Sequence-based Structured Prediction for Semantic Parsing

Chunyang Xiao, Marc Dymetman, Claire Gardent

Sequence-based Structured Prediction for Semantic Parsing

Chunyang Xiao and **Marc Dymetman**
Xerox Research Centre Europe
6-8 Chemin de Maupertuis
Meylan, 38240, France
chunyang.xiao, marc.dymetman
@xrce.xerox.com

Claire Gardent
CNRS, LORIA, UMR 7503
Vandoeuvre-lès-Nancy,
F-54500, France
claire.gardent@loria.fr

Abstract

We propose an approach for semantic parsing that uses a recurrent neural network to map a natural language question into a logical form representation of a KB query. Building on recent work by (Wang et al., 2015), the interpretable logical forms, which are structured objects obeying certain constraints, are enumerated by an underlying grammar and are paired with their canonical realizations. In order to use sequence prediction, we need to sequentialize these logical forms. We compare three sequentializations: a direct linearization of the logical form, a linearization of the associated canonical realization, and a sequence consisting of derivation steps relative to the underlying grammar. We also show how grammatical constraints on the derivation sequence can easily be integrated inside the RNN-based sequential predictor. Our experiments show important improvements over previous results for the same dataset, and also demonstrate the advantage of incorporating the grammatical constraints.

1 Introduction

Learning to map natural language utterances (NL) to logical forms (LF), a process known as semantic parsing, has received a lot of attention recently, in particular in the context of building Question-Answering systems (Kwiatkowski et al., 2013; Berant et al., 2013; Berant and Liang, 2014). In this paper, we focus on such a task where the NL question may be semantically complex, leading to a logical form query with a fair amount of compositionality, in a spirit close to (Pasupat and Liang, 2015).

Given the recently shown effectiveness of RNNs (Recurrent Neural Networks), in particular Long Short Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997), for performing sequence prediction in NLP applications such as machine translation (Sutskever et al., 2014) and natural language generation (Wen et al., 2015), we try to exploit similar techniques for our task. However we observe that, contrary to those applications which try to predict intrinsically *sequential* objects (texts), our task involves producing a *structured* object, namely a logical form that is tree-like by nature and also has to respect certain *a priori* constraints in order to be interpretable against the knowledge base.

In our case, building on the work "Building a Semantic Parser Overnight" (Wang et al., 2015), which we will refer to as SPO, the LFs are generated by a grammar which is known *a priori*, and it is this grammar that makes explicit the structural constraints that have to be satisfied by the LFs. The SPO grammar, along with generating logical forms, generates so-called "canonical forms" (CF), which are direct textual realizations of the LF that, although they are not "natural" English, transparently convey the meaning of the LF (see Fig. 1 for an example).

Based on this grammar, we explore three different ways of representing the LF structure through a sequence of items. The first one (LF Prediction, or LFP), and simplest, consists in just linearizing the LF tree into a sequence of individual tokens; the second one (CFP) represents the LF through its associated CF, which is itself a sequence of words; and finally the third one (DSP) represents the LF through a derivation sequence (DS), namely the sequence of grammar rules that were chosen to produce this LF.

We then predict the LF via LSTM-based models that take as input the NL question and map it into

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 1341–1350,
Berlin, Germany, August 7-12, 2016.

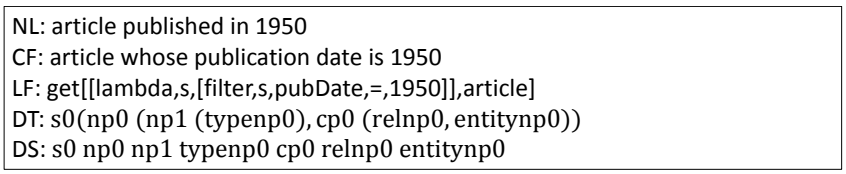

Figure 1: Example of natural language utterance (NL) from the SPO dataset and associated representations considered in this work. CF: canonical form, LF: logical form, DT: derivation tree, DS: derivation sequence.

one of the three sequentializations. In the three cases, the LSTM predictor cannot on its own ensure the grammaticality of the predicted sequence, so that some sequences do not lead to well-formed LFs. However, in the DSP case (in contrast to LFP and CFP), it is easy to integrate inside the LSTM predictor local constraints which guarantee that only grammatical sequences will be produced.

