Learning an Executable Neural Semantic Parser

Citation for published version:

Digital Object Identifier (DOI):
10.1162/coli_a_00342

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Computational Linguistics

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Learning an Executable Neural Semantic Parser *

Jianpeng Cheng  
University of Edinburgh  

Siva Reddy  
Stanford University  

Vijay Saraswat  
IBM T.J. Watson Research  

Mirella Lapata  
University of Edinburgh

This paper describes a neural semantic parser that maps natural language utterances onto logical forms which can be executed against a task-specific environment, such as a knowledge base or a database, to produce a response. The parser generates tree-structured logical forms with a transition-based approach which combines a generic tree-generation algorithm with domain-general grammar defined by the logical language. The generation process is modeled by structured recurrent neural networks, which provide a rich encoding of the sentential context and generation history for making predictions. To tackle mismatches between natural language and logical form tokens, various attention mechanisms are explored. Finally, we consider different training settings for the neural semantic parser, including fully supervised training where annotated logical forms are given, weakly-supervised training where denotations are provided, and distant supervision where only unlabeled sentences and a knowledge base are available. Experiments across a wide range of datasets demonstrate the effectiveness of our parser.

1. Introduction

An important task in artificial intelligence is to develop systems that understand natural language and enable interactions between computers and humans. Semantic parsing has emerged as a key technology towards achieving this goal. Semantic parsers specify a mapping between natural language utterances and machine-understandable meaning representations, commonly known as logical forms. A logical form can be executed against a real-world environment, such as a knowledge base, to produce a response, often called a denotation. Table 1 shows examples

* Email: jianpeng.cheng@ed.ac.uk

Submission received: November 6, 2017 Revised version received: July 17, 2018 Accepted for publication: August 10, 2018

© 2018 Association for Computational Linguistics
of natural language queries, their corresponding logical forms, and denotations. The query *What is the longest river in Ohio?* is represented by the logical form `longest(and(type.river, location(Ohio)))`, which when executed against a database of US geography returns the answer *Ohio River*. In the second example, the logical form `count(daughterOf(Barack Obama))` corresponds to the query *How many daughters does Obama have?* and is executed against the Freebase knowledge base to return the answer 2.

In recent years, semantic parsing has attracted a great deal of attention due to its utility in a wide range of applications such as question answering (Kwiatkowski et al. 2011; Liang, Jordan, and Klein 2011), relation extraction (Krishnamurthy and Mitchell 2012), goal-oriented dialog (Wen et al. 2015), natural language interfaces (Popescu et al. 2004), robot control (Matuszek et al. 2012), and interpreting instructions (Chen and Mooney 2011; Artzi and Zettlemoyer 2013).

Early statistical semantic parsers (Zelle and Mooney 1996; Zettlemoyer and Collins 2005; Wong and Mooney 2006; Kwiatkowski et al. 2010) mostly requires training data in the form of utterances paired with annotated logical forms. More recently, alternative forms of supervision have been proposed to alleviate the annotation burden, e.g., training on utterance-denotation pairs (Clarke et al. 2010; Liang 2016; Kwiatkowski et al. 2013), or using distant supervision (Krishnamurthy and Mitchell 2012; Cai and Yates 2013). Despite different supervision signals, training and inference procedures in conventional semantic parsers rely largely on domain-specific grammars and engineering. A CKY-style chart parsing algorithm is commonly employed to parse a sentence in polynomial time.

The successful application of recurrent neural networks (Bahdanau, Cho, and Bengio 2015; Sutskever, Vinyals, and Le 2014) to a variety of NLP tasks has provided
Table 1: Examples of questions, corresponding logical forms, and their answers.

<table>
<thead>
<tr>
<th>Environment: A database of US geography</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utterance: What is the longest river in Ohio?</td>
</tr>
<tr>
<td>Logical form: longest((type.river, location(Ohio)))</td>
</tr>
<tr>
<td>Denotation: Ohio River</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment: Freebase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utterance: How many daughters does Obama have?</td>
</tr>
<tr>
<td>Logical form: count(daughterOf(Barack Obama))</td>
</tr>
<tr>
<td>Denotation: 2</td>
</tr>
</tbody>
</table>

A strong impetus to treat semantic parsing as a sequence transduction problem where an utterance is mapped to a target meaning representation in string format (Dong and Lapata 2016; Jia and Liang 2016; Kočiský et al. 2016). Neural semantic parsers generate a sentence in linear time, while reducing the need for domain-specific assumptions, grammar learning, and more generally extensive feature engineering. But this modeling flexibility comes at a cost since it is no longer possible to interpret how meaning composition is performed, given that logical forms are structured objects like trees or graphs. Such knowledge plays a critical role in understanding modeling limitations so as to build better semantic parsers. Moreover, without any task-specific knowledge, the learning problem is fairly unconstrained, both in terms of the possible derivations to consider and in terms of the target output which can be syntactically invalid.

In this work we propose a neural semantic parsing framework which combines recurrent neural networks and their ability to model long-range dependencies with a transition system to generate well-formed and meaningful logical forms. The transition system combines a generic tree-generation algorithm with a small set of domain-general grammar pertaining to the logical language to guarantee correctness. Our neural parser differs from conventional semantic parsers in two respects. Firstly, it does not require lexicon-level rules to specify the mapping between natural language
and logical form tokens. Instead, the parser is designed to handle cases where the lexicon is missing or incomplete thanks to a neural attention layer, which encodes a soft mapping between natural language and logical form tokens. This modeling choice greatly reduces the number of grammar rules used during inference to those only specifying domain-general aspects. Secondly, our parser is transition-based rather than chart-based. Although chart-based inference has met with popularity in conventional semantic parsers, it has difficulty in leveraging sentence-level features since the dynamic programming algorithm requires features defined over substructures. In comparison, our linear-time parser allows us to generate parse structures incrementally conditioned on the entire sentence.

We perform several experiments in downstream question-answering tasks and demonstrate the effectiveness of our approach across different training scenarios. These include full supervision with questions paired with annotated logical forms using the GEOQUERY (Zettlemoyer and Collins 2005) dataset, weak supervision with question-answer pairs using the WEBQUESTIONS (Berant et al. 2013a) and GRAPHQUESTIONS (Su et al. 2016) datasets and distant supervision without question-answer pairs, using the SPADES (Bisk et al. 2016) dataset. Experimental results show that our neural semantic parser is able to generate high quality logical forms and answer real-world questions on a wide range of domains.

The remainder of this article is structured as follows. Section 2 provides an overview of related work. Section 3 introduces our neural semantic parsing framework and discusses the various training scenarios to which it can be applied. Our experiments are described in Section 4 together with detailed analysis of system output. Discussion of future work concludes the paper in Section 5.
2. Related Work

The proposed framework has connections to several lines of research including various formalisms for representing natural language meaning, semantic parsing models, and the training regimes they adopt. We review related work in these areas below.

Semantic Formalism. Logical forms have played an important role in semantic parsing systems since their inception in the 1970s (Winograd 1972; Woods, Kaplan, and Nash-Webber 1972). The literature is rife with semantic formalisms which can be used to define logical forms. Examples include lambda calculus (Montague 1973) which has been used by many semantic parsers (Zettlemoyer and Collins 2005; Kwiatkowksi et al. 2010; Reddy, Lapata, and Steedman 2014) due to its expressiveness and flexibility to construct logical forms of great complexity, Combinatory Categorial Grammar (Steedman 2000), dependency-based compositional semantics (Liang, Jordan, and Klein 2011), frame semantics (Baker, Fillmore, and Lowe 1998) and abstract meaning representations (Banarescu et al. 2013).

In this work, we adopt a database querying language as the semantic formalism, namely the functional query language (FunQL; Zelle (1995)). FunQL maps first-order logical forms into function-argument structures, resulting in recursive, tree-structured, program representations. Although it lacks expressive power, FunQL has a modeling advantage for downstream tasks, since it is more natural to describe the manipulation of a simple world as procedural programs. This modeling advantage has been revealed in recent advances of neural programmings: recurrent neural networks have demonstrated great capability in inducing compositional programs (Reed and De Freitas 2016; Neelakantan, Le, and Sutskever 2016; Cai, Shin, and Song 2017). For example, they learn to perform grade-school additions, bubble sort and table
comprehension in procedures. Finally, some recent work (Yin and Neubig 2017; Iyer et al. 2017; Zhong, Xiong, and Socher 2017) uses other programming languages, such as the SQL as the semantic formalism.

Semantic Parsing Model. The problem of learning to map utterances to meaning representations has been studied extensively in the NLP community. Most data-driven semantic parsers consist of three key components: a grammar, a trainable model, and a parsing algorithm. The grammar defines the space of derivations from sentences to logical forms, and the model together with the parsing algorithm find the most likely derivation. The model, which can take for example the form of an SVM (Kate and Mooney 2006), a structured perceptron (Zettlemoyer and Collins 2007; Lu et al. 2008; Reddy, Lapata, and Steedman 2014; Reddy et al. 2016) or a log-linear model (Zettlemoyer and Collins 2005; Berant et al. 2013a), scores the set of candidate derivations generated from the grammar. During inference, a chart-based parsing algorithm is commonly used to predict the most likely semantic parse for a sentence.

