TUPA at MRP 2019: A Multi-Task Baseline System

Daniel Hershcovich[*] and **Ofir Arviv**[**]

[*]University of Copenhagen, Department of Computer Science
[**]Hebrew University of Jerusalem, School of Computer Science and Engineering
hershcovich@di.ku.dk, ofir.arviv@mail.huji.ac.il

Abstract

This paper describes the TUPA system submission to the shared task on Cross-Framework Meaning Representation Parsing (MRP) at the 2019 Conference for Computational Language Learning (CoNLL). TUPA provides a baseline point of comparison and is not considered in the official ranking of participating systems. While originally developed for UCCA only, TUPA has been generalized to support all MRP frameworks included in the task, and trained using multi-task learning to parse them all with a shared model. It is a transition-based parser with a BiLSTM encoder, augmented with BERT contextualized embeddings.

1 Introduction

TUPA (Transition-based UCCA/Universal Parser; Hershcovich et al., 2017) is a general transition-based parser for directed acyclic graphs (DAGs), originally designed for parsing text to graphs in the UCCA framework (Universal Conceptual Cognitive Annotation; Abend and Rappoport, 2013). It was used as the baseline system in SemEval 2019 Task 1: Cross-lingual Semantic Parsing with UCCA (Hershcovich et al., 2019b), where it was outranked by participating team submissions in all tracks (open and closed in English, German and French), but was also among the top 5 best-scoring systems in all tracks, and reached second place in the English closed tracks.

Being a general DAG parser, TUPA has been shown (Hershcovich et al., 2018a,b) to support other graph-based meaning representations and similar frameworks, including UD (Universal Dependencies; Nivre et al., 2019), which was the focus of CoNLL 2017 and 2018 Shared Tasks (Zeman et al., 2017, 2018); AMR (Abstract Meaning Representation; Banarescu et al., 2013), targeted in SemEval 2016 and 2017 Shared Tasks (May, 2016; May and Priyadarshi, 2017); and DM (DELPH-IN MRS Bi-Lexical Dependencies; Ivanova et al., 2012), one of the target representations, among PAS and PSD (Prague Semantic Dependencies; Hajic et al., 2012; Miyao et al., 2014), in the SemEval 2014 and 2015 Shared Tasks on SDP (Semantic Dependency Parsing; Oepen et al., 2014, 2015, 2016). DM is converted from DeepBank (Flickinger et al., 2012), a corpus of hand-corrected parses from LinGO ERG (Copestake and Flickinger, 2000), an HPSG (Pollard and Sag, 1994) using Minimal Recursion Semantics (Copestake et al., 2005). EDS (Elementary Dependency Structures; Oepen and Lønning, 2006) is another framework derived from ERG, encoding English Resource Semantics in a variable-free semantic dependency graph.

The CoNLL 2019 Shared Task (Oepen et al., 2019) combines five frameworks for graph-based meaning representation: DM, PSD, EDS, UCCA and AMR. For the task, TUPA was extended to support the MRP format and frameworks, and is used as a baseline system, both as a single-task system trained separately on each framework, and as a multi-task system trained on all of them. The code is publicly available.[1]

2 Intermediate Graph Representation

Meaning representation graphs in the shared tasks are distributed in, and expected to be parsed to, a uniform graph interchange format, serialized as JSON Lines.[2]

The formalism encapsulates annotation for graphs containing nodes (corresponding either to text tokens, concepts, or logical predications), with the following components: top nodes, node

[1] https://github.com/danielhers/tupa/tree/mrp

[2] http://mrp.nlpl.eu/index.php?page=4

Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 CoNLL, pages 28–39
Hong Kong, November 3, 2019.
https://doi.org/10.18653/v1/K19-2002

