StructVAE: Tree-structured Latent Variable Models for Semi-supervised Semantic Parsing

Pengcheng Yin, Chunting Zhou, Junxian He, Graham Neubig
Language Technologies Institute
Carnegie Mellon University
{pcyin,ctzhou,junxianh,gneubig}@cs.cmu.edu

Abstract

Semantic parsing is the task of transducing natural language (NL) utterances into formal meaning representations (MRs), commonly represented as tree structures. Annotating NL utterances with their corresponding MRs is expensive and time-consuming, and thus the limited availability of labeled data often becomes the bottleneck of data-driven, supervised models. We introduce StructVAE, a variational auto-encoding model for semi-supervised semantic parsing, which learns both from limited amounts of parallel data, and readily-available unlabeled NL utterances. StructVAE models latent MRs not observed in the unlabeled data as tree-structured latent variables. Experiments on semantic parsing on the Atis domain and Python code generation show that with extra unlabeled data, StructVAE outperforms strong supervised models.[1]

1 Introduction

Semantic parsing tackles the task of mapping natural language (NL) utterances into structured formal meaning representations (MRs). This includes parsing to general-purpose logical forms such as λ-calculus (Zettlemoyer and Collins, 2005, 2007) and the abstract meaning representation (AMR, Banarescu et al. (2013); Misra and Artzi (2016)), as well as parsing to computer-executable programs to solve problems such as question answering (Berant et al., 2013; Yih et al., 2015; Liang et al., 2017), or generation of domain-specific (*e.g.*, SQL) or general purpose programming languages (*e.g.*, Python) (Quirk et al., 2015; Yin and Neubig, 2017; Rabinovich et al., 2017).

[1]Code available at http://pcyin.me/struct_vae

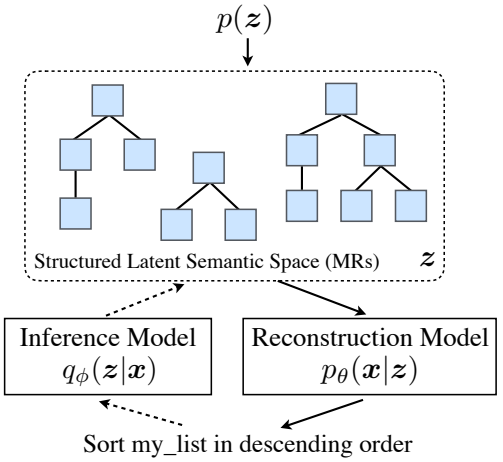


Figure 1: Graphical Representation of StructVAE

While these models have a long history (Zelle and Mooney, 1996; Tang and Mooney, 2001), recent advances are largely attributed to the success of neural network models (Xiao et al., 2016; Ling et al., 2016; Dong and Lapata, 2016; Iyer et al., 2017; Zhong et al., 2017). However, these models are also extremely *data hungry*: optimization of such models requires large amounts of training data of parallel NL utterances and manually annotated MRs, the creation of which can be expensive, cumbersome, and time-consuming. Therefore, the limited availability of parallel data has become the bottleneck of existing, purely supervised-based models. These data requirements can be alleviated with *weakly-supervised* learning, where the denotations (*e.g.*, answers in question answering) of MRs (*e.g.*, logical form queries) are used as indirect supervision (Clarke et al. (2010); Liang et al. (2011); Berant et al. (2013), *inter alia*), or *data-augmentation techniques* that automatically generate pseudo-parallel corpora using hand-crafted or induced grammars (Jia and Liang, 2016; Wang et al., 2015).

In this work, we focus on *semi-supervised* learning, aiming to learn from both limited

amounts of parallel NL-MR corpora, and *unlabeled* but readily-available NL utterances. We draw inspiration from recent success in applying variational auto-encoding (VAE) models in semi-supervised sequence-to-sequence learning (Miao and Blunsom, 2016; Kociský et al., 2016), and propose StructVAE — a principled deep generative approach for semi-supervised learning with tree-structured latent variables (Fig. 1). StructVAE is based on a generative story where the surface NL utterances are generated from tree-structured latent MRs following the standard VAE architecture: (1) an off-the-shelf semantic parser functions as the *inference model*, parsing an observed NL utterance into latent meaning representations (§ 3.2); (2) a *reconstruction model* decodes the latent MR into the original observed utterance (§ 3.1). This formulation enables our model to perform both standard supervised learning by optimizing the inference model (*i.e.*, the parser) using parallel corpora, and unsupervised learning by maximizing the variational lower bound of the likelihood of the unlabeled utterances (§ 3.3).

In addition to these contributions to semi-supervised semantic parsing, StructVAE contributes to generative model research as a whole, providing a recipe for training VAEs with *structured* latent variables. Such a structural latent space is contrast to existing VAE research using *flat* representations, such as continuous distributed representations (Kingma and Welling, 2013), discrete symbols (Miao and Blunsom, 2016), or hybrids of the two (Zhou and Neubig, 2017).

We apply StructVAE to semantic parsing on the Atis domain and Python code generation. As an auxiliary contribution, we implement a transition-based semantic parser, which uses Abstract Syntax Trees (ASTs, § 3.2) as intermediate MRs and achieves strong results on the two tasks. We then apply this parser as the inference model for semi-supervised learning, and show that with extra unlabeled data, StructVAE outperforms its supervised counterpart. We also demonstrate that StructVAE is compatible with different structured latent representations, applying it to a simple sequence-to-sequence parser which uses λ-calculus logical forms as MRs.

2 Semi-supervised Semantic Parsing

In this section we introduce the objectives for semi-supervised semantic parsing, and present high-level intuition in applying VAEs for this task.

2.1 Supervised and Semi-supervised Training

Formally, semantic parsing is the task of mapping utterance $\boldsymbol{x}$ to a meaning representation $\boldsymbol{z}$. As noted above, there are many varieties of MRs that can be represented as either graph structures (*e.g.*, AMR) or tree structures (*e.g.*, λ-calculus and ASTs for programming languages). In this work we specifically focus on tree-structured MRs (see Fig. 2 for a running example Python AST), although application of a similar framework to graph-structured representations is also feasible.

Traditionally, purely supervised semantic parsers train a probabilistic model $p_\phi(\boldsymbol{z}|\boldsymbol{x})$ using parallel data $\mathbb{L}$ of NL utterances and annotated MRs (*i.e.*, $\mathbb{L} = \{\langle \boldsymbol{x}, \boldsymbol{z} \rangle\}$). As noted in the introduction, one major bottleneck in this approach is the lack of such parallel data. Hence, we turn to semi-supervised learning, where the model additionally has access to a relatively large amount of unlabeled NL utterances $\mathbb{U} = \{\boldsymbol{x}\}$. Semi-supervised learning then aims to maximize the log-likelihood of examples in both $\mathbb{L}$ and $\mathbb{U}$:

$$\mathcal{J} = \underbrace{\sum_{\langle \boldsymbol{x}, \boldsymbol{z} \rangle \in \mathbb{L}} \log p_\phi(\boldsymbol{z}|\boldsymbol{x})}_{\text{supervised obj. } \mathcal{J}_s} + \alpha \underbrace{\sum_{\boldsymbol{x} \in \mathbb{U}} \log p(\boldsymbol{x})}_{\text{unsupervised obj. } \mathcal{J}_u} \quad (1)$$

The joint objective consists of two terms: (1) a supervised objective $\mathcal{J}_s$ that maximizes the conditional likelihood of annotated MRs, as in standard supervised training of semantic parsers; and (2) a unsupervised objective $\mathcal{J}_u$, which maximizes the marginal likelihood $p(\boldsymbol{x})$ of unlabeled NL utterances $\mathbb{U}$, controlled by a tuning parameter α. Intuitively, if the modeling of $p_\phi(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{x})$ is coupled (*e.g.*, they share parameters), then optimizing the marginal likelihood $p(\boldsymbol{x})$ using the unsupervised objective $\mathcal{J}_u$ would help the learning of the semantic parser $p_\phi(\boldsymbol{z}|\boldsymbol{x})$ (Zhu, 2005). StructVAE uses the variational auto-encoding framework to jointly optimize $p_\phi(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{x})$, as outlined in § 2.2 and detailed in § 3.

