Learning an Executable Neural Semantic Parser

Jianpeng Cheng
University of Edinburgh
jianpeng.cheng@ed.ac.uk

Siva Reddy
Stanford University
sivar@stanford.edu

Vijay Saraswat
IBM T.J. Watson Research
vjsaraswat@stanford.edu

Mirella Lapata
University of Edinburgh
mlap@inf.ed.ac.uk

This article describes a neural semantic parser that maps natural language utterances onto logical forms that 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, combining 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 data sets 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 toward achieving this goal. Semantic parsers specify a mapping between natural language utterances and machine-understandable meaning

Submission received: 6 November 2017; revised version received: 17 July 2018; accepted for publication: 10 August 2018.

doi:10.1162/COLI_a_00342

Table 1
Examples of questions, corresponding logical forms, and their answers.

Environment: A database of US geography
Utterance: What is the longest river in Ohio?
Logical form: `longest(and(type.river, location(Ohio)))`
Denotation: Ohio River

Environment: Freebase
Utterance: How many daughters does Obama have?
Logical form: `count(daughterOf(Barack Obama))`
Denotation: 2

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 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 because of 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; Kwiatkowksi et al. 2010) mostly require 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—for example, training on utterance-denotation pairs (Clarke et al. 2010; Kwiatkowski et al. 2013; Liang 2016), 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 used to parse a sentence in polynomial time.

The successful application of recurrent neural networks (Sutskever, Vinyals, and Le 2014; Bahdanau, Cho, and Bengio 2015) to a variety of NLP tasks has provided 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 because 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 that 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. First, 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. Second, 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 because 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) data set, *weak supervision* with question-answer pairs using the WEBQUESTIONS (Berant et al. 2013a) and GRAPHQUESTIONS (Su et al. 2016) data sets, and *distant supervision* without question-answer pairs, using the SPADES (Bisk et al. 2016) data set. 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 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 here.

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 that 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) because of 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, because 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 (Neelakantan, Le, and Sutskever 2016; Reed and De Freitas 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 (Iyer et al. 2017; Yin and Neubig 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 a support vector machine (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, because 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 set-up 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) have developed neural semantic parsers that treat semantic parsing as a sequence to a sequence learning problem. Jia and Liang (2016) also introduce a data augmentation approach that 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 that 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, but following a transition-based generation approach (Dyer et al. 2015,

2016). Our semantic parser is a generalization of the model presented in Cheng et al. (2017). Whereas 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 in statistical semantic parsing has mostly used annotated training data consisting of utterances paired with logical forms (Zelle and Mooney 1996; Kate, Wong, and Mooney 2005; Kate and Mooney 2006; Wong and Mooney 2006; Lu et al. 2008; Kwiatkowksi et al. 2010). The 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 that 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 crowdsourcing. For this reason, semantic parsing can be scaled to handle large, complex, and open domain problems. Examples include learning 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 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 are generated from the given resources. For example, Cai and Yates (2013) generate utterances paired with logical forms. Their approach searches for sentences containing certain entity pairs, and assumes (with some pruning technique) that the sentences express a certain relation from the knowledge base. In Krishnamurthy and Mitchell (2012, 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 with traditional semantic parsers, our framework reduces the number 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 the 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 a distant-supervision setting 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 that 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 $\mathcal{K}$ refer 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

$$[\![r(e_1)]\!] = \{e : (e_1, e) \in [\![r]\!]\} \tag{3}$$

  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:

$$[\![\texttt{count}(u)]\!] = \{|[\![u]\!]|\} \tag{4}$$

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:

$$[\![\texttt{argmax}(u, r)]\!] = \{e : e \in u \cap \forall e' \in u, r(e) \geq r(e')\} \tag{5}$$

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 $(=, !=, >, <, \geq, \leq)$ acting on the relation r is satisfied:

$$[\![\texttt{filter}_{>}(u, r, v)]\!] = \{e : e \in u \cap r(e) > v\} \tag{6}$$

For example, the query `filter`$_{>}$ `(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:

$$[\![\texttt{and}(u_1, u_2)]\!] = [\![u_1]\!] \cap [\![u_2]\!] \tag{7}$$

and `or` takes their union:

$$[\![\texttt{or}(u_1, u_2)]\!] = [\![u_1]\!] \cup [\![u_2]\!] \tag{8}$$

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., `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 that 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 that 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 $[(c_0, t_0), (c_1, t_1), \cdots, (c_m, t_m)]$. 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, π is the non-terminal token to be generated, σ is the terminal token to be generated, N is a stack of open non-terminals, and P is

Table 2
Transitions for the top–down and bottom–up generation system. Stack $\sum$ is represented as a list with its head to the right (with tail σ), same for stack N (with tail β).

