Learning Structured Natural Language Representations for Semantic Parsing

Jianpeng Cheng, Siva Reddy, Vijay Saraswat and Mirella Lapata

Presented by : **Rishika Agarwal**

Outline

- Introduction
- Problem Setting
- Model
- Training Objective
- Experimental Results
- Key takeaways

Outline

- **Introduction**
- Problem Setting
- Model
- Training Objective
- Experimental Results
- Key takeaways

Introduction: Semantic Parsing

Convert **natural language utterances** to **logical forms**, which can be executed to yield a **task-specific response**.

Eg:

natural language utterance : How many daughters does Obama have?

Logical form : answer(count(relatives.daughter(Obama)))

Task specific response (answer) : 2

Motivation

Applications of semantic parsing:

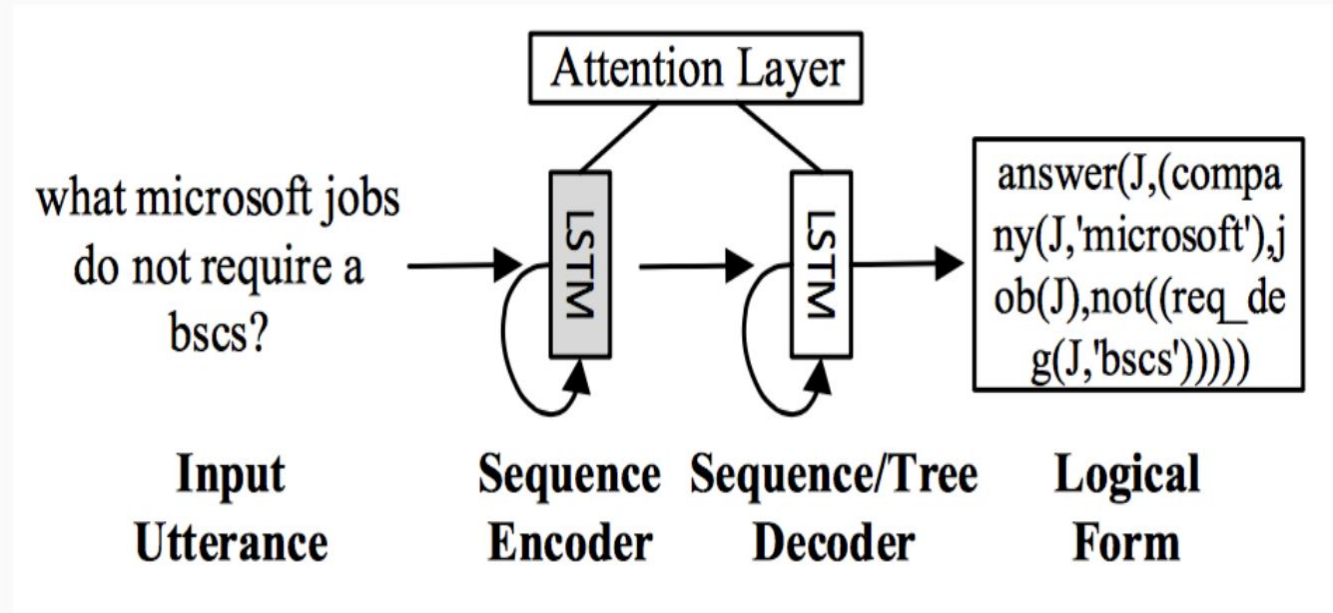
Question answering

Task-oriented dialog

Instructing robots

Neural Semantic Parsing

Neural Sequence to Sequence models : convert utterances into logical strings



Neural Semantic Parsing

Problems :

1) They generate a sequence of tokens (the output may contain extra or missing brackets)

2) They are not type-constrained (the output may be meaningless or ungrammatical).

Handling the problems

The proposed model handles these problems:

- **Tree-structured logical form**: ensures the outputs are well-formed.