2.2 VAEs for Semi-supervised Learning

From Eq. (1), our semi-supervised model must be able to calculate the probability $p(\boldsymbol{x})$ of unlabeled NL utterances. To model $p(\boldsymbol{x})$, we use VAEs, which provide a principled framework for generative models using neural networks (Kingma and Welling, 2013). As shown in Fig. 1, VAEs define a *generative story* (bold arrows in Fig. 1, explained in § 3.1) to model $p(\boldsymbol{x})$, where a latent MR $\boldsymbol{z}$ is

sampled from a prior, and then passed to the *reconstruction* model to decode into the surface utterance $\boldsymbol{x}$. There is also an *inference* model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ that allows us to infer the most probable latent MR $\boldsymbol{z}$ given the input $\boldsymbol{x}$ (dashed arrows in Fig. 1, explained in § 3.2). In our case, the inference process is equivalent to the task of semantic parsing if we set $q_\phi(\cdot) \triangleq p_\phi(\cdot)$. VAEs also provide a framework to compute an approximation of $p(\boldsymbol{x})$ using the inference and reconstruction models, allowing us to effectively optimize the unsupervised and supervised objectives in Eq. (1) in a joint fashion (Kingma et al. (2014), explained in § 3.3).

3 StructVAE: VAEs with Tree-structured Latent Variables

3.1 Generative Story

StructVAE follows the standard VAE architecture, and defines a generative story that explains how an NL utterance is generated: a latent meaning representation $\boldsymbol{z}$ is sampled from a prior distribution $p(\boldsymbol{z})$ over MRs, which encodes the latent semantics of the utterance. A *reconstruction* model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ then decodes the sampled MR $\boldsymbol{z}$ into the observed NL utterance $\boldsymbol{x}$.

Both the prior $p(\boldsymbol{z})$ and the reconstruction model $p(\boldsymbol{x}|\boldsymbol{z})$ takes tree-structured MRs as inputs. To model such inputs with rich internal structures, we follow Konstas et al. (2017), and model the distribution over a sequential surface representation of $\boldsymbol{z}$, $\boldsymbol{z}^s$ instead. Specifically, we have $p(\boldsymbol{z}) \triangleq p(\boldsymbol{z}^s)$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z}) \triangleq p_\theta(\boldsymbol{x}|\boldsymbol{z}^s)$[2]. For code generation, $\boldsymbol{z}^s$ is simply the surface source code of the AST $\boldsymbol{z}$. For semantic parsing, $\boldsymbol{z}^s$ is the linearized s-expression of the logical form. Linearization allows us to use standard sequence-to-sequence networks to model $p(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. As we will explain in § 4.3, we find these two components perform well with linearization.

Specifically, the prior is parameterized by a Long Short-Term Memory (LSTM) language model over $\boldsymbol{z}^s$. The reconstruction model is an attentional sequence-to-sequence network (Luong et al., 2015), augmented with a copying mechanism (Gu et al., 2016), allowing an out-of-vocabulary (OOV) entity in $\boldsymbol{z}^s$ to be copied to $\boldsymbol{x}$ (*e.g.*, the variable name `my_list` in Fig. 1 and its AST in Fig. 2). We refer readers to Appendix B for details of the neural network architecture.

[2]Linearizion is used by the prior and the reconstruction model only, and not by the inference model.

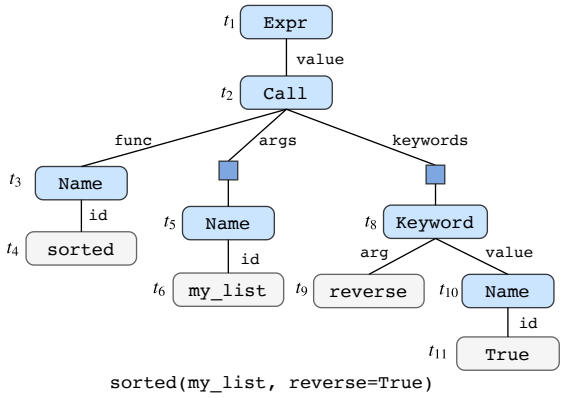
3.2 Inference Model

StructVAE models the semantic parser $p_\phi(\boldsymbol{z}|\boldsymbol{x})$ as the inference model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ in VAE (§ 2.2), which maps NL utterances $\boldsymbol{x}$ into tree-structured meaning representations $\boldsymbol{z}$. $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ can be any trainable semantic parser, with the corresponding MRs forming the structured latent semantic space. In this work, we primarily use a semantic parser based on the Abstract Syntax Description Language (ASDL) framework (Wang et al., 1997) as the inference model. The parser encodes $\boldsymbol{x}$ into ASTs (Fig. 2). ASTs are the native meaning representation scheme of source code in modern programming languages, and can also be adapted to represent other semantic structures, like λ-calculus logical forms (see § 4.2 for details). We remark that StructVAE works with other semantic parsers with different meaning representations as well (*e.g.*, using λ-calculus logical forms for semantic parsing on Atis, explained in § 4.3).

Our inference model is a transition-based parser inspired by recent work in neural semantic parsing and code generation. The transition system is an adaptation of Yin and Neubig (2017) (hereafter YN17), which decomposes the generation process of an AST into sequential applications of tree-construction actions following the ASDL grammar, thus ensuring the syntactic well-formedness of generated ASTs. Different from YN17, where ASTs are represented as a Context Free Grammar learned from a parsed corpus, we follow Rabinovich et al. (2017) and use ASTs defined under the ASDL formalism (§ 3.2.1).

3.2.1 Generating ASTs with ASDL Grammar

First, we present a brief introduction to ASDL. An AST can be generated by applying typed *constructors* in an ASDL grammar, such as those in Fig. 3 for the Python ASDL grammar. Each constructor specifies a language construct, and is assigned to a particular *composite type*. For example, the constructor `Call` has type `expr` (expression), and it denotes function calls. Constructors are associated with multiple *fields*. For instance, the `Call` constructor and has three fields: *func*, *args* and *keywords*. Like constructors, fields are also strongly typed. For example, the *func* field of `Call` has `expr` type. Fields with composite types are instantiated by constructors of the same type, while fields with *primitive* types store values (*e.g.*, identifier names or string literals). Each field also has



t	Frontier Field	Action
t_1	`stmt root`	`Expr(expr value)`
t_2	`expr value`	`Call(expr func, expr* args, keyword* keywords)`
t_3	`expr func`	`Name(identifier id)`
t_4	`identifier id`	GENTOKEN$[sorted]$
t_5	`expr* args`	`Name(identifier id)`
t_6	`identifier id`	GENTOKEN$[my_list]$
t_7	`expr* args`	REDUCE (close the frontier field)
t_8	`keyword* keywords`	`keyword(identifier arg, expr value)`
t_9	`identifier arg`	GENTOKEN$[reverse]$
t_{10}	`expr value`	`Name(identifier id)`
t_{11}	`identifier id`	GENTOKEN$[True]$
t_{12}	`keyword* keywords`	REDUCE (close the frontier field)

Figure 2: ***Left*** An example ASDL AST with its surface source code. Field names are labeled on upper arcs. Blue squares denote fields with *sequential* cardinality. Grey nodes denote primitive identifier fields, with annotated values. Fields are labeled with time steps at which they are generated. ***Right*** Action sequences used to construct the example AST. Frontier fields are denoted by their signature `(type name)`. Each constructor in the Action column refers to an APPLYCONSTR action.

```
stmt ↦ FunctionDef(identifier name,
                   arguments args, stmt* body)
     | ClassDef(identifier name, expr* bases, stmt* body)
     | Expr(expr value)
     | Return(expr? value)

expr ↦ Call(expr func, expr* args, keyword* keywords)
     | Name(identifier id)
     | Str(string s)
```

Figure 3: Excerpt of the python abstract syntax grammar (Python Software Foundation, 2016)

a cardinality (single, optional ?, and sequential $*$), specifying the number of values the field has.