With recent advances in neural networks and deep learning, there is a trend of reformulating semantic parsing as a machine translation problem. The idea is not novel, since semantic parsing has been previously studied with statistical machine translation approaches in both Wong and Mooney (2006) and Andreas, Vlachos, and Clark (2013). However, the task setup is important to be revisited since recurrent neural networks have been shown to be extremely useful in context modeling and sequence generation (Bahdanau, Cho, and Bengio 2015). Following this direction, Dong and Lapata (2016) and Jia and Liang (2016) develop neural semantic parsers which treat semantic parsing as a sequence to sequence learning problem. Jia and Liang (2016) also introduces a data augmentation approach which bootstraps a synchronous grammar from existing data and generates artificial examples as extra training data. Other related
work extends the vanilla sequence to sequence model in various ways, such as multi-
task learning (Fan et al. 2017), parsing cross-domain queries (Herzig and Berant 2017)
and context-dependent queries (Suhr, Iyer, and Artzi 2018), and applying the model to
other formalisms such as AMR (Konstas et al. 2017) and SQL (Zhong, Xiong, and Socher
2017).

The fact that logical forms have a syntactic structure has motivated some of the recent work on exploring structured neural decoders to generate tree or
graph structures, and grammar constrained decoders to ensure the outputs are
meaningful and executable. Related work includes Yin and Neubig (2017) who generate
abstract syntax trees for source code with a grammar constrained neural decoder.
Krishnamurthy, Dasigi, and Gardner (2017) also introduce a neural semantic parser
which decodes rules in a grammar to obtain well-typed logical forms. Rabinovich, Stern,
and Klein (2017) propose abstract syntax networks with a modular decoder, whose
multiple submodels (one per grammar construct) are composed to generate abstract
syntax trees in a top-down manner.

Our work shares similar motivation: we generate tree-structured, syntactically valid
logical forms, however, following a transition-based generation approach (Dyer et al.
2016, 2015). Our semantic parser is a generalization of the model presented in Cheng
et al. (2017). While they focus solely on top-down generation using hard attention, the
parser presented in this work generates logical forms following either a top-down or
bottom-up generation order and introduces additional attention mechanisms (i.e., soft
and structured attention) for handling mismatches between natural language and
logical form tokens. We empirically compare generation orders and attention variants,
elaborate on model details, and formalize how the neural parser can be effectively
trained under different types of supervision.
Training Regimes. Various types of supervision have been explored to train semantic parsers, ranging from full supervision with utterance-logical form pairs to unsupervised semantic parsing without given utterances. Early work of statistical semantic parsing has mostly used annotated training data consisting of utterances paired with logical forms (Zelle and Mooney 1996; Kate and Mooney 2006; Kate, Wong, and Mooney 2005; Wong and Mooney 2006; Lu et al. 2008; Kwiatkowski et al. 2010). Same applies to some of the recent work on neural semantic parsing (Dong and Lapata 2016; Jia and Liang 2016). This form of supervision is the most effective to train the parser, but is also expensive to obtain. In order to write down a correct logical form, the annotator not only needs to have expertise in the semantic formalism, but also has to ensure the logical form matches the utterance semantics and contains no grammatical mistakes. For this reason, fully supervised training applies more to small, close domain problems, such as querying the US geographical database (Zelle and Mooney 1996).

Over the past few years, developments have been made to train semantic parsers with weak supervision from utterance-denotation pairs (Clarke et al. 2010; Liang, Jordan, and Klein 2011; Berant et al. 2013a; Kwiatkowski et al. 2013; Pasupat and Liang 2015). The approach enables more efficient data collection, since denotations (such as answers to a question, responses to a system) are much easier to obtain via crowd sourcing. For this reason, semantic parsing can be scaled to handle large, complex and open domain problems. Examples include the work that learn semantic parsers from question-answer pairs on Freebase (Liang, Jordan, and Klein 2011; Berant et al. 2013a; Berant and Liang 2014; Liang et al. 2017; Cheng et al. 2017), from system feedbacks (Clarke et al. 2010; Chen and Mooney 2011; Artzi and Zettlemoyer 2013), from abstract examples (Goldman et al. 2018), and from human feedbacks (Iyer et al. 2017) or statements (Artzi and Zettlemoyer 2011).
Some work seeks for more clever ways of gathering data and trains semantic parsers with even weaker supervision. In a class of distant supervision methods, the input is solely a knowledge base and a corpus of unlabeled sentences. Artificial training data is generated from the given resources. For example, Cai and Yates (2013) generate utterance paired with logical forms. Their approach searches for sentences containing certain entity pairs, and assume (with some pruning technique) the sentences express a certain relation from the KB. In Krishnamurthy and Mitchell (2012) and Krishnamurthy and Mitchell (2014) whose authors work with the CCG formalism, an extra source of supervision is added. The semantic parser is trained to produce parses that syntactically agree with dependency structures. Reddy, Lapata, and Steedman (2014) generate utterance-denotation pairs by masking entity mentions in declarative sentences from a large corpus. A semantic parser is then trained to predict the denotations corresponding to the masked entities.

3. Neural Semantic Parsing Framework

We present a neural-network based semantic parser that maps an utterance into a logical form, which can be executed in the context of a knowledge base to produce a response. Compared to traditional semantic parsers, our framework reduces the amount of manually engineered features and domain-specific rules. As semantic formalism, we choose the functional query language (FunQL), which is recursive and tree-structured (Section 3.1). A transition-based tree generation algorithm is then defined to generate FunQL logical forms (Sections 3.2–3.4). The process of generating logical forms is modeled by recurrent neural networks—a powerful tool for encoding the context of a sentence and the generation history for making predictions (Section 3.5). We handle mismatches between natural language and knowledge base through various attention mechanisms (Section 3.7). Finally, we explore different training regimes.
(Section 3.8, including a fully supervised setting where each utterance is labeled with annotated logical forms, a weakly supervised setting where utterance-denotation pairs are available, and distant supervision where only a collection of unlabeled sentences and a knowledge base is given.

3.1 FunQL Semantic Representation

As mentioned earlier, we adopt FunQL as our semantic formalism. FunQL is a variable free recursive meaning representation language which maps simple first order logical forms to function-argument structures that abstract away from variables and quantifiers (Kate and Mooney 2006). The language is also closely related to lambda DCS (Liang 2013), which makes existential quantifiers implicit. Lambda DCS is more compact in the sense that it can use variables in rare cases to handle anaphora and build composite binary predicates.

The FunQL logical forms we define contain the following primitive functional operators. They overlap with simple lambda DCS (Berant et al. 2013a) but differ slightly in syntax to ease recursive generation of logical forms. Let $l$ denote a logical form, $[l]$ represent its denotation, and $\mathcal{K}$ refers to a knowledge base.

- **Unary base case:** An entity $e$ (e.g., Barack Obama) is a unary logical form whose denotation is a singleton set containing that entity:
  \[
  [e] = \{e\} \tag{1}
  \]

- **Binary base case:** A relation $r$ (e.g., daughterOf) is a binary logical form with denotation:
  \[
  [r] = \{(e_1, e_2) : (e_1, r, e_2) \in \mathcal{K}\} \tag{2}
  \]
A relation $r$ can be applied to an entity $e_1$ (written as $r(e_1)$) and returns as denotation the unary satisfying the relation:

$$[r(e_1)] = \{ e : (e_1, e) \in [r] \}$$  \hspace{1cm} (3)

For example, the expression `daughterOf(Barack Obama)` corresponds to the question “Who are Barack Obama’s daughters?”.

- `count` returns the cardinality of the unary set $u$:

$$[\text{count}(u)] = \{|[u]|\}$$  \hspace{1cm} (4)

For example, `count(daughterOf(Barack Obama))` represents the question “How many daughters does Barack Obama have?”.

- `argmax` or `argmin` return a subset of the unary set $u$ whose specific relation $r$ is maximum or minimum:

$$[\text{argmax}(u, r)] = \{ e : e \in u \land \forall e' \in u, r(e) \geq r(e') \}$$  \hspace{1cm} (5)

For example, the expression `argmax(daughterOf(Barack Obama), age)` corresponds to the utterance “Who is Barack Obama’s eldest daughter?”.

- `filter` returns a subset of the unary set $u$ where a comparative constraint ($\leq, \geq, =, \neq, >, <$) acting on the relation $r$ is satisfied:

$$[\text{filter}_{\geq}(u, r, v)] = \{ e : e \in u \land r(e) \geq v \}$$  \hspace{1cm} (6)
For example, the query \( \text{filter}_\geq \) (daughterOf(Barack Obama), age, 5) returns the daughters of Barack Obama who are older than five years.

- **and** takes the intersection of two urinary sets \( u_1 \) and \( u_2 \):

\[
\llbracket \text{and}(u_1, u_2) \rrbracket = \llbracket u_1 \rrbracket \cap \llbracket u_2 \rrbracket
\]  

(7)

while **or** takes their union:

\[
\llbracket \text{or}(u_1, u_2) \rrbracket = \llbracket u_1 \rrbracket \cup \llbracket u_2 \rrbracket
\]  

(8)

For example, the expression **and**(daughterOf(Barack Obama), InfluentialTeensByYear(2014)) would correspond to the query

"Which daughter of Barack Obama was named Most Influential Teens in the year 2014?".

The operators just defined give rise to compositional logical forms (e.g., \( \text{count} \) and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))).

The reason for using FunQL in our framework lies in its recursive nature which allows us to model the process of generating logical form as a sequence of transition operations, which can be decoded by powerful recurrent neural networks. We next describe how our semantic formalism is integrated with a transition-based tree-generation algorithm to produce tree-structured logical forms.

### 3.2 Tree Generation Algorithm

We introduce a generic tree generation algorithm which recursively generates tree constituents with a set of transition operations. The key insight underlying our
algorithm is to define a canonical traversal or generation order, which generates a tree as a transition sequence. A transition sequence for a tree is a sequence of configuration-transition pairs \([c_0, t_0), (c_1, t_1), \ldots, (c_m, t_m)]\). In this work, we consider two commonly used generation orders, namely top-down pre-order and bottom-up post-order.

