Broad-Coverage Semantic Parsing as Transduction

Sheng Zhang  Xutai Ma  Kevin Duh  Benjamin Van Durme
Johns Hopkins University
{zsheng2, xutai.ma}@jhu.edu
{kevinduh, vandurme}@cs.jhu.edu

Abstract

We unify different broad-coverage semantic parsing tasks under a transduction paradigm, and propose an attention-based neural framework that incrementally builds a meaning representation via a sequence of semantic relations. By leveraging multiple attention mechanisms, the transducer can be effectively trained without relying on a pre-trained aligner. Experiments conducted on three separate broad-coverage semantic parsing tasks – AMR, SDP and UCCA – demonstrate that our attention-based neural transducer improves the state of the art on both AMR and UCCA, and is competitive with the state of the art on SDP.

1 Introduction

Broad-coverage semantic parsing aims at mapping any natural language text, regardless of its domain, genre, or even the language itself, into a general-purpose meaning representation. As a long-standing topic of interest in computational linguistics, broad-coverage semantic parsing has targeted a number of meaning representation frameworks, including CCG (Steedman, 1996, 2001), DRS (Kamp and Reyle, 1993; Bos, 2008), AMR (Banarescu et al., 2013), UCCA (Abend and Rappoport, 2013), SDP (Oepen et al., 2014, 2015), and UDS (White et al., 2016). Each of these frameworks has their specific formal and linguistic assumptions. Such framework-specific “balkanization” results in a variety of framework-specific parsing approaches, and the state-of-the-art semantic parser for one framework is not always applicable to another. For instance, the state-of-the-art approaches to SDP parsing (Dozat and Manning, 2018; Peng et al., 2017a) are not directly transferable to AMR and UCCA because of the lack of explicit alignments between tokens in the sentence and nodes in the semantic graph.

While transition-based approaches are adaptable to different broad-coverage semantic parsing tasks (Wang et al., 2018; Hershovich et al., 2018; Damonte et al., 2017), when it comes to representations such as AMR whose nodes are unanchored to tokens in the sentence, a pre-trained aligner has to be used to produce the reference transition sequences (Wang et al., 2015; Damonte et al., 2017; Peng et al., 2017b). In contrast, there are attempts to develop attention-based approaches in a graph-based parsing paradigm (Dozat and Manning, 2018; Zhang et al., 2019), but they lack parsing incrementality, which is advocated in terms of computational efficiency and cognitive modeling (Nivre, 2004; Huang and Sagae, 2010).

In this paper, we approach different broad-coverage semantic parsing tasks under a unified framework of transduction. We propose an attention-based neural transducer that extends the two-stage semantic parser of Zhang et al. (2019) to directly transduce input text into a meaning representation in one stage. This transducer has properties of both transition-based approaches and graph-based approaches: on the one hand, it builds a meaning representation incrementally via a sequence of semantic relations, similar to a transition-based parser; on the other hand, it leverages multiple attention mechanisms used in recent graph-based parsers, thereby removing the need for pre-trained aligners.

Requiring only minor task-specific adaptations, we apply this framework to three separate broad-coverage semantic parsing tasks: AMR, SDP, and UCCA. Experimental results show that our neural transducer outperforms the state-of-the-art parsers on AMR (77.0% F1 on LDC2017T10 and 71.3%
F1 on LDC2014T12) and UCCA (76.6% F1 on the English-Wiki dataset v1.2), and is competitive with the state of the art on SDP (92.2% F1 on the English DELPH-IN MRS dataset).

2 Background and Related Work

We provide summary background on the meaning representations we target, and review related work on parsing for each.

Abstract Meaning Representation (AMR; Banarescu et al., 2013) encodes sentence-level semantics, such as predicate-argument information, reentrancies, named entities, negation and modality, into a rooted, directed, and usually acyclic graph with node and edge labels. AMR graphs abstract away from syntactic realizations, i.e., there is no explicit correspondence between elements of the graph and the surface utterance. Fig. 1(a) shows an example AMR graph.