Each node in an AST corresponds to a typed field in a constructor (except for the root node). Depending on the cardinality of the field, an AST node can be instantiated with one or multiple constructors. For instance, the *func* field in the example AST has single cardinality, and is instantiated with a `Name` constructor; while the *args* field with sequential cardinality could have multiple constructors (only one shown in this example).

Our parser employs a transition system to generate an AST using three types of actions. Fig. 2 (Right) lists the sequence of actions used to generate the example AST. The generation process starts from an initial derivation with only a root node of type `stmt` (statement), and proceeds according to the top-down, left-to-right traversal of the AST. At each time step, the parser applies an action to the *frontier field* of the derivation:

APPLYCONSTR$[c]$ actions apply a constructor c to the frontier composite field, expanding the derivation using the fields of c. For fields with single or optional cardinality, an APPLYCONSTR action instantiates the empty frontier field using the constructor, while for fields with sequential cardinality, it appends the constructor to the frontier field. For example, at t_2 the `Call` constructor is applied to the *value* field of `Expr`, and the derivation is expanded using its three child fields.

REDUCE actions complete generation of a field with optional or multiple cardinalities. For instance, the *args* field is instantiated by `Name` at t_5, and then closed by a REDUCE action at t_7.

GENTOKEN$[v]$ actions populate an empty primitive frontier field with token v. A primitive field whose value is a single token (*e.g.*, identifier fields) can be populated with a single GENTOKEN action. Fields of `string` type can be instantiated using multiple such actions, with a final GENTOKEN[`</f>`] action to terminate the generation of field values.

3.2.2 Modeling $q_\phi(\boldsymbol{z}|\boldsymbol{x})$

The probability of generating an AST $\boldsymbol{z}$ is naturally decomposed into the probabilities of the actions $\{a_t\}$ used to construct $\boldsymbol{z}$:

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \prod_t p(a_t|a_{<t}, \boldsymbol{x}).$$

Following YN17, we parameterize $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ using a sequence-to-sequence network with auxiliary recurrent connections following the topology of the AST. Interested readers are referred to Appendix B and Yin and Neubig (2017) for details of the neural network architecture.

3.3 Semi-supervised Learning

In this section we explain how to optimize the semi-supervised learning objective Eq. (1) in STRUCTVAE.

Supervised Learning For the supervised learning objective, we modify $\mathcal{J}_s$, and use the labeled data to optimize both the inference model (the se-

mantic parser) and the reconstruction model:

$$\mathcal{J}_s \triangleq \sum_{(\boldsymbol{x},\boldsymbol{z})\in\mathbb{L}} \big(\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) + \log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big) \quad (2)$$

Unsupervised Learning To optimize the unsupervised learning objective $\mathcal{J}_u$ in Eq. (1), we maximize the variational lower-bound of $\log p(\boldsymbol{x})$:

$$\log p(\boldsymbol{x}) \geq \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big(\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big) - \lambda \cdot \mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})] = \mathcal{L} \quad (3)$$

where $\mathrm{KL}[q_\phi||p]$ is the Kullback-Leibler (KL) divergence. Following common practice in optimizing VAEs, we introduce λ as a tuning parameter of the KL divergence to control the impact of the prior (Miao and Blunsom, 2016; Bowman et al., 2016).

To optimize the parameters of our model in the face of non-differentiable discrete latent variables, we follow Miao and Blunsom (2016), and approximate $\frac{\partial \mathcal{L}}{\partial \phi}$ using the score function estimator (a.k.a. REINFORCE, Williams (1992)):

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \phi} &= \frac{\partial}{\partial \phi}\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})} \underbrace{\Big(\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) - \lambda\big(\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p(\boldsymbol{z})\big)\Big)}_{\text{learning signal}} \\ &= \frac{\partial}{\partial \phi}\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})} l'(\boldsymbol{x},\boldsymbol{z}) \\ &\approx \frac{1}{|\mathcal{S}(\boldsymbol{x})|}\sum_{\boldsymbol{z}_i\in\mathcal{S}(\boldsymbol{x})} l'(\boldsymbol{x},\boldsymbol{z}_i)\frac{\partial \log q_\phi(\boldsymbol{z}_i|\boldsymbol{x})}{\partial \phi} \quad (4)\end{aligned}$$

where we approximate the gradient using a set of samples $\mathcal{S}(\boldsymbol{x})$ drawn from $q_\phi(\cdot|\boldsymbol{x})$. To ensure the quality of sampled latent MRs, we follow Guu et al. (2017) and use beam search. The term $l'(\boldsymbol{x},\boldsymbol{z})$ is defined as the *learning signal* (Miao and Blunsom, 2016). The learning signal weights the gradient for each latent sample $\boldsymbol{z}$. In REINFORCE, to cope with the high variance of the learning signal, it is common to use a baseline $b(\boldsymbol{x})$ to stabilize learning, and re-define the learning signal as

$$l(\boldsymbol{x},\boldsymbol{z}) \triangleq l'(\boldsymbol{x},\boldsymbol{z}) - b(\boldsymbol{x}). \quad (5)$$

Specifically, in StructVAE, we define

$$b(\boldsymbol{x}) = a \cdot \log p(\boldsymbol{x}) + c, \quad (6)$$

where $\log p(\boldsymbol{x})$ is a pre-trained LSTM language model. This is motivated by the empirical observation that $\log p(\boldsymbol{x})$ correlates well with the reconstruction score $\log p_\theta(\boldsymbol{x}|\boldsymbol{z})$, hence with $l'(\boldsymbol{x},\boldsymbol{z})$.

Finally, for the reconstruction model, its gradient can be easily computed:

$$\frac{\partial \mathcal{L}}{\partial \theta} \approx \frac{1}{|\mathcal{S}(\boldsymbol{x})|}\sum_{\boldsymbol{z}_i\in\mathcal{S}(\boldsymbol{x})} \frac{\partial \log p_\theta(\boldsymbol{x}|\boldsymbol{z}_i)}{\partial \theta}.$$

Discussion Perhaps the most intriguing question here is why semi-supervised learning could improve semantic parsing performance. While the underlying theoretical exposition still remains an active research problem (Singh et al., 2008), in this paper we try to empirically test some likely hypotheses. In Eq. (4), the gradient received by the inference model from each latent sample $\boldsymbol{z}$ is weighed by the learning signal $l(\boldsymbol{x},\boldsymbol{z})$. $l(\boldsymbol{x},\boldsymbol{z})$ can be viewed as the reward function in REINFORCE learning. It can also be viewed as weights associated with pseudo-training examples $\{\langle \boldsymbol{x},\boldsymbol{z}\rangle : \boldsymbol{z} \in \mathcal{S}(\boldsymbol{x})\}$ sampled from the inference model. Intuitively, a sample $\boldsymbol{z}$ with higher rewards should: (1) have $\boldsymbol{z}$ adequately encode the input, leading to high reconstruction score $\log p_\theta(\boldsymbol{x}|\boldsymbol{z})$; and (2) have $\boldsymbol{z}$ be succinct and natural, yielding high prior probability. Let $\boldsymbol{z}^*$ denote the gold-standard MR of $\boldsymbol{x}$. Consider the ideal case where $\boldsymbol{z}^* \in \mathcal{S}(\boldsymbol{x})$ and $l(\boldsymbol{x},\boldsymbol{z}^*)$ is positive, while $l(\boldsymbol{x},\boldsymbol{z}')$ is negative for other imperfect samples $\boldsymbol{z}' \in \mathcal{S}(\boldsymbol{x})$, $\boldsymbol{z}' \neq \boldsymbol{z}^*$. In this ideal case, $\langle \boldsymbol{x},\boldsymbol{z}^*\rangle$ would serve as a positive training example and other samples $\langle \boldsymbol{x},\boldsymbol{z}'\rangle$ would be treated as negative examples. Therefore, the inference model would receive informative gradient updates, and learn to discriminate between gold and imperfect MRs. This intuition is similar in spirit to recent efforts in interpreting gradient update rules in reinforcement learning (Guu et al., 2017). We will present more empirical statistics and observations in § 4.3.

