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 $\{z_1^s, z_2^s, \dots, z_{i-1}^s\}$. 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 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:

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

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.