	Top–down Transitions
NT(X)	$([\sigma\|\mathtt{X}'], \mathtt{X}, \varepsilon, [\beta\|\mathtt{X}'], P(\mathtt{X}')) \Rightarrow ([\sigma\|\mathtt{X}', \mathtt{X}], \varepsilon, \varepsilon, [\beta\|\mathtt{X}', \mathtt{X}], P(\mathtt{X}))$
TER(x)	$([\sigma\|\mathtt{X}'], \varepsilon, \mathtt{x}, [\beta\|\mathtt{X}'], \mathtt{P(X')}) \Rightarrow ([\sigma\|\mathtt{X}', \mathtt{x}], \varepsilon, \varepsilon, [\beta\|\mathtt{X}', \mathtt{x}], P(\mathtt{X}'))$
RED	$([\sigma\|\mathtt{X}', \mathtt{X}, \mathtt{x}], \varepsilon, \varepsilon, [\beta\|\mathtt{X}', \mathtt{X}], P(\mathtt{X})) \Rightarrow ([\sigma\|\mathtt{X}', \mathtt{X(x)}], \varepsilon, \varepsilon, [\beta\|\mathtt{X}'], P(\mathtt{X}'))$

	Bottom–up Transitions
TER(x)	$(\sigma, \varepsilon, \mathtt{x}) \Rightarrow ([\sigma\|\mathtt{x}], \varepsilon, \varepsilon)$
NT-RED(X)	$([\sigma\|\mathtt{x}], \mathtt{X}, \varepsilon) \Rightarrow ([\sigma\|\mathtt{X(x)}], \varepsilon, \varepsilon)$

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 $c_0 = ([], TOP, \varepsilon, [], \perp)$, where TOP stands for the root node of the tree, ε represents an empty string, and $\perp$ represents an unspecified function. The top–down system uses 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 (and 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, π is the token non-terminal to be generated, and σ 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 ε 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 `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 `count` takes one argument and `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, and 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 earlier with FunQL, whose grammar determines the space of allowed terminal and non-terminal symbols:

- `NT(X)` includes an operation that generates relations `NT(relation)`, and other domain-general operators in FunQL: `NT(and)`, `NT(or)`, `NT(count)`, `NT(argmax)`, `NT(argmin)`, and `NT(filter)`. Note that `NT(relation)` creates a placeholder for a relation, which is subsequently generated.
- `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 used by our parser in order to generate the logical form `count(and(daughterOf(Barack Obama), 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?*".

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

Table 3
Top–down generation of the logical form `count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))`. Elements on the stack are separated by || and the top of the stack is on the right.

Operation	Logical form token	Stack
NT(count)	count	count(
NT(and)	and	count(|| and(
NT(relation)	daughterOf	count(|| and(|| daughterOf
TER(entity)	Barack Obama	count(|| and(|| daughterOf(|| Barack Obama
RED		count(|| and(|| daughterOf(Barack Obama)
NT(relation)	InfluentialTeensByYear	count(|| and(|| daughterOf(Barack Obama) || InfluentialTeensByYear(
TER(entity)	2014	count(|| and(|| daughterOf(Barack Obama) || InfluentialTeensByYear(|| 2014
RED		count(|| and(|| daughterOf(Barack Obama) || InfluentialTeensByYear(2014)
RED		count(|| and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))
RED		count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014)))

Table 4
Bottom–up generation of the logical form `count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))`. Elements on the stack are separated by || and the top of the stack is on the right.

Operation	Logical form token	Stack
TER(entity)	Barack Obama	Barack Obama
NT-RED(relation)	daughterOf	daughterOf(Barack Obama)
TER(entity)	2014	daughterOf(Barack Obama) || 2014
NT-RED(relation)	InfluentialTeensByYear	daughterOf(Barack Obama) || InfluentialTeensByYear(2014)
NT-RED(and)	and	and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))
NT-RED(count)	count	count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014)))

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 `TER`s 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 constraint 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 this logical form generation algorithm with a structured neural network that 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 long short-term memory (LSTM) architecture (Hochreiter and Schmidhuber 1997). A bidirectional LSTM comprises 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 $\vec{c_t}$ and the hidden state $\vec{h_t}$ are updated with the following equations:

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ \hat{c}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \cdot [\overrightarrow{h_{t-1}}, x_t] \tag{9}$$

$$\overrightarrow{c_t} = f_t \odot \overrightarrow{c_{t-1}} + i_t \odot \hat{c}_t \tag{10}$$

$$\overrightarrow{h_t} = o_t \odot \tanh(\overrightarrow{c_t}) \tag{11}$$

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

$$\overrightarrow{h_t} = \overrightarrow{\text{LSTM}}(x_t, \overrightarrow{h_{t-1}}) \tag{12}$$

After encoding, a list of token representations $[\overrightarrow{h_1}, \overrightarrow{h_2}, \cdots, \overrightarrow{h_n}]$ within the forward context is obtained. Similarly, the backward LSTM computes a list of token representations $[\overleftarrow{h_1}, \overleftarrow{h_2}, \cdots, \overleftarrow{h_n}]$ within the backward context as

$$\overleftarrow{h_t} = \overleftarrow{\text{LSTM}}(x_t, \overleftarrow{h_{t+1}}) \tag{13}$$

Finally, each input token x_i is represented by the concatenation of its forward and backward LSTM state vectors, denoted by $h_i = \overrightarrow{h_i} : \overleftarrow{h_i}$. 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 because its alignment to logical form tokens is not predetermined in the general semantic parsing scenario. We denote the buffer b as $b = [h_1, \cdots, h_k]$, where k denotes the length of the utterance.