4 Experiments

4.1 Datasets

In our semi-supervised semantic parsing experiments, it is of interest how StructVAE could further improve upon a supervised parser with extra unlabeled data. We evaluate on two datasets:

Semantic Parsing We use the Atis dataset, a collection of 5,410 telephone inquiries of flight booking (*e.g.*, *"Show me flights from ci0 to ci1"*). The target MRs are defined using λ-calculus logical forms (*e.g.*, "`lambda $0 e (and (flight $0) (from $ci0) (to $ci1))`"). We use the pre-processed dataset released by Dong and Lapata (2016), where entities (*e.g.*, cities) are canonicalized using typed slots (*e.g.*, `ci0`). To predict λ-

calculus logical forms using our transition-based parser, we use the ASDL grammar defined by Rabinovich et al. (2017) to convert between logical forms and ASTs (see Appendix C for details).

Code Generation The DJANGO dataset (Oda et al., 2015) contains 18,805 lines of Python source code extracted from the Django web framework. Each line of code is annotated with an NL utterance. Source code in the DJANGO dataset exhibits a wide variety of real-world use cases of Python, including IO operation, data structure manipulation, class/function definition, *etc*. We use the pre-processed version released by Yin and Neubig (2017) and use the `astor` package to convert ASDL ASTs into Python source code.

4.2 Setup

Labeled and Unlabeled Data STRUCTVAE requires access to extra unlabeled NL utterances for semi-supervised learning. However, the datasets we use do not accompany with such data. We therefore simulate the semi-supervised learning scenario by randomly sub-sampling K examples from the training split of each dataset as the labeled set $\mathbb{L}$. To make the most use of the NL utterances in the dataset, we construct the unlabeled set $\mathbb{U}$ using all NL utterances in the training set[3,4].

Training Procedure Optimizing the unsupervised learning objective Eq. (3) requires sampling structured MRs from the inference model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Due to the complexity of the semantic parsing problem, we cannot expect any valid samples from randomly initialized $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. We therefore pre-train the inference and reconstruction models using the supervised objective Eq. (2) until convergence, and then optimize using the semi-supervised learning objective Eq. (1). Throughout all experiments we set α (Eq. (1)) and λ (Eq. (3)) to 0.1. The sample size $|\mathcal{S}(\boldsymbol{x})|$ is 5. We observe that the variance of the learning signal could still be high when low-quality samples are drawn from the inference model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. We therefore clip all learning signals lower than $k = -20.0$. Early-stopping is used to avoid over-fitting. We also pre-train the prior $p(\boldsymbol{z})$ (§ 3.3) and the baseline function Eq. (6). Readers are referred to Appendix D for more detail of the configurations.

[3]We also tried constructing $\mathbb{U}$ using the disjoint portion of the NL utterances not presented in the labeled set $\mathbb{L}$, but found this yields slightly worse performance, probably due to lacking enough unlabeled data. Interpreting these results would be an interesting avenue for future work.

[4]While it might be relatively easy to acquire additional unlabeled utterances in practical settings (*e.g.*, through query logs of a search engine), unfortunately most academic semantic parsing datasets, like the ones used in this work, do not feature large sets of in-domain unlabeled data. We therefore perform simulated experiments instead.

$\lvert\mathbb{L}\rvert$	SUP.	SELFTRAIN	STRUCTVAE
500	63.2	65.3	**66.0**
1,000	74.6	74.2	**75.7**
2,000	80.4	**83.3**	82.4
3,000	82.8	**83.6**	**83.6**
4,434 (All)	**85.3**	–	84.5

Previous Methods	ACC.
ZC07 (Zettlemoyer and Collins, 2007)	84.6
WKZ14 (Wang et al., 2014)	**91.3**
SEQ2TREE (Dong and Lapata, 2016)†	84.6
ASN (Rabinovich et al., 2017)†	85.3
+ supervised attention	85.9

Table 1: Performance on ATIS w.r.t. the size of labeled training data $\mathbb{L}$. †Existing neural network-based methods

$\lvert\mathbb{L}\rvert$	SUP.	SELFTRAIN	STRUCTVAE
1,000	49.9	49.5	**52.0**
2,000	56.6	55.8	**59.0**
3,000	61.0	61.4	**62.4**
5,000	63.2	64.5	**65.6**
8,000	70.3	69.6	**71.5**
12,000	71.1	71.6	**72.0**
16,000 (All)	**73.7**	–	72.3

Previous Method	ACC.
YN17 (Yin and Neubig, 2017)	71.6

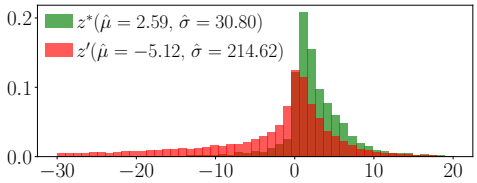
Table 2: Performance on DJANGO w.r.t. the size of labeled training data $\mathbb{L}$

Metric As standard in semantic parsing research, we evaluate by exact-match **accuracy**.

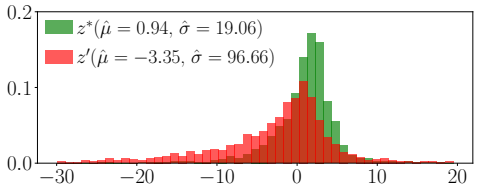
4.3 Main Results

Tab. 1 and Tab. 2 list the results on ATIS and DJANGO, resp, with varying amounts of labeled data $\mathbb{L}$. We also present results of training the transition-based parser using only the supervised objective (**SUP.**, Eq. (2)). We also compare STRUCTVAE with self-training (**SELFTRAIN**), a semi-supervised learning baseline which uses the supervised parser to predict MRs for unlabeled utterances in $\mathbb{U} - \mathbb{L}$, and adds the predicted examples to the training set to fine-tune the supervised model. Results for STRUCTVAE are averaged over four runs to account for the additional fluctuation caused by REINFORCE training.

Supervised System Comparison First, to highlight the effectiveness of our transition parser based on ASDL grammar (hence the reliability of



(a) DJANGO

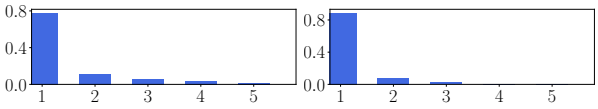


(b) ATIS

Figure 4: Histograms of learning signals on DJANGO ($|\mathbb{L}| = 5000$) and ATIS ($|\mathbb{L}| = 2000$). Difference in sample means is statistically significant ($p < 0.05$).

our supervised baseline), we compare the supervised version of our parser with existing parsing models. On ATIS, our supervised parser trained on the full data is competitive with existing neural network based models, surpassing the SEQ2TREE model, and on par with the Abstract Syntax Network (ASN) without using extra supervision. On DJANGO, our model significantly outperforms the YN17 system, probably because the transition system used by our parser is defined natively to construct ASDL ASTs, reducing the number of actions for generating each example. On DJANGO, the average number of actions is 14.3, compared with 20.3 reported in YN17.