In summary, the contribution of our paper is twofold. Firstly, we propose to use sequence prediction for semantic parsing. Our experimental results show some significant improvements over previous systems. Secondly, we propose to predict derivation sequences taking into account grammatical constraints and we show that the model performs better than sequence prediction models not exploiting this knowledge. These results are obtained without employing any reranking or linguistic features such as POS tags, edit distance, paraphrase features, etc., which makes the proposed methodology even more promising.

2 Background on SPO

The SPO paper (Wang et al., 2015) proposes an approach for quickly developing semantic parsers for new knowledge bases and domains when no training data initially exists. In this approach, a small underlying grammar is used to generate canonical forms and pair them with logical forms. Crowdsourcing is then used to paraphrase each of these canonical forms into several natural utterances. The crowdsourcing thus creates a dataset (SPO dataset in the sequel) consisting of (NL, CF, LF) tuples where NL is a natural language question with CF and LF the canonical and the logical form associated with this question.

SPO learns a semantic parser on this dataset by firstly learning a log-linear similarity model based on a number of features (word matches, ppdb matches, matches between semantic types and POSs, etc.) between NL and the corresponding (CF, LF) pair. At decoding time, SPO parses a natural utterance NL by searching among the derivations of the grammar for one for which the projected (CF, LF) is most similar to the NL based on this log-linear model. The search is based on a so-called "floating parser" (Pasupat and Liang, 2015), a modification of a standard chart-parser, which is able to guide the search based on the similarity features.

In contrast, our approach does not search among the derivations for the one that maximizes a match with the NL, but instead directly tries to predict a decision sequence that can be mapped to the LF.

The SPO system together with its dataset were released to the public[1] and our work exploits this release.

3 Approach

3.1 Grammars and Derivations

```
s0:  s(S) → np(S).
np0: np(get[CP,NP]) → np(NP), cp(CP).
np1: np(NP) → typenp(NP).
cp0: cp([lambda,s,[filter,s,RELNP,=,ENTNP]]) →
       [whose], relnp(RELNP), [is], entitynp(ENTNP).
...
typenp0:   typenp(article) → [article].
relnp0:    relnp(pubDate) → [publication, date]
entitynp0: entitynp(1950) → [1950].
...
```

Figure 2: Some general rules (top) and domain-specific rules (bottom) in DCG format.

The core grammatical resource released by SPO is a generic grammar connecting logical forms with canonical form realizations. They also provide seven domain-specific lexica that can be used in combination with the generic grammar to obtain domain-specific grammars which generate (LF, CF) pairs in each domain, in such a way that LF can then be used to query the corresponding knowledge base. While SPO also released a set of

[1] https://github.com/percyliang/sempre

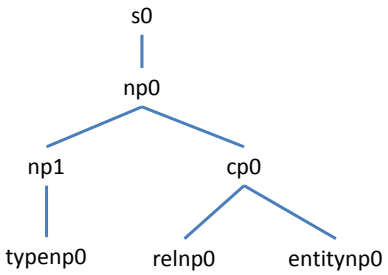

Figure 3: A derivation tree. Its leftmost derivation sequence is [s0, np0, np1, typenp0, cp0, relnp0, entitynp0].

typenp0	*article*
	article
relnp0	*publication date*
	pubDate
entitynp0	*1950*
	1950
cp0	*whose publication date is 1950*
	[lambda,s,[filter,s,pubDate,=,1950]
np1	*article*
	article
np0	*article whose publication date is 1950*
	get[[lambda,s,[filter,s,pubDate,=,1950]],article]
s0	*article whose publication date is 1950*
	get[[lambda,s,[filter,s,pubDate,=,1950]],article]

Figure 4: Projection of the derivation tree nodes into (i) a canonical form and (ii) a logical form.

