# Language to Logical Form with Neural Attention

**Li Dong** and **Mirella Lapata**
Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB
li.dong@ed.ac.uk, mlap@inf.ed.ac.uk

## Abstract

Semantic parsing aims at mapping natural language to machine interpretable meaning representations. Traditional approaches rely on high-quality lexicons, manually-built templates, and linguistic features which are either domain- or representation-specific. In this paper we present a general method based on an attention-enhanced encoder-decoder model. We encode input utterances into vector representations, and generate their logical forms by conditioning the output sequences or trees on the encoding vectors. Experimental results on four datasets show that our approach performs competitively without using hand-engineered features and is easy to adapt across domains and meaning representations.

## 1 Introduction

Semantic parsing is the task of translating text to a formal meaning representation such as logical forms or structured queries. There has recently been a surge of interest in developing machine learning methods for semantic parsing (see the references in Section 2), due in part to the existence of corpora containing utterances annotated with formal meaning representations. Figure 1 shows an example of a question (left hand-side) and its annotated logical form (right hand-side), taken from JOBS (Tang and Mooney, 2001), a well-known semantic parsing benchmark. In order to predict the correct logical form for a given utterance, most previous systems rely on predefined templates and manually designed features, which often render the parsing model domain- or representation-specific. In this work, we aim to use a simple yet effective method to bridge the gap between natural language and logical form with minimal domain knowledge.

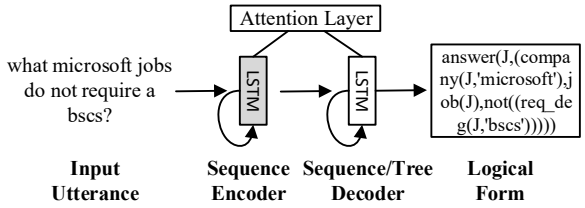



Figure 1: Input utterances and their logical forms are encoded and decoded with neural networks. An attention layer is used to learn soft alignments.

Encoder-decoder architectures based on recurrent neural networks have been successfully applied to a variety of NLP tasks ranging from syntactic parsing (Vinyals et al., 2015a), to machine translation (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014), and image description generation (Karpathy and Fei-Fei, 2015; Vinyals et al., 2015b). As shown in Figure 1, we adapt the general encoder-decoder paradigm to the semantic parsing task. Our model learns from natural language descriptions paired with meaning representations; it encodes sentences and decodes logical forms using recurrent neural networks with long short-term memory (LSTM) units. We present two model variants, the first one treats semantic parsing as a vanilla sequence transduction task, whereas our second model is equipped with a hierarchical tree decoder which explicitly captures the compositional structure of logical forms. We also introduce an attention mechanism (Bahdanau et al., 2015; Luong et al., 2015b) allowing the model to learn soft alignments between natural language and logical forms and present an argument identification step to handle rare mentions of entities and numbers.

Evaluation results demonstrate that compared to previous methods our model achieves similar or better performance across datasets and meaning representations, despite using no hand-engineered domain- or representation-specific features.



*Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 33–43,
Berlin, Germany, August 7-12, 2016. 

## 2 Related Work

Our work synthesizes two strands of research, namely semantic parsing and the encoder-decoder architecture with neural networks.

The problem of learning semantic parsers has received significant attention, dating back to Woods (1973). Many approaches learn from sentences paired with logical forms following various modeling strategies. Examples include the use of parsing models (Miller et al., 1996; Ge and Mooney, 2005; Lu et al., 2008; Zhao and Huang, 2015), inductive logic programming (Zelle and Mooney, 1996; Tang and Mooney, 2000; Thompson and Mooney, 2003), probabilistic automata (He and Young, 2006), string/tree-to-tree transformation rules (Kate et al., 2005), classifiers based on string kernels (Kate and Mooney, 2006), machine translation (Wong and Mooney, 2006; Wong and Mooney, 2007; Andreas et al., 2013), and combinatory categorial grammar induction techniques (Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007; Kwiatkowski et al., 2010; Kwiatkowski et al., 2011). Other work learns semantic parsers without relying on logical-from annotations, e.g., from sentences paired with conversational logs (Artzi and Zettlemoyer, 2011), system demonstrations (Chen and Mooney, 2011; Goldwasser and Roth, 2011; Artzi and Zettlemoyer, 2013), question-answer pairs (Clarke et al., 2010; Liang et al., 2013), and distant supervision (Krishnamurthy and Mitchell, 2012; Cai and Yates, 2013; Reddy et al., 2014).

Our model learns from natural language descriptions paired with meaning representations. Most previous systems rely on high-quality lexicons, manually-built templates, and features which are either domain- or representation-specific. We instead present a general method that can be easily adapted to different domains and meaning representations. We adopt the general encoder-decoder framework based on neural networks which has been recently repurposed for various NLP tasks such as syntactic parsing (Vinyals et al., 2015a), machine translation (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014), image description generation (Karpathy and Fei-Fei, 2015; Vinyals et al., 2015b), question answering (Hermann et al., 2015), and summarization (Rush et al., 2015).

Mei et al. (2016) use a sequence-to-sequence model to map navigational instructions to actions. Our model works on more well-defined meaning representations (such as Prolog and lambda calculus) and is conceptually simpler; it does not employ bidirectionality or multi-level alignments. Grefenstette et al. (2014) propose a different architecture for semantic parsing based on the combination of two neural network models. The first model learns shared representations from pairs of questions and their translations into knowledge base queries, whereas the second model generates the queries conditioned on the learned representations. However, they do not report empirical evaluation results.