Semi-supervised Learning Next, we discuss our main comparison between STRUCTVAE with the supervised version of the parser (recall that the supervised parser is used as the inference model in STRUCTVAE, § 3.2). First, comparing our proposed STRUCTVAE with the supervised parser when there are extra unlabeled data (*i.e.*, $|\mathbb{L}| < 4,434$ for ATIS and $|\mathbb{L}| < 16,000$ for DJANGO), semi-supervised learning with STRUCTVAE consistently achieves better performance. Notably, on DJANGO, our model registers results as competitive as previous state-of-the-art method (YN17) using only *half* the training data (71.5 when $|\mathbb{L}| = 8000$ v.s. 71.6 for YN17). This demonstrates that STRUCTVAE is capable of learning from unlabeled NL utterances by inferring high quality, structurally rich latent meaning representations, further improving the performance of its supervised counterpart that is already competitive. Second, comparing STRUCTVAE with self-training, we find STRUCTVAE outperforms SELFTRAIN in eight out of ten settings, while SELFTRAIN



(a) DJANGO (b) ATIS

Figure 5: Distribution of the rank of $l(\boldsymbol{x}, \boldsymbol{z}^*)$ in sampled set

under-performs the supervised parser in four out of ten settings. This shows self-training does not necessarily yield stable gains while STRUCTVAE does. Intuitively, STRUCTVAE would perform better since it benefits from the additional signal of the quality of MRs from the reconstruction model (§ 3.3), for which we present more analysis in our next set of experiments.

For the sake of completeness, we also report the results of STRUCTVAE when $\mathbb{L}$ is the full training set. Note that in this scenario there is no extra unlabeled data disjoint with the labeled set, and not surprisingly, STRUCTVAE does not outperform the supervised parser. In addition to the supervised objective Eq. (2) used by the supervised parser, STRUCTVAE has the extra unsupervised objective Eq. (3), which uses sampled (probably incorrect) MRs to update the model. When there is no extra unlabeled data, those sampled (incorrect) MRs add noise to the optimization process, causing STRUCTVAE to under-perform.

Study of Learning Signals As discussed in § 3.3, in semi-supervised learning, the gradient received by the inference model from each sampled latent MR is weighted by the learning signal. Empirically, we would expect that on average, the learning signals of gold-standard samples $\boldsymbol{z}^*$, $l(\boldsymbol{x}, \boldsymbol{z}^*)$, are positive, larger than those of other (imperfect) samples $\boldsymbol{z}'$, $l(\boldsymbol{x}, \boldsymbol{z}')$. We therefore study the statistics of $l(\boldsymbol{x}, \boldsymbol{z}^*)$ and $l(\boldsymbol{x}, \boldsymbol{z}')$ for all utterances $\boldsymbol{x} \in \mathbb{U} - \mathbb{L}$, *i.e.*, the set of utterances which are not included in the labeled set.[5] The statistics are obtained by performing inference using trained models. Figures 4a and 4b depict the histograms of learning signals on DJANGO and ATIS, resp. We observe that the learning signals for gold samples concentrate on positive intervals. We also show the mean and variance of the learning signals. On average, we have $l(\boldsymbol{x}, \boldsymbol{z}^*)$ being positive and $l(\boldsymbol{x}, \boldsymbol{z})$ negative. Also note that the distribution of $l(\boldsymbol{x}, \boldsymbol{z}^*)$ has smaller variance and is more concentrated. Therefore the inference model receives informative gradient updates to discriminate between gold and imperfect

[5] We focus on cases where $\boldsymbol{z}^*$ is in the sample set $\mathcal{S}(\boldsymbol{x})$.

NL	*join p and cmd into a file path, substitute it for f*	
z_1^s	`f = os.path.join(p, cmd)` ✓	
	$\log q(z\|x) = -1.00$	$\log p(x\|z) = -2.00$
	$\log p(z) = -24.33$	$l(x, z) = 9.14$
z_2^s	`p = path.join(p, cmd)` ✗	
	$\log q(z\|x) = -8.12$	$\log p(x\|z) = -20.96$
	$\log p(z) = -27.89$	$l(x, z) = -9.47$
NL	*append i-th element of existing to child_loggers*	
z_1^s	`child_loggers.append(existing[i])` ✓	
	$\log q(z\|x) = -2.38$	$\log p(x\|z) = -9.66$
	$\log p(z) = -13.52$	$l(x, z) = 1.32$
z_2^s	`child_loggers.append(existing[existing])` ✗	
	$\log q(z\|x) = -1.83$	$\log p(x\|z) = -16.11$
	$\log p(z) = -12.43$	$l(x, z) = -5.08$
NL	*split string pks by ',', substitute the result for primary_keys*	
z_1^s	`primary_keys = pks.split(',')` ✓	
	$\log q(z\|x) = -2.38$	$\log p(x\|z) = -11.39$
	$\log p(z) = -10.24$	$l(x, z) = 2.05$
z_2^s	`primary_keys = pks.split + ','` ✗	
	$\log q(z\|x) = -0.84$	$\log p(x\|z) = -14.87$
	$\log p(z) = -20.41$	$l(x, z) = -2.60$

Table 3: Inferred latent MRs on DJANGO ($|\mathbb{L}| = 5000$). For simplicity we show the surface representation of MRs (z^s, source code) instead.

samples. Next, we plot the distribution of the rank of $l(x, z^*)$, among the learning signals of all samples of x, $\{l(x, z_i) : z_i \in \mathcal{S}(x)\}$. Results are shown in Fig. 5. We observe that the gold samples z^* have the largest learning signals in around 80% cases. We also find that when z^* has the largest learning signal, its average difference with the learning signal of the highest-scoring incorrect sample is 1.27 and 0.96 on DJANGO and ATIS, respectively.

Finally, to study the relative contribution of the reconstruction score $\log p(x|z)$ and the prior $\log p(z)$ to the learning signal, we present examples of inferred latent MRs during training (Tab. 3). Examples 1&2 show that the reconstruction score serves as an informative quality measure of the latent MR, assigning the correct samples z_1^s with high $\log p(x|z)$, leading to positive learning signals. This is in line with our assumption that a good latent MR should adequately encode the semantics of the utterance. Example 3 shows that the prior is also effective in identifying "unnatural" MRs (*e.g.*, it is rare to add a function and a string literal, as in z_2^s). These results also suggest that the prior and the reconstruction model perform well with linearization of MRs. Finally, note that in Examples 2&3 the learning signals for the correct samples z_1^s are positive even if their inference scores $q(z|x)$ are lower than those of z_2^s.

$\|\mathbb{L}\|$	SUPERVISED	STRUCTVAE-SEQ
500	47.3	**55.6**
1,000	62.5	**73.1**
2,000	73.9	**74.8**
3,000	80.6	**81.3**
4,434 (All)	**84.6**	84.2

Table 4: Performance of the STRUCTVAE-SEQ on ATIS w.r.t. the size of labeled training data $\mathbb{L}$

ATIS				DJANGO			
$\|\mathbb{L}\|$	SUP.	MLP	LM	$\|\mathbb{L}\|$	SUP.	MLP	LM
500	63.2	*61.5*†	**66.0**	1,000	49.9	*47.0*†	**52.0**
1,000	74.6	**76.3**	75.7	5,000	63.2	*62.5*†	**65.6**
2,000	80.4	**82.9**	82.4	8,000	70.3	*67.6*†	**71.5**
3,000	82.8	*81.4*†	**83.6**	12,000	71.1	71.6	**72.0**

Table 5: Comparison of STRUCTVAE with different baseline functions $b(x)$, *italic*†: semi-supervised learning with the MLP baseline is worse than supervised results.