Generation History Encoding. The generation history—that is, the partially completed subtrees—is encoded with a variant of stack-LSTM (Dyer et al. 2015). 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

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

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 $s_{t-1:t}$.[1] The representation of the completed subtree u is then computed as

$$u = W_u \cdot [p_u : c_u] \tag{15}$$

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 methods, such as a recurrent neural network (Kuncoro et al. 2017). Finally, the subtree embedding u serves as the input to the LSTM and updates $s_{t-1:t}$ to s_t as

$$s_t = \text{LSTM}(u, s_{t-1:t}) \tag{16}$$

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:

$$s_t = \text{LSTM}(y_t, s_{t-1}) \tag{17}$$

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

1 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 subsequently in the text).

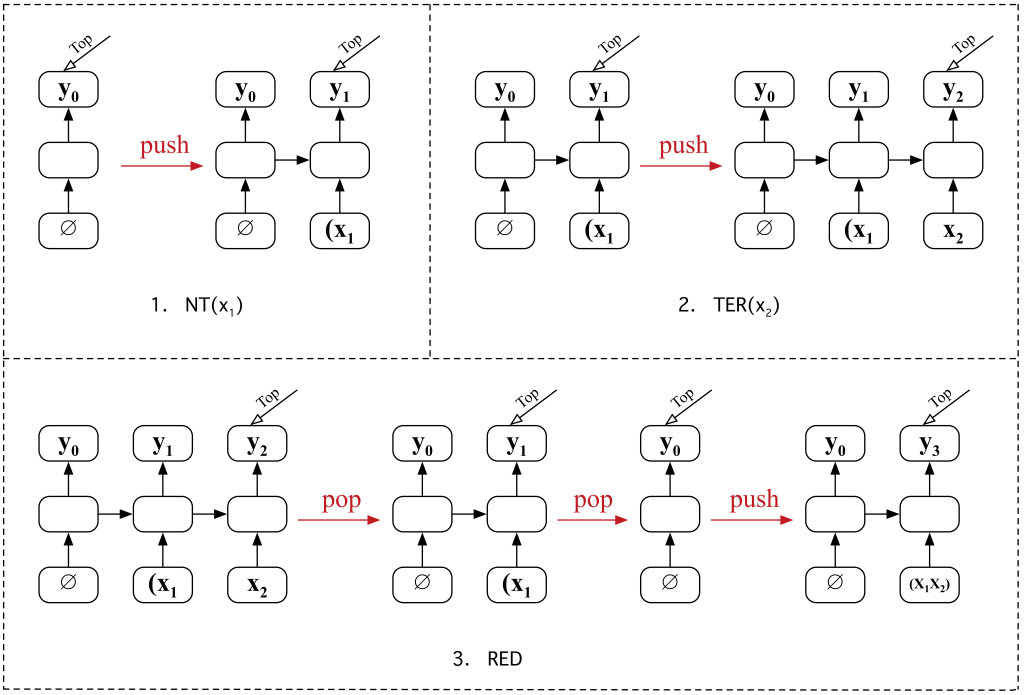

Figure 1
A stack-LSTM extends a standard LSTM with the addition of a stack pointer (shown as Top in the figure). The example shows how the configuration of the stack changes when the operations `NT`, `TER`, and `RED` are applied in sequence. The initial stack is presumed empty for illustration purposes. 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.

intermediate state $s_{t-1:t}$. Similar to the top–down system, we compute the representation of the newly combined subtree u as

$$u = W_u \cdot [p_u : c_u] \tag{18}$$

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

$$s_t = \text{LSTM}(u, s_{t-1:t}) \tag{19}$$

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

tokens (see Tables 3 and 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:

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

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(relation)`, `NT-RED(relation)`, `TER(relation)`, and `TER(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:

$$y_{t+1} \sim g(b, s_t) \tag{21}$$

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 that 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 because 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:

$$u_t^i = V \tanh(W_b b_i + W_s s_t) \tag{22}$$

$$\alpha_t^i = \text{softmax}(u_t^i) \tag{23}$$

$$\bar{b}_t = \sum_i \alpha_t^i b_i \tag{24}$$

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, softmax is taken to obtain the parameters of the multinomial distribution over actions:

$$a_{t+1} \sim \text{softmax}(W_{oa} \tanh(W_f[\bar{b}_t, s_t])) \tag{25}$$

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 `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 first mapping the utterance to a domain-general logical form that 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 2.