The top-down system is specified by the tuple \(c = (\sum, \pi, \sigma, N, P)\) where \(\sum\) is a stack used to store partially complete tree fragments, \(\pi\) is non-terminal token to be generated, \(\sigma\) is the terminal token to be generated, \(N\) is a stack of open non-terminals, and \(P\) is a function indexing the position of a non-terminal pointer. The pointer indicates where subsequent children nodes should be attached (e.g., \(P(X)\) means that the pointer is pointing to the non-terminal \(X\)). The initial configuration is \(c_0 = ([], TOP, \epsilon, [], \perp)\), where \(TOP\) stands for the root node of the tree, \(\epsilon\) represents an empty string, and \(\perp\) represents an unspecified function. The top-down system employs three transition operations defined in Table 2:

- **NT(X)** creates a new subtree non-terminal node denoted by \(X\). The non-terminal \(X\) is pushed on top of the stack and written as \(X(\) while subsequent tree nodes are generated as children underneath \(X\).
• **TER**(*x*) creates a new child node denoted by *x*. The terminal *x* is pushed on top of the stack, written as *x*.

• **RED** is the reduce operation which indicates that the current subtree being generated is complete. The non-terminal root of the current subtree is closed and subsequent children nodes will be attached to the predecessor open non-terminal. Stack-wise, **RED** recursively pops children (which can be either terminals or completed subtrees) on top until an open non-terminal is encountered. The non-terminal is popped as well, after which a completed subtree is pushed back to the stack as a single closed constituent, written for example as *X1*( *X2*, *X3* ).

We define the **bottom-up** system by tuple \( c = (\sum, \pi, \sigma) \) where \( \sum \) is a stack used to store partially complete tree fragments, \( \pi \) is the token non-terminal to be generated, and \( \sigma \) is the token terminal to be generated. We take the initial configuration to be \( c_0 = ([], x_l, \varepsilon) \), where \( x_l \) stands for the leftmost terminal node of the tree, and \( \varepsilon \) represents an empty string. The bottom-up generation uses two transition operations defined in Table 2:

• **TER**(*x*) creates a new terminal node denoted by *x*. The terminal *x* is pushed on top of the stack, written as *x*.

• **NT-RED**(*X*) builds a new subtree by attaching a parent node (denoted by *X*) to children nodes on top of the stack. The children nodes can be either terminals or smaller subtrees. Similarly to **RED** in the top-down case, children nodes are first popped from the stack, and subsequently combined with the parent *X* to form a subtree. The subtree is pushed back to the stack as a single constituent, written for example as *X1*( *X2*, *X3* ). A
challenge with $\text{NT-RED}(X)$ is to decide how many children should be
popped and included in the new subtree. In this work, the number of
children is dictated by the number of arguments expected by $X$ which is in
turn constrained by the logical language. For example, from the FunQL
grammar it is clear that $\text{count}$ takes one argument and $\text{argmax}$ takes two.
The language we use does not contain non-terminal functions with a
variable number of arguments.

Top-down traversal is defined by three generic operations, while bottom-up order
applies two operations only (since it combines reduce with non-terminal generation).
However, the operation predictions required are the same for the two systems. The
reason is that the reduce operation in the top-down system is deterministic when the
FunQL grammar is used as a constraint (we return to this point in Section 3.4).

3.3 Generating Tree-structured Logical Forms

To generate tree-structured logical forms, we integrate the generic tree generation
operations described above with FunQL, whose grammar determines the space of
allowed terminal and non-terminal symbols:

- $\text{NT}(X)$ includes an operation that generates relations $\text{NT}(\text{relation})$, and
  other domain-general operators in FunQL: $\text{NT}(\text{and}), \text{NT}(\text{or}),$
  $\text{NT}(\text{count}), \text{NT}(\text{argmax}), \text{NT}(\text{argmin})$ and $\text{NT}(\text{filter})$. Note that
  $\text{NT}(\text{relation})$ creates a placeholder for a relation, which is subsequently
generated.
Table 3: Top-down generation of the logical form $\text{count}(\text{and}(\text{daughterOf}(\text{Barack Obama}), \text{InfluentialTeensByYear}(2014)))$. Elements on the stack are separated by $||$ and the top of the stack is on the right.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Logical form token</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT (count)</td>
<td>count</td>
<td>count</td>
</tr>
<tr>
<td>NT (and)</td>
<td>and</td>
<td>count</td>
</tr>
<tr>
<td>NT (relation)</td>
<td>daughterOf</td>
<td>count</td>
</tr>
<tr>
<td>TER (entity)</td>
<td>Barack Obama</td>
<td>count</td>
</tr>
<tr>
<td>RED</td>
<td></td>
<td>count</td>
</tr>
<tr>
<td>NT (relation)</td>
<td>InfluentialTeensByYear</td>
<td>count</td>
</tr>
<tr>
<td>TER (entity)</td>
<td>2014</td>
<td>count</td>
</tr>
<tr>
<td>RED</td>
<td></td>
<td>count</td>
</tr>
<tr>
<td>RED</td>
<td></td>
<td>count</td>
</tr>
</tbody>
</table>

- TER(X) includes two operations: TER(relation) for generating relations and TER(entity) for generating entities. Both operations create a placeholder for a relation or an entity, which is subsequently generated.
- NT-RED(X) includes NT-RED(relation), NT-RED(and), NT-RED(or), NT-RED(count), NT-RED(argmax), NT-RED(argmin) and NT-RED(filter). Again, NT-RED(relation) creates a placeholder for a relation, which is subsequently generated.

Table 3 illustrates the sequence of operations employed by our parser in order to generate the logical form $\text{count}(\text{and}(\text{daughterOf}(\text{Barack Obama}), \text{InfluentialTeensByYear}(2014)))$ top-down. Table 4 shows how the same logical form is generated bottom-up. Note that the examples are simplified for illustration purposes; the logical form is generated conditioned on an input utterance, such as “How many daughters of Barack Obama were named Most Influential Teens in the year 2014?”. 
Table 4: Bottom-up generation of the logical form \( \text{count}(\text{and}(\text{daughterOf}(\text{Barack Obama}), \text{InfluentialTeensByYear}(2014))) \). Elements on the stack are separated by \(||\) and the top of the stack is on the right.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Logical form token</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>TER(entity)</td>
<td>Barack Obama</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>NT-RED(relation)</td>
<td>daughterOf</td>
<td>daughterOf(Barack Obama)</td>
</tr>
<tr>
<td>TER(entity)</td>
<td>2014</td>
<td>daughterOf(Barack Obama)</td>
</tr>
<tr>
<td>NT-RED(relation)</td>
<td>InfluentialTeensByYear</td>
<td>daughterOf(Barack Obama)</td>
</tr>
<tr>
<td>NT-RED(and)</td>
<td>and</td>
<td>and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))</td>
</tr>
<tr>
<td>NT-RED(count)</td>
<td>count</td>
<td>count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014)))</td>
</tr>
</tbody>
</table>

### 3.4 Constraints

A challenge in neural semantic parsing lies in generating well-formed and meaningful logical forms. To this end, we incorporate two types of constraints in our system. The first ones are structural constraints to ensure that the outputs are syntactically valid logical forms. For the top-down system these constraints include:

- The first operation must be NT;
- RED cannot directly follow NT;
- The maximum number of open non-terminal symbols allowed on the stack is 10. NT is disabled when the maximum number is reached;
- The maximum number of (open and closed) non-terminal symbols allowed on the stack is 10. NT is disabled when the maximum number is reached.

Tree constraints for the bottom-up system are:

- The first operation must be TER;
- The maximum number of consecutive TERs allowed is 5;
• The maximum number of terminal symbols allowed on the stack is the number of words in the sentence. TER is disallowed when the maximum number is reached.

The second type of constraints relate to the FunQL-grammar itself, ensuring that the generated logical forms are meaningful for execution:

• The type of argument expected by each non-terminal symbol must follow the FunQL grammar;

• The number of arguments expected by each non-terminal symbol must follow the FunQL grammar;

• When the expected number of arguments for a non-terminal symbol is reached, a RED operation must be called for the top-down system; for the bottom-up system this constrain is built within the NT-RED operation, since it reduces the expected number of arguments based on a specific non-terminal symbol.

3.5 Neural Network Realizer

We model the above logical form generation algorithm with a structured neural network which encodes the utterance and the generation history, and then predicts a sequence of transition operations as well as logical form tokens based on the encoded information. In the following, we present details for each component in the network.

Utterance Encoding. An utterance $x$ is encoded with a bidirectional LSTM architecture (Hochreiter and Schmidhuber 1997). A bidirectional LSTM is comprised of a forward LSTM and a backward LSTM. The forward LSTM processes a variable-length sequence $x = (x_1, x_2, \cdots, x_n)$ by incrementally adding new content into a single memory slot,
with gates controlling the extent to which new content should be memorized, old content should be erased, and current content should be exposed. At time step $t$, the memory $c_t$ and the hidden state $h_t$ are updated with the following equations:

$$
\begin{bmatrix}
    i_t \\
    f_t \\
    o_t \\
    \hat{c}_t
\end{bmatrix} =
\begin{bmatrix}
    \sigma \\
    \sigma \\
    \sigma \\
    \tanh
\end{bmatrix}
W \cdot
\begin{bmatrix}
    h_{t-1} \\
    x_t
\end{bmatrix}
$$

(9)

$$
c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t
$$

(10)

$$
h_t = o_t \odot \tanh(c_t)
$$

(11)

where $i$, $f$, and $o$ are gate activations; $W$ denotes the weight matrix. For simplicity, we denote the recurrent computation of the forward LSTM as:

$$
h_t = \overrightarrow{\text{LSTM}}(x_t, h_{t-1})
$$

(12)

After encoding, a list of token representations $[h_1, h_2, \ldots, h_n]$ within the forward context is obtained. Similarly, the backward LSTM computes a list of token representations $[\hat{h}_1, \hat{h}_2, \ldots, \hat{h}_n]$ within the backward context as:

$$
\hat{h}_t = \overleftarrow{\text{LSTM}}(x_t, \hat{h}_{t+1})
$$

(13)

Finally, each input token $x_i$ is represented by the concatenation of its forward and backward LSTM state vectors, denoted by $h_i = h_i \oplus \hat{h}_i$. The list storing token vectors for the entire utterance $x$ can be considered as a buffer, in analogy to syntactic parsing. A notable difference is that tokens in the buffer will not be removed since its alignment to logical form tokens is not pre-determined in the general semantic parsing scenario. We denote the buffer $b$ as $b = [h_1, \ldots, h_k]$, where $k$ denotes the length of the utterance.
Generation History Encoding. The generation history, aka partially completed subtrees, is encoded with a variant of stack-LSTM \cite{Dyer2015}. Such an encoder captures not only previously generated tree tokens but also tree structures. We first discuss the stack-based LSTM in the top-down transition system and then present modifications to account for the bottom-up system.