This result further demonstrates that learning signals provide informative gradient weights for optimizing the inference model.

Generalizing to Other Latent MRs Our main results are obtained using a strong AST-based semantic parser as the inference model, with copy-augmented reconstruction model and an LSTM language model as the prior. However, there are many other ways to represent and infer structure in semantic parsing (Carpenter, 1998; Steedman, 2000), and thus it is of interest whether our basic STRUCTVAE framework generalizes to other semantic representations. To examine this, we test STRUCTVAE using λ-calculus logical forms as latent MRs for semantic parsing on the ATIS domain. We use standard sequence-to-sequence networks with attention (Luong et al., 2015) as inference and reconstruction models. The inference model is trained to construct a tree-structured logical form using the transition actions defined in Cheng et al. (2017). We use a classical tri-gram Kneser-Ney language model as the prior. Tab. 4 lists the results for this STRUCTVAE-SEQ model.

We can see that even with this very different model structure STRUCTVAE still provides significant gains, demonstrating its compatibility with different inference/reconstruction networks and priors. Interestingly, compared with the results in Tab. 1, we found that the gains are especially larger with few labeled examples — STRUCTVAE-SEQ achieves improvements of 8-10 points when $|\mathbb{L}| < 1000$. These results suggest that semi-supervision is especially useful in improving a mediocre parser in low resource settings.

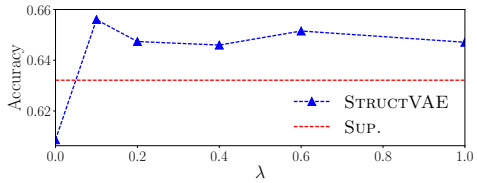


Figure 6: Performance on DJANGO ($|\mathbb{L}| = 5000$) w.r.t. the KL weight λ

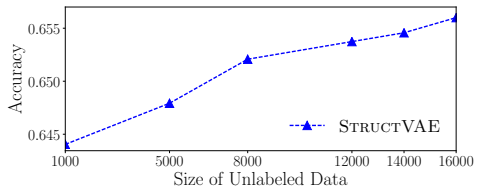


Figure 7: Performance on DJANGO ($|\mathbb{L}| = 5000$) w.r.t. the size of unlabeled data $\mathbb{U}$

Impact of Baseline Functions In § 3.3 we discussed our design of the baseline function $b(\boldsymbol{x})$ incorporated in the learning signal (Eq. (4)) to stabilize learning, which is based on a language model (LM) over utterances (Eq. (6)). We compare this baseline with a commonly used one in REINFORCE training: the multi-layer perceptron (MLP). The MLP takes as input the last hidden state of the utterance given by the encoding LSTM of the inference model. Tab. 5 lists the results over sampled settings. We found that although StructVAE with the MLP baseline sometimes registers better performance on ATIS, in most settings it is worse than our LM baseline, and could be even worse than the supervised parser. On the other hand, our LM baseline correlates well with the learning signal, yielding stable improvements over the supervised parser. This suggests the importance of using carefully designed baselines in REINFORCE learning, especially when the reward signal has large range (*e.g.*, log-likelihoods).

Impact of the Prior $p(\boldsymbol{z})$ Fig. 6 depicts the performance of StructVAE as a function of the KL term weight λ in Eq. (3). When StructVAE degenerates to a vanilla auto-encoder without the prior distribution (*i.e.*, $\lambda = 0$), it under-performs the supervised baseline. This is in line with our observation in Tab. 3 showing that the prior helps identify unnatural samples. The performance of the model also drops when $\lambda > 0.1$, suggesting that empirically controlling the influence of the prior to the inference model is important.

Impact of Unlabeled Data Size Fig. 7 illustrates the accuracies w.r.t. the size of unlabeled data. StructVAE yields consistent gains as the size of the unlabeled data increases.

5 Related Works

Semi-supervised Learning for NLP Semi-supervised learning comes with a long history (Zhu, 2005), with applications in NLP from early work of self-training (Yarowsky, 1995), and graph-based methods (Das and Smith, 2011), to recent advances in auto-encoders (Cheng et al., 2016; Socher et al., 2011; Zhang et al., 2017) and deep generative methods (Xu et al., 2017). Our work follows the line of neural variational inference for text processing (Miao et al., 2016), and resembles Miao and Blunsom (2016), which uses VAEs to model summaries as discrete latent variables for semi-supervised summarization, while we extend the VAE architecture for more complex, tree-structured latent variables.

Semantic Parsing Most existing works alleviate issues of limited parallel data through weakly-supervised learning, using the denotations of MRs as indirect supervision (Reddy et al., 2014; Krishnamurthy et al., 2016; Neelakantan et al., 2016; Pasupat and Liang, 2015; Yin et al., 2016). For semi-supervised learning of semantic parsing, Kate and Mooney (2007) first explore using transductive SVMs to learn from a semantic parser's predictions. Konstas et al. (2017) apply self-training to bootstrap an existing parser for AMR parsing. Kociský et al. (2016) employ VAEs for semantic parsing, but in contrast to StructVAE's structured representation of MRs, they model NL utterances as flat latent variables, and learn from unlabeled MR data. There have also been efforts in unsupervised semantic parsing, which exploits external linguistic analysis of utterances (*e.g.*, dependency trees) and the schema of target knowledge bases to infer the latent MRs (Poon and Domingos, 2009; Poon, 2013). Another line of research is domain adaptation, which seeks to transfer a semantic parser learned from a source domain to the target domain of interest, therefore alleviating the need of parallel data from the target domain (Su and Yan, 2017; Fan et al., 2017; Herzig and Berant, 2018).

6 Conclusion

We propose StructVAE, a deep generative model with tree-structured latent variables for semi-supervised semantic parsing. We apply StructVAE to semantic parsing and code generation tasks, and show it outperforms a strong supervised parser using extra unlabeled data.

References

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

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of LAW-ID@ACL*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of EMNLP*.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *Proceedings of the SIGNLL*.

Bob Carpenter. 1998. *Type-logical Semantics*.

Jianpeng Cheng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. 2017. Learning structured natural language representations for semantic parsing. In *Proceedings of ACL*.

Yong Cheng, Wei Xu, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Semi-supervised learning for neural machine translation. In *Proceedings of ACL*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of CoNLL*.

Dipanjan Das and Noah A. Smith. 2011. Semi-supervised frame-semantic parsing for unknown predicates. In *Proceedings of HLT*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of ACL*.

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

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of ACL*.

Kelvin Guu, Panupong Pasupat, Evan Zheran Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of ACL*.

Jonathan Herzig and Jonathan Berant. 2018. Decoupling structure and lexicon for zero-shot semantic parsing. *arXiv preprint arXiv:1804.07918* .

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of ACL*.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of ACL*.

Rohit J. Kate and Raymond J. Mooney. 2007. Semi-supervised learning for semantic parsing using support vector machines. In *Proceedings of NAACL-HLT*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. 2014. Semi-supervised learning with deep generative models. In *Proceedings of NIPS*.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* .

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

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of ACL*.

Jayant Krishnamurthy, Oyvind Tafjord, and Aniruddha Kembhavi. 2016. Semantic parsing to probabilistic programs for situated question answering. In *Proceedings of EMNLP*.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of ACL*.

Percy Liang, Michael I. Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of ACL*.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of ACL*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*.

Yishu Miao and Phil Blunsom. 2016. Language as a latent variable: Discrete generative models for sentence compression. In *Proceedings of EMNLP*.

Yishu Miao, Lei Yu, and Phil Blunsom. 2016. Neural variational inference for text processing. In *Proceedings of ICML*.

