurations, the penultimate layer of hidden activations is extracted for word embeddings.

Table 1: The labelling classifier accuracy under different configurations.

| Configuration            | Accuracy |
|-------------------------|----------|
| BERTBASE_UNCASED        | 72.3%    |
| BERTLARGE_UNCASED       | 74.6%    |
| BERTLARGE_CASED         | 74.7%    |
| RoBERTABASE             | 76.1%    |
| RoBERTALARGE            | 77.5%    |

The performance of different contextual embeddings configurations is listed in Table 1. This agrees with their performance from previous evaluations [14]; larger models perform better than their smaller counterparts, and RoBERTa outperforms BERT, confirming the previous finding that RoBERTa is a robustly optimized configuration of BERT. In terms of the hyperparameters for the classifier, the best performance is obtained with a six-layer net each with 768-dim hidden layers, and trained using gradient descent with momentum.

4.2 Different activation layers

As shown in [5], a different layer of activates would lead to noticeable performance difference in the Named Entity Recognition task. Two different activation-extraction methods are examined here with the RoBERTALARGE model, the better performing model from the experiment above. The extraction methods are a) use the penultimate layer directly and b) extract the last four hidden layers and compute the arithmetic mean among the layers. Hyperparameter optimization is performed again in each of the settings.
Table 2: The labelling classifier accuracy using different activation-extraction method from RoBERTa\textsubscript{LARGE}.

| Configuration                | Accuracy |
|------------------------------|----------|
| Second-to-Last Hidden        | 77.3%    |
| Average of Last Four Hidden  | 77.0%    |

The results in Table 2 show the score of the average method is a slightly lower than its alternative. This is different from the finding from \cite{5} with BERT, which could be explained by the different nature of the tasks and could be informative to future design decisions.

4.3 Different averaging method

As introduced in Section 3 after obtaining the spans of sub-word tokens corresponding to a word-level token, different methods exist to compute the final feature vectors representing the word-level tokens. In our experiments, the headword token’s embeddings and the average of the spanned sub-word embeddings from RoBERTa\textsubscript{LARGE}’s penultimate layer are evaluated.

Table 3: The labelling classifier accuracy using different method of resolving multiple sub-word level units corresponding to one word level unit from RoBERTa\textsubscript{LARGE}.

| Configuration            | Accuracy |
|--------------------------|----------|
| Head Token of the span   | 76.9%    |
| Average of Tokens in the span | 77.3% |

Results in Table 3 shows that averaging the spanning tokens’ performs slightly better than only taking the head token, which is also what Devlin et al. \cite{5} employed when extracting BERT embeddings for the named-entities-recognition task. The averaging method seems to contain more discriminative information for the classifier than just the head token method, though in theory, each token should have attended to all the other context tokens.

4.4 Additional Concept Features

The results above demonstrate, even in the best condition, the labelling classifier using purely contextual word vectors obtains 77% accuracy on the dev set, while the one using non-contextual word vectors achieves 3% higher than that, as shown in Table 4. We attribute this poor performance to the absence of concept embeddings in AMRBERGER. This situation is effectively rescued by leveraging additional GloVe embeddings for node concepts on top of the contextual embeddings for words, as shown in the third column of Table 4. We observe that with the help of these concept embeddings, AMRBERGER significantly improves in labelling classification, outperforming AMREAGER.

The Other Classification Tasks and Fine-grained Smatch comparison Table 4 tabulates the performance of the other classifiers and the fine-grained Smatch score of the best performing configuration of AMREAGER, AMRBERGER and AMRBERGER plus concept features.

We observe that albeit AMRBERGER’s poor accuracy in labelling and reentrancy classification in comparison to AMREAGER, it outperforms AMREAGER by 4% in transition classification, the most frequently used component of the system. As a result, they achieve the same overall Smatch statistics. The mixed fine-grained Smatch performance agree with the strengths and weaknesses of each system’s classifiers. With additional concept features, AMRBERGER+Concept outperforms AMREAGER in all metrics except in reentrancy, though only by a small margin. In the end, its Smatch score is 2 points higher than AMREAGER.

4.5 Ablation Study

As analyzed in \cite{13, 26, 27}, contextual word embeddings, especially BERT, can capture strong syntactical information about words. We therefore experiment with a variation of AMREAGER and
Table 4: Performance of the models on the pre-split development set of the LDC2020T02 dataset. \(\alpha\) (Transition), \(\alpha\) (labelling), \(\alpha\) (reentrancy), stands for the classifiers’ accuracy on the development set. The others are the SMATCH and its decomposition scores.

| Metric            | AMR_EAGER | AMR_BERGER | AMR_BERGER+Concept |
|-------------------|-----------|------------|--------------------|
| \(\alpha\) (Transition) | 84%       | 88%        | 88%                |
| \(\alpha\) (labelling) | 80%       | 77%        | 82%                |
| \(\alpha\) (Reentrancy) | 98%       | 96%        | 97%                |
| Smatch            | 63        | 63         | 65                 |
| Unlabeled         | 67        | 68         | 68                 |
| No WSD            | 63        | 64         | 64                 |
| Reentrancy        | 45        | 44         | 46                 |
| Concepts          | 81        | 81         | 81                 |
| Named Ent.        | 78        | 77         | 80                 |
| Wikification      | 61        | 61         | 61                 |
| Negations         | 46        | 46         | 46                 |
| SRL               | 61        | 61         | 63                 |

Table 5: Different system’s labelling classification accuracy on the dev set, all without dependency features.

| Configuration    | Accuracy |
|------------------|----------|
| AMR_EAGER-dep    | 77.4%    |
| AMR_BERGER-dep   | 75.9%    |
| AMR_BERGER-dep+concept | 81.1% |

AMR_BERGER that does not use explicit syntactic dependency features, to probe how the use of contextual embeddings would help in this situation.