## 3 Problem Formulation

Our aim is to learn a model which maps natural language input $q = x_1 \cdots x_{|q|}$ to a logical form representation of its meaning $a = y_1 \cdots y_{|a|}$. The conditional probability $p(a|q)$ is decomposed as:

$$p(a|q) = \prod_{t=1}^{|a|} p(y_t|y_{<t}, q) \tag{1}$$

where $y_{<t} = y_1 \cdots y_{t-1}$.

Our method consists of an **encoder** which encodes natural language input $q$ into a vector representation and a **decoder** which learns to generate $y_1, \cdots, y_{|a|}$ conditioned on the encoding vector. In the following we describe two models varying in the way in which $p(a|q)$ is computed.

### 3.1 Sequence-to-Sequence Model

This model regards both input $q$ and output $a$ as sequences. As shown in Figure 2, the encoder and decoder are two different $L$-layer recurrent neural networks with long short-term memory (LSTM) units which recursively process tokens one by one. The first $|q|$ time steps belong to the encoder, while the following $|a|$ time steps belong to the decoder. Let $\mathbf{h}_t^l \in \mathbb{R}^n$ denote the hidden vector at time step $t$ and layer $l$. $\mathbf{h}_t^l$ is then computed by:

$$\mathbf{h}_t^l = \text{LSTM}\left(\mathbf{h}_{t-1}^l, \mathbf{h}_t^{l-1}\right) \tag{2}$$

where LSTM refers to the LSTM function being used. In our experiments we follow the architecture described in Zaremba et al. (2015), however other types of gated activation functions are possible (e.g., Cho et al. (2014)). For the encoder, $\mathbf{h}_t^0 = \mathbf{W}_q \mathbf{e}(x_t)$ is the word vector of the current input token, with $\mathbf{W}_q \in \mathbb{R}^{n \times |V_q|}$ being a parameter matrix, and $\mathbf{e}(\cdot)$ the index of the corresponding

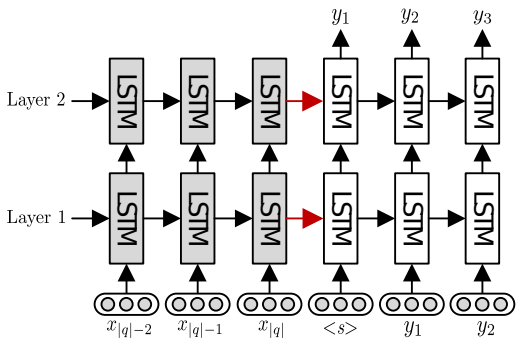



Figure 2: Sequence-to-sequence (SEQ2SEQ) model with two-layer recurrent neural networks.

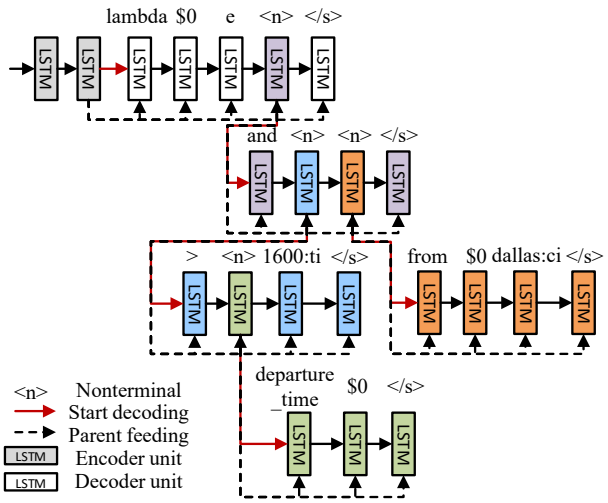



Figure 3: Sequence-to-tree (SEQ2TREE) model with a hierarchical tree decoder.

token. For the decoder, $\mathbf{h}_t^0 = \mathbf{W}_a \mathbf{e}(y_{t-1})$ is the word vector of the previous predicted word, where $\mathbf{W}_a \in \mathbb{R}^{n \times |V_a|}$. Notice that the encoder and decoder have different LSTM parameters.

Once the tokens of the input sequence $x_1, \cdots, x_{|q|}$ are encoded into vectors, they are used to initialize the hidden states of the first time step in the decoder. Next, the hidden vector of the topmost LSTM $\mathbf{h}_t^L$ in the decoder is used to predict the $t$-th output token as:

$$p\left(y_t | y_{<t}, q\right) = \text{softmax}\left(\mathbf{W}_o \mathbf{h}_t^L\right)^\intercal \mathbf{e}\left(y_t\right) \quad (3)$$

where $\mathbf{W}_o \in \mathbb{R}^{|V_a| \times n}$ is a parameter matrix, and $\mathbf{e}\left(y_t\right) \in \{0,1\}^{|V_a|}$ a one-hot vector for computing $y_t$'s probability from the predicted distribution.

We augment every sequence with a "start-of-sequence" <*s*> and "end-of-sequence" </*s*> token. The generation process terminates once </*s*> is predicted. The conditional probability of generating the whole sequence $p\left(a|q\right)$ is then obtained using Equation (1).

### 3.2 Sequence-to-Tree Model

The SEQ2SEQ model has a potential drawback in that it ignores the hierarchical structure of logical forms. As a result, it needs to memorize various pieces of auxiliary information (e.g., bracket pairs) to generate well-formed output. In the following we present a hierarchical tree decoder which is more faithful to the compositional nature of meaning representations. A schematic description of the model is shown in Figure 3.