Dipendra K. Misra and Yoav Artzi. 2016. Neural shift-reduce CCG semantic parsing. In *Proceedings of EMNLP*.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. In *Proceedings of ICLR*.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation (T). In *Proceedings of ASE*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of ACL*.

Hoifung Poon. 2013. Grounded unsupervised semantic parsing. In *Proceedings of ACL*.

Hoifung Poon and Pedro Domingos. 2009. Unsupervised semantic parsing. In *Proceedings of EMNLP*.

Python Software Foundation. 2016. Python abstract grammar. https://docs.python.org/2/library/ast.html.

Chris Quirk, Raymond J. Mooney, and Michel Galley. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of ACL*.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of ACL*.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of ACL* .

Aarti Singh, Robert D. Nowak, and Xiaojin Zhu. 2008. Unlabeled data: Now it helps, now it doesn't. In *Proceedings of NIPS*.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of EMNLP*.

Mark Steedman. 2000. *The Syntactic Process*.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *Proceedings of EMNLP*.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of ECML*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Proceedings of NIPS*.

Adrienne Wang, Tom Kwiatkowski, and Luke Zettlemoyer. 2014. Morpho-syntactic lexical generalization for ccg semantic parsing. In *Proceedings of EMNLP*.

Daniel C. Wang, Andrew W. Appel, Jeffrey L. Korn, and Christopher S. Serra. 1997. The zephyr abstract syntax description language. In *Proceedings of DSL*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of ACL*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* .

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Proceedings of ACL*.

Weidi Xu, Haoze Sun, Chao Deng, and Ying Tan. 2017. Variational autoencoder for semi-supervised text classification. In *Proceedings of AAAI*.

David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of ACL*.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of ACL*.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016. Neural enquirer: Learning to query tables in natural language. In *Proceedings of IJCAI*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of ACL*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* .

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of AAAI*.

Luke Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form structured classification with probabilistic categorial grammars. In *Proceedings of UAI*.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of EMNLP-CoNLL*.

Xiao Zhang, Yong Jiang, Hao Peng, Kewei Tu, and Dan Goldwasser. 2017. Semi-supervised structured prediction with neural crf autoencoder. In *Proceedings of EMNLP*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* .

Chunting Zhou and Graham Neubig. 2017. Multi-space variational encoder-decoders for semi-supervised labeled sequence transduction. In *Proceedings of ACL*.

Xiaojin Zhu. 2005. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison.

STRUCTVAE: Tree-structured Latent Variable Models for Semi-supervised Semantic Parsing

Supplementary Materials

A Generating Samples from STRUCTVAE

STRUCTVAE is a generative model of natural language, and therefore can be used to sample latent MRs and the corresponding NL utterances. This amounts to draw a latent MR $\boldsymbol{z}$ from the prior $p(\boldsymbol{z})$, and sample an NL utterance $\boldsymbol{x}$ from the reconstruction model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. Since we use the sequential representation $\boldsymbol{z}^s$ in the prior, to guarantee the syntactic well-formedness of sampled MRs from $p(\boldsymbol{z})$, we use a syntactic checker and reject any syntactically-incorrect samples[6]. Tab. 6 and Tab. 7 present samples from DJANGO and ATIS, respectively. These examples demonstrate that STRUCTVAE is capable of generating syntactically diverse NL utterances.

latent MR	`def __init__(self, *args, **kwargs): pass`
surface NL	*Define the method __init__ with 3 arguments: self, unpacked list args and unpacked dictionary kwargs*
latent MR	`elif isinstance(target, six.string_types): pass`
surface NL	*Otherwise if target is an instance of six.string_types*
latent MR	`for k, v in unk.items(): pass`
surface NL	*For every k and v in return value of the method unk.items*
latent MR	`return cursor.fetchone()[0]`
surface NL	*Call the method cursor.fetchone , return the first element of the result*
latent MR	`sys.stderr.write(_STR_ % e)`
surface NL	*Call the method sys.stderr, write with an argument _STR_ formated with e*
latent MR	`opts = getattr(self, _STR_, None)`
surface NL	*Get the _STR_ attribute of the self object, if it exists substitute it for opts, if not opts is None*

Table 6: Sampled latent meaning representations (presented in surface source code) and NL utterances from DJANGO.

latent MR	`(argmax $0 (and (flight $0) (meal $0 lunch:me) (from $0 ci0) (to $0 ci1)) (departure_time $0))`
surface NL	*Show me the latest flight from ci0 to ci1 that serves lunch*
latent MR	`(min $0 (exists $1 (and (from $1 ci0) (to $1 ci1) (day_number $1 dn0) (month $1 mn0) (round_trip $1) (= (fare $1) $0))))`
surface NL	*I want the cheapest round trip fare from ci0 to ci1 on mn0 dn0*
latent MR	`(lambda $0 e (and (flight $0) (from $0 ci0) (to $0 ci1) (weekday $0)))`
surface NL	*Please list weekday flight between ci0 and ci1*
latent MR	`(lambda $0 e (and (flight $0) (has_meal $0) (during_day $0 evening:pd) (from $0 ci1) (to $0 ci0) (day_number $0 dn0) (month $0 mn0)))`
surface NL	*What are the flight from ci1 to ci0 on the evening of mn0 dn0 that serves a meal*
latent MR	`(lambda $0 e (and (flight $0) (oneway $0) (class_type $0 first:cl) (from $0 ci0) (to $0 ci1) (day $0 da0)))`
surface NL	*Show me one way flight from ci0 to ci1 on a da0 with first class fare*
latent MR	`(lambda $0 e (exists $1 (and (rental_car $1) (to_city $1 ci0) (= (ground_fare $1) $0))))`
surface NL	*What would be cost of car rental car in ci0*

Table 7: Sampled latent meaning representations (presented in surface λ-calculus expression) and NL utterances from ATIS. Verbs are recovered to their correct form instead of the lemmatized version as in the pre-processed dataset.

[6] We found most samples from $p(\boldsymbol{z})$ are syntactically well-formed, with 98.9% and 95.3% well-formed samples out of $100K$ samples on ATIS and DJANGO, respectively.

B Neural Network Architecture

B.1 Prior $p(\boldsymbol{z})$

The prior $p(\boldsymbol{z})$ is a standard LSTM language model (Zaremba et al., 2014). We use the sequence representation of $\boldsymbol{z}$, $\boldsymbol{z}^s$, to model $p(\boldsymbol{z})$. Specifically, let $\boldsymbol{z}^s = \{z_i^s\}_{i=1}^{|\boldsymbol{z}^s|}$ consisting of $|\boldsymbol{z}^s|$ tokens, we have

$$p(\boldsymbol{z}^s) = \prod_{i=1}^{|\boldsymbol{z}^s|} p(z_i^s | \boldsymbol{z}_{<i}^s),$$

where $\boldsymbol{z}_{<i}^s$ denote the sequence of history tokens $\{z_1^s, z_2^s, \ldots, z_{i-1}^s\}$. At each time step i, the probability of predicting z_i^s given the context is modeled by an LSTM network

$$p(z_i^s | \boldsymbol{z}_{<i}^s) = \text{softmax}(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{b})$$
$$\boldsymbol{h}_i = f_{\text{LSTM}}(\boldsymbol{e}(z_{i-1}^s), \boldsymbol{h}_{i-1})$$

where $\boldsymbol{h}_i$ denote the hidden state of the LSTM at time step i, and $\boldsymbol{e}(\cdot)$ is an embedding function.