Since its first general release in 2014, AMR has been a popular target of data-driven semantic parsing, notably in two SemEval shared tasks (May, 2016; May and Priyadarshi, 2017). Graph-based parsers build AMRs by identifying concepts and scoring edges between them, either in a pipeline (Flanigan et al., 2014), or jointly (Zhou et al., 2016; Lyu and Titov, 2018; Zhang et al., 2019). This two-stage parsing process limits the parser incrementality. Transition-based parsers either transform dependency trees into AMRs (Wang et al., 2015, 2016; Goodman et al., 2016), or employ transition systems specifically tailored to AMR parsing (Damonte et al., 2017; Ballesteros and Al-Onaizan, 2017). Transition-based parsers rely on pre-trained aligner produce the reference transitions. Grammar-based parsers leverage external semantic resources to derive AMRs compositionally based on CCG rules (Artzi et al., 2015), or SHRG rules (Peng et al., 2015). Another line of work uses neural model translation models to convert sentences into linearized AMRs (Barzdins and Gosko, 2016; Peng et al., 2017b), but has relied on data augmentation to produce effective parsers (van Noord and Bos, 2017; Konstas et al., 2017). Our parser differs from the previous ones in that it has incrementality without relying on pre-trained aligners, and can be effectively trained without data augmentation.

Semantic Dependency Parsing (SDP) was introduced in 2014 and 2015 SemEval shared tasks (Oepen et al., 2014, 2015). It is centered around three semantic formalisms – DM (DELPH-IN MRS; Flickinger et al., 2012; Oepen and Lønning, 2006), PAS (Predicate-Argument Structures; Miyao and Tsujii, 2004), and PSD (Prague Semantic Dependencies; Hajic et al., 2012) – representing predicate-argument relations between content words in a sentence. Their annotations have been converted into bi-lexical dependencies, forming directed graphs whose nodes injectively correspond to surface lexical units, and edges represent semantic relations between nodes. In this work, we focus on only the DM formalism. Fig. 1(b) shows an example DM graph.

Most recent parsers for SDP are graph-based: Peng et al. (2017a, 2018) use a max-margin classifier on top of a BiLSTM, with the factored score for each graph over predicates, unlabeled arcs, and arc labels. Multi-task learning approaches and disjoint data have been used to improve the parser performance. Dozat and Manning (2018) extend an LSTM-based syntactic dependency parser to produce graph-structured dependencies, and carefully tune it to state of the art performance. Wang et al. (2018) extend the transition system of Choi and McCallum (2013) to produce non-projective trees, and use improved versions of stack-LSTMs (Dyer et al., 2015) to learn representation for key components. All of these are specialized for bi-lexical dependency parsing, whereas our parser can effectively produce both bi-lexical semantics graphs, and graphs that are less anchored to the surface utterance.

Universal Conceptual Cognitive Annotation (UCCA; Abend and Rappoport, 2013) targets a level of semantic granularity that abstracts away from syntactic paraphrases in a typologically-motivated, cross-linguistic fashion. Sentence representations in UCCA are directed acyclic graphs (DAG), where terminal nodes correspond to surface lexical tokens, and non-terminal nodes to semantic units that participate in super-ordinate relations. Edges are labeled, indicating the role of a child in the relation the parent represents. Fig. 1(c) shows an example UCCA DAG.

The first UCCA parser is proposed by Hershcovich et al. (2017), where they extend a transition system to produce DAGs. To leverage other semantic resources, Hershcovich et al. (2018) is one of the few attempts to present (lossy) conversion from AMR, SDP and Universal Dependencies (UD; Nivre et al., 2016) to a unified UCCA-
Figure 1: Meaning representation in the task-specific format – (a) AMR, (b) DM, and (c) UCCA – for an example sentence “Pierre Vinken expressed his concern”. Meaning representation (d), (e) and (f) are in the unified arborescence format, which are converted from (a), (b) and (c) respectively.

Based DAG format. They explore multi-task learning under the unified format. While multi-task learning improves UCCA parsing results, it shows poor performance on AMR, SDP and UD parsing. In contrast, different semantic parsing tasks are formalized in our unified transduction paradigm with no loss, and our approach achieves state-of-the-art or competitive performance on each task, using only single-task data.