The present model shares the same encoder with the sequence-to-sequence model described in Section 3.1 (essentially it learns to encode input $q$ as vectors). However, its decoder is fundamentally different as it generates logical forms in a top-down manner. In order to represent tree structure, we define a "nonterminal" <*n*> token which indicates subtrees. As shown in Figure 3, we preprocess the logical form "*lambda $0 e (and (>(departure_time $0) 1600:ti) (from $0 dallas:ci))*" to a tree by replacing tokens between pairs of brackets with nonterminals. Special tokens <*s*> and <*(*> denote the beginning of a sequence and nonterminal sequence, respectively (omitted from Figure 3 due to lack of space). Token </*s*> represents the end of sequence.

After encoding input $q$, the hierarchical tree decoder uses recurrent neural networks to generate tokens at depth 1 of the subtree corresponding to parts of logical form $a$. If the predicted token is <*n*>, we decode the sequence by conditioning on the nonterminal's hidden vector. This process terminates when no more nonterminals are emitted. In other words, a sequence decoder is used to hierarchically generate the tree structure.

In contrast to the sequence decoder described in Section 3.1, the current hidden state does not only depend on its previous time step. In order to better utilize the parent nonterminal's information, we introduce a *parent-feeding* connection where the hidden vector of the parent nonterminal is concatenated with the inputs and fed into LSTM.

As an example, Figure 4 shows the decoding tree corresponding to the logical form "*A B (C)*", where $y_1 \cdots y_6$ are predicted tokens, and $t_1 \cdots t_6$ denote different time steps. Span "*(C)*" corresponds to a subtree. Decoding in this example has two steps: once input $q$ has been encoded, we first generate $y_1 \cdots y_4$ at depth 1 until token </*s*> is

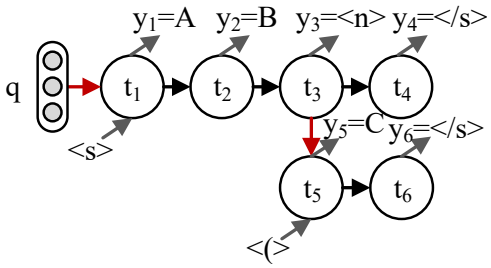



Figure 4: A SEQ2TREE decoding example for the logical form "*A B (C)*".

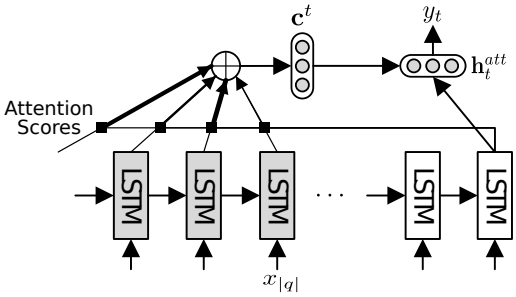



Figure 5: Attention scores are computed by the current hidden vector and all the hidden vectors of encoder. Then, the encoder-side context vector $\mathbf{c}^t$ is obtained in the form of a weighted sum, which is further used to predict $y_t$.

predicted; next, we generate $y_5, y_6$ by conditioning on nonterminal $t_3$'s hidden vectors. The probability $p(a|q)$ is the product of these two sequence decoding steps:

$$p(a|q) = p(y_1 y_2 y_3 y_4|q)\, p(y_5 y_6|y_{\leq 3}, q) \quad (4)$$

where Equation (3) is used for the prediction of each output token.

### 3.3 Attention Mechanism

As shown in Equation (3), the hidden vectors of the input sequence are not directly used in the decoding process. However, it makes intuitively sense to consider relevant information from the input to better predict the current token. Following this idea, various techniques have been proposed to integrate encoder-side information (in the form of a context vector) at each time step of the decoder (Bahdanau et al., 2015; Luong et al., 2015b; Xu et al., 2015).

As shown in Figure 5, in order to find relevant encoder-side context for the current hidden state $\mathbf{h}_t^L$ of decoder, we compute its attention score with the $k$-th hidden state in the encoder as:

$$s_k^t = \frac{\exp\{\mathbf{h}_k^L \cdot \mathbf{h}_t^L\}}{\sum_{j=1}^{|q|} \exp\{\mathbf{h}_j^L \cdot \mathbf{h}_t^L\}} \quad (5)$$

where $\mathbf{h}_1^L, \cdots, \mathbf{h}_{|q|}^L$ are the top-layer hidden vectors of the encoder. Then, the context vector is the weighted sum of the hidden vectors in the encoder:

$$\mathbf{c}^t = \sum_{k=1}^{|q|} s_k^t \mathbf{h}_k^L \quad (6)$$

In lieu of Equation (3), we further use this context vector which acts as a summary of the encoder to compute the probability of generating $y_t$ as:

$$\mathbf{h}_t^{att} = \tanh\left(\mathbf{W}_1 \mathbf{h}_t^L + \mathbf{W}_2 \mathbf{c}^t\right) \quad (7)$$

$$p(y_t|y_{<t}, q) = \text{softmax}\left(\mathbf{W}_o \mathbf{h}_t^{att}\right)^{\mathsf{T}} \mathbf{e}(y_t) \quad (8)$$

where $\mathbf{W}_o \in \mathbb{R}^{|V_a| \times n}$ and $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times n}$ are three parameter matrices, and $\mathbf{e}(y_t)$ is a one-hot vector used to obtain $y_t$'s probability.