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:

$$u_t^i = V \tanh(W_b b_i + W_s s_t) \tag{26}$$

$$\alpha_t^i = \text{softmax}(u_t^i) \tag{27}$$

$$\bar{b}_t = \sum_i \alpha_t^i b_i \tag{28}$$

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[\bar{b}_t, s_t])) \tag{29}$$

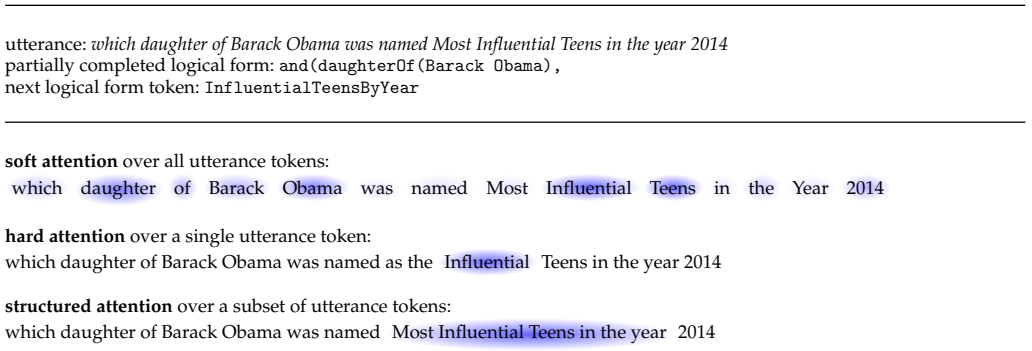

Figure 2
Different attention mechanisms for predicting the next logical form token. The example utterance is *which daughter of Barack Obama was named Most Influential Teens in the year 2014?* and the corresponding logical form to be generated is `and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))`. The figure shows attention for predicting `InfluentialTeensByYear`. Darker shading indicates higher values.

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 (Lafferty, Mccallum, and Pereira 2001). Assume that at time step t each token in the buffer (e.g., the ith token) is assigned an attention label $A_t^i \in \{0, 1\}$. The conditional random field defines $p(A_t)$, the probability of the sequence of attention labels at time step t as

$$p(A_t) = \frac{\exp \sum_i W_f \cdot \psi(A_t^{i-1}, A_t^i, b_i, s_t)}{\sum_{A_t^1, \cdots, A_t^n} \exp \sum_i W_f \cdot \psi(A_t^{i-1}, A_t^i, b_i, s_t)} \tag{30}$$

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

$$u_t^i \cdot a_t^i \tag{31}$$

where u_t^i is the token-specific attention score computed in Equation (26); the transition feature

$$A_t^{i-1} \cdot A_t^i \tag{32}$$

and the context-dependent transition feature

$$u_t^i \cdot A_t^{i-1} \cdot A_t^i \tag{33}$$

The marginal probability $p(A_t^i = 1)$ 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 3. To compare with standard soft attention, we denote this procedure as

$$\alpha_t^i = \text{forward-backward}(u_t^i) \tag{34}$$

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

$$\bar{b}_t = \sum_i \alpha_t^i b_i \tag{35}$$

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

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[\bar{b}_t, s_t])) \tag{36}$$

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 $u_t^i = V \tanh(W_b b_i + W_s s_t)$. The logit will be used to compute the first and third feature in ψ.
2. Forward algorithm: Initialize $\beta(A_t^1) = 1$.
   For $i \in \{2 \cdots n\}$, $A_t^i \in \{0,1\} : \beta(A_t^i) = \sum_{A_t^{i-1} \in \{0,1\}} \beta(A_t^{i-1}) \times \psi(A_t^{i-1}, A_t^i, b_i, s_t)$,
   where ψ is the context-dependent feature vector.
3. Backward algorithm: Initialize $\gamma(A_t^n) = 1$.
   For $i \in \{1 \cdots (n-1)\}$, $A_t^i \in \{0,1\}$:
   $\gamma(A_t^i) = \sum_{A_t^{i+1} \in \{0,1\}} \gamma(A_t^{i+1}) \times \psi(A_t^i, A_t^{i+1}, b_i, s_t)$, where ψ is the
   context-dependent feature vector.
4. Compute the marginal probability α_t^i of each input token b_i:
   $\alpha_t^i = \beta(A_t^i) \times \gamma(A_t^i)$
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 3
The structured attention model for token prediction.

The structured attention layer is soft and fully differentiable and allows us to model attention over phrases because 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:

$$u_t^i = V \tanh(W_b b_i + W_s s_t) \tag{37}$$

$$x_t \sim \text{softmax}(u_t^i) \tag{38}$$

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

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[b_t, s_t])) \tag{39}$$

Hard attention is nevertheless optimization-wise challenging; it requires sampling symbols (i.e., non-differentiable representations) inside an end-to-end module that 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 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 ith 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

$$p(A_t^i = 1) = \text{logistic}(u_t^i) \tag{40}$$

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

$$\bar{b}_t = \frac{1}{\sum_i A_t^i} \sum_i A_t^i b_i \tag{41}$$

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

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[\bar{b}_t, s_t])) \tag{42}$$

3.8 Model Training

We now discuss how our neural semantic parser can be trained under different conditions, that is, 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:

$$\mathcal{L} = \sum_{(x,l)\in\mathcal{T}} \log p(l|x) \tag{43}$$

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

$$\log p(l|x) = \log p(a|x) + \log p(y|x,a) \tag{44}$$

Soft Attention. Our objective consists of two terms, one for the action sequence,

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

and one for the logical form token sequence,

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

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

Table 5
Example data for various semantic parsing training regimes.