- **Domain-general constraints :** ensure outputs are meaningful and executable

Goals of this work

- Improve neural semantic parsing
- Interpret neural semantic parsing

Outline

- Introduction
- **Problem Formulation**
- Model
- Training Objective
- Experimental Results
- Key takeaways

Problem Formulation: Notations

- $\boldsymbol{\kappa}$: knowledge base or a reasoning system
- **x** : a natural language utterance
- **G:** grounded meaning representation of x
- **y**: denotation of G

Our problem is to learn a semantic parser that maps **x** to **G** via an intermediate ungrounded representation **U**

When **G** is executed against $\boldsymbol{\kappa}$, it outputs denotation **y**

Problem Formulation: Notations

Eg:

κ : Knowledge bank

x : How many daughters does Obama have?

G : answer(count(relatives.daughter(Obama)))

y : 2

Grounded and Ungrounded Meaning Representation (G, U)

- Both U and G represented in FunQL
- Advantage of FunQL : convenient to be predicted with RNNs
- U : consists of natural language predicates and domain-general predicates.
- G: consists only of domain-general predicates

Grounded and Ungrounded Meaning Representation (G, U)

Eg: which states do not border texas:

U : answer(exclude(states(all), border(texas)))

G : answer(exclude(state(all), next_to(texas)))

states and *border* are natural language predicates.

Some domain-general predicates

Predicate	Usage	Sub-categories
answer	denotation wrapper	—
type	entity type checking	*stateid*, *cityid*, *riverid*, etc.
all	querying for an entire set of entities	—
aggregation	one-argument meta predicates for sets	*count*, *largest*, *smallest*, etc.
logical connectors	two-argument meta predicates for sets	*intersect*, *union*, *exclude*

Table 1: List of domain-general predicates.

Problem Formulation

- They constrain ungrounded representations to be **structurally isomorphic** to grounded ones

- So to get the target logical form G, just replace predicates in U with symbols in knowledge base

- Will see in detail later

Outline

- Introduction
- Problem Formulation
- **Model**
- Training Objective
- Experimental Results
- Key takeaways

Model

Recall the flow:

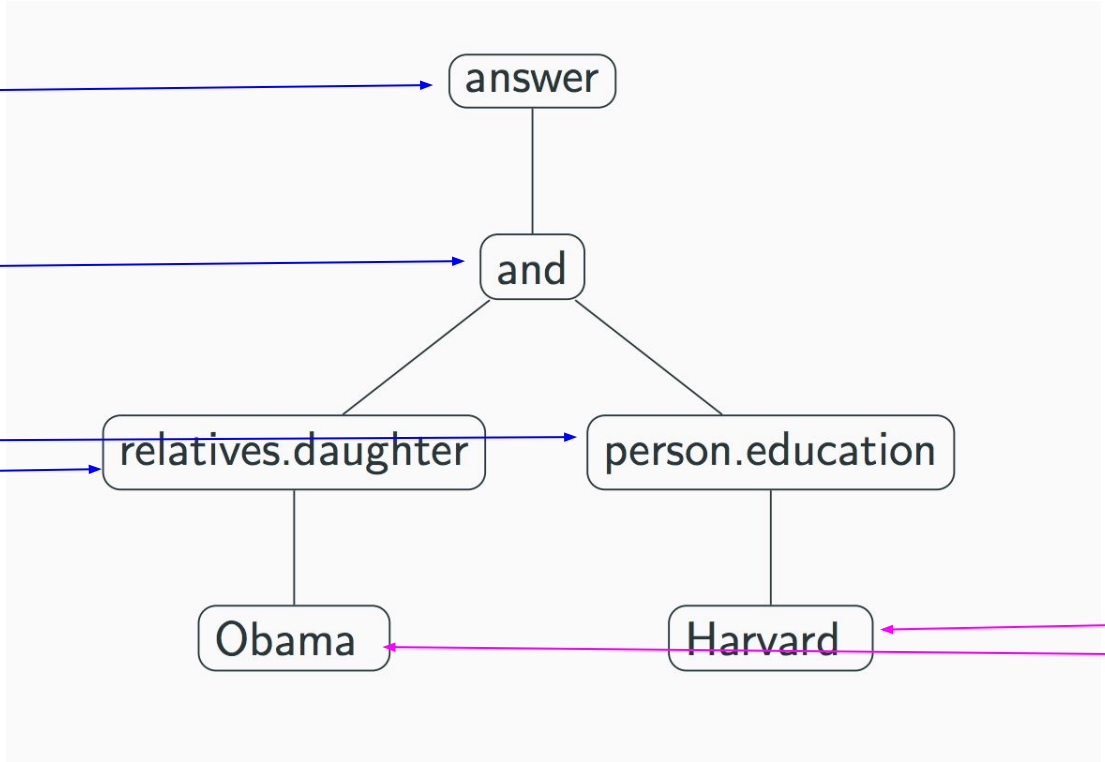
- Convert utterance (x) to an intermediate representation (U)
- Ground U to knowledge base to get G