### 3.4 Model Training

Our goal is to maximize the likelihood of the generated logical forms given natural language utterances as input. So the objective function is:

$$\text{minimize} - \sum_{(q,a)\in\mathcal{D}} \log p(a|q) \quad (9)$$

where $\mathcal{D}$ is the set of all natural language-logical form training pairs, and $p(a|q)$ is computed as shown in Equation (1).

The RMSProp algorithm (Tieleman and Hinton, 2012) is employed to solve this non-convex optimization problem. Moreover, dropout is used for regularizing the model (Zaremba et al., 2015). Specifically, dropout operators are used between different LSTM layers and for the hidden layers before the softmax classifiers. This technique can substantially reduce overfitting, especially on datasets of small size.

### 3.5 Inference

At test time, we predict the logical form for an input utterance $q$ by:

$$\hat{a} = \arg\max_{a'} p\left(a'|q\right) \quad (10)$$

where $a'$ represents a candidate output. However, it is impractical to iterate over all possible results to obtain the optimal prediction. According to Equation (1), we decompose the probability $p(a|q)$ so that we can use greedy search (or beam search) to generate tokens one by one.

**Algorithm 1** Decoding for SEQ2TREE

**Input:** $q$: Natural language utterance
**Output:** $\hat{a}$: Decoding result

$\triangleright$ *Push the encoding result to a queue*
$Q.init(\{hid : \mathsf{SeqEnc}(q)\})$
$\triangleright$ *Decode until no more nonterminals*
**while** $(c \leftarrow Q.pop()) \neq \varnothing$ **do**
    $\triangleright$ *Call sequence decoder*
    $c.child \leftarrow \mathsf{SeqDec}(c.hid)$
    $\triangleright$ *Push new nonterminals to queue*
    **for** $n \leftarrow$ nonterminal in $c.child$ **do**
        $Q.push(\{hid : \mathsf{HidVec}(n)\})$
$\hat{a} \leftarrow$ convert decoding tree to output sequence

Algorithm 1 describes the decoding process for SEQ2TREE. The time complexity of both decoders is $\mathcal{O}(|a|)$, where $|a|$ is the length of output. The extra computation of SEQ2TREE compared with SEQ2SEQ is to maintain the nonterminal queue, which can be ignored because most of time is spent on matrix operations. We implement the hierarchical tree decoder in a batch mode, so that it can fully utilize GPUs. Specifically, as shown in Algorithm 1, every time we pop multiple nonterminals from the queue and decode these nonterminals in one batch.

### 3.6 Argument Identification

The majority of semantic parsing datasets have been developed with question-answering in mind. In the typical application setting, natural language questions are mapped into logical forms and executed on a knowledge base to obtain an answer. Due to the nature of the question-answering task, many natural language utterances contain entities or numbers that are often parsed as arguments in the logical form. Some of them are unavoidably rare or do not appear in the training set at all (this is especially true for small-scale datasets). Conventional sequence encoders simply replace rare words with a special unknown word symbol (Luong et al., 2015a; Jean et al., 2015), which would be detrimental for semantic parsing.

We have developed a simple procedure for argument identification. Specifically, we identify entities and numbers in input questions and replace them with their type names and unique IDs. For instance, we pre-process the training example "*jobs with a salary of 40000*" and its logical form "*job(ANS), salary_greater_than(ANS, 40000, year)*" as "*jobs with a salary of* $\underline{num_0}$" and "*job(ANS), salary_greater_than(ANS,* $\underline{num_0}$*, year)*". We use the pre-processed examples as training data. At inference time, we also mask entities and numbers with their types and IDs. Once we obtain the decoding result, a post-processing step recovers all the markers $\underline{type_i}$ to their corresponding logical constants.

## 4 Experiments

We compare our method against multiple previous systems on four datasets. We describe these datasets below, and present our experimental settings and results. Finally, we conduct model analysis in order to understand what the model learns. The code is available at `https://github.com/donglixp/lang2logic`.

### 4.1 Datasets

Our model was trained on the following datasets, covering different domains and using different meaning representations. Examples for each domain are shown in Table 1.

**JOBS** This benchmark dataset contains 640 queries to a database of job listings. Specifically, questions are paired with Prolog-style queries. We used the same training-test split as Zettlemoyer and Collins (2005) which contains 500 training and 140 test instances. Values for the variables company, degree, language, platform, location, job area, and number are identified.

**GEO** This is a standard semantic parsing benchmark which contains 880 queries to a database of U.S. geography. GEO has 880 instances split into a training set of 680 training examples and 200 test examples (Zettlemoyer and Collins, 2005). We used the same meaning representation based on lambda-calculus as Kwiatkowski et al. (2011). Values for the variables city, state, country, river, and number are identified.

**ATIS** This dataset has 5,410 queries to a flight booking system. The standard split has 4,480 training instances, 480 development instances, and 450 test instances. Sentences are paired with lambda-calculus expressions. Values for the variables date, time, city, aircraft code, airport, airline, and number are identified.