Java-based parsers and generators for these grammars, for our own purposes we found it convenient to translate the grammars into the formalism of Definite Clause Grammars (Pereira and Warren, 1980), a classical unification-based extension of CFGs, which — through a standard Prolog interpreter such as SWIPL[2] — provide direct support for jointly generating textual realizations and logical forms and also for parsing text into logical forms; we found this translation process to be rather straightforward and we were able to cover all of the SPO grammars.

Figure 2 lists a few DCG rules, general rules first, then lexical rules, for the SPO "publications" domain. Nonterminals are indicated in bold, terminals in italics. We provide each rule with a unique identifier (e.g. s0, np0, ...), which is obtained by concatenating the name of its head nonterminal with a position number relative to the rules that may expand this nonterminal; we can then consider that the nonterminal (e.g. **np**) is the "type" of all its expanding rules (e.g. np0, np1, ...).

According to standard DCG notation, uppercase items `S, NP, CP, RELNP, ENTNP` denote unification variables that become instantiated during processing. In our case unificaion variables range over logical forms and each nonterminal has a single argument denoting a partially instantiated associated logical form. For instance, in the cp0 rule, **`relnp`** is associated with the logical form `RELNP`, **`entitynp`** with the logical form `ENTNP`, and the LHS nonterminal **cp** is then associated with the logical form `[lambda, s, [filter, s, RELNP, =, ENTNP]]`.[3]

In Figure 3, we display a *derivation tree* DT (or simply *derivation*) relative to this grammar, where each node is labelled with a rule identifier. This tree projects on the one hand onto the canonical form *article whose publication date is 1950*, on the other hand onto the logical form `get[[lambda,s,[filter,s,pubDate,=,1950]],article]`.

Figure 4 shows how these projections are obtained by bottom-up composition. For instance, the textual projection of node cp0 is obtained from the textual representations of nodes relnp0 and entitynp0, according to the RHS of the rule cp0, while its logical form projection is obtained by instantiation of the variables `RELNP` and `ENTNP` respectively to the LFs associated with relnp0 and entitynp0.

Relative to these projections, one may note a fundamental difference between derivation trees DT and their projections CF and LF: while the well-formedness of DT can simply be assessed locally by checking that each node expansion is valid according to the grammar, there is in principle no such easy, local, checking possible for the canonical or the logical form; in fact, in order to check the validity of a proposed CF (resp. LF), one needs to *search* for *some* DT that projects onto this CF (resp LF). The first process, of course, is known as "parsing", the second process as "generation". While parsing has polynomial complexity for grammars with a context-free backbone such as the ones considered here, deciding whether a logical form is well-formed or not could in principle be undecidable for certain forms of LF composition.[4]

[2]http://www.swi-prolog.org/

[3]This logical form is written here in DCG list notation; in the more "Lispian" format used by SPO, it would be written `(lambda s (filter s RELNP = ENTNP))`.

[4]The term 'projection' is borrowed from the notion of *bimorphism* in formal language theory (Shieber, 2014) and refers in particular to the fact that the overall logical form is constructed by bottom-up composition of logical forms asso-

To be able to leverage sequence prediction models, we can associate with each derivation tree DT its leftmost *derivation sequence* DS, which corresponds to a preorder traversal of the tree. For the tree of Figure 3, this sequence is [s0, np0, np1, typenp0, cp0, relnp0, entitynp0]. When the grammar is known (in fact, as soon as the CFG core of the grammar is known), two properties of the DS hold (we omit the easy algorithms underlying these properties; they involve using a prefix of the DS for constructing a partial derivation tree in a top-down fashion):

1. knowing the DS uniquely identifies the derivation tree.

2. knowing a prefix of the DS (for instance [s0, np0, np1, typenp0]) completely determines the *type* of the next item (here, this type is **cp**).

The first property implies that if we are able to predict DS, we are also able to predict DT, and therefore also LF and CF. The second property implies that the sequential prediction of DS is strongly constrained by *a priori* knowledge of the underlying grammar: instead of having to select the next item among all the possible rules in the grammar, we only have to select among those rules that are headed by a specific nonterminal. Under a simple condition on the grammar (namely that there are no "unproductive" rules, rules that can never produce an output[5]), following such constrained selection for the next rule guarantees that the derivation sequence will always lead to a valid derivation tree.