In **top-down** transitions, operations **NT** and **TER** change the stack-LSTM representation $s_t$ as in a vanilla LSTM as:

$$s_t = \text{LSTM}(y_t, s_{t-1})$$ \hspace{1cm} (14)

where $y_t$ denotes the newly generated non-terminal or terminal token. A **RED** operation recursively pops the stack-LSTM states as well as corresponding tree tokens on the output stack. The popping stops when a non-terminal state is reached and popped, after which the stack-LSTM reaches an intermediate state $s_{t-1,t}$.\footnote{We use $s_{t-1:t}$ to denote the intermediate transit state from time step $t - 1$ to $t$, after terminal tokens are popped from the stack; $s_t$ denotes the final LSTM state after the subtree representation is pushed back to the stack (as explained in the following).} The representation of the completed subtree $u$ is then computed as:

$$u = W_u \cdot [p_u : c_u]$$ \hspace{1cm} (15)

where $p_u$ denotes the parent (non-terminal) embedding of the subtree, $c_u$ denotes the average of the children (terminal or completed subtree) embeddings, and $W_u$ denotes the weight matrix. Note that $c_u$ can also be computed with more advanced method such as a recurrent neural network \cite{Kuncoro2017}. Finally, the subtree embedding $u$ serves as the input to the LSTM and updates $s_{t-1:t}$ to $s_t$ as:

$$s_t = \text{LSTM}(u, s_{t-1:t})$$ \hspace{1cm} (16)
Figure 1 provides a graphical view on how the three operations change the configuration of a stack-LSTM.

In comparison, the **bottom-up** transition system uses the same TER operation to update the stack-LSTM representation \( s_t \) when a terminal \( y_t \) is newly generated:

\[
s_t = \text{LSTM}(y_t, s_{t-1})
\]  

(17)

Differently, the effects of NT and RED are merged into a NT-RED(\( X \)) operation. When NT-RED(\( X \)) is invoked, a non-terminal \( y_t \) is first predicted and then the stack-LSTM starts popping its states on the stack. The number of pops is decided by the amount of argument expected by \( y_t \). After that, a subtree can be obtained by combining the non-terminal \( y_t \) and the newly popped terminal tokens, while the stack-LSTM reaches an intermediate state \( s_{t-1:t} \). Similar to the top-down system, we compute the representation of the newly combined subtree \( u \) as:

\[
u = W_u \cdot [p_u : c_u]
\]  

(18)

where \( p_u \) denotes the parent (non-terminal) embedding of the subtree, \( c_u \) denotes the average of the children (terminal or completed subtree) embeddings, and \( W_u \) denotes the weight matrix. Finally, the subtree embedding \( u \) serves as the input to the LSTM and updates \( s_{t-1:t} \) to \( s_t \) as:

\[
s_t = \text{LSTM}(u, s_{t-1:t})
\]  

(19)

The key difference here is that a non-terminal tree token is never pushed alone to update the stack-LSTM, but rather as part of a completed subtree that does the update.

**Making Predictions.** Given encodings of the utterance and generation history, our model makes two types of predictions pertaining to transition operations and logical form.
Computational Linguistics Volume xx, Number xx

Figure 1: A stack-LSTM extends a standard LSTM with the addition of a stack pointer (shown as Top in the figure). The example shows how the configuration of the stack changes when the operations NT, TER, and RED are applied in sequence. The initial stack is presumed empty for illustration purposes. We only show how the stack-LSTM updates its states, not how subsequent predictions are made which depend not only on the hidden state of the stack-LSTM, but also on the natural language utterance.

tokens (see Tables 3, 4). First, at every time step, the next transition operation $o_{t+1}$ is predicted based on utterance encoding $b$ and generation history $s_t$:

$$o_{t+1} \sim f(b, s_t)$$  \hspace{1cm} (20)$$

where $f$ is a neural network that computes the parameters of a multinomial distribution over the action space which is restricted by the constraints discussed in Section 3.4.

Next, the logical form token underlying each generation operation must be emitted. When the generation operation contains one of the domain-general non-terminals count, argmax, argmin, and, or, and filter (e.g., NT(count)), the logical form token is the corresponding non-terminal (e.g., count). When the generation operation
involves one of the placeholders for entity or relation (e.g., \( NT(\text{relation}) \), \( NT-RED(\text{relation}) \), \( TER(\text{relation}) \) and \( TER(\text{entity}) \)), a domain-specific logical form token \( y_{t+1} \) (i.e., an entity or a relation) is predicted in a fashion similar to action prediction:

\[
y_{t+1} \sim g(b_{t}, s_{t})
\]

where \( g \) is a neural network that computes the parameters of a multinomial distribution over the token space.

A remaining challenge lies in designing predictive functions \( f \) (for the next action) and \( g \) (for the next logical form token) in the context of semantic parsing. We explore various attention mechanisms which we discuss in the next sections.

### 3.6 Next Action Prediction

This section explains how we model function \( f \) for predicting the next action. We draw inspiration from previous work on transition-based syntactic parsing and compute a feature vector representing the current state of the generation system (Dyer et al. 2016). This feature vector typically leverages the buffer which stores unprocessed tokens in the utterance and the stack which stores tokens in the partially completed parse tree. A major difference in our semantic parsing context is that the buffer configuration does not change deterministically with respect to the stack since the alignment between natural language tokens and logical-form tokens is not explicitly specified. This gives rise to the challenge of extracting features representing the buffer at different time steps. To this end, we compute at each time step \( t \) a single adaptive representation of the buffer \( \bar{b}_{t} \) with a soft attention mechanism:

\[
u_{t} = V \tanh(W_{b}b_{t} + W_{s}s_{t})
\]
\[ \alpha_i^t = \text{softmax}(u_i^t) \]  
\[ \bar{b}_t = \sum_i \alpha_i^t b_i \]  

where \( W_b \) and \( W_s \) are weight matrices and \( V \) is a weight vector. We then combine the representation of the buffer and the stack with a feed-forward neural network (Equation (25)) to yield a feature vector for the generation system. Finally, \text{softmax} is taken to obtain the parameters of the multinomial distribution over actions:

\[ a_{t+1} \sim \text{softmax}(W_{oa} \tanh(W_f[\bar{b}_t, s_t])) \]  

where \( W_{oa} \) and \( W_f \) are weight matrices.

### 3.7 Next Token Prediction

This section presents various functions \( g \) for predicting the next logical form token (i.e., a specific entity or relation). A hurdle in semantic parsing concerns handling mismatches between natural language and logical tokens in the target knowledge base. For example, both utterances “Where did X graduate from” and “Where did X get his PhD” would trigger the same predicate \text{education} in Freebase. Traditional semantic parsers map utterances directly to domain-specific logical forms relying exclusively on a set of lexicons either predefined or learned for the target domain with only limited coverage. Recent approaches alleviate this issue by firstly mapping the utterance to a domain-general logical form which aims to capture language-specific semantic aspects, after which ontology matching is performed to handle mismatches (Kwiatkowski et al. 2013; Reddy, Lapata, and Steedman 2014; Reddy et al. 2016). Beyond efficiency considerations, it remains unclear which domain-general representation is best suited to domain-specific semantic parsing.
Neural networks provide an alternative solution: the matching between natural language and domain-specific predicates is accomplished via an attention layer, which encodes a context-sensitive probabilistic lexicon. This is analogous to the application of the attention mechanism in machine translation (Bahdanau, Cho, and Bengio 2015), which is used as an alternative to conventional phrase tables. In this work, we consider a practical domain-specific semantic parsing scenario where we are given no lexicon. We first introduce the basic form of attention used to predict logical form tokens and then discuss various extensions as shown in Figure 3.

**Soft Attention.** In the case where no lexicon is provided, we use a soft attention layer similar to action prediction. The parameters of the soft attention layer prior to softmax are shared with those used in action prediction:

\[
\begin{align*}
    u_i^t &= V \tanh(W_b b_i + W_s s_t) \\
    \alpha_i^t &= \text{softmax}(u_i^t) \\
    \bar{b}_t &= \sum_i \alpha_i^t b_i \\
    y_{t+1} &\sim \text{softmax}(W_{yo} \tanh(W_f [\bar{b}_t, s_t]))
\end{align*}
\]  

which outputs the parameters of the multinomial distribution over logical form tokens (either predicates or entities). When dealing with extremely large knowledge bases, the output space can be pruned and restricted with an entity linking procedure. This method requires us to identity potential entity candidates in the sentence, and then generate only entities belonging to this subset and the relations linking them.
Structured Soft Attention. We also explored a structured attention layer (Kim et al. 2017; Liu and Lapata 2018) to encourage the model to attend to contiguous natural language phrases when generating a logical token, while still being differentiable.