**IFTTT** Quirk et al. (2015) created this dataset by extracting a large number of if-this-then-that

| Dataset | Length | Example |
|---|---|---|
| JOBS | 9.80 | *what microsoft jobs do not require a bscs?* |
| | 22.90 | answer(company(J,'microsoft'),job(J),not((req_deg(J,'bscs')))) |
| GEO | 7.60 | *what is the population of the state with the largest area?* |
| | 19.10 | (population:i (argmax $0 (state:t $0) (area:i $0))) |
| ATIS | 11.10 | *dallas to san francisco leaving after 4 in the afternoon please* |
| | 28.10 | (lambda $0 e (and (>(departure_time $0) 1600:ti) (from $0 dallas:ci) (to $0 san_francisco:ci))) |
| IFTTT | 6.95 | *Turn on heater when temperature drops below 58 degree* |
| | 21.80 | TRIGGER: Weather - Current_temperature_drops_below - ((Temperature (58)) (Degrees_in (f))) |
| | | ACTION: WeMo_Insight_Switch - Turn_on - ((Which_switch? (""))) |

Table 1: Examples of natural language descriptions and their meaning representations from four datasets. The average length of input and output sequences is shown in the second column.

recipes from the IFTTT website[1]. Recipes are simple programs with exactly one trigger and one action which users specify on the site. Whenever the conditions of the trigger are satisfied, the action is performed. Actions typically revolve around home security (e.g., "*turn on my lights when I arrive home*"), automation (e.g., "*text me if the door opens*"), well-being (e.g., "*remind me to drink water if I've been at a bar for more than two hours*"), and so on. Triggers and actions are selected from different channels (160 in total) representing various types of services, devices (e.g., Android), and knowledge sources (such as ESPN or Gmail). In the dataset, there are 552 trigger functions from 128 channels, and 229 action functions from 99 channels. We used Quirk et al.'s (2015) original split which contains $77,495$ training, $5,171$ development, and $4,294$ test examples. The IFTTT programs are represented as abstract syntax trees and are paired with natural language descriptions provided by users (see Table 1). Here, numbers and URLs are identified.

### 4.2 Settings

Natural language sentences were lowercased; misspellings were corrected using a dictionary based on the Wikipedia list of common misspellings. Words were stemmed using NLTK (Bird et al., 2009). For IFTTT, we filtered tokens, channels and functions which appeared less than five times in the training set. For the other datasets, we filtered input words which did not occur at least two times in the training set, but kept all tokens in the logical forms. Plain string matching was employed to identify augments as described in Section 3.6. More sophisticated approaches could be used, however we leave this future work.

Model hyper-parameters were cross-validated on the training set for JOBS and GEO. We used the standard development sets for ATIS and IFTTT. We used the RMSProp algorithm (with batch size set to 20) to update the parameters. The smoothing constant of RMSProp was $0.95$. Gradients were clipped at $5$ to alleviate the exploding gradient problem (Pascanu et al., 2013). Parameters were randomly initialized from a uniform distribution $\mathcal{U}(-0.08, 0.08)$. A two-layer LSTM was used for IFTTT, while a one-layer LSTM was employed for the other domains. The dropout rate was selected from $\{0.2, 0.3, 0.4, 0.5\}$. Dimensions of hidden vector and word embedding were selected from $\{150, 200, 250\}$. Early stopping was used to determine the number of epochs. Input sentences were reversed before feeding into the encoder (Sutskever et al., 2014). We use greedy search to generate logical forms during inference. Notice that two decoders with shared word embeddings were used to predict triggers and actions for IFTTT, and two softmax classifiers are used to classify channels and functions.

| Method | Accuracy |
|---|---|
| COCKTAIL (Tang and Mooney, 2001) | 79.4 |
| PRECISE (Popescu et al., 2003) | 88.0 |
| ZC05 (Zettlemoyer and Collins, 2005) | 79.3 |
| DCS+L (Liang et al., 2013) | 90.7 |
| TISP (Zhao and Huang, 2015) | 85.0 |
| SEQ2SEQ | 87.1 |
| − attention | 77.9 |
| − argument | 70.7 |
| SEQ2TREE | 90.0 |
| − attention | 83.6 |

Table 2: Evaluation results on JOBS.

### 4.3 Results

We first discuss the performance of our model on JOBS, GEO, and ATIS, and then examine our results on IFTTT. Tables 2–4 present comparisons against a variety of systems previously described

[1] http://www.ifttt.com

| Method | Accuracy |
|---|---|
| SCISSOR (Ge and Mooney, 2005) | 72.3 |
| KRISP (Kate and Mooney, 2006) | 71.7 |
| WASP (Wong and Mooney, 2006) | 74.8 |
| $\lambda$-WASP (Wong and Mooney, 2007) | 86.6 |
| LNLZ08 (Lu et al., 2008) | 81.8 |
| ZC05 (Zettlemoyer and Collins, 2005) | 79.3 |
| ZC07 (Zettlemoyer and Collins, 2007) | 86.1 |
| UBL (Kwiatkowski et al., 2010) | 87.9 |
| FUBL (Kwiatkowski et al., 2011) | 88.6 |
| KCAZ13 (Kwiatkowski et al., 2013) | 89.0 |
| DCS+L (Liang et al., 2013) | 87.9 |
| TISP (Zhao and Huang, 2015) | 88.9 |
| SEQ2SEQ | 84.6 |
| − attention | 72.9 |
| − argument | 68.6 |
| SEQ2TREE | 87.1 |
| − attention | 76.8 |

Table 3: Evaluation results on GEO. 10-fold cross-validation is used for the systems shown in the top half of the table. The standard split of ZC05 is used for all other systems.