3 Unified Transduction Problem

3.1 Unified Arborescence Format

We first introduce a unified target format for different broad-coverage semantic parsing tasks. Meaning representation in the unified format is an arborescence (aka, a directed rooted tree), which is converted from its corresponding task-specific semantic graph via the following reversible steps:

**AMR**
- Reentrancy is what can make an AMR graph not an arborescence (it introduces cycles).
- Following Zhang et al. (2019), we convert an AMR graph into an arborescence by duplicating nodes that have reentrant relations; that is, whenever a node has a reentrant relation, we make a copy of that node and use the copy to participate in the relation, thereby resulting in an arborescence.
- Next, in order to preserve the reentrancy information, we assign a node index to each node. Duplicated nodes are assigned the same index as the original node. Fig. 1(d) shows an AMR arborescence converted from Fig. 1(a): two “person” nodes have the same node index 2. The original AMR graph can be recovered by merging identically indexed nodes.

**DM**
- We first break the DM graph into a set of weakly connected subgraphs.
- For each subgraph, if it has the top node, we treat top as root; otherwise, we treat the node with the max outdegree as root.
- We then run depth-first traversal over each subgraph from its root to yield an arborescence, and repeat the following three steps until no more edges can be added to the arborescence: (1) we run breadth-first traversal over the arborescence from the root until we find a node that has an incoming edge not belonging to the arborescence; (2) we reverse the edge and add a -of suffix to the edge label; (3) we run depth-first search from that node to include more edges to the arborescence. During the whole process, we add node indices and duplicate reentrant nodes in the same way as AMR conversion. Finally, we connect arborescences by adding a null edge from top to other arborescence roots. Fig. 1(e) shows a DM arborescence converted from Fig. 1(b). The original DM graph can be recovered by removing null edges, merging identically indexed nodes, and reversing edges with -of suffix.

**UCCA**
- To date, official UCCA evaluation only considers UCCA’s foundational layer, which is al-
Pierre Vinken expressed his concern.

Semantic Relations
Decoder Output
Semantic Relations
Encoder
Input Text
Decoding Module
Input Text
Embedding Module
Decoder
Source Node Module
Relation Type Module
Target Node Module

Figure 2: The encoder-decoder architecture of our attention-based neural transducer. An encoder encodes the input text into hidden states. A decoder is composed by three modules: a target node module, a relation type module, and a source node module. At each decoding time step, the decoder takes the previous semantic relation as input, and outputs a new semantic relation in a factorized way: firstly, the target node module produces a new target node; secondly, the source node module points to a preceding node as a new source node; finally, the relation type module predicts the relation type between source and target nodes.

Let $\mathcal{Y}$ be the output space. The unified transduction problem is to seek the most-likely sequence of semantic relations $\hat{Y}$ given $X$:

$$
\hat{Y} = \arg \max_{Y \in \mathcal{Y}} P(Y \mid X)
$$

$$
= \arg \max_{Y \in \mathcal{Y}} \prod_{i}^{m} P(y_{i} \mid y_{<i}, X)
$$

4 Transducer

To tackle the unified transduction problem, we introduce an attention-based neural transducer that extends Zhang et al. (2019)’s attention-based parser. Their attention-based parser addresses semantic parsing in a two-stage process: it first employs an extended variant of pointer-generator network (See et al., 2017) to convert the input text into a list of nodes, and then uses a deep biaffine graph-based parser (Dozat and Manning, 2016) with a maximum spanning tree (MST) algorithm to create edges. In contrast, our attention-based neural transducer directly transduces the input text into a meaning representation in one stage via a sequence of semantic relations. A high-level model architecture of our transducer is depicted in Fig. 2: an encoder first encodes the input text into hidden states; and then conditioned on the hidden states, at each decoding time step, a decoder takes the previous semantic relation as input, and outputs a new semantic relation, which includes a target node, a relation type, and a source node.
Specifically, there is a significant difference between Zhang et al. (2019) and our model: Zhang et al. (2019) first predicts nodes, and then edges. These two stages are done separately (except that a shared encoder is used). At the node prediction stage, their model has no knowledge of edges, and therefore node prediction is performed purely based on previous nodes. At the edge prediction stage, their model predicts the head of each node in parallel. Head prediction of one node has no constraints or impact on another. As a result, MST algorithms have to be used to search for a valid prediction. In comparison, our model does not have two separate stages for node and edge prediction. At each decoding step, our model predicts not only a node, but also the incoming edge to the node, which includes a source and a relation type. See Fig. 2 for an example. The predicted node and incoming edge together with previous predictions form a partial semantic graph, which is used as input of the next decoding step for the next node and incoming edge prediction. Our model therefore makes predictions based on the partial semantic graph, which helps prune the output space for both nodes and edges. Since at each decoding step, we assume the incoming edge is always from a preceding node (see Section 4.3 for the details), the predicted semantic graph is guaranteed to be a valid arborescence, and a MST algorithm is no longer needed.