At this point, a theoretical observation should be made: *there is no finite-state mechanism on the sequence of rule-names that can control whether the next rule-name is valid or not*.[6] The relevance of that observation for us is that the RNNs that we use are basically finite-state devices (with a huge number of states, but still finite-state), and therefore we do not expect them in principle to be able to always produce valid derivation sequences *unless* they can exploit the underlying grammar for constraining the next choice.

ciated with lower nodes in the derivation tree. In our DCG grammars, this composition actually involves more complex operations (such as "beta-reduction") than the simple copyings illustrated in the small excerpt of Fig. 2.

[5]The general grammar ensures a good coverage of possible logical and canonical forms. However, when this general grammar is used in particular domains, some rules are not relevant any more (i.e. become "unproductive"), but these can be easily eliminated at compile time.

[6]This is easy to see by considering a CFG generating the non finite-state language $a^n b^n$.

3.2 Sequence prediction models

In all these models, we start from a natural utterance NL and we predict a sequence of target items, according to a common sequence prediction architecture that will be described in section 3.3.

3.2.1 Predicting logical form (LFP model)

The most direct approach is to directly predict a linearization of the logical form from NL, the input question. While an LF such as that of Figure 1 is really a structured object respecting certain implicit constraints (balanced parentheses, consistency of the variables bound by lambda expressions, and more generally, conformity with the underlying grammar), the linearization treats it simply as a sequence of tokens: `get [ [ lambda s [ filter s pubDate = 1950 ] ] article ]`. At training time, the LFP model only sees such sequences, and at test time, the next token in the target sequence is then predicted without taking into account any structural constraints. The training regime is the standard one attempting to minimize the cross-entropy of the model relative to the logical forms in the training set.

3.2.2 Predicting derivation sequence (DSP-X models)

Rather than predicting LF directly, we can choose to predict a derivation sequence DS, that is, a sequence of rule-names, and then project it onto LF. We consider three variants of this model.

DSP This basic derivation sequence prediction model is trained on pairs (NL, DS) with the standard training regime. At test time, it is possible for this model to predict ill-formed sequences, which do not correspond to grammatical derivation trees, and therefore do not project onto any logical form.

DSP-C This is a Constrained variant of DSP where we use the underlying grammar to constrain the next rule-name. We train this model exactly as the previous one, but at test time, when sampling the next rule-name inside the RNN, we reject any rule that is not a possible continuation.

DSP-CL This last model is also constrained, but uses a different training regime, with Constrained Loss. In the standard learning regime (used for the

Target sequence	DS	CF	LF
Length	10.5	11.8	47.0
Vocabulary Size	106.0	55.8	59.9

Table 1: Characteristics of different target sequences.

two previous models), the incremental loss when predicting the next item y_t of the sequence is computed as $-\log p(y_t)$, where $p(y_t)$ is the probability of y_t according to the RNN model, normalized (through the computation of a softmax) over *all* the potential values of y_t (namely, here, all the rules in the grammar). By contrast, in the CL learning regime, the incremental loss is computed as $-\log p'(y_t)$, where $p'(y_t)$ is normalized *only over the values of* y_t *that are possible continuations* once the grammar-induced constraints are taken into account, ignoring whatever weights the RNN predictor may (wrongly) believe should be put on impossible continuations. In other words, the DSP-CL model incorporates the prior knowledge about well-formed derivation sequences that we have thanks to the grammar. It computes the actual cross-entropy loss according to the underlying generative process of the model that is used once the constraints are taken into account.

3.2.3 Predicting canonical form (CFP model)

The last possibility we explore is to predict the sequence of words in the canonical form CF, and then use our grammar to parse this CF into its corresponding LF, which we then execute against the knowledge base.[7]

Table 1 provides length and vocabulary-size statistics for the LFP, DSP and CFP tasks.

We see that, typically, for the different domains, DS is a shorter sequence than LF or CF, but its vocabulary size (i.e. number of rules) is larger than that of LF or CF. However DS is unique in allowing us to easily validate grammatical constraints. We also note that the CF is less lengthy than the LF, which uses a number of non "word-like" symbols such as parentheses, lambda variables, and the like.