We observe from Table 5 that even without dependency features, AMR_EAGER has higher accuracy than when AMR_BERGER has the explicit dependency signals. On the other hand, the accuracy of the classifiers using contextual word embeddings suffer less from the removal of the dependency features.

5 Conclusions

In this study, we found contextual embeddings alone are not able to provide much performance improvements in our transition based AMR parser, as demonstrated by the classifiers’ accuracy and the final SMATCH scores. This is not entirely consistent with the kind of performance gain shown in much of the other works in the space, indicating potential limitations of the current approach, and encourages us to understand deeper about the nature of these pretrained models in future works. A hybrid approach with contextual embedding for words and non-contextual embeddings for concepts achieves the best performance, highlighting the importance of employing the concept features in AMR parsing. Nonetheless, as demonstrated by the ablation study, contextual embeddings do show evidence of capturing richer information about the sentence, namely syntactical knowledge of words.

6 Acknowledgment

I thank my advisor Shay B. Cohen for his support and insightful discussion throughout this project.

References

[1] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. Abstract meaning representation for sembanking. In Proceedings
of the 7th linguistic annotation workshop and interoperability with discourse, pages 178–186, 2013.

[2] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. Journal of machine learning research, 13(2), 2012.

[3] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 740–750, 2014.

[4] M. Damonte, S. B. Cohen, and G. Satta. An incremental parser for abstract meaning representation. In Proceedings of EACL, 2017.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[6] S. Dohare, H. Karnick, and V. Gupta. Text summarization using abstract meaning representation. arXiv preprint arXiv:1706.01678, 2017.

[7] J. Flanigan, S. Thomson, J. G. Carbonell, C. Dyer, and N. A. Smith. A discriminative graph-based parser for the abstract meaning representation. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1426–1436, 2014.

[8] Z. S. Harris. Distributional structure. Word, 10(2-3):146–162, 1954.

[9] B. Jones, J. Andreas, D. Bauer, K. M. Hermann, and K. Knight. Semantics-based machine translation with hyperedge replacement grammars. In Proceedings of COLING 2012, pages 1359–1376, 2012.

[10] P. R. Kingsbury and M. Palmer. From treebank to propbank. In LREC, pages 1989–1993. Citeseer, 2002.

[11] I. Konstas, S. Iyer, M. Yatskar, Y. Choi, and L. Zettlemoyer. Neural amr: Sequence-to-sequence models for parsing and generation. arXiv preprint arXiv:1704.08381, 2017.

[12] F. Liu, F. Jeffrey, T. Sam, and S. Norman. Smith noah a. 2015. toward abstractive summarization using semantic representations. In Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, Denver, Colorado, page 10771086.

[13] N. F. Liu, M. Gardner, Y. Belinkov, M. E. Peters, and N. A. Smith. Linguistic knowledge and transferability of contextual representations. 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 1073–1094. Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1112. URL https://www.aclweb.org/anthology/N19-1112.

[14] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019.

[15] C. Lyu and I. Titov. Amr parsing as graph prediction with latent alignment. arXiv preprint arXiv:1805.05286, 2018.

[16] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, pages 55–60, 2014.

[17] C. M. Matthiessen, M. Christian, J. A. Bateman, and M. Matthiessen. Text generation and systemic-functional linguistics: experiences from English and Japanese. Burns & Oates, 1991.

[18] A. Mitra and C. Baral. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In AAAI, pages 2779–2785, 2016.
[19] T. Naseem, A. Shah, H. Wan, R. Florian, S. Roukos, and M. Ballesteros. Rewarding smatch: Transition-based amr parsing with reinforcement learning. *arXiv preprint arXiv:1905.13370*, 2019.

[20] J. Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the workshop on incremental parsing: Bringing engineering and cognition together*, pages 50–57, 2004.

[21] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[22] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding with unsupervised learning. *Technical report, OpenAI*, 2018.

[23] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[24] P. Shi and J. Lin. Simple bert models for relation extraction and semantic role labeling. *arXiv preprint arXiv:1904.05255*, 2019.

[25] L. Song, D. Gildea, Y. Zhang, Z. Wang, and J. Su. Semantic neural machine translation using amr. *Transactions of the Association for Computational Linguistics*, 7:19–31, 2019.

[26] I. Tenney, D. Das, and E. Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://www.aclweb.org/anthology/P19-1452.

[27] I. Tenney, P. Xia, B. Chen, A. Wang, A. Poliak, R. T. McCoy, N. Kim, B. V. Durme, S. Bowman, D. Das, and E. Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SJzSgnRcKX.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[29] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[30] Q. Zhou, Y. Zhang, D. Ji, and H. Tang. Amr parsing with latent structural information. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4306–4319, 2020.