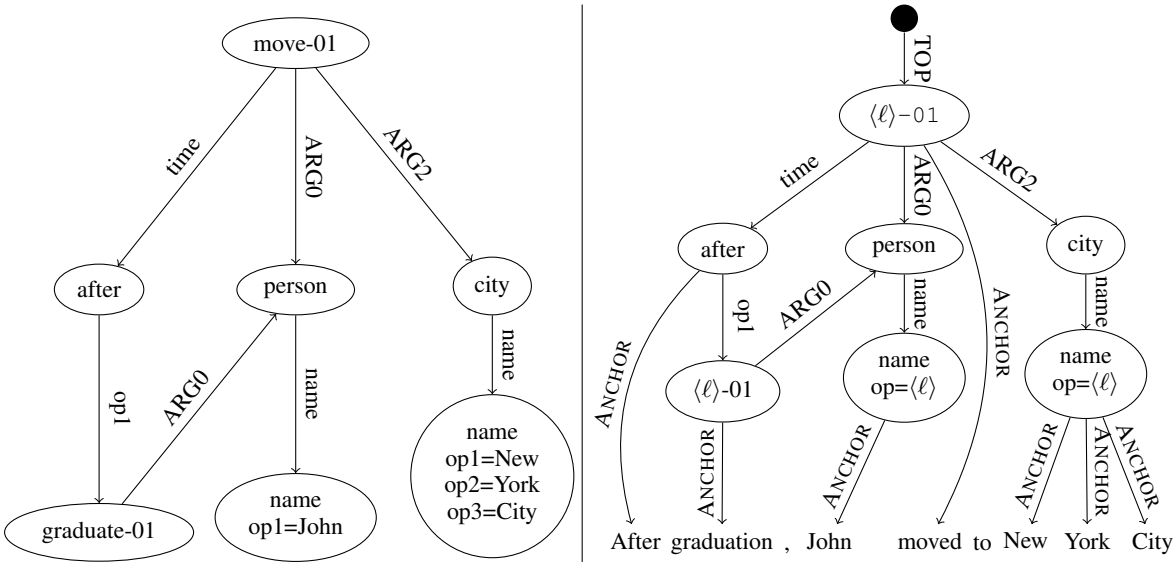

Figure 1: Left: AMR graph, in the MRP formalism, for the sentence "After graduation, John moved to New York City." Edge labels are shown on the edges. Node labels are shown inside the nodes, along with any node properties (in the form `property=value`). The text tokens are not part of the graph, and are matched to nodes by automatic alignment (anchoring). Right: converted AMR graph in the intermediate graph representation. Same as in the intermediate graph representation for all frameworks, it contains a virtual root node attached to the graph's top node with a TOP edge, and virtual terminal nodes corresponding to text tokens, attached according to the anchoring (or, for AMR, the provided automatic alignments) with ANCHOR edges. Same as for all frameworks with node labels and properties (i.e., all but UCCA), labels and properties are replaced with placeholders corresponding to anchored tokens, where possible. The placeholder $\langle \ell \rangle$ corresponds to the concatenated lemmas of anchored tokens. Specifically for AMR, name operator properties (e.g., op* for New York City) are collapsed to single properties.

labels, node properties, node anchoring, directed edges, edge labels, and edge attributes.

While all frameworks represent top nodes, and include directed, labeled edges, UCCA does not contain node labels and properties, AMR lacks node anchoring, and only UCCA has edge attributes (distinguishing primary/remote edges).

2.1 Roots and Anchors

TUPA supports parsing to rooted graphs with labeled edges, and with the text tokens as terminals (leaves), which is the standard format for UCCA graphs. However, MRP graphs are not given in this format, since there may be multiple roots and the text tokens are only matched to the nodes by anchoring (and not by explicit edges).

For the CoNLL 2019 Shared Task, TUPA was extended to support node labels, node properties, and edge attributes (see §3.1). Top nodes and anchoring are combined into the graph by adding a virtual root node and virtual terminal nodes, respectively, during preprocessing.

A virtual terminal node is created per token according to the tokenization predicted by UDPipe (Straka and Straková, 2017) and provided as companion data by the task organizers. All top nodes are attached as children of the virtual root with a TOP-labeled edge.

Nodes with anchoring are attached to the virtual terminals associated with the tokens whose character spans intersect with their anchoring, with ANCHOR-labeled edges. Note that anchoring is automatically determined for training in the case of AMR, using the alignments from the companion data, computed by the ISI aligner (Pourdamghani et al., 2014). There is no special treatment of non-trivial anchoring for EDS: in case a node is anchored to multiple tokens (as is the case for multi-word expressions), they are all attached with ANCHOR-labeled edges, resulting in possibly multiple parents for some virtual terminal nodes.

During inference, after TUPA returns an output graph, the virtual root and terminals are removed as postprocessing to return the final graph. Top nodes and anchoring are then inserted accordingly.

2.2 Placeholder Insertion

The number of distinct node labels and properties is very large for most frameworks, resulting in severe sparsity, as they are taken from an open vo-

Before Transition				Transition	After Transition				
Stack	Buffer	N.	Edges		Stack	Buffer	Nodes	Edges	Extra Effect
S	$x \mid B$	V	E	SHIFT	$S \mid x$	B	V	E	
$S \mid x$	B	V	E	REDUCE	S	B	V	E	
$S \mid x$	B	V	E	NODE$_X$	$S \mid x$	$y \mid B$	$V \cup \{y\}$	$E \mid (y, x)$	$\ell_E(y, x) \leftarrow X$
$S \mid x$	B	V	E	CHILD$_X$	$S \mid x$	$y \mid B$	$V \cup \{y\}$	$E \mid (x, y)$	$\ell_E(x, y) \leftarrow X$
$S \mid x$	B	V	E	LABEL$_X$	$S \mid x$	B	V	E	$\ell_V(x) \leftarrow X$
$S \mid x$	B	V	E	PROPERTY$_X$	$S \mid x$	B	V	E	$p(x) \leftarrow X$
$S \mid y, x$	B	V	E	LEFT-EDGE$_X$	$S \mid y, x$	B	V	$E \mid (x, y)$	$\ell_E(x, y) \leftarrow X$
$S \mid x, y$	B	V	E	RIGHT-EDGE$_X$	$S \mid x, y$	B	V	$E \mid (x, y)$	$\ell_E(x, y) \leftarrow X$
S	B	V	$E \mid (x, y)$	ATTRIBUTE$_X$	S	B	V	$E \mid (x, y)$	$a(x) \leftarrow X$
$S \mid x, y$	B	V	E	SWAP	$S \mid y$	$x \mid B$	V	E	
[root]	$\emptyset$	V	E	FINISH	$\emptyset$	$\emptyset$	V	E	terminal state

Figure 2: The TUPA-MRP transition set. We write the stack with its top to the right and the buffer with its head to the left; the set of edges is also ordered with the latest edge on the right. NODE, LABEL, PROPERTY and ATTRIBUTE require that $x \neq$ root; CHILD, LABEL, PROPERTY, LEFT-EDGE and RIGHT-EDGE require that $x \notin w_{1:n}$; ATTRIBUTE requires that $y \notin w_{1:n}$; LEFT-EDGE and RIGHT-EDGE require that $y \neq$ root and that there is no directed path from y to x; and SWAP requires that $\mathrm{i}(x) < \mathrm{i}(y)$, where $\mathrm{i}(x)$ is the swap index (see §3.5).

cabulary of e.g. word senses and named entities. However, many are simply copies of text tokens and their lemmas.

To reduce the number of unique node labels and properties, we use the (possibly automatic) anchoring and UDPipe preprocessing to introduce placeholders in the values. For example, a node labeled `move-01` anchored to the token `moved` will be instead labeled $\langle\ell\rangle$`-01`, where $\langle\ell\rangle$ is a placeholder for the token's lemma. In this way we reduce the number of node labels in the AMR training set, for example, from tens of thousands to 7,300, of which 2,000 occur only once and are treated as unknown. We use similar placeholders for the token's surface form. Placeholders are resolved back to the full value after an output graph is produced by the parser, according to the anchoring in the graph. While nodes labels and properties sometimes have a non-trivial relationship to the text tokens, in most cases they contain the lemma or surface form, making this a simple and effective solution.

While more sophisticated alignment rules have been developed (Flanigan et al., 2014; Pourdamghani et al., 2014), such as using entity linking (Daiber et al., 2013), as employed by Bjerva et al. (2016); van Noord and Bos (2017), in this baseline system we are employing a simple strategy without relying on external, potentially non-whitelisted resources.

Named entities in AMR are expressed by `name`-labeled nodes, with a property for each token in the name, with keys `op1`, `op2`, etc. We instead collapse these properties to a single `op` property whose label is the concatenation of the name tokens, with special separator symbols. This value is in turn replaced by a placeholder, if the node is anchored and the anchored tokens match the property. Figure 1 demonstrates an AMR graph before and after the conversion to the intermediate graph representation.

3 Transition-based Meaning Representation Parser

TUPA is a transition-based parser (Nivre, 2003), constructing graphs incrementally from input tokens by applying *transitions* (*actions*) to the *parser state* (*configuration*). The parser state is composed of a buffer B of tokens and nodes to be processed, a stack S of nodes currently being processed, and an incrementally constructed graph G. Some states are marked as *terminal*, meaning that G is the final output. The input to the parser is a sequence of tokens: $w_1, \ldots, w_n$. Parsing starts with a (virtual) root node on the stack, and the input tokens in the buffer, as (virtual) terminal nodes.

Given a gold-standard graph and a parser state, an *oracle* returns the set of gold transitions to apply at the next step, i.e., all transitions that preserve the reachability of the gold target graph.[3] A classifier is trained using the oracle to select

[3]This type of oracle is similar to a *dynamic oracle* (Goldberg and Nivre, 2012; Goldberg, 2013), but in TUPA it only supports the case where the current parser state is valid, i.e., only gold transitions have been applied since the initial state. Training with exploration is thus not supported (yet).

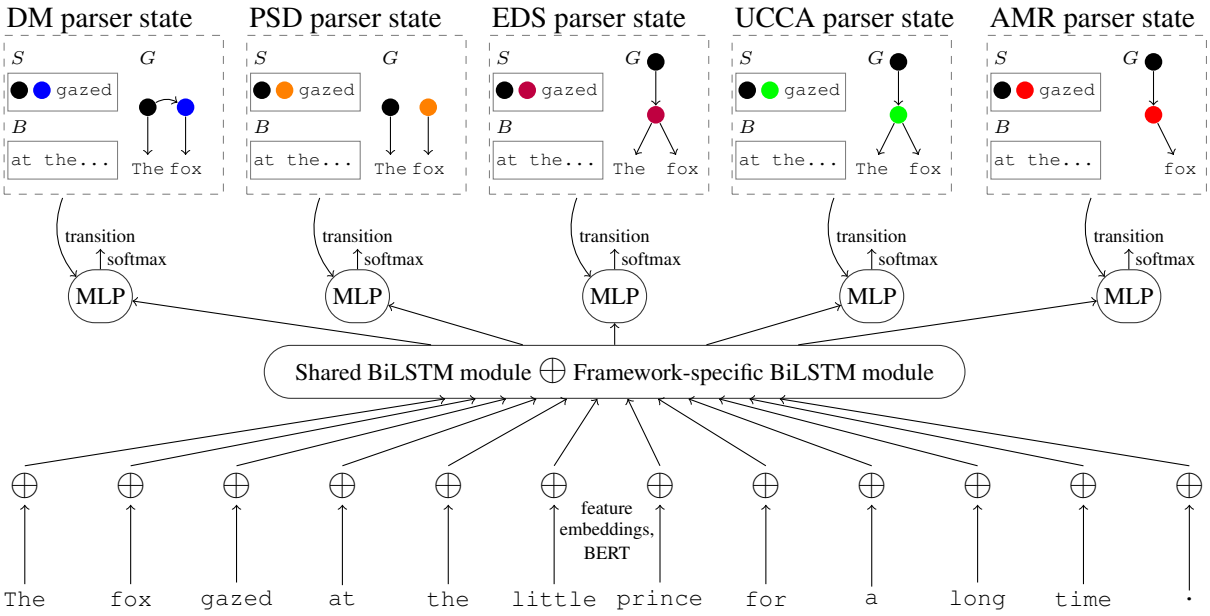

Figure 3: Illustration of the TUPA model, adapted from Hershcovich et al. (2018a), at an intermediate point in the process of parsing the sentence "The fox gazed at the little prince for a long time." Top: parser state (stack, buffer and intermediate graph) for each framework. Bottom: encoder architecture. Input feature embeddings are concatenate with BERT embeddings for each token. Vector representations for the input tokens are then computed by two layers of shared and framework-specific bidirectional LSTMs. At each point in the parsing process, the encoded vectors for specific tokens (from specific location in the stack/buffer) are concatenated with embedding and numeric features from the parser state (for existing edge labels, number of children, etc.), and fed into the MLP for selecting the next transition. Note that parsing the different frameworks is not performed jointly; the illustration only expresses the parameter sharing scheme.

the next transition based on features encoding the parser's current state, where the training objective is to maximize the sum of log-likelihoods of all gold transitions at each step. If there are multiple gold transitions, the highest-scoring one is taken in training. Inference is performed greedily: the highest-scoring transition is always taken.

Formally, the incrementally constructed graph G consists of $(V, E, \ell_V, \ell_E, p, a)$, where V is the set of *nodes*, E is the sequence of directed *edges*, $\ell_V : V \to L_V$ is the *node label* function, L_V being the set of possible node labels, $\ell_E : E \to L_E$ is the *edge label* function, L_E being the set of possible edge labels, $p : V \to \mathcal{P}(P)$ is the *node property* function, P being the set of possible node property-value pairs, and $a : E \to \mathcal{P}(A)$ is the *edge attribute* function, A being the set of possible edge attribute-value pairs (a node may have any number of properties; an edge may have any number of attributes).

3.1 Transition Set

The set of possible transitions in TUPA is based on a combination of transition sets from other parsers, designed to support reentrancies (Sagae and Tsujii, 2008; Tokgöz and Eryiğit, 2015), discontinuities (Nivre, 2009; Maier, 2015; Maier and Lichte, 2016) and non-terminal nodes (Zhu et al., 2013). Beyond the original TUPA transitions (Hershcovich et al., 2017, 2018a), for the CoNLL 2019 Shared Task, transitions are added to support node labels, node properties, and edge attributes. Additionally, top nodes and node anchoring are encoded by special edges from a virtual root node and to virtual terminal nodes (corresponding to text tokens), respectively (see §2).

The TUPA-MRP transition set is shown in Figure 2. It includes the following original TUPA transitions: the standard SHIFT and REDUCE operations (to move a node from the buffer to the stack and to discard a stack node, respectively), NODE$_X$ for creating a new non-terminal node and an X-labeled edge (so that the new node is a parent of the stack top), LEFT-EDGE$_X$ and RIGHT-EDGE$_X$ to create a new X-labeled edge, SWAP to handle discontinuous nodes (moving the second topmost stack node back to the buffer), and FINISH to mark the state as terminal.

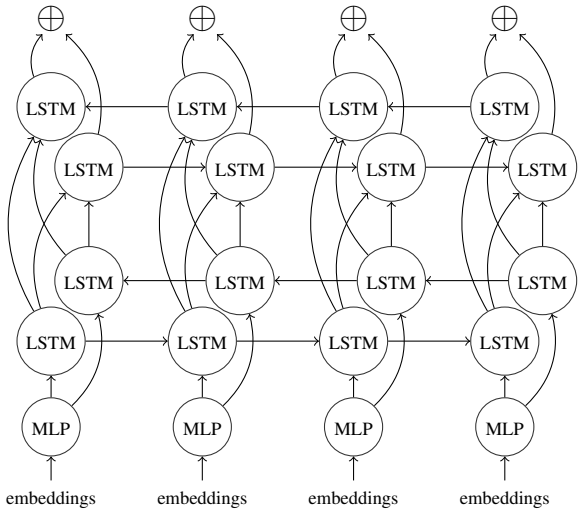

Figure 4: BiLSTM module, illustrated for an input sequence of four tokens.

Besides the original TUPA transitions, TUPA-MRP contains a CHILD transition to create unanchored children for existing nodes (like NODE, but the new node is a *child* of the stack top),[4] a LABEL transition to select a label for an existing node (either the stack top of the second topmost stack node), a PROPERTY transition to select a property-value pair for an existing node, and an ATTRIBUTE transition to select an attribute-value pair for an existing edge (the last created edge).

The original TUPA transitions LEFT-REMOTE$_X$ and RIGHT-REMOTE$_X$, creating new *remote* edges (a UCCA-specific distinction), are omitted. Remote edges are encoded instead as edges with the `remote` attribute, and are supported by the combination of EDGE and ATTRIBUTE transitions. In contrast to the original TUPA transitions, EDGE transitions are allowed to attach multiple parents to a node.

3.2 Transition Classifier

To predict the next transition at each step, TUPA uses a BiLSTM module followed by an MLP and a softmax layer for classification (Kiperwasser and Goldberg, 2016). The model is illustrated in Figure 3.

The BiLSTM module (illustrated in more detail in Figure 4) is applied before the transition sequence starts, running over the input tokenized sequence. It consists of a pre-BiLSTM MLP with feature embeddings (§3.3) and pre-trained contextualized embeddings (§3.4) concatenated as inputs, followed by (multiple layers of) a bidirectional recurrent neural network (Schuster and Paliwal, 1997; Graves, 2008) with a long short-term memory cell (Hochreiter and Schmidhuber, 1997).

While edge labels are combined into the identity of the transition (so that for example, LEFT-EDGE$_P$ and LEFT-EDGE$_S$ are separate transitions in the output), there is just one transition for each of LABEL, PROPERTY and ATTRIBUTE. After each time one of these transition is selected, an additional classifier is evoked with the set of possible values for the currently parsed framework. This hard separation is made due to the large number of node labels and properties in the MRP frameworks. Since there is only one possible edge attribute value (`remote` for UCCA), performing this transition always results in this value being selected.

3.3 Features

In both training and testing, we use vector embeddings representing the lemmas, coarse POS tags (UPOS) and fine-grained POS tags (XPOS). These feature values are provided by UDPipe as companion data by the task organizers. In addition, we use punctuation and gap type features (Maier and Lichte, 2016), and previously predicted node and edge labels, node properties, edge attributes and parser actions. These embeddings are initialized randomly (Glorot and Bengio, 2010).

To the feature embeddings, we concatenate numeric features representing the node height, number of parents and children, and the ratio between the number of terminals to total number of nodes in the graph G. Numeric features are taken as they are, whereas categorical features are mapped to real-valued embedding vectors. For each non-terminal node, we select a *head terminal* for feature extraction, by traversing down the graph, selecting the first outgoing edge each time according to alphabetical order of labels.

3.4 Pre-trained Contextualized Embeddings

Contextualized representation models such as BERT (Devlin et al., 2019) have recently achieved state-of-the-art results on a diverse array of downstream NLP tasks, gaining improved results compared to non-contextual representations. We use

[4] While UCCA contains unanchored (*implicit*) nodes corresponding to non-instantiated arguments or predicates, the original TUPA disregards them as they are not included in standard UCCA evaluation. The CoNLL 2019 Shared Task omits implicit UCCA nodes too, in fact, but the CHILD transition is included to support unanchored nodes in AMR, and is not used otherwise.

the weighted sum of last four hidden layers of a BERT pre-trained model as extra input features.[5]

BERT uses a wordpiece tokenizer (Wu et al., 2016), which segments all text into sub-word units, while TUPA uses the UDPipe tokenization. To maintain alignment between wordpieces and tokens, we use a summation of the outputs of BERT vectors corresponding to the wordpieces of each token as its representation.

3.5 Constraints

As each annotation scheme has different constraints on the allowed graph structures, we apply these constraints separately for each task. During training and parsing, the relevant constraint set rules out some of the transitions according to the parser state.

Some constraints are task-specific, others are generic. For example, in AMR, a node with an incoming NAME edge must have the NAME label. In UCCA, a node may have at most one outgoing edge with label $\in \{\text{PROCESS}, \text{STATE}\}$.

An example of a generic constraint is that stack nodes that have been swapped should not be swapped again, to avoid infinite loops in inference. To implement this constraint, we define a *swap index* for each node, assigned when the node is created. At initialization, only the root node and terminals exist. We assign the root a swap index of 0, and for each terminal, its position in the text (starting at 1). Whenever a node is created as a result of a NODE or CHILD transition, its swap index is the arithmetic mean of the swap indices of the stack top and buffer head. While this constraint may theoretically limit the ability to parse arbitrary graphs, in practice we find that all graphs in the shared task training set can still be reached without violating it.

4 Multi-Task Learning

Whereas in the single-task setting TUPA is trained separately on each framework as described above, in the multi-task setting, all frameworks share a BiLSTM for encoding the input. In addition, each framework has a framework-specific BiLSTM, private to it. Each framework has its own MLP on top of the concatenation of the shared and framework-specific BiLSTM (see Figure 3).

[5] We used the `bert-large-cased` model from https://github.com/huggingface/pytorch-transformers.

Hyperparameter	Value
Lemma dim.	200
UPOS dim.	20
XPOS dim.	20
Dep. rel. dim.	10
Punct. dim.	1
Action dim.	3
Node label dim.	20
Node prop. dim.	20
Edge label dim.	20
Edge attrib. dim.	1
MLP layers	2
MLP dim.	50
Shared BiLSTM layers	2
Shared BiLSTM dim.	500
Shared pre-BiLSTM MLP layers	1
Shared pre-BiLSTM MLP dim.	300
Private BiLSTM layers	2
Private BiLSTM dim.	500
Private pre-BiLSTM MLP layers	1
Private pre-BiLSTM MLP dim.	300

Table 1: Hyperparameter settings.

For node labels and properties and for edge attributes (when applicable), an additional "axis" (private BiLSTM and MLP) is added per framework (e.g., AMR node labels are predicted separately and with an identical architecture to AMR transitions, except the output dimension is different). This is true for the single-task setting too, so in fact the single-task setting is multi-task over {transitions, node labels, node properties, edge attributes}.

5 Training details

The model is implemented using DyNet v2.1 (Neubig et al., 2017).[6] Unless otherwise noted, we use the default values provided by the package. We use the same hyperparameters as used in previous experiments on UCCA parsing (Hershcovich et al., 2018a), without any hyperparameter tuning on the CoNLL 2019 data.

5.1 Hyperparameters

We use dropout (Srivastava et al., 2014) between MLP layers, and recurrent dropout (Gal and Ghahramani, 2016) between BiLSTM layers, both with $p = 0.4$. We also use word, lemma, coarse- and fine-grained POS tag dropout with $\alpha = 0.2$

[6] http://dynet.io

Official	TUPA (single-task)		TUPA (multi-task)		Best System	
	ALL	LPPS	ALL	LPPS	ALL	LPPS
DM	55.54	58.60	42.69	39.45	95.50 (Bai and Zhao, 2019)	94.96 (Che et al., 2019)
PSD	51.76	58.87	52.65	54.53	91.28 (Donatelli et al., 2019)	88.46 (Li et al., 2019)
EDS	81.00	81.36	73.95	74.81	91.85 (Zhang et al., 2019)	92.55 (Zhang et al., 2019)
UCCA	27.56	40.06	23.65	41.03	81.67 (Che et al., 2019)	82.61 (Che et al., 2019)
AMR	44.73	47.04	33.75	43.37	73.38 (Cao et al., 2019)	73.11 (Donatelli et al., 2019)
Overall	57.70	57.55	45.34	50.64	86.20 (Che et al., 2019)	84.88 (Donatelli et al., 2019)

Table 2: Official test MRP F-scores (in %) for TUPA (single-task and multi-task). For comparison, the highest score achieved for each framework and evaluation set is shown.

(Kiperwasser and Goldberg, 2016): in training, the embedding for a feature value w is replaced with a zero vector with a probability of $\frac{\alpha}{\#(w)+\alpha}$, where $\#(w)$ is the number of occurrences of w observed. In addition, we use *node dropout* (Hershcovich et al., 2018a): with a probability of 0.1 at each step, all features associated with a single node in the parser state are replaced with zero vectors. For optimization we use a minibatch size of 100, decaying all weights by 10^{-5} at each update, and train with stochastic gradient descent for 50 epochs with a learning rate of 0.1, followed by AMSGrad (Sashank J. Reddi, 2018) for 250 epochs with $\alpha = 0.001, \beta_1 = 0.9$ and $\beta_2 = 0.999$. Table 1 lists other hyperparameter settings.

5.2 Official Evaluation

For the official evaluation, we did not use a development set, and trained on the full training set for as many epochs as the evaluation period allowed for. The multi-task model completed just 3 epoch of training. The single task models completed 12 epochs for DM, 22 epochs for PSD, 14 epochs for EDS, 100 epochs for UCCA (the maximum number we allowed) and 13 epochs for AMR.

Due to an oversight resulting from code re-use, in the official evaluation we used non-whitelisted resources. Specifically, for AMR, we used a constraint forcing any node whose label corresponds to a PropBank (Palmer et al., 2005) frame to only have the core arguments defined for the frame. We obtained the possible arguments per frame from the PropBank frame files.[7] Additionally, for the intermediate graph representation, we used placeholders for tokens' negation, verb, noun and adjective form, as well as organizational and relational roles, from a pre-defined lexicon included in the AMR official resources.[8] This is similar to the delexicalization employed by Buys and Blunsom (2017a) for AMR parsing.

5.3 Post-evaluation Training

After the evaluation period, we continued training for a longer period of time, using a slightly modified system: we used only resources whitelisted by the task organizers in the post-evaluation training, removing the constraints and placeholders based on PropBank and AMR lexicons.

In this setting, training is done over a shuffled mix of the training set for all frameworks (no special sampling is done to balance the number of instances per framework), and a development set of 500 instances per framework (see §5.1). We select the epoch with the best average MRP F-score score on a development set, selected by sampling 500 random training instances from each framework (the development instances are excluded from the training set). The large multi-task model only completed 4 training epochs in the available time, the single-task models completed 24 epochs for DM, 31 epochs for PSD, 25 epochs for EDS, 69 epochs for UCCA and 23 epochs for AMR.

6 Results

Table 2 presents the averaged scores on the test sets in the official evaluation (§5.2), for TUPA and for the best-performing system in each framework and evaluation set. Since non-whitelisted resources were used, the TUPA scores cannot be taken as a baseline. Furthermore, due to insufficient training time, all models but the UCCA one are underfitting, while the UCCA model is overfitting due to excessive training without early stopping (no development set was used in this setting).

[7] https://github.com/propbank/propbank-frames

[8] https://amr.isi.edu/download.html

Post-Evaluation	MRP Test Scores				Native Evaluation Test Scores				Trans./Token Ratio
	TUPA (single-task)		TUPA (multi-task)		TUPA (single-task)		TUPA (multi-task)		
	ALL	LPPS	ALL	LPPS	ALL	LPPS	ALL	LPPS	
DM	75.57	80.46	62.16	66.07	77.16	79.27	72.65	71.80	8.4
PSD	70.86	70.62	65.95	68.05	69.53	72.03	61.27	65.81	6.7
EDS	84.85	85.36	79.39	80.25	72.38	72.68	79.84	80.29	12.8
UCCA	77.69	82.15	64.05	73.11	57.42	65.90	35.60	50.29	8.4
AMR	53.85	53.47	39.00	42.62	53.05	52.52	38.11	40.47	6.6
Overall	75.73	77.63	66.01	68.58					8.4

Table 3: Post-evaluation test scores (in %) for TUPA (single-task and multi-task), using the MRP F-score (left), and using Native Evaluation (middle): labeled SDP F-score for DM and PSD, EDM F-score for EDS, primary labeled F-score for UCCA, and Smatch for AMR. The rightmost column (Trans./Token Ratio) shows the mean ratio between length of oracle transition sequence and sentence length, over the training set.

6.1 Post-evaluation Results

Table 3 presents the averaged scores on the test sets for the post-evaluation trained models (§5.3). Strikingly, the multi-task TUPA consistently falls behind the single-task one, for each framework separately and in the overall score. This stems from several factors, namely that the sharing strategy could be improved, but mainly since the multi-task model is probably underfitting due to insufficient training. We conclude that better efficiency and faster training is crucial for practical applicability of this approach. Perhaps a smaller multi-task model would have performed better by training on more data in the available time frame.

6.2 Diagnostic Evaluation

The rightmost column of Table 3 displays the mean ratio between length of oracle transitions sequence and sentence length by framework, over the shared task training set. Scores are clearly better as the framework has longer oracle transition sequences, perhaps because many of the transitions are "easy" as they correspond to structural elements of the graphs or properties copied from the input tokens.

6.3 Comparability with Previous Results

Previous published results of applying TUPA to UCCA parsing (Hershcovich et al., 2017, 2018a, 2019b,a) used a different version of the parser, without contextualized word representations from BERT.

For comparability with previous results, we train and test an identical model to the one presented in this paper, on the SemEval 2019 Task 1 data (Hershcovich et al., 2019b), which is UCCA-only, but contains tracks in English, German and French. For this experiment, we use `bert-multilingual` instead of `bert-large-cased`, and train a shared model over all three languages. A 50-dimensional learned language embedding vector is concatenated to the input. Word, lemma and XPOS features are not used. No multi-task learning with other frameworks is employed. The results are shown in Table 4. While improvement is achieved uniformly over the previous TUPA scores, even with BERT, TUPA is outperformed by the shared task winners (Jiang et al., 2019). Note that Jiang et al. (2019) also used `bert-multilingual` in the open tracks.

We also train and test TUPA with BERT embeddings on v1.0 of the UCCA English Web Treebank (EWT) reviews dataset (Hershcovich et al., 2019a). While the EWT reviews are included in the MRP shared task UCCA data, the different format and preprocessing makes for slightly different scores, so we report the scores for comparability with previous work in Table 5. We again see pronounced improvements from incorporating pre-trained contextualized embeddings into the model.