[7] Although the general intention of SPO is to unambiguously reflect the logical form through the canonical form (which is the basis on which Turkers provide their paraphrases), we do encounter some cases where, although the CF is well-formed and therefore parsable by the grammar, several parses are actually possible, some of which do not correspond to queries for which the KB can return an answer. In these cases, we return the first parse whose logical form does return an answer. Such situations could be eliminated by refining the SPO grammar to a moderate extent, but we did not pursue this.

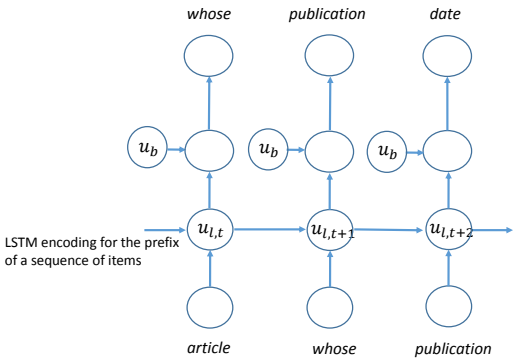

Figure 5: Our neural network model which is shared between all the systems. An MLP encodes the sentence in unigrams and bigrams and produces u_b. An LSTM encodes the prefix of the predicted sequence generating $u_{l,t}$ for each step t. The two representations are then fed into a final MLP to predict the next choice of the target sequence.

3.3 Sequence prediction architecture

3.3.1 Neural network model

The goal of our neural network is to estimate the conditional probability $p(y_1, \ldots, y_{T'}|x_1, \ldots, x_T)$ where $(x_1, \ldots, x_T)$ is a natural language question and $(y_1, \ldots, y_{T'})$ is a target sequence (linearized LF, CF or derivation sequence). In all three cases, we use the same neural network model, which we explain in this subsection.

Suppose that the content of the NL is captured in a real-valued vector u_b, while the prefix of the target sequence up to time t is captured in another real-valued vector $u_{l,t}$. Now, the probability of the target sequence given the input question can be estimated as:

$$p(y_1, \ldots y_{T'}|x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t|u_b, y_1, \ldots y_{t-1})$$
$$= \prod_{t=1}^{T'} p(y_t|u_b, u_{l,t-1})$$

In all our systems, the u_b capturing the content of the NL is calculated from the concatenation of a vector u_1 reading the sentence based on unigrams and another vector u_2 reading the sentence based on bigrams. Mathematically, $u_1 = \tanh(W_1 v_1)$ where v_1 is the 1-hot unigram encoding of the NL

and $u_2 = \tanh(W_2 v_2)$ where v_2 is its 1-hot bigram encoding. Then $u_b = \tanh(Wu)$, where u is the concatenation of u_1 and u_2. W_1, W_2 and W are among the parameters to be learnt. For regularization purposes, a dropout procedure (Srivastava et al., 2014) is applied to u_1 and u_2.

The prefix of the target sequence up to time t is modelled with the vector $u_{l,t}$ generated by the latest hidden state of an LSTM (Hochreiter and Schmidhuber, 1997); LSTM is appropriate here in order to capture the long distance dependencies inside the target sequence. The vector $u_{l,t}$ is then concatenated with u_b (forming u_{bl} in the equation below) before passing through a two-layer MLP (Multi-Layer Perceptron) for the final prediction:

$$p(y_{t+1}|u_{l,t}, u_b) = \text{softmax}(W_2' \tanh(W_1' u_{bl}))$$

Using deep structures such as this MLP for RNN prediction has been shown to be beneficial in previous work (Pascanu et al., 2013).

The overall network architecture is summarized in Figure 5. We train the whole network to minimize the cross entropy between the predicted sequence of items and the reference sequence.

This network architecture can easily support other representations for the input sentence than unigrams and bigrams, as long as they are real-valued vectors of fixed length. We can just concatenate them with u_1 and u_2 and generate u_b as previously. In fact, in initial experiments, we did concatenate an additional representation which reads the sentence through an LSTM, but the performance was not improved.