The structured attention layer we adopt is a linear-chain conditional random field (CRF; Lafferty, McCallum, and Pereira 2001). Assume that at time step $t$ each token in the buffer (e.g., the $i$th token) is assigned an attention label $A_i^t \in \{0, 1\}$. The CRF defines $p(A_t)$, the probability of the sequence of attention labels at time step $t$ as:

$$p(A_t) = \frac{\exp \sum_i W_f \cdot \psi(A_{t-1}^i, A_i^t, b_i, s_i)}{\sum_{A_1^t, \ldots, A_n^t} \exp \sum_i W_f \cdot \psi(A_{t-1}^i, A_i^t, b_i, s_i)}$$  \hspace{1cm} (30)

where $\sum_i$ sums over all tokens and $\sum_{A_1^t, \ldots, A_n^t}$ sums over all possible sequences of attention labels. $W_f$ is a weight vector and $\psi(A_{t-1}^i, A_i^t, b_i, s_i)$ a feature vector. In this work the feature vector is defined with three dimensions: the state feature for each token:

$$u_i^t \cdot a_i^t$$  \hspace{1cm} (31)

where $u_i^t$ is the token-specific attention score computed in Equation (26); the transition feature:

$$A_{i-1}^t \cdot A_i^t$$  \hspace{1cm} (32)

and the context-dependent transition feature

$$u_i^t \cdot A_{i-1}^t \cdot A_i^t$$  \hspace{1cm} (33)

The marginal probability $p(A_i^t = 1)$ of each token being selected is computed with the forward-backward message passing algorithm (Lafferty, McCallum, and Pereira 2001). The procedure is shown in Figure 2. To compare with standard soft attention,
**Objective:** Predict the next logical form token given the current stack representation \( s_t \) and \( n \) input token representations in the buffer \( b_1 \cdots b_n \).

**Steps:**

1. Compute the logit \( u^t_i \) of each input token \( b_i \) as \( u^t_i = V \tanh(W_s b_i + W_t s_t) \). The logit will be used to compute the first and third feature in \( \psi \).

2. **Forward algorithm:** Initialize \( \beta(A^1_t) = 1 \).
   For \( i \in \{2 \cdots n\}, A^i_t \in \{0, 1\} \):
   \[
   \beta(A^i_t) = \sum_{A^{i-1}_t \in \{0, 1\}} \beta(A^{i-1}_t) \times \psi(A^{i-1}_t, A^i_t, b_i, s_t),
   \]
   where \( \psi \) is the context-dependent feature vector.

3. **Backward algorithm:** Initialize \( \gamma(A^n_t) = 1 \).
   For \( i \in \{1 \cdots (n-1)\}, A^i_t \in \{0, 1\} \):
   \[
   \gamma(A^i_t) = \sum_{A^{i+1}_t \in \{0, 1\}} \gamma(A^{i+1}_t) \times \psi(A^i_t, A^{i+1}_t, b_i, s_t),
   \]
   where \( \psi \) is the context-dependent feature vector.

4. Compute the marginal probability \( \alpha^t_i \) of each input token \( b_i \):
   \[
   \alpha^t_i = \beta(A^i_t) \times \gamma(A^i_t)
   \]

5. **Apply soft attention to compute an adaptive buffer representation:**
   \[
   \bar{b}_t = \sum_i \alpha^t_i b_i
   \]

6. **Predict the next token:**
   \[
   y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f [\bar{b}_t, s_t]))
   \]

7. **Compute the error and backpropagate.**

Figure 2: The structured attention model for token prediction.

we denote this procedure as:

\[
\alpha^t_i = \text{forward-backward}(u^t_i)
\]  

(34)

The marginal probabilities are used as in standard soft attention to compute an adaptive buffer representation:

\[
\bar{b}_t = \sum_i \alpha^t_i b_i
\]

(35)

which is then used to compute a distribution of output logical form tokens:

\[
y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f [\bar{b}_t, s_t]))
\]

(36)
The structured attention layer is soft and fully differentiable and allows us to model attention over phrases since the forward-backward algorithm implicitly sums over an exponentially-sized set of substructures through dynamic programming.

**Hard Attention.** Soft attention learns a complete mapping between natural language and logical tokens with a differentiable neural layer. At every time step, every natural language token in the utterance is assigned the probability of triggering every logical predicate. This offers little in the way of interpretability. In order to render the inner workings of the model more transparent we explore the use of a hard attention mechanism as a means of rationalizing neural predictions.

At each time step, hard attention samples from the attention probability a single natural language token $x_t$:

$$u_t^i = V \tanh(W_b b_t + W_s s_t)$$

$$x_t \sim \text{softmax}(u_t^i)$$

The representation of $x_t$, denoted by $b_t$ is then used to predict the logical token $y_t$:

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f [b_t, s_t]))$$

Hard attention is nevertheless optimization-wise challenging; it requires sampling symbols (aka non-differentiable representations) inside an end-to-end module which may incur high variance. In practice, we adopt a baseline method to reduce the variance of the predictor which we discuss in Section 3.8.1.

**Binomial Hard Attention.** Learning difficulties aside, a limitation of hard attention lies in selecting a single token to attend to at each time step. In practice, a logical form
Figure 3: Different attention mechanisms for predicting the next logical form token. The example utterance is *which daughter of Barack Obama was named Most Influential Teens in the year 2014?* and the corresponding logical form to be generated is \( \text{and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))} \). The figure shows attention for predicting *InfluentialTeensByYear*. Darker shading indicates higher values.

Predicate is often triggered by a natural language phrase or a multi-word expression. A way to overcome this limitation is to compute a binomial distribution for every token separately, indicating the probability of the token being selected. Then an attention label is assigned to each token based on this probability (e.g., with threshold 0.5). Let \( A^t_i \in \{0, 1\} \) denote the attention label of the \( i \)th token at time step \( t \). Using the unnormalized attention score \( u^t_i \) computed in Equation (26), we obtain the probability \( p(A^t_i = 1) \) as:

\[
p(A^t_i = 1) = \text{logistic}(u^t_i)
\]  

where logistic denotes a logistic regression classifier. We compute adaptive buffer representation as an average of the selected token embeddings:

\[
\bar{b}_t = \frac{1}{\sum_i A^t_i} \sum_i A^t_i b_i
\]  

which is then used to compute a distribution of the output logical form tokens:

\[
y_{t+1} \sim \text{softmax}(W_{ogy} \tanh(W_f[\bar{b}_t, s_t]))
\]
Table 5: Example data for various semantic parsing training regimes.

<table>
<thead>
<tr>
<th>Training Regime</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full supervision</td>
<td>utterance-logical form pairs</td>
</tr>
<tr>
<td></td>
<td>utterance: <em>which daughter of Barack Obama was named Most Influential Teens in the year 2014?</em></td>
</tr>
<tr>
<td></td>
<td>logical form: and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))</td>
</tr>
<tr>
<td>Weak supervision</td>
<td>utterance-denotation pairs</td>
</tr>
<tr>
<td></td>
<td>utterance: <em>which daughter of Barack Obama was named Most Influential Teens in the year 2014?</em></td>
</tr>
<tr>
<td></td>
<td>denotation: Malia Obama</td>
</tr>
<tr>
<td>Distant supervision</td>
<td>entity-masked utterances</td>
</tr>
<tr>
<td></td>
<td>utterance: Malia Obama, the daughter of Barack Obama, was named Most Influential Teens in the year 2014.</td>
</tr>
<tr>
<td></td>
<td>artificial utterance: <em>blank</em>, the daughter of Barack Obama, was named Most Influential Teens in the year 2014.</td>
</tr>
<tr>
<td></td>
<td>denotation: Malia Obama</td>
</tr>
</tbody>
</table>

3.8 Model Training

We now discuss how our neural semantic parser can be trained under different conditions, i.e., with access to utterances annotated with logical forms, when only denotations are provided, and finally, when neither logical forms nor denotations are available (see Table 5).

3.8.1 Learning from Utterance-Logical Form Pairs. The most straightforward training setup is fully supervised making use of utterance-logical form pairs. Consider utterance $x$ with logical form $l$ whose structure is determined by a sequence of transition operations $a$ and a sequence of logical form tokens $y$. Our ultimate goal is to maximize the conditional likelihood of the logical form given the utterance for all training data:

$$
\mathcal{L} = \sum_{(x,l) \in T} \log p(l|x)
$$

which can be decomposed into the action likelihood and the token likelihood:

$$
\log p(l|x) = \log p(a|x) + \log p(y|x, a)
$$
Soft attention. The above objective consists of two terms, one for the action sequence:

$$
\mathcal{L}_a = \sum_{(x,l) \in T} \log p(a|x) = \sum_{(x,l) \in T} \sum_{t=1}^{n} \log p(a_t|x) \tag{45}
$$

and one for the logical form token sequence:

$$
\mathcal{L}_y = \sum_{(x,l) \in T} \log p(y|x, a) = \sum_{(x,l) \in T} \sum_{t=1}^{n} \log p(y_t|x, a_t) \tag{46}
$$

These constitute the training objective for fully differentiable neural semantic parsers, when (basic or structured) soft attention is used.