| Method | Accuracy |
|---|---|
| ZC07 (Zettlemoyer and Collins, 2007) | 84.6 |
| UBL (Kwiatkowski et al., 2010) | 71.4 |
| FUBL (Kwiatkowski et al., 2011) | 82.8 |
| GUSP-FULL (Poon, 2013) | 74.8 |
| GUSP++ (Poon, 2013) | 83.5 |
| TISP (Zhao and Huang, 2015) | 84.2 |
| SEQ2SEQ | 84.2 |
| − attention | 75.7 |
| − argument | 72.3 |
| SEQ2TREE | 84.6 |
| − attention | 77.5 |

Table 4: Evaluation results on ATIS.

| Method | Channel | +Func | F1 |
|---|---|---|---|
| retrieval | 28.9 | 20.2 | 41.7 |
| phrasal | 19.3 | 11.3 | 35.3 |
| sync | 18.1 | 10.6 | 35.1 |
| classifier | 48.8 | 35.2 | 48.4 |
| posclass | 50.0 | 36.9 | 49.3 |
| SEQ2SEQ | 54.3 | 39.2 | 50.1 |
| − attention | 54.0 | 37.9 | 49.8 |
| − argument | 53.9 | 38.6 | 49.7 |
| SEQ2TREE | 55.2 | 40.1 | 50.4 |
| − attention | 54.3 | 38.2 | 50.0 |

(a) Omit non-English.

| Method | Channel | +Func | F1 |
|---|---|---|---|
| retrieval | 36.8 | 25.4 | 49.0 |
| phrasal | 27.8 | 16.4 | 39.9 |
| sync | 26.7 | 15.5 | 37.6 |
| classifier | 64.8 | 47.2 | 56.5 |
| posclass | 67.2 | 50.4 | 57.7 |
| SEQ2SEQ | 68.8 | 50.5 | 60.3 |
| − attention | 68.7 | 48.9 | 59.5 |
| − argument | 68.8 | 50.4 | 59.7 |
| SEQ2TREE | 69.6 | 51.4 | 60.4 |
| − attention | 68.7 | 49.5 | 60.2 |

(b) Omit non-English & unintelligible.

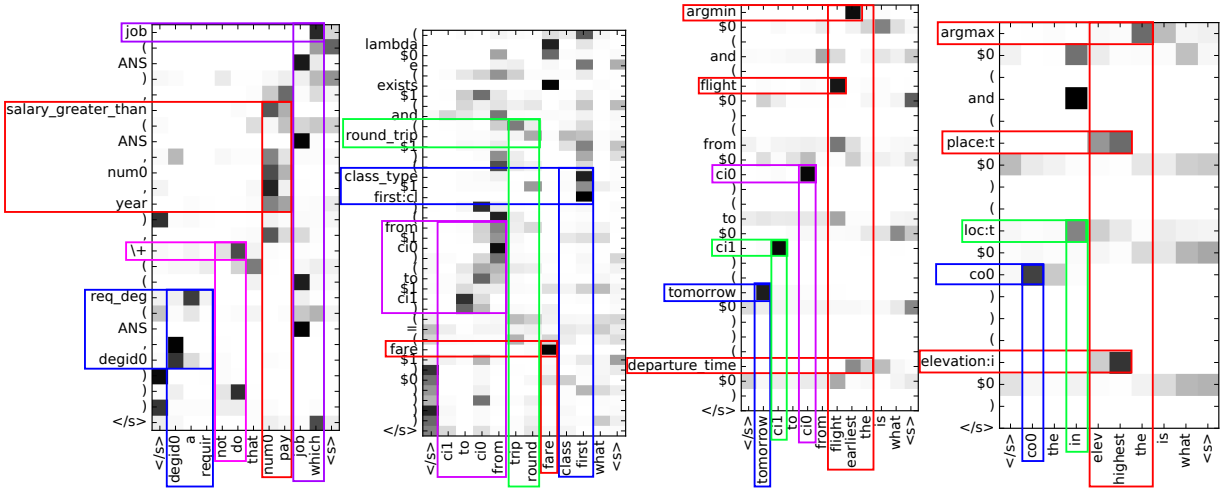
| Method | Channel | +Func | F1 |
|---|---|---|---|
| retrieval | 43.3 | 32.3 | 56.2 |
| phrasal | 37.2 | 23.5 | 45.5 |
| sync | 36.5 | 24.1 | 42.8 |
| classifier | 79.3 | 66.2 | 65.0 |
| posclass | 81.4 | 71.0 | 66.5 |
| SEQ2SEQ | 87.8 | 75.2 | 73.7 |
| − attention | 88.3 | 73.8 | 72.9 |
| − argument | 86.8 | 74.9 | 70.8 |
| SEQ2TREE | 89.7 | 78.4 | 74.2 |
| − attention | 87.6 | 74.9 | 73.5 |

(c) $\geq 3$ turkers agree with gold.

Table 5: Evaluation results on IFTTT.

in the literature. We report results with the full models (SEQ2SEQ, SEQ2TREE) and two ablation variants, i.e., without an attention mechanism (−attention) and without argument identification (−argument). We report accuracy which is defined as the proportion of the input sentences that are correctly parsed to their gold standard logical forms. Notice that DCS+L, KCAZ13 and GUSP output answers directly, so accuracy in this setting is defined as the percentage of correct answers.