3.3.2 Decoding the target sequence

We implemented a uniform-cost search algorithm (Russell and Norvig, 2003) to decode the best decision sequence as the sequence with the highest probability. The algorithm finishes in a reasonable time for two reasons: 1) as indicated by Table 1, the vocabulary size of each domain is relatively small, and 2) we found that our model predicts relatively peaked distributions. Of course, it would also be easy to use a beam-search procedure, for situations where these conditions would not hold.

4 Experiments

4.1 Setup

We conduct our experiments on the SPO dataset. To test the overall performance of a semantic parser, the SPO dataset contains seven domains focusing on different linguistic phenomena such as multi-arity relations, sublexical compositionality etc. The utterances in each domain are annotated both with logical forms (LFs) and canonical forms (CFs). The number of such utterances vary from 800 to 4000 depending on the domain. The size of training data is indeed small but as the target vocabulary is always in the domain, thus very small as well, it is actually possible to learn a reasonable semantic parser.

In the SPO dataset, the natural utterances were split randomly into 80%-20% for training and test, and we use the same sets. We perform an additional 80%-20% random split on the SPO training data and keep the 20% as development set to choose certain hyperparameters of our model. Once the hyperparameters are chosen, we retrain on the whole training data before testing.

For LFP experiments, we directly tokenize the LF, as explained earlier, and for CFP experiments we directly use the CF. For DSP experiments (DSP, DSP-C, DSP-CL) where our training data consist of (NL, DS) pairs, the derivation sequences are obtained by parsing each canonical form using the DCG grammar of section 3.

We compare our different systems to SPO. While we only use unigram and bigram features on the NL, SPO uses a number of features of different kinds: linguistic features on NL such as POS tags, lexical features computing the similarity between words in NL and words in CF, semantic features on types and denotations, and also features based on PPDB (Ganitkevitch et al., 2013).

At test time, like SPO, we evaluate our system on the proportion of questions for which the system is able to find the correct answer in the knowledge base.

4.2 Implementation details

We choose the embedding vectors u_1 for unigrams and u_2 for bigrams to have 50 dimensions. The vector u_b representing the sentence content has 200 dimensions. The word embedding layer has 100 dimensions, which is also the case of the hidden layer of the LSTM $u_{l,t}$. Thus u_{bl} which is the concatenation of u_b and $u_{l,t}$ has 300 dimensions and we fix the next layer to u_{bl} to have 100 dimensions. The model is implemented in Keras[8] on top of Theano (Bergstra et al., 2010). For all the exper-

[8] https://github.com/fchollet/keras

iments, we train our models using rmsprop (Tieleman and Hinton., 2012) as the backpropagation algorithm[9]. We use our development set to select the number of training epochs, the dropout factor over unigrams representation and the dropout factor over bigrams representation, by employing a grid search over these hyperparameters: epochs in $\{20, 40, 60\}$, unigrams dropout in $\{0.05, 0.1\}$ and bigrams dropout in $\{0.1, 0.2, 0.3\}$.

4.3 Experimental results

4.3.1 Results on test data

Table 2 shows the test results of SPO and of our different systems over the seven domains.

It can be seen that all of our sequence-based systems are performing better than SPO by a large margin on these tests. When averaging over the seven domains, our 'worst' system DSP scores at 64.7% compared to SPO at 57.1%.

We note that these positive results hold despite the fact that DSP has the handicap that it may generate ungrammatical sequences relative to the underlying grammar, which do not lead to interpretable LFs. The LFP and CFP models, with higher performance than DSP, also may generate ungrammatical sequences.

The best results overall are obtained by the DSP-C system, which does take into account the grammatical constraints. This model performs not only considerably better than its DSP baseline (72.7% over 64.7%), but also better than the models LFP and CFP. Somewhat contrary to our expectations, the DSP-CL model, which exploits constraints not only during decoding, but also during training, performs somewhat worse than the DSP-C, which only exploits them during decoding.

We note that, for all the sequence based models, we strictly base our results on the performance of the *first* sequence predicted by the model. It would probably be possible to improve them further by reranking n-best sequence lists using a set of features similar to those used by SPO.