Hard attention. When hard attention is used for token prediction, the objective $\mathcal{L}_a$ remains the same but $\mathcal{L}_y$ differs. This is because the attention layer is non-differentiable for errors to backpropagate through. We use the alternative REINFORCE-style algorithm (Williams 1992) for backpropagation. In this scenario, the neural attention layer is used as a policy predictor to emit an attention choice, while subsequent neural layers are used as the value function to compute a reward—a lower bound of the log likelihood $\log p(y|x, a)$. Let $u_t$ denote the latent attention choice$^2$ at each time step $t$; we maximize the expected log likelihood of the logical form token given the overall attention choice for all examples, which by Jensen’s Inequality is the lower bound on the log likelihood $\log p(y|x, a)$:

$$
\mathcal{L}_y = \sum_{(x,l) \in T} \sum_{u} [p(u|x, a) \log p(y|u, x, a)] \\
\leq \sum_{(x,l) \in T} \log \sum_{u} [p(u|x, a)p(y|u, x, a)] \\
= \sum_{(x,l) \in T} \log p(y|x, a) \tag{47}
$$

$^2$ In standard hard attention, the choice is a single token in the sentence; while in binomial hard attention, it is a phrase.
The gradient of $L_y$ with respect to model parameters $\theta$ is given by:

$$
\frac{\partial L_y}{\partial \theta} = \sum_{(x,l) \in T} \sum_u p(u|x,a) \left( \frac{\partial \log p(y|u,x,a)}{\partial \theta} + \log p(y|u,x,a) \frac{\partial p(u|x,a)}{\partial \theta} \right) + \log p(y|u,x,a) \frac{\partial \log p(u|x,a)}{\partial \theta} \frac{p(u|x,a)}{p(u|x,a)}
$$

(48)

which is estimated by the Monte Carlo estimator with $K$ samples. This gradient estimator incurs high variance because the reward term $\log p(y|u^k,x,a)$ is dependent on the samples of $u^k$. An input-dependent baseline is used to reduce the variance, which adjusts the gradient update as:

$$
\frac{\partial L_y}{\partial \theta} = \sum_{(x,l) \in T} \frac{1}{N} \sum_{k=1}^K \left[ \frac{\partial \log p(y|u^k,x,a)}{\partial \theta} + \log p(y|u^k,x,a) \frac{\partial \log p(u^k|x,a)}{\partial \theta} \right]
$$

(49)

As baseline, we use the soft attention token predictor described earlier. The effect is to encourage attention samples that return a higher reward than standard soft attention, while discouraging those resulting in a lower reward. For each training case, we approximate the expected gradient with a single sample of $u^k$.

3.8.2 Learning from Utterance-Denotation Pairs. Unfortunately, training data consisting of utterances and their corresponding logical forms is difficult to obtain at large scale, and as a result limited to a few domains with a small number of logical predicates. An alternative to full supervision is a weakly supervised setting where the semantic parser is trained on utterance-denotation pairs, where logical forms are treated as latent.
In the following we firstly provide a brief review of conventional weakly supervised semantic parsing systems (Berant et al. 2013a), and then explain the extension of our neural semantic parser to a similar setting. Conventional weakly-supervised semantic parsing systems separate the parser from the learner (Liang 2016). A chart-based (non-parametrized) parser will recursively build derivations for each span of an utterance, eventually obtaining a list of candidate derivations mapping the utterance to its logical form. The learner (which is often a log-linear model) defines features useful for scoring and ranking the set of candidate derivations, and is trained based on the correctness of their denotations. As mentioned in Liang (2016), the chart-based parser brings a disadvantage since the system does not support incremental contextual interpretation, because features of a span can only depend on the sub-derivations in that span, as a requirement of dynamic programming.

Different from chart-based parsers, a neural semantic parser is itself a parametrized model and is able to leverage global utterance features (via attention) for decoding. However, training the neural parser directly with utterance-denotation pairs is challenging since the decoder does not have access to gold standard logical forms for backpropagation. Moreover, the neural decoder is a conditional generative model which generates logical forms in a greedy fashion and therefore lacks the ability to make global judgments of logical forms. To this end, we follow conventional setup in integrating our neural semantic parser with a log-linear ranker, to cope with the weak supervision signal. The role of the neural parser is to generate a list of candidate logical forms, while the ranker is able to leverage global features of utterance-logical form-denotation triplets to select which candidate to use for execution.
The objective of the log-linear ranker is to maximize the log marginal likelihood of the denotation \( d \) via latent logical forms \( l \):

\[
\log p(d|x) = \log \sum_{l \in L} p(l|x)p(d|x, l)
\]  

(50)

where \( L \) denotes the set of candidate logical forms generated by the neural parser. Note that \( p(d|x, l) \) equates to 1 if the logical form executes to the correct denotation and 0 otherwise. For this reason, we can also write the above equation as \( \log \sum_{l \in L(c)} p(l|x) \), where \( L(c) \) is the set of consistent logical forms which execute to the correct denotation.

Specifically \( p(l|x) \) is computed with a log-linear model:

\[
p(l|x) = \frac{\exp(\phi(x, l)\theta)}{\sum_{l' \in L} \exp(\phi(x, l')\theta)}
\]  

(51)

where \( L \) is the set of candidate logical forms; \( \phi \) is the feature function that maps an utterance-logical form pair (and also the corresponding denotation) into a feature vector; and \( \theta \) denotes the weight parameter of the model.

Training such a system involves the following steps. Given an input utterance, the neural parser first generates a list of candidate logical forms via beam search. Then these candidate logical forms are executed and those which yield the correct denotation are marked as consistent logical forms. The neural parser is then trained to maximize the likelihood of these consistent logical forms \( \sum_{l \in L_c} \log p(l|x) \). Meanwhile, the ranker is trained to maximize the marginal likelihood of denotations \( \log p(d|x) \).

Clearly, if the parser does not generate any consistent logical forms, no model parameters will be updated. A challenge in this training paradigm is the fact that we rely exclusively on beam search to find good logical forms from an exponential search space. In the beginning of training, neural parameters are far from optimal, and as a result good logical forms are likely to fall outside the beam. We alleviate this problem
by performing entity linking which greatly reduces the search space. We determine the identity of the entities mentioned in the utterance according to the knowledge base and restrict the neural parser to generating logical forms containing only those entities.

### 3.8.3 Distant Supervision.

Despite allowing to scale semantic parsing to large open-domain problems (Kwiatkowski et al. 2013; Berant et al. 2013a; Yao and Van Durme 2014), the creation of utterance-denotation pairs still relies on labor-intensive crowdsourcing. A promising research direction is to employ a sort of distant supervision, where training data (e.g., artificial utterance-denotations pairs) is artificially generated with given resources (e.g., a knowledge base, Wikipedia documents). In this work, we additionally train the weakly-supervised neural semantic parser with a distant supervision approach proposed by Reddy, Lapata, and Steedman (2014). In this setting, the given data is a corpus of entity-recognized sentences and a knowledge base. Utterance-denotation pairs are artificially created by replacing entity mentions in the sentences with variables. Then, the semantic parser is trained to predict the denotation for the variable that includes the mentioned entity. For example, given the declarative sentence *NVIDIA was founded by Jen-Hsun Huang and Chris Malachowsky*, the distant supervision approach creates the utterance *NVIDIA was founded by Jen-Hsun_Huang and _blank_* paired with the corresponding denotation *Chris Malachowsky*. In some cases, even stronger constraints can be applied. For example, if the mention is preceded by the word *the*, then the correct denotation includes exactly one entity. In sum, the approach converts the corpus of entity-recognized sentences into artificial utterance-denotation pairs on which the weakly supervised model described in Section 3.8.2 can be trained. We also aim to evaluate if this approach is helpful for practical question answering.
4. Experiments

In this section, we present our experimental setup for assessing the performance of the neural semantic parsing framework. We present the datasets on which our model was trained and tested, discuss implementation details, and finally report and analyze semantic parsing results.

4.1 Datasets

We evaluated our model on the following datasets which cover different domains and require different types of supervision.

**GeoQuery** (Zelle and Mooney 1996) contains 880 questions and database queries about US geography. The utterances are compositional, but the language is simple and vocabulary size small (698 entities and 24 relations). Model training on this dataset is fully supervised (Section 3.8.1).

**WebQuestions** (Berant et al. 2013b) contains 5,810 question-answer pairs. It is based on Freebase and the questions are not very compositional. However, they are real questions asked by people on the web.

**GraphQuestions** (Su et al. 2016) contains 5,166 question-answer pairs which were created by showing 500 Freebase graph queries to Amazon Mechanical Turk workers and asking them to paraphrase them into natural language. Model training on WebQuestions and GraphQuestions is weakly supervised (Section 3.8.2).

**Spades** (Bisk et al. 2016) contains 93,319 questions derived from CLUEWEB09 (Gabrilovich, Ringgaard, and Subramanya 2013) sentences. Specifically, the questions were created by randomly removing an entity, thus producing sentence-denotation pairs (Reddy, Lapata, and Steedman 2014). The sentences include two or more entities.
and although they are not very compositional, they constitute a large-scale dataset for neural network training with distant supervision (Section 3.8.3).

4.2 Implementation Details

**Shared Parameters.** Across training regimes, the dimensions of word vector, logical form token vector, and LSTM hidden state are 50, 50, and 150 respectively. Word embeddings were initialized with Glove embeddings (Pennington, Socher, and Manning 2014). All other embeddings were randomly initialized. We used one LSTM layer in forward and backward directions. Dropout was used on the combined feature representation of the buffer and the stack (Equation (25)), which computes the softmax activation of the next action or token. The dropout rate was set to 0.5. Finally, momentum SGD (Sutskever et al. 2013) was used as the optimization method to update the parameters of the model.