Full supervision: utterance-logical form pairs
utterance: *which daughter of Barack Obama was named Most Influential Teens in the year 2014?*
logical form: `and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))`

Weak supervision: utterance-denotation pairs
utterance: *which daughter of Barack Obama was named Most Influential Teens in the year 2014?*
denotation: `Malia Obama`

Distant supervision: entity-masked utterances
utterance: *Malia Obama, the daughter of Barack Obama, was named Most Influential Teens in the year 2014.*
artificial utterance: *_blank_, the daughter of Barack Obama, was named Most Influential Teens in the year 2014.*
denotation: `Malia Obama`

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 $\log p(y|x,a)$:

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

The gradient of $\mathcal{L}_y$ with respect to model parameters θ is given by

$$\begin{aligned}\frac{\partial \mathcal{L}_y}{\partial \theta} &= \sum_{(x,l)\in\mathcal{T}} \sum_{u} p(u|x,a) \frac{\partial \log p(y|u,x,a)}{\partial \theta} + \log p(y|u,x,a) \frac{\partial p(u|x,a)}{\partial \theta} \\ &= \sum_{(x,l)\in\mathcal{T}} \sum_{u} p(u|x,a) \frac{\partial \log p(y|u,x,a)}{\partial \theta} + \log p(y|u,x,a) \frac{\partial \log p(u|x,a)}{\partial \theta} p(u|x,a) \\ &= \sum_{(x,l)\in\mathcal{T}} \sum_{u} p(u|x,a) \left[\frac{\partial \log p(y|u,x,a)}{\partial \theta} + \log p(y|u,x,a) \frac{\partial \log p(u|x,a)}{\partial \theta}\right] \\ &\approx \sum_{(x,l)\in\mathcal{T}} \frac{1}{N} \sum_{k=1}^{K} \left[\frac{\partial \log p(y|u^k,x,a)}{\partial \theta} + \log p(y|u^k,x,a) \frac{\partial \log p(u^k|x,a)}{\partial \theta}\right]\end{aligned} \tag{48}$$

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

$$\frac{\partial \mathcal{L}_y}{\partial \theta} = \sum_{(x,l)\in\mathcal{T}} \frac{1}{N} \sum_{k=1}^{K} \left[\frac{\partial \log p(y|u^k,x,a)}{\partial \theta} + (\log p(y|u^k,x,a) - b) \frac{\partial \log p(u^k|x,a)}{\partial \theta}\right] \tag{49}$$

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

2 In standard hard attention, the choice is a single token in the sentence, whereas in binomial hard attention it is a phrase.

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 are 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 first 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 because the decoder does not have access to gold standard logical forms for backpropagation. Moreover, the neural decoder is a conditional generative model that generates logical forms in a greedy fashion and therefore lacks the ability to make global judgments of logical forms. To this end, we follow the conventional set-up 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:

$$\log p(d|x) = \log \sum_{l \in L} p(l|x) p(d|x, l) \tag{50}$$

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

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

$$p(l|x) = \frac{\exp(\phi(x, l)\theta)}{\sum_{l' \in L} \exp(\phi(x, l')\theta)} \tag{51}$$

where L is the set of candidate logical forms; ϕ is the feature function that maps an utterance-logical form pair (and also the corresponding denotation) into a feature vector; and θ 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 that 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 use a sort of distant supervision, where training data (e.g., artificial utterance-denotations pairs) are 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 whether this approach is helpful for practical question answering.

4. Experiments

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

4.1 Data Sets

We evaluated our model on the following data sets, 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 data set 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 that 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 data set 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. Among the four data sets, only GEOQUERY contains annotated logical forms that can be used to directly train a neural semantic parser. For the other three data sets, 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 data sets that 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, and 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 that correctly parsed to their gold standard logical forms. Among TNSP models, a top–down system with structured (soft) attention performs 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 that 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

Table 6
Fully supervised experimental results on the GEOQUERY data set. 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).

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

performance improvements over these systems when using comparable data sources for training. Jia and Liang (2016) achieve better results with synthetic data that expand GEOQUERY; we could adopt their approach to improve model performance, however, we leave this to future work. Our system is on 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 data sets we use average F1 (Berant et al. 2013a) as our evaluation metric. We report results on WEBQUESTIONS (Table 7a) and GRAPHQUESTIONS (Table 7b). 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 (Dong and Lapata 2016).

On WEBQUESTIONS, the best performing TNSP system generates logical forms based on top–down pre-order while using 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 Bast and Haussmann (2015) develop a question-answering system, which, contrary to ours, cannot produce meaning representations, whereas Berant and Liang (2015) propose a sophisticated agenda-based parser that is trained borrowing ideas from imitation learning. Reddy et al. (2016) learn a semantic parser via intermediate representations that 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. Xu et al. (2016) represent the state of the art on 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 7), 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 question-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 that 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

Table 7
Weakly supervised experimental results on two data sets. Results with additional resources are shown in parentheses.