Overall, SEQ2TREE is superior to SEQ2SEQ. This is to be expected since SEQ2TREE explicitly models compositional structure. On the JOBS and GEO datasets which contain logical forms with nested structures, SEQ2TREE outperforms SEQ2SEQ by 2.9% and 2.5%, respectively. SEQ2TREE achieves better accuracy over SEQ2SEQ on ATIS too, however, the difference is smaller, since ATIS is a simpler domain without complex nested structures. We find that adding attention substantially improves performance on all three datasets. This underlines the importance of utilizing soft alignments between inputs and outputs. We further analyze what the attention layer learns in Figure 6. Moreover, our results show that argument identification is critical for small-scale datasets. For example, about 92% of city names appear less than 4 times in the GEO training set, so it is difficult to learn reliable parameters for these words. In relation to previous work, the proposed models achieve comparable or better performance. Importantly, we use the same framework (SEQ2SEQ or SEQ2TREE) across datasets and meaning representations (Prolog-style logical forms in JOBS and lambda calculus in the other two datasets) without modification. Despite this relatively simple approach, we observe that SEQ2TREE ranks second on JOBS, and is tied for first place with ZC07 on ATIS.




(a) which jobs pay num0 that do not require a degid0

(b) what's first class fare round trip from ci0 to ci1

(c) what is the earliest flight from ci0 to ci1 tomorrow

(d) what is the highest elevation in the co0

Figure 6: Alignments (same color rectangles) produced by the attention mechanism (darker color represents higher attention score). Input sentences are reversed and stemmed. Model output is shown for SEQ2SEQ (a, b) and SEQ2TREE (c, d).

We illustrate examples of alignments produced by SEQ2SEQ in Figures 6a and 6b. Alignments produced by SEQ2TREE are shown in Figures 6c and 6d. Matrices of attention scores are computed using Equation (5) and are represented in grayscale. Aligned input words and logical form predicates are enclosed in (same color) rectangles whose overlapping areas contain the attention scores. Also notice that attention scores are computed by LSTM hidden vectors which encode context information rather than just the words in their current positions. The examples demonstrate that the attention mechanism can successfully model the correspondence between sentences and logical forms, capturing reordering (Figure 6b), many-to-many (Figure 6a), and many-to-one alignments (Figures 6c,d).

For IFTTT, we follow the same evaluation protocol introduced in Quirk et al. (2015). The dataset is extremely noisy and measuring accuracy is problematic since predicted abstract syntax trees (ASTs) almost never exactly match the gold standard. Quirk et al. view an AST as a set of productions and compute balanced F1 instead which we also adopt. The first column in Table 5 shows the percentage of channels selected correctly for both triggers and actions. The second column measures accuracy for *both* channels and functions. The last column shows balanced F1 against the gold tree over all productions in the proposed derivation. We compare our model against posclass, the method introduced in Quirk et al. and several of their baselines. posclass is reminiscent of KRISP (Kate and Mooney, 2006), it learns distributions over productions given input sentences represented as a bag of linguistic features. The retrieval baseline finds the closest description in the training data based on character string-edit-distance and returns the recipe for that training program. The phrasal method uses phrase-based machine translation to generate the recipe, whereas sync extracts synchronous grammar rules from the data, essentially recreating WASP (Wong and Mooney, 2006). Finally, they use a binary classifier to predict whether a production should be present in the derivation tree corresponding to the description.

Quirk et al. (2015) report results on the full test data and smaller subsets after noise filtering, e.g., when non-English and unintelligible descriptions are removed (Tables 5a and 5b). They also ran their system on a high-quality subset of description-program pairs which were found in the gold standard and at least three humans managed to independently reproduce (Table 5c). Across all subsets our models outperforms posclass and related baselines. Again we observe that SEQ2TREE consistently outperforms SEQ2SEQ, albeit with a small margin. Compared to the previous datasets, the attention mechanism and our argument iden-

tification method yield less of an improvement. This may be due to the size of Quirk et al. (2015) and the way it was created – user curated descriptions are often of low quality, and thus align very loosely to their corresponding ASTs.

### 4.4 Error Analysis

Finally, we inspected the output of our model in order to identify the most common causes of errors which we summarize below.

**Under-Mapping** The attention model used in our experiments does not take the alignment history into consideration. So, some question words, expecially in longer questions, may be ignored in the decoding process. This is a common problem for encoder-decoder models and can be addressed by explicitly modelling the decoding coverage of the source words (Tu et al., 2016; Cohn et al., 2016). Keeping track of the attention history would help adjust future attention and guide the decoder towards untranslated source words.

**Argument Identification** Some mentions are incorrectly identified as arguments. For example, the word *may* is sometimes identified as a month when it is simply a modal verb. Moreover, some argument mentions are ambiguous. For instance, *6 o'clock* can be used to express either *6 am* or *6 pm*. We could disambiguate arguments based on contextual information. The execution results of logical forms could also help prune unreasonable arguments.

**Rare Words** Because the data size of JOBS, GEO, and ATIS is relatively small, some question words are rare in the training set, which makes it hard to estimate reliable parameters for them. One solution would be to learn word embeddings on unannotated text data, and then use these as pretrained vectors for question words.

## 5 Conclusions

In this paper we presented an encoder-decoder neural network model for mapping natural language descriptions to their meaning representations. We encode natural language utterances into vectors and generate their corresponding logical forms as sequences or trees using recurrent neural networks with long short-term memory units. Experimental results show that enhancing the model with a hierarchical tree decoder and an attention mechanism improves performance across the board. Extensive comparisons with previous methods show that our approach performs competitively, without recourse to domain- or representation-specific features. Directions for future work are many and varied. For example, it would be interesting to learn a model from question-answer pairs without access to target logical forms. Beyond semantic parsing, we would also like to apply our SEQ2TREE model to related structured prediction tasks such as constituency parsing.