**Entity Resolution.** Amongst the four datasets described above, only GEOQUERY contains annotated logical forms which can be used to directly train a neural semantic parser. For the other three datasets, supervision is indirect via consistent logical forms validated on denotations (see Section 3.8.2). As mentioned earlier, we use entity linking to reduce the search space for consistent logical forms. Entity mentions in SPADES are automatically annotated with Freebase entities (Gabrilovich, Ringgaard, and Subramanya 2013). For WEBQUESTIONS and GRAPHQUESTIONS we perform entity linking following the procedure described in Reddy et al. (2016). We identify potential entity spans using seven handcrafted part-of-speech patterns and associate them with Freebase entities obtained from the Freebase/KG API (http://developers.google.com/freebase/). For each candidate entity span, we retrieve the top 10 entities according to the API. We treat each possibility as a candidate entity to construct candidate utterances with beam search of size 500, among which we look for the consistent logical forms.
**Discriminative Ranker.** For datasets which use denotations as supervision, our semantic parsing system additionally includes a discriminative ranker, whose role is to select the final logical form to execute from a list of candidates generated by the neural semantic parser. At test time, the generation process is accomplished by beam search with beam size 300. The ranker which is a log-linear model is trained with momentum SGD (Sutskever et al. 2013). As features, we consider the embedding cosine similarity between the utterance (excluding stop-words) and the logical form, the token overlap count between the two, and also similar features between the lemmatized utterance and the logical form. In addition, we include as features the embedding cosine similarity between the question words and the logical form, the similarity between the question words (e.g., what, who, where, whose, date, which, how many, count) and relations in the logical form, and the similarity between the question words and answer type as indicated by the last word in the Freebase relation (Xu et al. 2016). Finally, we add as a feature the length of the denotation given by the logical form (Berant et al. 2013a).

### 4.3 Results

In this section, we present the experimental results of our Transition-based Neural Semantic Parser (TNSP). We present various instantiations of our own model as well as comparisons against semantic parsers proposed in the literature.

Experimental results on GEOQUERY are shown in Table 6. The first block contains conventional statistical semantic parsers, previously proposed neural models are presented in the second block, whereas variants of TNSP are shown in the third block. Specifically we build various top-down and bottom-up TNSP models using the various types of attention introduced in Section 3.7. We report accuracy which is defined as the proportion of utterances which correctly parsed to their gold standard logical forms. Amongst TNSP models, a top-down system with structured (soft) attention performs
Table 6: Fully supervised experimental results on the GEOQUERY dataset. For Jia and Liang (2016), we include two of their results: one is a standard neural sequence to sequence model; and the other is the same model trained with a data augmentation algorithm on the labeled data (reported in parentheses).

<table>
<thead>
<tr>
<th>Models</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zettlemoyer and Collins (2005)</td>
<td>79.3</td>
</tr>
<tr>
<td>Zettlemoyer and Collins (2007)</td>
<td>86.1</td>
</tr>
<tr>
<td>Kwiatkowski et al. (2010)</td>
<td>87.9</td>
</tr>
<tr>
<td>Kwiatkowski et al. (2011)</td>
<td>88.6</td>
</tr>
<tr>
<td>Kwiatkowski et al. (2013)</td>
<td>88.0</td>
</tr>
<tr>
<td>Zhao and Huang (2015)</td>
<td>88.9</td>
</tr>
<tr>
<td>Liang, Jordan, and Klein (2011)</td>
<td>91.1</td>
</tr>
<tr>
<td>Dong and Lapata (2016)</td>
<td>84.6</td>
</tr>
<tr>
<td>Jia and Liang (2016)</td>
<td>85.0 (89.1)</td>
</tr>
<tr>
<td>Rabinovich, Stern, and Klein (2017)</td>
<td>87.1</td>
</tr>
<tr>
<td>TNSP, soft attention, top-down</td>
<td>86.8</td>
</tr>
<tr>
<td>TNSP, soft structured attention, top-down</td>
<td>87.1</td>
</tr>
<tr>
<td>TNSP, hard attention, top-down</td>
<td>85.3</td>
</tr>
<tr>
<td>TNSP, binomial hard attention, top-down</td>
<td>85.5</td>
</tr>
<tr>
<td>TNSP, soft attention, bottom-up</td>
<td>86.1</td>
</tr>
<tr>
<td>TNSP, soft structured attention, bottom-up</td>
<td>86.8</td>
</tr>
<tr>
<td>TNSP, hard attention, bottom-up</td>
<td>85.3</td>
</tr>
<tr>
<td>TNSP, binomial hard attention, bottom-up</td>
<td>85.3</td>
</tr>
</tbody>
</table>

best. Overall, we observe that differences between top-down and bottom-up systems are small; it is mostly the attention mechanism that affects performance, with hard attention performing worst and soft attention performing best for both top-down and bottom-up systems. TNSP outperforms previously proposed neural semantic parsers which treat semantic parsing as a sequence transduction problem and use LSTMs to map utterances to logical forms (Dong and Lapata 2016; Jia and Liang 2016). TNSP brings performance improvements over these systems when using comparable data sources for training. Jia and Liang (2016) achieve better results with synthetic data that expands GEOQUERY; we could adopt their approach to improve model performance, however, we leave this to future work. Our system is on the same par with the model of Rabinovich, Stern, and Klein (2017) who also output well-formed trees in a top-down manner using a decoder.
built of many submodels, each associated with a specific construct in the underlying grammar.

Results for the weakly supervised training scenario are shown in Table 7. For all Freebase related datasets we use average F1 \( (\text{Berant et al. 2013a}) \) as our evaluation metric. We report results on \textsc{WebQuestions} and \textsc{GraphQuestions} in Tables 7a and 7b, respectively. The first block in the tables groups conventional statistical semantic parsers, the second block presents related neural models, and the third block variants of TNSP. For fair comparison, we also built a baseline sequence-to-sequence model enhanced with an attention mechanism \( (\text{Dong and Lapata 2016}) \).

On \textsc{WebQuestions}, the best performing TNSP system generates logical forms based on top-down pre-order while employing soft attention. The same top-down system with structured attention performs closely. Again we observe that bottom-up preorder lags behind. In general, our semantic parser obtains performance on par with the best symbolic systems (see the first block in Table 7a). It is important to note that \text{Bast and Haussmann (2015)} develop a question answering system, which contrary to ours cannot produce meaning representations whereas \text{Berant and Liang (2015)} propose a sophisticated agenda-based parser which is trained borrowing ideas from imitation learning. \text{Reddy et al. (2016)} learn a semantic parser via intermediate representations which they generate based on the output of a dependency parser. TNSP performs competitively despite not having access to linguistically-informed syntactic structure. Regarding neural systems (see the second block in Table 7a), our model outperforms the sequence-to-sequence baseline and other related neural architectures using similar resources. \text{Xu et al. (2016)} represent the state of the art on \textsc{WebQuestions}. Their system uses Wikipedia to prune out erroneous candidate answers extracted from Freebase. Our model would also benefit from a similar post-processing.
With respect to GRAPHQUESTIONS, we report F1 for various TNSP models (third block in Table 7b), and conventional statistical semantic parsers (first block in Table 7b). The first three systems are presented in Su et al. (2016). Again, we observe that a top-down variant of TNSP with soft attention performs best. It is superior to the sequence-to-
sequence baseline and obtains performance comparable to Reddy et al. (2017) without making use of an external syntactic parser. The model of Dong et al. (2017) is state of the art on GRAPHQUESTIONS. Their method is trained end-to-end using questions-answer pairs as a supervision signal together with question paraphrases as a means of capturing different ways of expressing the same content. Importantly, their system is optimized with question-answering in mind, and does not produce logical forms.

When learning from denotations, a challenge concerns the handling of an exponentially large set of logical forms. In our approach, we rely on the neural semantic parser to generate a list of candidate logical forms by beam search. Ideally, we hope the beam size is large enough to include good logical forms which will be subsequently selected by the discriminative ranker. Figure 4 shows the effect of varying beam size on GRAPHQUESTIONS (development set) when training executes for two epochs using the TNSP soft attention model with top-down generation order. We report the number of utterances that are answerable (i.e., an utterance is considered answerable if the beam includes one or more good logical forms leading to the correct denotation) and the number of utterances that are correctly answered eventually. As the beam size increases, the gap between utterances that are answerable and those that are answered correctly becomes larger. And the curve for correctly answered utterances gradually plateaus and the performance does not improve. This indicates a trade-off between generating candidates that cover good logical forms and picking the best logical form for execution: when the beam size is large, there is a higher chance for good logical forms to be included but also for the discriminative ranker to make mistakes.

GRAPHQUESTIONS consists of four types of questions. As shown in Table 8, the first type are relational questions (denoted by relation). An example of a relational question is what periodic table block contains oxygen; the second type contains count
Figure 4: Fraction of utterances that are answerable versus those correctly predicted with varying beam size on the GRAPHQUESTIONS development set.

Table 8: Breakdown of questions answered by type for the GRAPHQUESTIONS.

<table>
<thead>
<tr>
<th>Question type</th>
<th>Number</th>
<th>% Answerable</th>
<th>% Correctly answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation</td>
<td>1938</td>
<td>0.499</td>
<td>0.213</td>
</tr>
<tr>
<td>count</td>
<td>309</td>
<td>0.421</td>
<td>0.032</td>
</tr>
<tr>
<td>aggregation</td>
<td>226</td>
<td>0.363</td>
<td>0.075</td>
</tr>
<tr>
<td>filter</td>
<td>135</td>
<td>0.459</td>
<td>0.096</td>
</tr>
<tr>
<td>All</td>
<td>2,608</td>
<td>0.476</td>
<td>0.173</td>
</tr>
</tbody>
</table>

questions (denoted by count). An example is *how many firefighters does the new york city fire department have*; the third type includes aggregation questions requiring argmax or argmin (denoted by aggregation). An example is *what human stampede injured the most people*; the last type are filter questions which requires comparisons by $>$, $\geq$, $<$ and $\leq$ (denoted by filter). An example is *which presidents of the united states weigh not less than 80.0 kg*. Table 8 shows the number of questions broken down by type, as well as the proportion of answerable and correctly answered questions. As the results reveal, relation questions are the simplest to answer which is expected since relation questions are non-compositional and their logical forms are easy to find by
beam search. The remaining types of questions are rather difficult to answer: although the system is able to discover logical forms that lead to the correct denotation during beam search, the ranker is not able to identify the right logical forms to execute. Aside from the compositional nature of these questions which makes them hard to answer, another difficulty is that such questions are a minority in the dataset posing a learning challenge for the ranker to identify them. As future work, we plan to train separate rankers for different question types.