(a) WEBQUESTIONS

Models	F1
SEMPRE (Berant et al. 2013b)	35.7
JACANA (Yao and Van Durme 2014)	33.0
PARASEMPRE (Berant and Liang 2014)	39.9
AQQU (Bast and Haussmann 2015)	49.4
AGENDAIL (Berant and Liang 2015)	49.7
DEPLAMBDA (Reddy et al. 2016)	50.3
SUBGRAPH (Bordes, Chopra, and Weston 2014)	39.2
MCCNN (Dong et al. 2015)	40.8
STAGG (Yih et al. 2015)	52.5
MCNN (Xu et al. 2016)	53.3
Sequence-to-sequence	48.3
TNSP, soft attention, top–down	50.1
TNSP, soft structured attention, top–down	49.8
TNSP, hard attention, top–down	49.4
TNSP, binomial hard attention, top–down	48.7
TNSP, soft attention, bottom–up	49.6
TNSP, soft structured attention, bottom–up	49.5
TNSP, hard attention, bottom–up	48.4
TNSP, binomial hard attention, bottom–up	48.7

(b) GRAPHQUESTIONS

Models	F1
sempre (Berant et al. 2013b)	10.8
parasempre (Berant and Liang 2014)	12.8
jacana (Yao and Van Durme 2014)	5.1
SimpleGraph (Reddy et al. 2016)	15.9
UDepLambda (Reddy et al. 2017)	17.6
Sequence-to-sequence	16.2
PARA4QA (Dong et al. 2017)	20.4
TNSP, soft attention, top–down	17.3
TNSP, soft structured attention, top–down	17.1
TNSP, hard attention, top–down	16.2
TNSP, binomial hard attention, top–down	16.4
TNSP, soft attention, bottom–up	16.9
TNSP, soft structured attention, bottom–up	17.1
TNSP, hard attention, bottom–up	16.8
TNSP, binomial hard attention, bottom–up	16.5

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

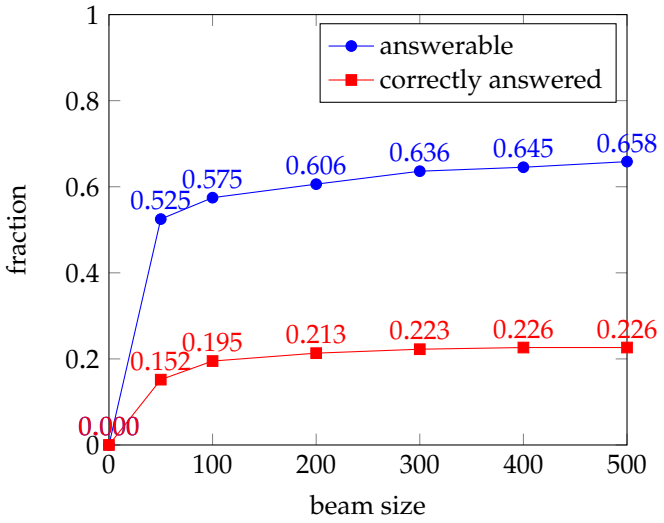

Figure 4
Fraction of utterances that are answerable versus those correctly predicted with varying beam size on the GRAPHQUESTIONS development set.

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 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 that require 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

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

Question type	Number	% Answerable	% Correctly answered
`relation`	1,938	0.499	0.213
`count`	309	0.421	0.032
`aggregation`	226	0.363	0.075
`filter`	135	0.459	0.096
All	2,608	0.476	0.173

Table 9
Distantly supervised experimental results on the SPADES data set.

Models	F1
Unsupervised CCG (Bisk et al. 2016)	24.8
Semi-supervised CCG (Bisk et al. 2016)	28.4
Supervised CCG (Bisk et al. 2016)	30.9
Rule-based system (Bisk et al. 2016)	31.4
Sequence-to-sequence	28.6
TNSP, soft attention, top–down	32.4
TNSP, soft structured attention, top–down	32.1
TNSP, hard attention, top–down	31.5
TNSP, binomial hard attention, top–down	29.8
TNSP, soft attention, bottom–up	32.1
TNSP, soft structured attention, bottom–up	31.4
TNSP, hard attention, bottom–up	30.7
TNSP, binomial hard attention, bottom–up	30.4

from the compositional nature of these questions, which makes them hard to answer, another difficulty is that such questions are a minority in the data set, 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 data set 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 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 amounts of SPADES training data is used.

As the figure shows, using artificial training data allows us 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. Because 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.

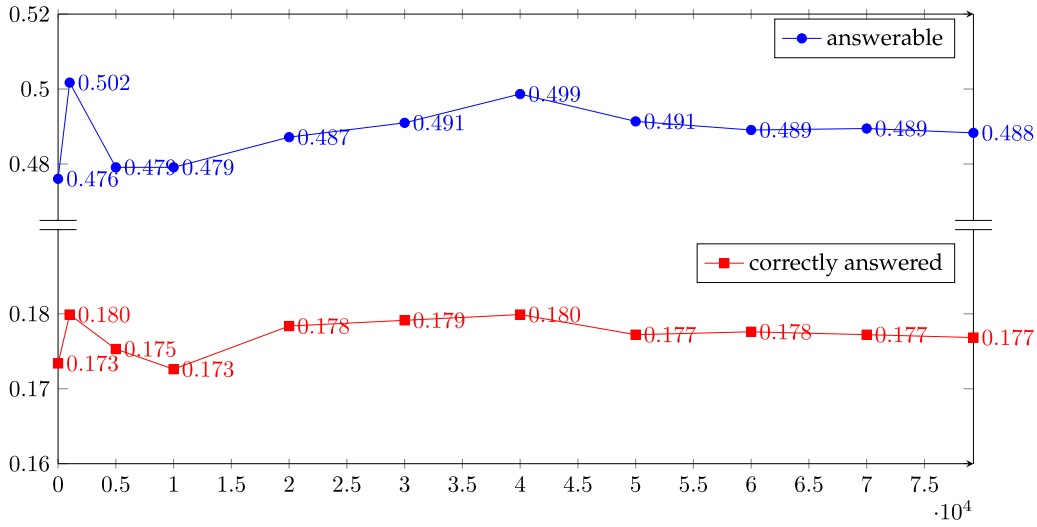

Figure 5
Fraction of answerable and correctly answered questions in the GRAPHQUESTIONS when different amounts of the SPADES data is used.