4.1 Encoder

At the encoding stage, we employ an encoder embedding module to convert the input text into vector representations, and a BiLSTM is used to encode vector representations into hidden states.

Encoder Embedding Module concatenates word-level embeddings from GloVe (Pennington et al., 2014) and BERT\(^2\) (Devlin et al., 2018), char-level embeddings from CharCNN (Kim et al., 2016), and randomly initialized embeddings for POS tags.

For AMR, it includes extra randomly initialized embeddings for anonymization indicators that tell the encoder whether a token is an anonymized token from preprocessing.

For UCCA, it includes extra randomly initialized embeddings for NER tags, syntactic dependency labels, punctuation indicators, and shapes that are provided in the UCCA official dataset.

Multi-layer BiLSTM (Hochreiter and Schmidhuber, 1997) is defined as:

\[
s_t^l = \left[ \frac{s_t^l}{s_t^{l-1}} \right] = \left[ \frac{\text{LSTM}(s_{t-1}^l, s_{t-1}^l)}{\text{LSTM}(s_{t-1}^l, s_{t+1}^l)} \right], \quad (1)
\]

where \(s_t^l\) is the \(l\)-th layer hidden state at time step \(t\); \(s_t^l\) is the embedding module output for token \(x_t\).

4.2 Decoder

Decoder Embedding Module at decoding time step \(i\) converts elements in the input semantic relation \(\langle u_i, d_i^u, r_i, v_i, d_i^v \rangle\) into vector representations \((u_i, d_i^u, r_i, v_i, d_i^v)\).\(^3\)

\(u_i\) and \(v_i\) are concatenations of word-level embeddings from GloVe, char-level embeddings from CharCNN, and randomly initialized embeddings for POS tags. POS tags for source and target nodes are inferred at runtime: if a node is copied from input text, the POS tag of the corresponding token is used; if it is copied from a preceding node, the POS tag of the preceding node is used; otherwise, an UNK tag is used.

\(d_i^u, d_i^v\) and \(r_i\) are randomly initialized embeddings for source node index, target node index, and relation type.

Next, the decoder outputs a new semantic relation in a factorized way depicted in Fig. 2: First, a target node module takes vector representations of the previous semantic relation, and predicts a target node label as well as its index. Then, a source node module predicts a source node via pointing to a preceding node. Lastly, a relation type module takes the predicted source and target nodes, and predicts the relation type between them.

Target Node Module converts vector representations of the input semantic relation into a hidden state \(z_i\) in the following way:

\[
z_i = \text{FFN}^{(\text{relation})}(\langle \text{h}_i; c_i; r_i; u_i; d_i^v \rangle)
\]

\[
h_i = \text{LSTM}(h_{i-1}; h_{i-1})
\]

\[
\text{FFN}(x) = Wx + b
\]

where an \(l\)-layer LSTM generates contextual representation \(\text{h}_i^l\) for target node \(v_i\) (for initialization, \(h_0 = [v_i; d_i^v]\), \(h_0 = [s_1; s_n]\)). A feed-forward neural network \(\text{FFN}^{(\text{relation})}\) generates the hidden

---

\(^2\)We use average pooling in the same way as Zhang et al. (2019) to get word-level embeddings from BERT.