**Acknowledgments** We would like to thank Luke Zettlemoyer and Tom Kwiatkowski for sharing the ATIS dataset. The support of the European Research Council under award number 681760 "Translating Multiple Modalities into Text" is gratefully acknowledged.

## References

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In *Proceedings of the 51st ACL*, pages 47–52, Sofia, Bulgaria.

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the 2011 EMNLP*, pages 421–432, Edinburgh, United Kingdom.

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the ICLR*, San Diego, California.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

Qingqing Cai and Alexander Yates. 2013. Semantic parsing freebase: Towards open-domain semantic parsing. In *2nd Joint Conference on Lexical and Computational Semantics*, pages 328–338, Atlanta, Georgia.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 15th AAAI*, pages 859–865, San Francisco, California.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 EMNLP*, pages 1724–1734, Doha, Qatar.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of CONLL*, pages 18–27, Uppsala, Sweden.

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. 2016. Incorporating structural alignment biases into an attentional neural translation model. In *Proceedings of the 2016 NAACL*, San Diego, California.

Ruifang Ge and Raymond J. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of CoNLL*, pages 9–16, Ann Arbor, Michigan.

Dan Goldwasser and Dan Roth. 2011. Learning from natural instructions. In *Proceedings of the 22nd IJCAI*, pages 1794–1800, Barcelona, Spain.

Edward Grefenstette, Phil Blunsom, Nando de Freitas, and Karl Moritz Hermann. 2014. A deep architecture for semantic parsing. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, Atlanta, Georgia.

Yulan He and Steve Young. 2006. Semantic processing using the hidden vector state model. *Speech Communication*, 48(3-4):262–275.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28*, pages 1684–1692. Curran Associates, Inc.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *Proceedings of 53rd ACL and 7th IJCNLP*, pages 1–10, Beijing, China.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of the 2013 EMNLP*, pages 1700–1709, Seattle, Washington.

Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of CVPR*, pages 3128–3137, Boston, Massachusetts.

Rohit J. Kate and Raymond J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st COLING and 44th ACL*, pages 913–920, Sydney, Australia.

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the 20th AAAI*, pages 1062–1068, Pittsburgh, Pennsylvania.

Jayant Krishnamurthy and Tom Mitchell. 2012. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 EMNLP*, pages 754–765, Jeju Island, Korea.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 EMNLP*, pages 1223–1233, Cambridge, Massachusetts.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 EMNLP*, pages 1512–1523, Edinburgh, United Kingdom.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 EMNLP*, pages 1545–1556, Seattle, Washington.

Percy Liang, Michael I. Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Wei Lu, 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 EMNLP*, pages 783–792, Honolulu, Hawaii.

Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. 2015a. Addressing the rare word problem in neural machine translation. In *Proceedings of the 53rd ACL and 7th IJCNLP*, pages 11–19, Beijing, China.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015b. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 EMNLP*, pages 1412–1421, Lisbon, Portugal.

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Proceedings of the 30th AAAI*, Phoenix, Arizona. to appear.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *ACL*, pages 55–61.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th ICML*, pages 1310–1318, Atlanta, Georgia.

Hoifung Poon. 2013. Grounded unsupervised semantic parsing. In *Proceedings of the 51st ACL*, pages 933–943, Sofia, Bulgaria.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th IUI*, pages 149–157, Miami, Florida.

Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of 53rd ACL and 7th IJCNLP*, pages 878–888, Beijing, China.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *TACL*, 2(Oct):377–392.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 EMNLP*, pages 379–389, Lisbon, Portugal.

Ilya Sutskever, 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. Curran Associates, Inc.

Lappoon R. Tang and Raymond J. Mooney. 2000. Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In *Proceedings of the 2000 EMNLP*, pages 133–141, Hong Kong, China.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th ECML*, pages 466–477, Freiburg, Germany.

Cynthia A. Thomspon and Raymond J. Mooney. 2003. Acquiring word-meaning mappings for natural language interfaces. *Journal of Artifical Intelligence Research*, 18:1–44.

T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. Technical report.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *Proceedings of the 54th ACL*, Berlin, Germany.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015a. Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28*, pages 2755–2763. Curran Associates, Inc.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015b. Show and tell: A neural image caption generator. In *Proceedings of CVPR*, pages 3156–3164, Boston, Massachusetts.

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

Yuk Wah Wong and Raymond J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th ACL*, pages 960–967, Prague, Czech Republic.

W. A. Woods. 1973. Progress in natural language understanding: An application to lunar geology. In *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, pages 441–450, New York, NY.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd ICML*, pages 2048–2057, Lille, France.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2015. Recurrent neural network regularization. In *Proceedings of the ICLR*, San Diego, California.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the 19th AAAI*, pages 1050–1055, Portland, Oregon.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st UAI*, pages 658–666, Toronto, ON.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *In Proceedings of the EMNLP-CoNLL*, pages 678–687, Prague, Czech Republic.

Kai Zhao and Liang Huang. 2015. Type-driven incremental semantic parsing with polymorphism. In *Proceedings of the 2015 NAACL*, pages 1416–1421, Denver, Colorado.