Finally, Table 9 presents experimental results on SPADES which serves as a testbed for our distant supervision setting. Previous work on this dataset has used a semantic parsing framework where natural language is converted to an intermediate syntactic representation and then grounded to Freebase. Specifically, Bisk et al. (2016) evaluate the effectiveness of four different CCG parsers on the semantic parsing task when varying the amount of supervision required. As can be seen, TNSP outperforms all CCG variants (from unsupervised to fully supervised) without having access to any manually

<table>
<thead>
<tr>
<th>Models</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupervised CCG (Bisk et al. 2016)</td>
<td>24.8</td>
</tr>
<tr>
<td>Semi-supervised CCG (Bisk et al. 2016)</td>
<td>28.4</td>
</tr>
<tr>
<td>Supervised CCG (Bisk et al. 2016)</td>
<td>30.9</td>
</tr>
<tr>
<td>Rule-based system (Bisk et al. 2016)</td>
<td>31.4</td>
</tr>
<tr>
<td>Sequence-to-sequence</td>
<td>28.6</td>
</tr>
<tr>
<td>TNSP, soft attention, top-down</td>
<td>32.4</td>
</tr>
<tr>
<td>TNSP, soft structured attention, top-down</td>
<td>32.1</td>
</tr>
<tr>
<td>TNSP, hard attention, top-down</td>
<td>31.5</td>
</tr>
<tr>
<td>TNSP, binomial hard attention, top-down</td>
<td>29.8</td>
</tr>
<tr>
<td>TNSP, soft attention, bottom-up</td>
<td>32.1</td>
</tr>
<tr>
<td>TNSP, soft structured attention, bottom-up</td>
<td>31.4</td>
</tr>
<tr>
<td>TNSP, hard attention, bottom-up</td>
<td>30.7</td>
</tr>
<tr>
<td>TNSP, binomial hard attention, bottom-up</td>
<td>30.4</td>
</tr>
</tbody>
</table>
annotated derivations or lexicons. Again, we observe that a top-down TNSP system with
soft attention performs best and is superior to the sequence-to-sequence baseline.

The results on SPADES hold promise for scaling semantic parsing by using distant
supervision. In fact, artificial data could potentially help improve weakly supervised
question answering models trained on utterance-denotation pairs. To this end, we
use the entity-masked declarative sentences paired with their denotations in SPADES
as additional training data for GRAPHQUESTIONS. We train the neural semantic
parser with the combined training data and evaluate on the GRAPHQUESTIONS.
We use the top-down, soft-attention TNSP model with a beam search size of 300.
During each epoch of training, the model was first trained with a mixture of the
additional SPADES data and the original training data. Figure 5 shows the fraction of
answerable and correctly answered questions generated by the neural semantic parser
on GRAPHQUESTIONS. Note that the original GRAPHQUESTIONS training set consists
of 1,794 examples and we report numbers when different amount of SPADES training
data is used.

As the figure shows, using artificially training data is able to improve the neural
semantic parser on a question answering task to some extent. This suggests that distant
supervision is a promising direction for building practical semantic parsing systems.
Since artificial training data can be abundantly generated to fit a neural parser, the
approach can be used for data argumentation when question-answer pairs are limited.

However, we observe that the maximum gain occurs when 1,000 extra training
examples are used, a size comparable to the original training set. After that no further
improvements are made when more training examples are used. We hypothesize
this is due to the disparities between utterance-denotation pairs created in distant
supervision and utterance-denotation pairs gathered from real users. For example,
given the declarative sentence *NVIDIA was founded by Jen-Hsun Huang and Chris Malachowsky*, the distant supervision approach creates the utterance *NVIDIA was founded by Jen-Hsun_Huang and _blank_ and the corresponding denotation *Chris Malachowsky*. However, the actual question users may ask is *Who founded NVIDIA together with Jen-Hsun_Huang*. This poses a challenge if the neural network is trained on one type of utterance and tested on another. We observe that the distribution mismatch outweighs the addition of artificial data quickly. Future work will focus on how to alleviate this problem by generating more realistic data with an advanced question generation module.

Another factor limiting performance is that *SPADES* mainly consists of relational questions without high-level predicates, such as *count, filter* and *aggregation* which substantially harder to answer correctly (see Table 8).
Table 10: Hard attention and structure attention when predicting the relation in each question. The corresponding logical predicate is shown in brackets.

To summarize, across experiments and training regimes, we observe that TNSP performs competitively while producing meaningful and well-formed logical forms. One characteristic of the neural semantic parser is that it generates tree-structured representations in an arbitrarily canonical order, as a sequence of transition operations. We investigated two such orders, top-down pre-order and bottom-up post-order. Experimentally, we observed that pre-order generation provides marginal benefits over post-order generation. One reason for this is that compared to sibling information which the bottom-up system uses, parent information used by the top-down system is more important for subtree prediction.

We explored three attention mechanisms in our work, including soft attention, hard attention, and structured attention. Quantitatively, we observe that soft attention
always outperforms hard attention in all three training setups. This can be attributed to the differentiability of the soft attention layer. The structured attention layer is also differentiable since it computes the marginal probability of each token being selected with a dynamic programming procedure. We observe that on GEOQUERY which represents the fully supervised setting, structured attention offers marginal gains over soft attention. But in other datasets where logical forms are not given, the more structurally aware attention mechanism does not improve over soft attention, possibly due to the weaker supervision signal. However, it should be noted that the structured attention layer at each decoding step requires the forward-backward algorithm, which has time complexity $O(2n^2)$ (where $n$ denotes the utterance length) and therefore much slower than soft attention which has linear ($O(n)$) complexity.

An advantage of hard and structured attention is that it allows us to inspect which natural language tokens are being selected when predicting a relation or entity in the logical form. For hard attention, the selection boils down to a token sampling procedure; whereas for structured attention, the tokens selected can be interpreted with the Viterbi algorithm which assigns the most likely label for each token. Table 10 shows examples of hard and structured attention when predicting the key relational logical predicate. These examples were selected from GRAPHQUESTIONS using the top-down TNSP system. The table contains both meaningful token selections (where the selected tokens denote an informative relation) and non-meaningful ones.

5. Conclusions

In this paper, we described a general neural semantic parsing framework which operates with functional query language and generates tree-structured logical forms with transition-based neural networks. To tackle mismatches between natural language and logical form tokens, we introduced various attention mechanisms in the generation
process. We also considered different training regimes, including fully supervised training where annotated logical forms are given, weakly-supervised training when denotations are provided, and distant supervision where only unlabeled sentences and a knowledge base are available. Compared to previous neural semantic parsers, our model generates well-formed logical forms, and is more interpretable — hard and structured attention can be used to inspect what the model has learned.

When the training data consists of utterance-denotation pairs, we employ a generative parser-discriminative ranker framework: the role of the parser is to (beam) search for candidate logical forms, which are subsequently re-scored by the ranker. This is in contrast to recent work (Neelakantan et al. 2017) on weakly-supervised neural semantic parsing, where the parser is directly trained by reinforcement learning using denotations as reward. Advantageously, our framework employs beam search (in contrast to greedy decoding) to increase the likelihood of discovering correct logical forms in a candidate set. Meanwhile, the discriminative ranker is able to leverage global features on utterance-logical form-denotation triplets to score logical forms. In future, we will compare the presented parser-ranker framework with reinforcement learning-based parsers.

Directions for future work are many and varied. Since the current semantic parser generates tree structured logical forms conditioned on an input utterance, we could additionally exploit input information beyond sequences such as dependency tree representations, resembling a tree-to-tree transduction model. To tackle long-term dependencies in the generation process, an intra-attention mechanism could be used (Cheng, Dong, and Lapata 2016; Vaswani et al. 2017). Secondly, when learning from denotations, it is possible that the beam search output contains spurious logical forms which lead to correct answers accidentally but do not represent the actual meaning of an
utterance. Such logical forms are misleading training signals and should be removed, e.g., with a generative neural network component (Cheng, Lopez, and Lapata 2017) which scores how well a logical form represents the utterance semantics. Last but not least, since our semantic parsing framework provides a decomposition between domain-generic tree generation and the selection of domain-specific constants, we would like to further explore training the semantic parser in a multi-domain setup (Herzig and Berant 2017), where the domain-generic parameters are shared.

References


Cai, Jonathon, Richard Shin, and Dawn Song. 2017. Making neural programming...
architectures generalize via recursion.


Clarke, James, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world’s response. In Proceedings of the 14th Conference on Computational Natural Language Learning, pages 18–27, Uppsala, Sweden.


Gabrilovich, Evgeniy, Michael Ringgaard, and Amarnag Subramanya. 2013. FACC1: Freebase annotation of ClueWeb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0).


Liu, Yang and Mirella Lapata. 2018. Learning structured text representations.
Cheng, Reddy, Saraswat and Lapata

Learning an Executable Neural Semantic Parser

Transactions of the Association for Computational Linguistics, pages 63–75.


Neelakantan, Arvind, Quoc V Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2017. Learning a natural language interface with neural programmer. 5th International Conference on Learning Representations.


Reed, Scott and Nando De Freitas. 2016. Neural programmer-interpreters. 4th International Conference on Learning Representations.


Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia


