Grammar-Constrained Neural Semantic Parsing with LR Parsers

Artur Baranowski

Technische Hochschule Köln
Institut für Nachrichtentechnik

German Aerospace Center (DLR)
Institute for Software Technology

artur.baranowski@mailbox.org

Nico Hochgeschwender

Hochschule Bonn-Rhein-Sieg
Department of Computer Science

German Aerospace Center (DLR)
Institute for Software Technology

nico.hochgeschwender@h-brs.de

Abstract

Target meaning representations for semantic parsing tasks are often based on programming or query languages, such as SQL, and can be formalized by a context-free grammar. Assuming a priori knowledge of the target domain, such grammars can be exploited to enforce syntactical constraints when predicting logical forms. To that end, we assess how syntactical parsers can be integrated into modern encoder-decoder frameworks. Specifically, we implement an attentional SEQ2SEQ model that uses an LR parser to maintain syntactically valid sequences throughout the decoding procedure. Compared to other approaches to grammar-guided decoding that modify the underlying neural network architecture or attempt to derive full parse trees, our approach is conceptually simpler, adds less computational overhead during inference and integrates seamlessly with current SEQ2SEQ frameworks. We present preliminary evaluation results against a recurrent SEQ2SEQ baseline on GEOQUERY and ATIS and demonstrate improved performance while enforcing grammatical constraints.

1 Introduction

Semantic parsing aims at delivering granular, structured representations of natural language utterances, referred to as *meaning representations* or *logical forms*. Thus, it goes beyond shallow semantic analysis involving argument identification and role labeling (Collobert et al., 2011; Roth and Lapata, 2016). Meaning representations based on programming or query languages (PYTHON, SQL) are describable by (deterministic) context-free grammars and used for general purpose code generation (Xiao et al., 2016; Yin and Neubig, 2017). Our work targets this particular subset of logical forms. A context-free grammar may be exploited to constrain a semantic parser to only produce token sequences derivable from the grammar. Specifically, we investigate how syntax constraints can be enforced in semantic parsers based on modern encoder-decoder frameworks in a non-intrusive, computationally inexpensive way at inference time. We show that enforcing grammatical constraints with LR parsers is particularly well suited for modern autoregressive neural network architectures used in neural machine translation (Sutskever et al., 2014; Vaswani et al., 2017). We do not require any modifications to standard SEQ2SEQ neural network architectures and make very little assumptions about the inputs and outputs of such models. In contrast, most grammar-constrained decoders attempt to model the grammar explicitly within the neural network, complicating the architecture. Moreover, they predict complete syntax trees or derivation sequences. Our approach predicts source code token streams, preserving syntactic validity throughout the decoding procedure. Enforcing syntactical constraints relieves neural networks models from having to learn the syntactic structure of the target language, which is particularly beneficial for ensuring balanced expressions over long ranges (Bahdanau et al., 2014; Ling et al., 2016). Also, when integrating our models into larger application environments, we may want to preclude specific failure modes (i.e., syntax errors) when executing the generated program snippets to increase robustness. Preliminary evaluation results on the GEOQUERY and ATIS data sets demonstrate that simply enforcing syntactical constraints on the pre-

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 1275–1279

dicted lexical tokens at inference time improves the performance of the semantic parser against a recurrent SEQ2SEQ baseline.

2 Related Work

Enforcing grammatical constraints within neural network models has sparked a fair amount of research interest. (Xiao et al., 2016) take a derivational viewpoint when decoding derivation trees, demonstrating improved performance when accounting for grammatical constraints. They predict leftmost derivation sequences, each uniquely associated with a corresponding derivation tree. Employing a constrained loss over probabilities $p'(\hat{\mathbf{y}}_t)$, where $\hat{\mathbf{y}}_t$ are the permissible continuations of a derivation sequence, constraints are enforced at training time. We take inspiration from (Xiao et al., 2016), however, enforce grammatical constraints at inference time and on lexical token streams in a bottom-up fashion, eliminating the need to derive entire syntax trees and effectively reducing the sequence length. Similarly, (Yin and Neubig, 2017) predict entire syntax trees sequentially using a SEQ2SEQ model, starting from the root node and generating tree nodes in depth-first, left-to-right order, deterministically converting them to the corresponding surface code. They define a dedicated grammar model that predicts action sequences that either apply a production rule or generate a lexical token. (Krishnamurthy et al., 2017) additionally ensure that decoder predictions satisfy type constraints by providing a type-constrained grammar. (Rabinovich et al., 2017) propose a decoder that employs a separate neural network module for each construct in the grammar. The decoder generates an abstract syntax tree (AST) through mutual recursion between modules. At each decoding step, the decoder either generates a symbol or propagates the decoder state to the next module. (Yin and Neubig, 2018) developed a transition-based abstract syntax parser (TRANX) guided by a grammar specified under ASDL (Wang et al., 1997). TRANX uses ASTs as general-purpose intermediate meaning representations, decoupling the semantic parsing procedure from domain-specific grammars. A user-defined grammar converts ASTs to domain-specific meaning representations. Similar to (Yin and Neubig, 2017) an AST is generated using a sequence of tree-constructing actions. All approaches enforce syntactical constraints by first predicting the tree-structured syntax tree top-down. Instead, we propose to directly generate lexical tokens (the values of syntax tree leaves) and constrain the decoding process by means of an bottom-up LR parser.

3 Problem Statement

Informally, we aim at translating a set of natural language utterances X to a structured representation of their meaning. We assume the syntax of target meaning representations is describable by a deterministic context-free grammar and that it is known at training time. Given a grammar G, our goal is to enforce the constraints imposed by G during decoding. That is, the image of our model f shall be the language generated by G.

$$f\colon X \rightarrow L(G) \tag{1}$$

We achieve this by means of a recurrent encoder-decoder model as proposed by (Sutskever et al., 2014) and an LR parser. We briefly introduce recurrent encoder-decoder NMT models and the specifics of our model.

3.1 SEQ2SEQ Model

All modern encoder-decoder frameworks define a probability distribution $P(\mathbf{y}|\mathbf{x})$ (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014), where in our case, $\mathbf{x}$ represents a natural language input. For a target source code string, $\mathbf{y}$ represents the token stream generated by a lexical analyzer and a corresponding lexical grammar (see section 3.2). $P(\mathbf{y}|\mathbf{x})$ is factorized as:

$$p(y_1, ..., y_\kappa|\mathbf{x}) = \prod_{t=1}^{\kappa} p(y_t|\mathbf{x}, y_1, ..., y_{t-1}) \tag{2}$$

The encoder portion of the neural network encodes $\mathbf{x}$ into a vector-valued, so-called *context*. Conditioned on the context and all previous decoder hidden states, the decoder generates the output tokens $y = (y_1, ..., y_\kappa)$. Both encoder and decoder are distinct recurrent neural networks (LSTM's in our case). The decoder generates a sequence of hidden states and outputs a hidden state $\mathbf{h}_t^L$ at the topmost, L-th layer at timestep t. The individual factors in Eq. 2 are finally obtained using a feedforward neural network with a softmax layer that maps each hidden state to a probability distribution over the token vocabulary V_D of the decoder. We optimize standard cross-entropy loss.

3.2 Grammar-Constrained Decoder

LR parsers are used to verify that a given token stream is derivable from a deterministic context-free grammar G. The parsing stage is usually preceded by lexical analysis. During lexical analysis, a source code string is converted into a sequence of tokens, ready to be consumed by the parser. LR parsers employ ACTION and GOTO tables associated with the grammar, governing the applicable shift-reduce decisions the parser can make on each token input and determining the error states of the parser. The decoder stage in the SEQ2SEQ model can be viewed as taking the role of the lexical analyzer in the parsing process (see Figure 1).