\(^3\)While training, the input semantic relation is from the reference sequence of relations; at test time, it is the previous decoder output semantic relation.
state \( z_i \) of the input semantic relation by combining contextual representation \( h_i^l \) for target node \( v_i \), encoder context vector \( c_i \), and vector representations \( r_i, u_i, d_i^p \) for relation type \( r_i \), source node label \( u_i \) and source node index \( d_i^p \).

Encoder context vector \( c_i \) is a weighted-sum of encoder hidden states \( s_{1:n}^j \). The weight is attention \( a_i^{(\text{enc})} \) from the decoder at decoding step \( i \) to encoder hidden states:

\[
a_i^{(\text{enc})} = \text{softmax}(\text{MLP}^{(\text{enc})}([h_i^l; s_{1:n}^j]))
\]

\[
\text{MLP}(x) = \text{ELU}(Wx + b)
\]

Given the hidden state \( z_i \) for input semantic relation, we use an extended variant of pointer-generator network to compute the probability distribution of next target node label \( v_{i+1} \):

\[
P(v_{i+1}) = p_{\text{gen}}p_i^{(\text{voc})} + p_{\text{enc}}a_i^{(\text{enc})} + p_{\text{dec}}a_i^{(\text{dec})}
\]

\[
p_i^{(\text{voc})} = \text{softmax}(\text{FFN}^{(\text{voc})}(z_i))
\]

\[
a_i^{(\text{dec})} = \text{softmax}(\text{MLP}^{(\text{dec})}([z_i; z_{i-1:n}]))
\]

\[
p_{\text{gen}}, p_{\text{enc}}, p_{\text{dec}} = \text{softmax}(\text{FFN}^{(\text{switch})}(z_i))
\]

The next target node index \( d_i^{v_{i+1}} \) is assigned based on the following rule:

\[
d_i^{v_{i+1}} = \begin{cases} d_j, & \text{if } v_{i+1} \text{ copies its antecedent } v_j, \\ i + 1, & \text{otherwise}. \end{cases}
\]

**Source Node Module** produces the next source node label \( u_{i+1} \) via pointing to a node label among preceding target node labels (the dotted arrows shown in Fig. 2). The probability distribution of next source node label \( u_{i+1} \) is defined as

\[
P(u_{i+1}) = \text{softmax}(\text{BIAFFINE}(h_{i+1}^{(\text{start})}, h_{i+1}^{(\text{end})}))
\]

where BIAFFINE is a biaffine function (Dozat and Manning, 2016). \( h_{i+1}^{(\text{start})} \) is the vector representation for the start of the pointer. \( h_{i+1}^{(\text{end})} \) are vector representations for possible ends of the pointer. They are computed by two multi-layer perceptrons:

\[
h_{i+1}^{(\text{start})} = MLP^{(\text{start})}(h_{i+1}^l) \quad (12)
\]

\[
h_{i+1}^{(\text{end})} = MLP^{(\text{end})}(h_{i+1}^l) \quad (13)
\]

Note that \( h_{i+1}^l \) is the LSTM hidden state for target node \( v_{i+1} \), generated by Equation (3) in the target node module. We reuse LSTM hidden states from the target node module such that we can train the decoder modules jointly.

Then, the next source node index \( d_i^{u_{i+1}} \) is the same as the target node the module points to.

**Relation Type Module** also reuses LSTM hidden states from the target node module to compute the probability distribution of next relation type \( r_{i+1} \). Assuming that the source node module points to target node label \( v_j \) as the next source node label, the next relation type probability distribution is computed by:

\[
P(r_{i+1}) = \text{softmax}(\text{BILINEAR}(h_{i+1}^{(\text{rel-src})}, h_j^{(\text{rel-tgt})}))
\]

\[
h_{i+1}^{(\text{rel-src})} = MLP^{(\text{rel-src})}(h_j^l) \quad (14)
\]

\[
h_j^{(\text{rel-tgt})} = MLP^{(\text{rel-tgt})}(h_{i+1}^l) \quad (16)
\]

### 4.3 Training

To ensure that at each decoding step, the source node can be found in the preceding nodes, we create the reference sequence of semantic relations by running a pre-order traversal over the reference arborescence. The pre-order traversal only determines the order between a node and its children. As for the order of its children, we sort them in alphabetic order in the case of AMR, following Zhang et al. (2019). In the case of SDP, we sort the children based on their order in the input text. In the case of UCCA, we sort the children based on their UCCA node ID.