4.4 Analysis of results

4.4.1 Grammatical errors

We just observed that CFP and LFP perform well on test data although the sequences generated are not guaranteed to be grammatical. We analysed the percentage of grammatical errors made by these models and also by DSP for three domains, which we report in Table 3.[10]

	Basketball	Publication	Housing
LFP	6.6	3.7	1.6
CFP	1.8	1.9	2.2
DSP	9.5	11.8	5.8
DSP-C(L)	0.0	0.0	0.0

Table 3: Grammatical error rate of different systems on test.

The table shows that LFP and especially CFP make few grammatical errors while DSP makes them more frequently. For DSP-C and DSP-CL, the error rate is always 0 since by construction, the derivations must be well-formed. Note that as DSP is not constrained by prior knowledge about the grammar, the grammatical error rate can be high – even higher than CFP or LFP because DSP typically has to choose among more symbols, see Table 1.

4.4.2 Difference between DSP-C and DSP-CL

We observed that the DSP-CL model performs somewhat worse than DSP-C in our experiments. While we were a bit surprised by that behavior, given that the DSP-CL has strong theoretical motivations, let us note that the two models are quite different. To stress the difference, suppose that, for a certain prediction step, only two rules are considered as possible by the grammar, among the many rules of the grammar. Suppose that the LSTM gives probabilities 0.004 and 0.006 respectively to these two rules, the rest of the mass being on the ungrammatical rules. While the DSP-C model associates respective losses of $-\log 0.004, -\log 0.006$ with the two rules, the DSP-CL model normalizes the probabilites first, resulting in smaller losses $-\log 0.4, -\log 0.6$.

As we choose the best complete sequence during decoding, it means that DSP-C will be more likely to prefer to follow a different path in such a case, in order not to incur a loss of at least $-\log 0.006$. Intuitively, this means that DSP-C will prefer paths where the LSTM *on its own*

[9] All the hyperparameters of rmsprop as well as options for initializing the neural network are left at their default values in Keras.

[10] Our DCG permits to compute this error rate directly for canonical forms and derivation sequences. For logical forms, we made an estimation by executing them against the knowledge base and eliminating the cases where the errors are not due to the ungrammaticality of the logical form.

	Basketball	Social	Publication	Blocks	Calendar	Housing	Restaurants	Avg
SPO	46.3	48.2	59.0	41.9	74.4	54.0	75.9	57.1
LFP	73.1	70.2	72.0	55.4	71.4	61.9	76.5	68.6
CFP	80.3	79.5	70.2	54.1	73.2	**63.5**	71.1	70.3
DSP	71.6	67.5	64.0	53.9	64.3	55.0	76.8	64.7
DSP-C	80.5	**80.0**	**75.8**	**55.6**	**75.0**	61.9	**80.1**	**72.7**
DSP-CL	**80.6**	77.6	70.2	53.1	**75.0**	59.3	74.4	70.0

Table 2: Test results over different domains on SPO dataset. The numbers reported correspond to the proportion of cases in which the predicted LF is interpretable against the KB *and* returns the correct answer. LFP = Logical Form Prediction, CFP = Canonical Form Prediction, DSP = Derivation Sequence Prediction, DSP-C = Derivation Sequence constrained using grammatical knowledge, DSP-CL = Derivation Sequence using a loss function constrained by grammatical knowledge.

gives small probability to ungrammatical choices, a property not shared by DSP-CL. However, a more complete understanding of the difference will need more investigation.

5 Related Work and Discussion

In recent work on developing semantic parsers for open-domain and domain-specific question answering, various methods have been proposed to handle the mismatch between natural language questions and knowledge base representations including, graph matching, paraphrasing and embeddings techniques.

Reddy et al. (2014) exploits a weak supervision signal to learn a mapping between the logical form associated by a CCG based semantic parser with the input question and the appropriate logical form in Freebase (Bollacker et al., 2008).

Paraphrase-based approaches (Fader et al., 2013; Berant and Liang, 2014) generate variants of the input question using a simple hand-written grammar and then rank these using a paraphrase model. That is, in their setting, the logical form assigned to the input question is that of the generated sentence which is most similar to the input question.

Finally, Bordes et al. (2014b; 2014a) learn a similarity function between a natural language question and the knowledge base formula encoding its answer.