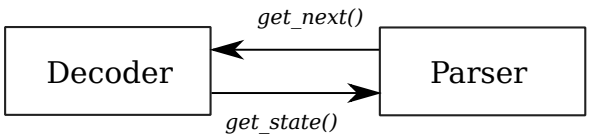

Figure 1: The parser requests the next token from the decoder. The decoder queries the current parser state to determine the set of applicable tokens, generates a token and returns it to the parser.

Algorithm 1 : LR(1) Assisted Decoder

```
let s ← top of stack state;
let h ← encoder context vector;
let a ← Token(SOS, '<SOS>');

while ACTION(s, a) ≠ acc do
    let e_t ← EXPECTED(s);
    if |e_t| > 1: h, a ← DECODE(h, e_t, a);
    else: a ← e_t;

    if ACTION(s, a) = shift i:
        push state i onto stack;
    elif ACTION(s, a) = reduce [i, A → β]:
        pop |β| symbols off stack;
        push GOTO(i, A) onto stack;

    s ← top of stack state;
```

The decoder determines the set of applicable tokens in its vocabulary by consulting the parser's ACTION table. We generate a probability distribution as described in section 3.1 over the actions (identified by lexical tokens) in the current parser state and return the most likely token to the parser. The parser consumes the token, updates its state, and requests the next token. This process continues until the parser encounters a token that indicates acceptance (an EOF token "$"). The neural network model is implemented using PYTORCH [1]. The parser implementation relies on the parsing toolkit LARK [2]. The output vocabulary V_D consists of all source code tokens defined by the given grammar. Literals, such as string or integer values, are usually tokenized by matching them with a regular expression. We explicitly include all occurrences of literal values in the data sets as distinct tokens in the vocabulary. Algorithm 1 describes our modified LR parsing procedure relying on the decoder module providing the token stream. The procedure is initialized with the context vector obtained from the encoder and the parser start state. Given the state s, we determine the set of possible tokens e_t by looking in the parser's ACTION table. Conditioned on the previous hidden state, we invoke the DECODE function and generate an output distribution $\hat{\mathbf{y}}$ over *all* output vocabulary tokens. We finally choose $a = \max(\hat{\mathbf{y}}_{e_t})$ as our prediction, where $\hat{\mathbf{y}}_{e_t}$ are the elements of $\hat{\mathbf{y}}$ indexed by e_t. The next hidden state h_t is returned, and the parser updates its state by parsing a. On shift actions, we push the associated state i onto the stack and request the next token. On reduce actions, we pop the recognized handle off the stack and push the left-hand side of the production onto the stack. The decoding procedure concludes when the parser encounters a token that indicates acceptance (corresponding to action "acc"). Note, that the decoder is invoked only if $|e_t| > 1$, i.e., when there is more than one applicable token. Otherwise, we simply set $a = e_t$. The additional computational overhead of running a single parsing step is constant at each decoding step. Although most programming languages are close to deterministic, generalizing our approach to GLR parsers (and thus to context-free grammars) may incur an additional computational cost proportional to the non-determinism in the grammar (Tomita, 1985).

3.3 Model Training

Algorithm 1 is only used during inference. Thus, during model training, the decoder may generate sequences $s \notin L(G)$. Furthermore, since the decoder is only invoked when $|e_t| > 1$, there is a mismatch between sequences seen during training and during test time. To account for this mismatch, each target sample is parsed prior to training, and

[1] https://github.com/pytorch
[2] https://github.com/lark-parser/lark

for each state s for which $\text{EXPECTED}(s) = 1$, we filter the corresponding target sequence element from the target sample.

4 Experimental Evaluation

We present preliminary evaluation results and compare our approach to a recurrent SEQ2SEQ baseline (see section 3.1) and an attentional SEQ2SEQ model as reported in (Finegan-Dollak et al., 2018). Our attentional model extends the recurrent baseline with an attention layer as proposed by (Bahdanau et al., 2014).

4.1 Datasets

For our trials we use the canonicalized and annotated semantic parsing data sets for text-to-SQL tasks provided by (Finegan-Dollak et al., 2018). Compared to data sets like WIKISQL, GEOQUERY and ATIS feature complex queries with low levels of redundancy. We hypothesize that the benefits of a grammar-constrained decoder will be particularly pronounced in data sets with high complexity and variability. To ensure comparability, we use identical training, validation and test splits as (Finegan-Dollak et al., 2018).