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 that 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 are substantially harder to answer correctly (see Table 8).

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 with 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 set-ups. This can be attributed to the differentiability of the soft attention layer. The structured attention layer is also differentiable because 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 data sets where logical forms are not given, the more structurally aware attention mechanism does not improve over soft attention, possibly because of 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 is much slower than soft attention, which has linear ($O(n)$) complexity.

Table 10
Hard attention and structured attention when predicting the relation in each question. The corresponding logical predicate is shown in parentheses.

hard attention
good selections:
the brickyard 400 was hosted at what **venue**? (`base.nascar.nascar_venue`)
christian faith branched from what **religion**? (`religion.religion`)
which **paintings** are discovered in lascaux? (`base.caveart.painting`)
bad selections:
which **violent** events started on 1995-04-07? (`base.disaster2.attack`)
who was **the** aircraft designer of the b-747? (`aviation.aircraft_designer`)
the boinc has been **used** in which services? (`base.centreforeresearch.service`)
neutral selections:
how does ultram **act** in the body? (`medicine.drug_mechanism_of_action`)
microsoft has created which **programming** languages? (`computer.programming_language`)
find un agencies **founded** in 1957 (`base.unitednations.united_nations_agency`).
structured attention
good selections:
the brickyard 400 was **hosted at what venue**? (`base.nascar.nascar_venue`)
which **violent events started on** 1995-04-07? (`base.disaster2.attack`)
how does ultram **act in the body**? (`medicine.drug_mechanism_of_action`)
bad selections:
what is ehrlich's **affiliation**? (`education.department`)
for which war was the italian armistice **signed**? (`base.morelaw.war`)
the boinc **has been used in** which services? (`base.centreforeresearch.service`)
neutral selections:
where was the brickyard 400 **held**? (`base.nascar.nascar_venue`)
by whom was paul **influenced**? (`influence.influence_node`)
how does ultram **act in the body**? (`medicine.drug_mechanism_of_action`)

An advantage of hard and structured attention is that they allow 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 article, we described a general neural semantic parsing framework that 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 where denotations are provided, and distant supervision where only unlabeled sentences and a knowledge base are available. Compared with 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 use 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 uses 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 the future, we will compare the presented parser-ranker framework with reinforcement learning-based parsers.

Directions for future work are many and varied. Because 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). Additionaly, when learning from denotations, it is possible that the beam search output contains spurious logical forms that 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, because 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 set-up (Herzig and Berant 2017), where the domain-generic parameters are shared.

References

Andreas, Jacob, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In *Proceedings of the Association for Computational Linguistics*, pages 47–52, Sofia.

Artzi, Yoav and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 421–432, Edinburgh.

Artzi, Yoav and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1):49–62.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 4th International Conference on Learning Representations*, San Diego.

Baker, Collin F., Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 86–90, Montreal.

Banarescu, Laura, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia.

Bast, Hannah and Elmar Haussmann. 2015. More accurate question answering on Freebase. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, pages 1431–1440, Melbourne.

Berant, Jonathan, Andrew Chou, Roy Frostig, and Percy Liang. 2013a. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, WA.

Berant, Jonathan, Andrew Chou, Roy Frostig, and Percy Liang. 2013b. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, WA.

Berant, Jonathan and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, MD.

Berant, Jonathan and Percy Liang. 2015. Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics*, 3:545–558.

Bisk, Yonatan, Siva Reddy, John Blitzer, Julia Hockenmaier, and Mark Steedman. 2016. Evaluating induced CCG parsers on grounded semantic parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2022–2027, Austin, TX.

Bordes, Antoine, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620, Doha.

Cai, Jonathon, Richard Shin, and Dawn Song. 2017. Making neural programming architectures generalize via recursion. `arXiv:1074.06611`.

Cai, Qingqing and Alexander Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 423–433, Sofia.

Chen, David L. and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th Conference on Artificial Intelligence*, volume 2, pages 859–865, San Francisco, CA.

Cheng, Jianpeng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, TX.

Cheng, Jianpeng, Adam Lopez, and Mirella Lapata. 2017. A generative parser with a discriminative recognition algorithm. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 118–124, Vancouver.

Cheng, Jianpeng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. 2017. Learning structured natural language representations for semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*

(Volume 1: Long Papers), pages 44–55, Vancouver.

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.

Dong, Li and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin.

Dong, Li, Jonathan Mallinson, Siva Reddy, and Mirella Lapata. 2017. Learning to paraphrase for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 886–897, Copenhagen.

Dong, Li, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over Freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 260–269, Beijing.

Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing.

Dyer, Chris, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, CA.

Fan, Xing, Emilio Monti, Lambert Mathias, and Markus Dreyer. 2017. Transfer learning for neural semantic parsing. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 48–56, Vancouver.

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). `lemurproject.org/clueweb12/FACC1`.

Goldman, Omer, Veronica Latcinnik, Udi Naveh, Amir Globerson, and Jonathan Berant. 2018. Weakly-supervised semantic parsing with abstract examples, pages 1809–1819, Melbourne.

Herzig, Jonathan and Jonathan Berant. 2017. Neural semantic parsing over multiple knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 623–628, Vancouver.

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

Iyer, Srinivasan, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver.

Jia, Robin and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin.

Kate, Rohit J. and Raymond J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 913–920, Sydney.

Kate, Rohit J., Yuk Wah Wong, and Raymond J. Mooney. 2005. Learning to Transform Natural to Formal Languages. In *Proceedings for the 20th National Conference on Artificial Intelligence*, pages 1062–1068, Pittsburgh, PA.

Kim, Yoon, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Structured attention networks. In *5th International Conference on Learning Representations*, Toulon.

Konstas, Ioannis, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 146–157, Vancouver.

Kočiský, Tomáš, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. 2016. Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of*

the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1078–1087, Austin, TX.
Krishnamurthy, Jayant, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen.
Krishnamurthy, Jayant and Tom Mitchell. 2012. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 754–765, Jeju Island.
Krishnamurthy, Jayant and Tom M. Mitchell. 2014. Joint syntactic and semantic parsing with combinatory categorial grammar. In *Proceedings of the Association for Computational Linguistics*, pages 1188–1198, Baltimore, MD.
Kuncoro, Adhiguna, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia.
Kwiatkowksi, Tom, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA.
Kwiatkowski, Tom, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556, Seattle, WA.
Kwiatkowski, Tom, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523, Edinburgh.
Lafferty, John, Andrew Mccallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–298, San Francisco, CA.
Liang, Chen, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 23–33, Vancouver.
Liang, Percy. 2013. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*.
Liang, Percy. 2016. Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, 59(9):68–76.
Liang, Percy, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 590–599, Portland, OR.
Liu, Yang and Mirella Lapata. 2018. Learning structured text representations. *Transactions of the Association for Computational Linguistics*, 6:63–75.
Lu, Wei, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. 2008. A generative model for parsing natural language to meaning representations. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 783–792, Honolulu, HI.
Matuszek, Cynthia, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. 2012. A joint model of language and perception for grounded attribute learning. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1671–1678, Edinburgh.
Montague, Richard. 1973. The proper treatment of quantification in ordinary English. In K. J. J. Hintikka, J. M. E. Moravcsik, and P. Suppes, editors, *Approaches to Natural Language*, volume 49 of Synthese Library. Springer Netherlands, pages 221–242.
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*, Toulon.
Neelakantan, Arvind, Quoc V. Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. *4th International Conference on Learning Representations*, San Juan.

Pasupat, Panupong and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1470–1480, Beijing.

Pennington, Jeffrey, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha.

Popescu, Ana-Maria, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of COLING 2004*, pages 141–147, Geneva.

Rabinovich, Maxim, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver.

Reddy, Siva, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392.

Reddy, Siva, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

Reddy, Siva, Oscar Täckström, Slav Petrov, Mark Steedman, and Mirella Lapata. 2017. Universal semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 89–101, Copenhagen.

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

Steedman, Mark. 2000. The *Syntactic Process*, MIT Press.

Su, Yu, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for qa evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572, Austin, TX.

Suhr, Alane, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2238–2249, New Orleans.

Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, Atlanta, GA.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112, Montreal.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008, Montreal.

Wen, Tsung Hsien, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721, Lisbon.

Williams, Ronald J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Winograd, Terry. 1972. Understanding natural language. *Cognitive Psychology*, 3(1):1–191.

Wong, Yuk Wah and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York, NY.

Woods, William A., Ronald M. Kaplan, and Bonnie Nash-Webber. 1972. The Lunar sciences: Natural language information system: Final report. Technical Report BBN Report 2378, Bolt Beranek and Newman.

Xu, Kun, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question answering on Freebase via relation extraction and textual evidence. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*

(Volume 1: Long Papers), pages 2326–2336, Berlin.

Yao, Xuchen and Benjamin Van Durme. 2014. Information extraction over structured data: Question answering with Freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 956–966, Baltimore, MD.

Yih, Wen tau, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing.

Yin, Pengcheng and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver.

Zelle, John M. and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 1050–1055, Portland, OR.

Zelle, John Marvin. 1995. Using inductive logic programming to automate the construction of natural language parsers. Ph.D. thesis, University of Texas at Austin.

Zettlemoyer, Luke and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague.

Zettlemoyer, Luke S. and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pages 658–666, Edinburgh.

Zhao, Kai and Liang Huang. 2015. Type-driven incremental semantic parsing with polymorphism. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1416–1421, Denver, CO.

Zhong, Victor, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.