Given a training pair \( \langle X, Y \rangle \), the optimization objective is to maximize the decomposed conditional log likelihood \( \sum_i \log P(y_i \mid y_{<i}, X) \), which is approximated by:

\[
\sum_i \log (P(u_i)) + \log (P(r_i)) + \log (P(v_i))
\]

We also employ label smoothing (Szegedy et al., 2016) to prevent overfitting, and include a coverage loss (See et al., 2017) to penalize repetitive nodes: \( \text{covloss}_i = \sum \min(a_i^{(\text{enc})}[t], \text{cov}[t]), \) where \( \text{cov}[t] = \sum_{j=0}^{i-1} a_j^{(\text{enc})} \).
4.4 Prediction

Our transducer at each decoding time step looks for the source node from the preceding nodes, which ensures that the output of a greedy search is already a valid arborescence $Y$:

$$P(Y \mid X) = \prod_{i} \max_{u_{i}} P(u_{i}) \max_{r_{i}} P(r_{i}) \max_{v_{i}} P(v_{i})$$

Therefore, a MST algorithm such as the Chu-Liu-Edmonds algorithm at $O(EV)$ used in Zhang et al. (2019) is no longer needed, and the decoding speed of our transducer is $O(V)$. Moreover, since our transducer builds the meaning representation via a sequence of semantic relations, we implement a beam search over relation in Algo. 1. Compared to the beam search of Zhang et al. (2019) that only returns top-$k$ nodes, our beam search finds the top-$k$ relation scores, which includes source nodes, relation types and target nodes.

Algorithm 1: Beam Search over Semantic Relations.

```
Input : The input text X.
Output: A sequence of relations $Y = \{y_1, ..., y_m\}$.
// Initialization.
i, score ← 0, 0;
Y, finished ← {}, {};
beam ← \{\{Y, score\}\};
// Encoding.
encode(X);
// Decoding.
for i ← 1 to MaxLength do
    new_beam ← {};
    \{Y, score\} ← beam.pop();
    for v_i in topK(P(v_i)) do
        if v_i = EOS then
            finished.push(\{Y, score\});
        else
            for u_i ← v_0 to v_i-1 do
                Y ← Y ∪ \{(u_i, r_i, v_i)\};
                score ← score + log (P(u_i)) +
                            log (P(r_i)) + log (P(v_i));
                new_beam.push(\{Y, score\});
            end
        end
    end
    beam ← new_beam.topK();
end
// Finishing.
while beam.not_empty() do
    \{Y, score\} ← beam.pop();
    finished.push(\{Y, score\});
end
\{Y, score\} ← finished.topK(k=1);
return Y;
```

4 $E$ denotes the number of edges. $V$ the number of nodes.

5 Data Pre- and Post-processing

AMR Pre- and post-processing steps are similar to those of Zhang et al. (2019): in preprocessing, we anonymize subgraphs of entities, remove senses, and convert resultant AMR graphs into the unified format; in post-processing, we assign the most frequent sense for nodes, restore Wikipedia links using the DBpedia Spotlight API (Daiber et al., 2013), add polarity attributes based on rules observed from training data, and recover the original AMR format from the unified format.

DM No pre- or post-processing is done to DM except converting them into the unified format, and recovering them from predictions.

UCCA During training, multi-sentence input text and its corresponding DAG are split into single-sentence training pairs based on rules observed from training data. At test time, we split multi-sentence input text, and join the predicted graphs into one. We also convert the original format to the unified format in preprocessing, and recover the original DAG format in post-processing.

<table>
<thead>
<tr>
<th>Hidden Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glove</td>
</tr>
<tr>
<td>BERT</td>
</tr>
<tr>
<td>POS / NER / Dep / Shapes</td>
</tr>
<tr>
<td>Anonymization / Node index</td>
</tr>
<tr>
<td>CharCNN kernel size</td>
</tr>
<tr>
<td>CharCNN channel size</td>
</tr>
<tr>
<td>Encoder</td>
</tr>
<tr>
<td>Decoder</td>
</tr>
<tr>
<td>Biaffine input size</td>
</tr>
<tr>
<td>Bilinear input size</td>
</tr>
<tr>
<td>AMR 1.0</td>
</tr>
<tr>
<td>AMR 1.0</td>
</tr>
<tr>
<td>UCCA 0.33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Learning rate</td>
</tr>
<tr>
<td>Maximum gradient norm</td>
</tr>
<tr>
<td>Coverage loss weight $\lambda$</td>
</tr>
<tr>
<td>Label smoothing $\epsilon$</td>
</tr>
<tr>
<td>Beam size</td>
</tr>
<tr>
<td>Batch size</td>
</tr>
<tr>
<td>Dropout rate</td>
</tr>
<tr>
<td>UCCA 0.33</td>
</tr>
<tr>
<td>AMR 0.33</td>
</tr>
<tr>
<td>AMR 1.0</td>
</tr>
<tr>
<td>AMR 1.0</td>
</tr>
<tr>
<td>UCCA 10000</td>
</tr>
<tr>
<td>UCCA 12000</td>
</tr>
<tr>
<td>UCCA 256</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 1: Hyperparameter settings</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Table 1: Hyperparameter settings</th>
</tr>
</thead>
</table>

3790
6 Experiments

6.1 Data and Setup

We evaluate our approach on three separate broad-coverage semantic parsing tasks: (1) AMR 2.0 (LDC2017T10) and 1.0 (LDC2014T12); (2) the English DM dataset from SemEval 2015 Task 18 (LDC2016T10); (3) the UCCA English Wikipedia Corpus v1.2 (Abend and Rappoport, 2013). The train/dev/test split follows the official setup. Our model is trained on two GeForce GTX TITAN X GPUs with early stop based on the dev set. We fix BERT parameters similar to Zhang et al. (2019) due to the limited GPU memory. Hyperparameter setting for each task is provided in Table 1.

6.2 Results

<table>
<thead>
<tr>
<th>Data</th>
<th>Parser</th>
<th>F1(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMR 2.0</td>
<td>Cai and Lam (2019)</td>
<td>73.2</td>
</tr>
<tr>
<td></td>
<td>Lyu and Titov (2018)</td>
<td>74.4±0.2</td>
</tr>
<tr>
<td></td>
<td>Lindemann et al. (2019)</td>
<td>75.3±0.1</td>
</tr>
<tr>
<td></td>
<td>Naseem et al. (2019)</td>
<td>75.5</td>
</tr>
<tr>
<td></td>
<td>Zhang et al. (2019)</td>
<td>76.3±0.1</td>
</tr>
<tr>
<td></td>
<td>- w/o beam search</td>
<td>75.3±0.1</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>77.0±0.1</td>
</tr>
<tr>
<td></td>
<td>- w/o beam search</td>
<td>76.4±0.1</td>
</tr>
<tr>
<td>AMR 1.0</td>
<td>Flanigan et al. (2016)</td>
<td>66.0</td>
</tr>
<tr>
<td></td>
<td>Pust et al. (2015)</td>
<td>67.1</td>
</tr>
<tr>
<td></td>
<td>Wang and Xue (2017)</td>
<td>68.1</td>
</tr>
<tr>
<td></td>
<td>Guo and Lu (2018)</td>
<td>68.3±0.4</td>
</tr>
<tr>
<td></td>
<td>Zhang et al. (2019)</td>
<td>70.2±0.1</td>
</tr>
<tr>
<td></td>
<td>- w/o beam search</td>
<td>69.2±0.1</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>71.3±0.1</td>
</tr>
<tr>
<td></td>
<td>- w/o beam search</td>
<td>70.4±0.1</td>
</tr>
</tbody>
</table>

Table 2: SMATCH F1 on AMR 2.0 and 1.0 test sets. Standard deviation is computed over 3 runs.

Table 3: Fine-grained F1 scores on the AMR 2.0 test set. L’18 is Lyu and Titov (2018); N’19 is Naseem et al. (2019); Z’19 is Zhang et al. (2019).

Table 4: Labeled F1 (%) scores on the English DM in-domain (WSJ) and out-of-domain (Brown corpus) test sets. (open) denotes results from the open track.
mantic parsing tasks such as AMR and UCCA. In contrast, our transducer is more general, and is competitive to the best SDP parser.

<table>
<thead>
<tr>
<th>Parser</th>
<th>F1 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hershcovich et al. (2017)</td>
<td>71.1</td>
</tr>
<tr>
<td>Hershcovich et al. (2018): single</td>
<td>71.2</td>
</tr>
<tr>
<td>Hershcovich et al. (2018): MTL</td>
<td>74.3</td>
</tr>
<tr>
<td>Ours</td>
<td>76.6±0.1</td>
</tr>
<tr>
<td>- w/o non-terminal node labels</td>
<td>75.7±0.1</td>
</tr>
</tbody>
</table>

Table 5: Labeled F1 (%) scores for all edges including primary edges and remote edges. Standard deviation is computed over 3 runs.

UCCA Table 5 compares our results to the previous best published results (labeled F1 for all edges) on the English Wiki test set. Hershcovich et al. (2018) explore multi-task learning (MTL) to improve UCCA parsing, using AMR, DM and UD parsing as auxiliaries. While improvement is achieved UCCA parsing, their MTL model shows poor results on the auxiliary tasks: 64.7% unlabeled F1 on AMR, 27.2% unlabeled F1 on DM, and 4.9% UAS on UD. In comparison, our transducer improves the state of the art on AMR, and shows competitive results on DM. At the same time, it also outperforms the best published UCCA results by 2.3% F1. When converting UCCA DAGs to the unified format, we adopt a simple rule (Section 3.1) to add node labels to non-terminals. Table 5 shows that these node labels do improve the parsing performance from 75.7% to 76.6%.

6.3 Analysis

Validity Graph-based parsers like Dozat and Manning (2018); Zhang et al. (2019) make independent decisions on edge types. As a result, the same outgoing edge type can appear multiple times to a node. For instance, a node can have more than one ARG1 outgoing edge. Although F1 scores can be computed for graphs with such kind of nodes, these graphs are in fact invalid mean representations. Our neural transducer incrementally builds meaning representations: at each decoding step, it takes a semantic relation as input, and has memory of preceding edge type information, which implicitly places constraints on edge type prediction. We compute the number of invalid graphs predicted by the parser of Zhang et al. (2019) and our neural transducer on the AMR 2.0 test set, and find that our neural transducer reduces the number of invalid graphs by 8%.

Speed Besides the improvement on parsing accuracy, we also significantly speed up parsing. Table 6 compares the parsing speed of our transducer and Zhang et al. (2019) on the AMR 2.0 test set, under the same environment setup. Without relying on MST algorithms to produce a valid arborescence, our transducer is able to parse at 1.7x speed.

<table>
<thead>
<tr>
<th>Speed (tokens/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang et al. (2019)</td>
</tr>
<tr>
<td>Ours</td>
</tr>
</tbody>
</table>

Table 6: Parsing speed on the AMR 2.0 test set.

7 Conclusion

We cast three broad-coverage semantic parsing tasks into a unified transduction framework, and propose a neural transducer to tackle the problem. Given the input text, the transducer incrementally builds a meaning representation via a sequence of semantic relations. Experiments conducted on three tasks show that our approach improves the state of the art in both AMR and UCCA, and is competitive to the best parser in SDP.

This work can be viewed as a starting point for cross-framework semantic parsing. Also, compared with transition-based parsers (e.g. Damonte et al., 2017) and graph-based parsers (e.g. Dozat and Manning, 2018), our transductive framework does not require a pre-trained aligner, and it is capable of building a meaning representation that is less anchored to the input text. These advantages make it well suited to semantic parsing in cross-lingual settings (Zhang et al., 2018). In the future, we hope to explore its potential in cross-framework and cross-lingual semantic parsing.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work was supported in part by the JHU Human Language Technology Center of Excellence, and DARPA LORELEI and AIDA. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes. The views and conclusions contained in this publication are those of the authors and should not be interpreted as representing official policies or endorsements of DARPA or the U.S. Government.
References


pages 50–57, Stroudsburg, PA, USA. Association for Computational Linguistics.