7 Related Work

Transition-based meaning representation parsing dates back already to semantic dependency parsing work by Sagae and Tsujii (2008); Tokgöz and Eryiğit (2015), who support a DAG structure by allowing multiple parents to be created by EDGE transitions, and by Titov et al. (2009), who applied a SWAP transition (Nivre, 2008) for online reordering of nodes to support non-projectivity.

Transition-based parsing was applied to AMR

SemEval 2019	All	Prim.	Rem.
English-Wiki (open)			
TUPA (w/o BERT)	73.5	73.9	53.5
TUPA (w/ BERT)	77.8	78.3	57.4
Jiang et al. (2019)	**80.5**	**81.0**	**58.8**
English-20K (open)			
TUPA (w/o BERT)	68.4	69.4	25.9
TUPA (w/ BERT)	74.9	75.7	**44.0**
Jiang et al. (2019)	**76.7**	**77.7**	39.2
German-20K (open)			
TUPA (w/o BERT)	79.1	79.6	59.9
TUPA (w/ BERT)	81.3	81.6	**69.2**
Jiang et al. (2019)	**84.9**	**85.4**	64.1
French-20K (open)			
TUPA (w/o BERT)	48.7	49.6	2.4
TUPA (w/ BERT)	72.0	72.8	**45.8**
Jiang et al. (2019)	**75.2**	**76.0**	43.3

Table 4: Test UCCA F-score scores (in %) on all edges, primary edges and remote edges, on the SemEval 2019 Task 1 data. The previous published TUPA scores are shown (TUPA w/o BERT), as well as scores for TUPA with BERT contextualized embeddings, TUPA (w/ BERT), averaged over three separately trained models in each setting, differing only by random seed (standard deviation < 0.03); and the scores for the best-scoring system from that shared task.

by Wang et al. (2015b,a, 2016); Wang and Xue (2017); Guo and Lu (2018), who transformed syntactic dependencies into AMRs by a sequence of transitions. Subsequent work used transition-based parsing to create AMRs from text directly (Damonte et al., 2017; Ballesteros and Al-Onaizan, 2017; Naseem et al., 2019). Buys and Blunsom (2017b) developed a transition-based parser supporting both AMR and EDS.

8 Conclusion

We have presented TUPA, a baseline system in the CoNLL 2019 shared task on Cross-Framework Meaning Representation. TUPA is a general transition-based DAG parser, which is trained with multi-task learning on multiple frameworks. Its input representation is augmented with BERT contextualized embeddings.

Acknowledgments

We are grateful for the valuable feedback from the anonymous reviewers. We would like to thank the other task organizers, Stephan Oepen, Omri Abend, Jan Hajič, Tim O'Gorman and Nianwen Xue, for valuable discussions and tips on developing the baseline systems, as well as for providing the data, evaluation metrics and information on the various frameworks.

EWT	All	Prim.	Rem.
TUPA (w/o BERT)	71.0	72.1	47.0
TUPA (w/ BERT)	**75.2**	**76.1**	**54.8**

Table 5: Test UCCA F-score scores (in %) on all edges, primary edges and remote edges, on the UCCA EWT reviews data. TUPA (w/o BERT) is from (Hershcovich et al., 2019a). TUPA (w/ BERT) is averaged over three separately trained models in each setting, differing only by random seed (standard deviation < 0.03).

References

Omri Abend and Ari Rappoport. 2013. Universal Conceptual Cognitive Annotation (UCCA). In *Proc. of ACL*, pages 228–238.

Joakim Nivre et al. 2019. Universal dependencies 2.4. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Hongxiao Bai and Hai Zhao. 2019. SJTU at MRP 2019: A transition-based multi-task parser for cross-framework meaning representation parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 86 – 94, Hong Kong, China.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. 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.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proc. of the Linguistic Annotation Workshop*.

Johannes Bjerva, Johan Bos, and Hessel Haagsma. 2016. The meaning factory at SemEval-2016 task 8: Producing AMRs with boxer. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1179–1184, San Diego, California. Association for Computational Linguistics.

Jan Buys and Phil Blunsom. 2017a. Oxford at SemEval-2017 task 9: Neural AMR parsing with pointer-augmented attention. In *Proc. of SemEval*, pages 914–919.

Jan Buys and Phil Blunsom. 2017b. Robust incremental neural semantic graph parsing. In *Proc. of ACL*, pages 1215–1226.

Jie Cao, Yi Zhang, Adel Youssef, and Vivek Srikumar. 2019. Amazon at MRP 2019: Parsing meaning representations with lexical and phrasal anchoring. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 138 – 148, Hong Kong, China.

Wanxiang Che, Longxu Dou, Yang Xu, Yuxuan Wang, Yijia Liu, and Ting Liu. 2019. HIT-SCIR at MRP 2019: A unified pipeline for meaning representation parsing via efficient training and effective encoding. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 76 – 85, Hong Kong, China.

Ann Copestake and Dan Flickinger. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proc. of LREC*, pages 591–600.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3(2):281–332.

Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. 2013. Improving efficiency and accuracy in multilingual entity extraction. In *Proc. of I-Semantics*.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for Abstract Meaning Representation. In *Proc. of EACL*.

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.

Lucia Donatelli, Meaghan Fowlie, Jonas Groschwitz, Alexander Koller, Matthias Lindemann, Mario Mina, and Pia WeiSSenhorn. 2019. Saarland at MRP 2019: Compositional parsing across all graphbanks. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 66 – 75, Hong Kong, China.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the Abstract Meaning Representation. In *Proc. of ACL*, pages 1426–1436.

Daniel Flickinger, Yi Zhang, and Valia Kordoni. 2012. DeepBank: A dynamically annotated treebank of the Wall Street Journal. In *Proc. of Workshop on Treebanks and Linguistic Theories*, pages 85–96.

Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In D D Lee, M Sugiyama, U V Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Yoav Goldberg. 2013. Dynamic-oracle transition-based parsing with calibrated probabilistic output. In *Proc. of IWPT*.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proc. of COLING*, pages 959–976.

Alex Graves. 2008. Supervised sequence labelling with recurrent neural networks. *Ph. D. thesis*.

Zhijiang Guo and Wei Lu. 2018. Better transition-based AMR parsing with a refined search space. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722, Brussels, Belgium. Association for Computational Linguistics.

Jan Hajic, Eva Hajicová, Jarmila Panevová, Petr Sgall, Ondrej Bojar, Silvie Cinková, Eva Fucíková, Marie Mikulová, Petr Pajas, Jan Popelka, et al. 2012. Announcing prague czech-english dependency treebank 2.0. In *LREC*, pages 3153–3160.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proc. of ACL*, pages 1127–1138.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018a. Multitask parsing across semantic representations. In *Proc. of ACL*, pages 373–385.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018b. Universal dependency parsing with a general transition-based DAG parser. In *Proc. of CoNLL UD Shared Task*, pages 103–112.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2019a. Content differences in syntactic and semantic representation. In *Proc. of NAACL-HLT*, pages 478–488, Minneapolis, Minnesota. Association for Computational Linguistics.

Daniel Hershcovich, Zohar Aizenbud, Leshem Choshen, Elior Sulem, Ari Rappoport, and Omri Abend. 2019b. SemEval-2019 task 1: Cross-lingual semantic parsing with UCCA. In *Proc. of SemEval*, pages 1–10.

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

Angelina Ivanova, Stephan Oepen, Lilja Øvrelid, and Dan Flickinger. 2012. Who did what to whom? A contrastive study of syntacto-semantic dependencies. In *Proc. of LAW*, pages 2–11.

Wei Jiang, Zhenghua Li, Yu Zhang, and Min Zhang. 2019. HLT@SUDA at SemEval-2019 task 1: UCCA graph parsing as constituent tree parsing. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 11–15, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327.

Zuchao Li, Hai Zhao, Zhuosheng Zhang, Rui Wang, Masao Utiyama, and Eiichiro Sumita. 2019. SJTU–NICT at MRP 2019: Multi-task learning for end-to-end uniform semantic graph parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 45 – 54, Hong Kong, China.

Wolfgang Maier. 2015. Discontinuous incremental shift-reduce parsing. In *Proc. of ACL*, pages 1202–1212.

Wolfgang Maier and Timm Lichte. 2016. Discontinuous parsing with continuous trees. In *Proc. of Workshop on Discontinuous Structures in NLP*, pages 47–57.

Jonathan May. 2016. SemEval-2016 task 8: Meaning representation parsing. In *Proc. of SemEval*, pages 1063–1073.

Jonathan May and Jay Priyadarshi. 2017. SemEval-2017 task 9: Abstract Meaning Representation parsing and generation. In *Proc. of SemEval*, pages 536–545.

Yusuke Miyao, Stephan Oepen, and Daniel Zeman. 2014. In-house: An ensemble of pre-existing off-the-shelf parsers. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 335–340.

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.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The dynamic neural network toolkit. *CoRR*, abs/1701.03980.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT*, pages 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proc. of ACL*, pages 351–359.

Rik van Noord and Johan Bos. 2017. Dealing with co-reference in neural semantic parsing. In *Proc. of SemDeep*, pages 41–49.

Stephan Oepen, Omri Abend, Jan Hajič, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdeňka Urešová. 2019. MRP 2019: Cross-framework Meaning Representation Parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 1 – 27, Hong Kong, China.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Zdenka Uresova. 2016. Towards comparability of linguistic graph banks for semantic parsing. In *Proc. of LREC*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, and Zdeňka Urešová. 2015. SemEval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*, pages 915–926.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*, pages 63–72.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based mrs banking. In *LREC*, pages 1250–1255.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1).

Carl Pollard and Ivan A Sag. 1994. *Head-driven phrase structure grammar*. University of Chicago Press.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning English strings with Abstract Meaning Representation graphs. In *Proc. of EMNLP*, pages 425–429.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proc. of COLING*, pages 753–760.

Sanjiv Kumar Sashank J. Reddi, Satyen Kale. 2018. On the convergence of Adam and beyond. *ICLR*.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Milan Straka and Jana Straková. 2017. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proc. of CoNLL UD Shared Task*, pages 88–99, Vancouver, Canada.

Ivan Titov, James Henderson, Paola Merlo, and Gabriele Musillo. 2009. Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Alper Tokgöz and Gülsen Eryiğit. 2015. Transition-based dependency DAG parsing using dynamic oracles. In *Proc. of ACL Student Research Workshop*, pages 22–27.

Chuan Wang, Sameer Pradhan, Xiaoman Pan, Heng Ji, and Nianwen Xue. 2016. CAMR at SemEval-2016 task 8: An extended transition-based AMR parser. In *Proc. of SemEval*, pages 1173–1178.

Chuan Wang and Nianwen Xue. 2017. Getting the most out of AMR parsing. In *Proc. of EMNLP*, pages 1257–1268.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proc. of ACL*, pages 857–862.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. A transition-based algorithm for AMR parsing. In *Proc. of NAACL*, pages 366–375.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proc. of CoNLL UD Shared Task*, pages 1–21.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gökırmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadova, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada. Association for Computational Linguistics.

Yue Zhang, Wei Jiang, Qingrong Xia, Junjie Cao, Rui Wang, Zhenghua Li, and Min Zhang. 2019. Suda–alibaba at MRP 2019: Graph-based models with BERT. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 149 – 157, Hong Kong, China.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proc. of ACL*, pages 434–443.