4.2 Setup

We run trials without entity anonymization and with anonymized entities. We refer to trials with the standard dataset, i.e., the trials without anonymized entities, as *standard* trials. Trials with entity anonymization are referred to as *oracle* trials. Greedy-search was used for generating output sequences. We measured the exact match classification accuracy. A predicted token sequence that is identical to the token sequence in the corresponding test set example constitutes an exact match. Stochastic gradient descent with momentum (0.9) and a learning rate of 0.1 was used for each trial. The batch sizes ($\{16, 32, 64\}$ for GEOQUERY and $\{128, 256\}$ for ATIS), hidden and embedding dimensions ($\{64, 96, 128, 256\}$), the dropout rate for embeddings and hidden units ($\{0.05, 0.1, 0.2, 0.4\}$), the number of layers ($\{1, 2\}$) and the teacher-forcing ratio ($\{1.0, 0.9, 0.8, 0.7\}$) were determined using grid search. We tested the models with best validation set performance during training and set an early stopping criterion.

4.3 Results

In Table 1 and Table 2 we present the results of the evaluation. We see the greatest improvements in the oracle trials without an attention layer. This verifies that the main utility of enforcing syntactical constraints lies with resolving the complex syntactical structures of target logical forms. Correctly recognizing entities and inserting the appropriate literals into the query is more akin to a *slot-filling* task than a semantic parsing task, and we observe no added value in enforcing grammatical constraints to resolve such literals in the standard trials. Applying an attention mechanism to both our approach and the basic recurrent model of (Finegan-Dollak et al., 2018) further puts the results into perspective. An attention layer in recurrent SEQ2SEQ models helps with resolving long range dependencies that may occur when expanding non-terminals, for example, involving long sub-queries (Bahdanau et al., 2014). Similarly, long range dependencies are resolved by virtue of the LR parser, ensuring that any non-terminal node is fully expanded, even if it involves sub-expressions that are expanded to long token sequences. Thus, using an attention mechanism, syntactic relationships between tokens can be learned much better, although syntax errors cannot be completely precluded as with an LR parser.

	GEOQUERY	
	Standard	**Oracle**
Ours	34%	63%
+ Attention	51%	69%
Finegan-Dollak et al.	27%	49%
+ Attention	63%	73%

Table 1: Exact match accuracy on GEOQUERY.

	ATIS	
	Standard	**Oracle**
Ours	9%	48%
+ Attention	33%	55%
Finegan-Dollak et al.	8%	14%
+ Attention	46%	57%

Table 2: Exact match accuracy on ATIS.

5 Conclusion and Future Work

We showed that grammatical constraints can be enforced with LR parsers, imposing no assumptions

on the neural machine translation model used and adding little computational overhead. We intend to expand the trials to include other logical forms than SQL and comparable approaches to enforcing grammatical constraints (Xiao et al., 2016; Yin and Neubig, 2017). Moreover, we intend to generalize our approach to context-free grammars using GLR parsers and enforce grammatical constrains at training time.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA. Association for Computational Linguistics.

Jayant Krishnamurthy, 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, Denmark. Association for Computational Linguistics.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany. Association for Computational Linguistics.

Maxim Rabinovich, 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, Canada. Association for Computational Linguistics.

Michael Roth and Mirella Lapata. 2016. Neural semantic role labeling with dependency path embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1192–1202, Berlin, Germany. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA. MIT Press.

Masaru Tomita. 1985. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers, USA.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Daniel C. Wang, Andrew W. Appel, Jeff L. Korn, and Christopher S. Serra. 1997. The zephyr abstract syntax description language. In *Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997*, DSL'97, page 17, USA. USENIX Association.

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1341–1350, Berlin, Germany. Association for Computational Linguistics.

Pengcheng Yin 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, Canada. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 7–12, Brussels, Belgium. Association for Computational Linguistics.