Model: Generating Ungrounded Represenations (U)

- x mapped to U with a transition-based algorithm
- Transition system generates the representation by following a derivation tree
- Derivation tree contains a set of applied rules and follows some canonical generation order (e.g., depth-first)

x : Which of Obama's daughter studied in Harvard?

G : *answer(and(relatives.daughter(Obama) , person.education(Harvard)))*



Non terminals

NTs are predicates

Ts are entities, or the special token 'all'

Terminals

Tree generation actions

1. Generate non-terminal node **(NT)**
2. Generate terminal node **(TER)**
3. Complete subtree **(REDUCE)**

Tree generation actions

1. Generate non-terminal node **(NT)**

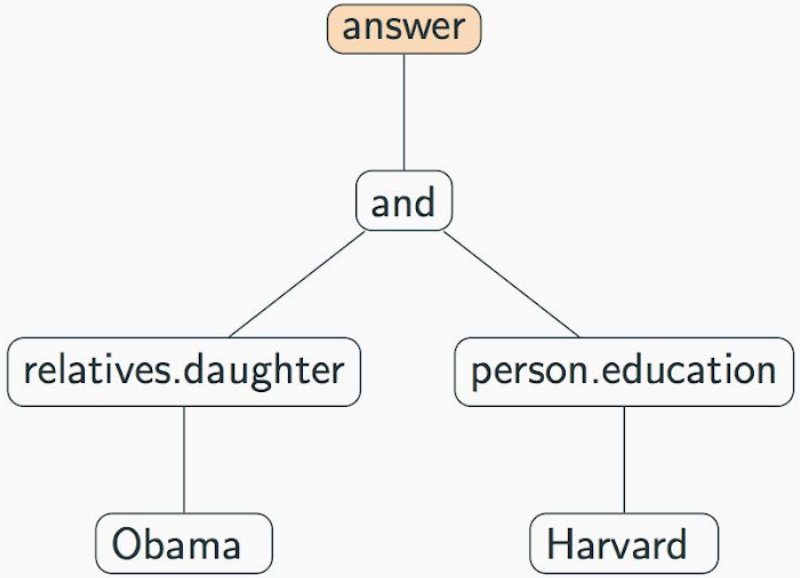
2. Generate terminal node **(TER)**

3. Complete subtree **(REDUCE)**

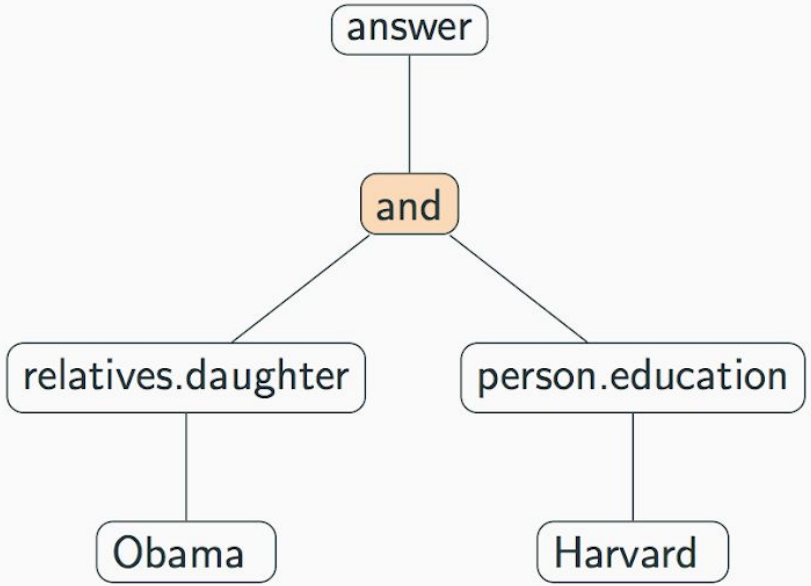
Recall RNNG

Combined with FunQL:

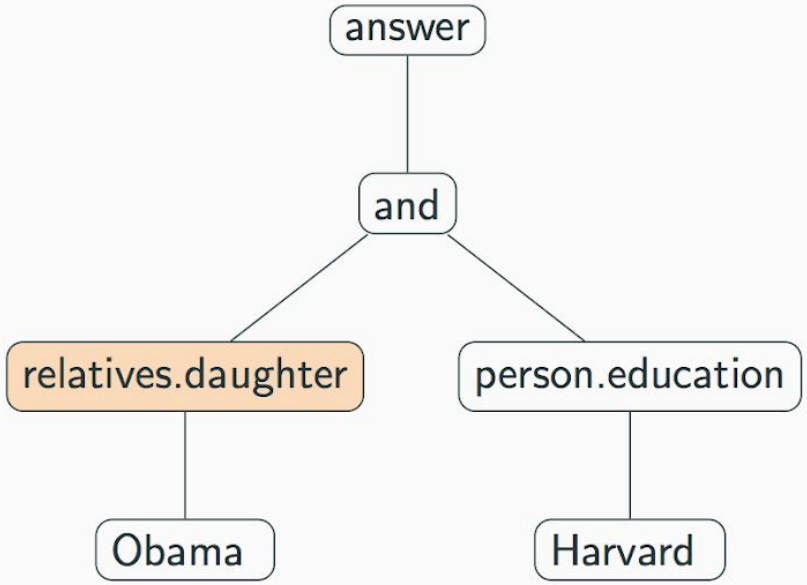
- NT further includes: count, argmax, argmin, and, relation,..
- TER further includes: entity , *all*



Oracle action sequence
NT(answer)

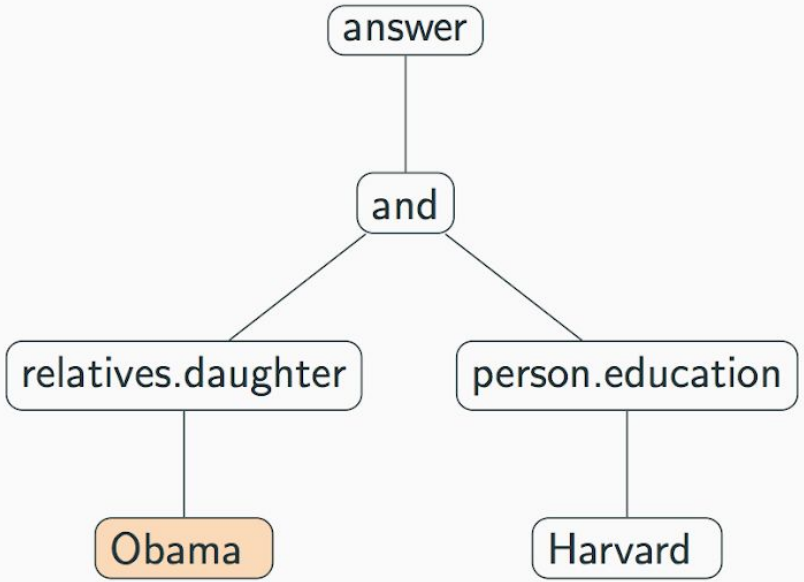


Oracle action sequence
NT(answer), NT(and)