B.2 Reconstruction Model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$

We implement a standard attentional sequence-to-sequence network (Luong et al., 2015) with copy mechanism as the reconstruction network $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. Formally, given a utterance $\boldsymbol{x}$ of n words $\{x_i\}_{i=1}^n$, the probability of generating a token x_i is marginalized over the probability of generating x_i from a closed-set vocabulary, and that of copying from the MR $\boldsymbol{z}^s$:

$$\begin{aligned} p(x_i | x_{<i}, \boldsymbol{z}^s) = & \, p(\text{gen} | x_{<i}, \boldsymbol{z}^s) p(x_i | \text{gen}, x_{<i}, \boldsymbol{z}^s) \\ & + p(\text{copy} | x_{<i}, \boldsymbol{z}^s) p(x_i | \text{copy}, x_{<i}, \boldsymbol{z}^s) \end{aligned}$$

where $p(\text{gen}|\cdot)$ and $p(\text{copy}|\cdot)$ are computed by $\text{softmax}(\boldsymbol{W}\tilde{\boldsymbol{s}}_i^c)$. $\tilde{\boldsymbol{s}}_i^c$ denotes the attentional vector (Luong et al., 2015) at the i-th time step:

$$\tilde{\boldsymbol{s}}_i^c = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_i^c; \boldsymbol{s}_i^c]). \tag{7}$$

Here, $\boldsymbol{s}_i^c$ is the i-th decoder hidden state of the reconstruction model, and $\boldsymbol{c}_i^c$ the context vector (Bahdanau et al., 2015) obtained by attending to the source encodings. The probability of copying the j-th token in $\boldsymbol{z}^s$, z_j^s, is given by a pointer network (Vinyals et al., 2015), derived from $\tilde{\boldsymbol{s}}_i^c$ and the encoding of z_j^s, $\boldsymbol{h}_j^{\boldsymbol{z}}$.

$$p(x_i = z_j^s | \text{copy}, x_{<i}, \boldsymbol{z}^s) = \frac{\exp\left(\boldsymbol{h}_j^{\boldsymbol{z}\intercal} \boldsymbol{W} \tilde{\boldsymbol{s}}_i^c\right)}{\sum_{j'=1}^{|\boldsymbol{z}^s|} \exp\left(\boldsymbol{h}_{j'}^{\boldsymbol{z}\intercal} \boldsymbol{W} \tilde{\boldsymbol{s}}_i^c\right)}$$

B.3 Inference Model $p_\phi(\boldsymbol{z}|\boldsymbol{x})$

Our inference model (*i.e.*, the semantic parser) is based on the code generation model proposed in Yin and Neubig (2017). As illustrated in Fig. 2 and elaborated in § 3.2, our transition parser constructs an abstract syntax tree specified under the ASDL formalism using a sequence of transition actions. The parser is a neural sequence-to-sequence network, whose recurrent decoder is augmented with auxiliary connections following the topology of ASTs. Specifically, at each decoding time step t, an LSTM decoder uses its internal hidden state $\boldsymbol{s}_t$ to keep track of the generation process of a derivation AST

$$\boldsymbol{s}_t = f_{\text{LSTM}}([\boldsymbol{a}_{t-1} : \tilde{\boldsymbol{s}}_{t-1} : \boldsymbol{p}_t], \boldsymbol{s}_{t-1})$$

where $[:]$ denotes vector concatenation. $\boldsymbol{a}_{t-1}$ is the embedding of the previous action. $\tilde{\boldsymbol{s}}_{t-1}$ is the input-feeding attentional vector as in Luong et al. (2015). $\boldsymbol{p}_t$ is a vector that captures the information of the parent frontier field in the derivation AST, which is the concatenation of four components: $\boldsymbol{n}_{\boldsymbol{f_t}}$, which is the embedding of the current frontier field n_{f_t} on the derivation; $\boldsymbol{e}_{\boldsymbol{f_t}}$, which is the embedding of the type of n_{f_t}; $\boldsymbol{s}_{p_t}$, which is the state of the decoder at which the frontier field n_{f_t} was generated by applying its parent constructor c_{p_t} to the derivation; $\boldsymbol{c}_{p_t}$, which is the embedding of the parent constructor c_{p_t}.

Given the current state of the decoder, $\boldsymbol{s}_t$, an attentional vector $\tilde{\boldsymbol{s}}_t$ is computed similar as Eq. (7) by attending to input the utterance $\boldsymbol{x}$. The attentional vector $\tilde{\boldsymbol{s}}_t$ is then used as the query vector to compute action probabilities, as elaborated in §4.2.2 of Yin and Neubig (2017).

C ASDL Grammar for ATIS

We use the ASDL grammar defined in Rabinovich et al. (2017) to deterministically convert between λ-calculus logical forms and ASDL ASTs:

```
expr = Variable(var variable)
     | Entity(ent entity)
     | Number(num number)
     | Apply(pred predicate, expr* arguments)
     | Argmax(var variable, expr domain, expr body)
     | Argmin(var variable, expr domain, expr body)
     | Count(var variable, expr body)
     | Exists(var variable, expr body)
     | Lambda(var variable, var_type type, expr body)
     | Max(var variable, expr body)
     | Min(var variable, expr body)
     | Sum(var variable, expr domain, expr body)
     | The(var variable, expr body)
     | Not(expr argument)
     | And(expr* arguments)
     | Or(expr* arguments)
     | Compare(cmp_op op, expr left, expr right)

cmp_op = Equal | LessThan | GreaterThan
```

D Model Configuration

Initialize Baselines $b(\boldsymbol{x})$ STRUCTVAE uses baselines $b(\boldsymbol{x})$ to reduce variance in training. For our proposed baseline based on the language model over utterances (Eq. (6)), we pre-train a language model using all NL utterances in the datasets. For terms a and c in Eq. (6), we determine their initial values by first train STRUCTVAE starting from $a = 1.0$ and $c = 0$ for a few epochs, and use their optimized values. Finally we initialize a to 0.5 and b to -2.0 for ATIS, and a to 0.9 and b to 2.0 for DJANGO. We perform the same procedure to initialize the bias term b_{MLP} in the MLP baseline, and have $b_{\text{MLP}} = -20.0$.

Pre-trained Priors $p(\boldsymbol{z})$ STRUCTVAE requires pre-trained priors $p(\boldsymbol{z})$ (§ 3.3). On ATIS, we train a prior for each labeled set $\mathbb{L}$ of size K using the MRs in $\mathbb{L}$. For DJANGO, we use all source code in Django that is not included in the annotated dataset.

Hyper-Parameters and Optimization For all experiments we use embeddings of size 128, and LSTM hidden size of 256. For the transition parser, we use the same hyper parameters as Yin and Neubig (2017), except for the node (field) type embedding, which is 64 for DJANGO and 32 for ATIS. To avoid over-fitting, we impose dropouts on the LSTM hidden states, with dropout rates validated among $\{0.2, 0.3, 0.4\}$. We train the model using Adam (Kingma and Ba, 2014), with a batch size of 10 and 25 for the supervised and unsupervised objectives, resp. We apply early stopping, and reload the best model and halve the learning rate when the performance on the development set does not increase after 5 epochs. We repeat this procedure for 5 times.

E SEQ2TREE Results on ATIS Data Splits

$\lvert\mathbb{L}\rvert$	SUP.	SEQ2TREE
500	63.2	57.1
1000	74.6	69.9
2000	80.4	71.7
3000	82.8	81.5

Table 8: Accuracies of SEQ2TREE and our supervised parser on different data splits of ATIS

We also present results of SEQ2TREE (Dong and Lapata, 2016) trained on the data splits used in Tab. 1, as shown in Tab. 8. Our supervised parser performs consistently better than SEQ2TREE. This is probably due to the fact that our transition-based parser encodes the grammar of the target logical form *a priori* under the ASDL specification, in contrast with SEQ2TREE which need to learn the grammar from the data. This would lead to improved performance when the amount of parallel training data is limited.