We depart from these approaches in that we learn a direct mapping between natural language questions and their corresponding logical form or equivalently, their corresponding derivation and canonical form. This simple, very direct approach to semantic parsing eschews the need for complex feature engineering and large external resources required by such paraphrase-based approaches as (Fader et al., 2013; Berant and Liang, 2014). It is conceptually simpler than the two steps, graph matching approach proposed by Reddy et al. (2014). And it can capture much more complex semantic representations than Bordes et al. (2014b; 2014a)'s embeddings based method.[11]

At a more abstract level, our approach differs from previous work in that it exploits the fact that logical forms are structured objects whose shape is determined by an underlying grammar. Using the power of RNN as sequence predictors, we learn to predict, from more or less explicit representations of this underlying grammar, equivalent but different representations of a sentence content namely, its canonical form, its logical form and its derivation sequence.

We observe that the best results are obtained by using the derivation sequence, when also exploiting the underlying grammatical constraints. However the results obtained by predicting directly the linearization of the logical form or canonical form are not far behind; we show that often, the predicted linearizations actually satisfy the underlying grammar. This observation can be related to the results obtained by Vinyals et al. (2014), who use an RNN-based model to map a sentence to the linearization of its parse tree,[12] and find that in most cases, the predicted sequence produces well-balanced parentheses. It would be interest-

[11] In (Bordes et al., 2014b; Bordes et al., 2014a), the logical forms denoting the question answers involve only few RDF triples consisting of a subject, a property and an object i.e., a binary relation and its arguments.

[12] Note a crucial difference with our approach. While in their case the underlying ("syntactic") grammar is only partially and implicitly represented by a set of parse annotations, in our case the explicit ("semantic") grammar is known *a priori* and can be exploited as such.

ing to see if our observation would be maintained for more complex LFs than the ones we tested on, where it might be more difficult for the RNN to predict not only the parentheses, but also the dependencies between several lambda variables inside the overall structure of the LF.

6 Conclusion and Future Work

We propose a sequence-based approach for the task of semantic parsing. We encode the target logical form, a structured object, through three types of sequences: direct linearization of the logical form, canonical form, derivation sequence in an underlying grammar. In all cases, we obtain competitive results with previously reported experiments. The most effective model is one using derivation sequences and taking into account the grammatical constraints.

In order to encode the underlying derivation tree, we chose to use a leftmost derivation sequence. But there are other possible choices that might make the encoding even more easily learnable by the LSTM, and we would like to explore those in future work.

In order to improve performance, other promising directions would involve adding re-reranking techniques and extending our neural networks with attention models in the spirit of (Bahdanau et al., 2015).

Acknowledgements

We would like to thank Guillaume Bouchard for his advice on a previous version of this work, as well as the anonymous reviewers for their constructive feedback. The authors gratefully acknowledge the support of the Association Nationale de la Recherche Technique (ANRT), Convention Industrielle de Formation par la Recherche (CIFRE) No. 2014/0476 .

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

J. Berant and P. Liang. 2014. Semantic parsing via paraphrasing. In *Annual Meeting for the Association for Computational Linguistics (ACL)*.

J. Berant, A. Chou, R. Frostig, and P. Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, jun. Oral.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250.

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

Antoine Bordes, Jason Weston, and Nicolas Usunier. 2014b. Open question answering with weakly supervised embedding models. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML-PKDD)*.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1608–1618.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. Ppdb: The paraphrase database. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 758–764.

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

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke S. Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Empirical Methods in Natural Language Processing, (EMNLP)*, pages 1545–1556.

Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2013. How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL Volume 1: Long Papers*.

Fernando C.N. Pereira and David H.D. Warren. 1980. Definite clause grammars for language analysis a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231 – 278.

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

Stuart J. Russell and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

Stuart M. Shieber. 2014. Bimorphisms and synchronous grammars. *J. Language Modelling*, 2(1):51–104.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958.

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

T. Tieleman and G. E. Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2014. Grammar as a foreign language. *Neural Information Processing Systems (NIPS)*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL Volume 1: Long Papers*, pages 1332–1342.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Peihao Su, David Vandyke, and Steve J. Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Empirical Methods in Natural Language Processing (EMNLP)*.