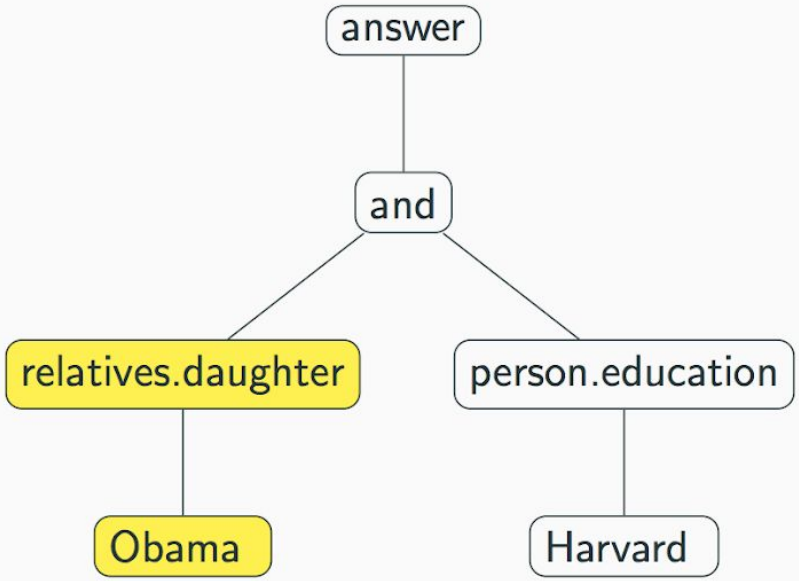


Oracle action sequence

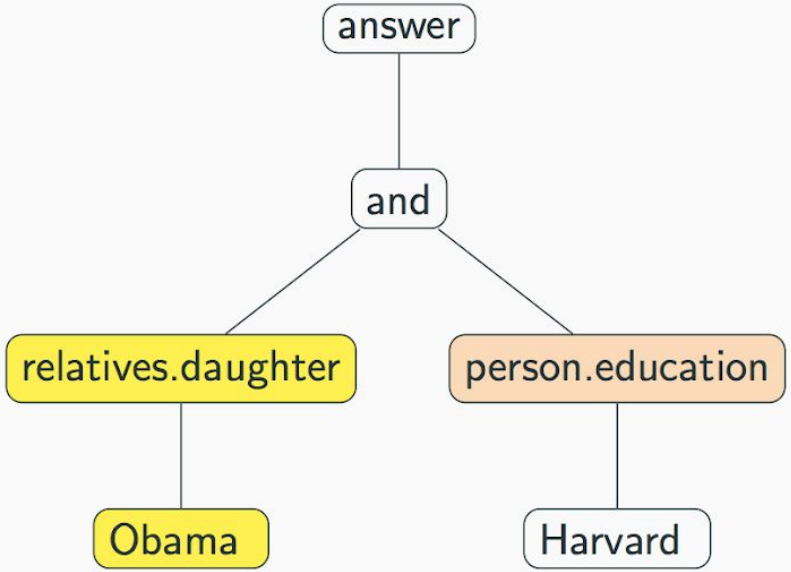
NT(answer), NT(and), **NT(relation)**



Oracle action sequence
NT(answer), NT(and), NT(relation), **TER(entity)**

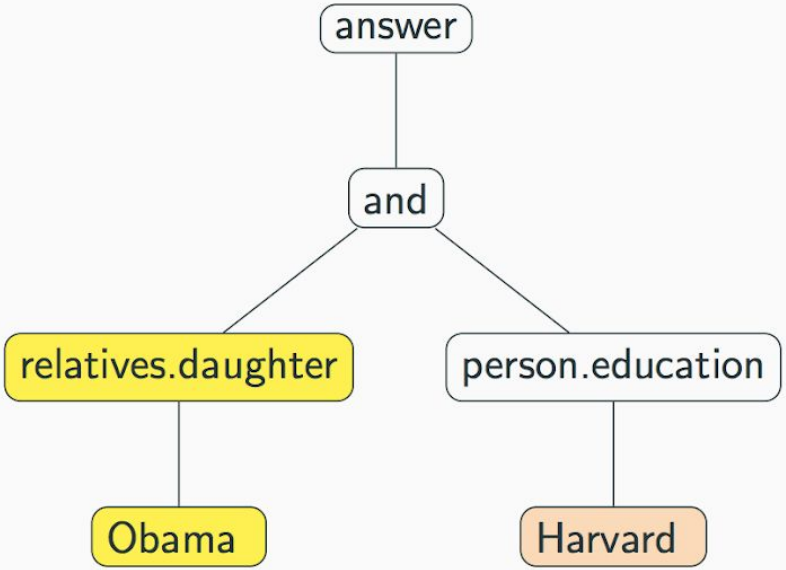


Oracle action sequence
NT(answer), NT(and), NT(relation), TER(entity), **REDUCE**



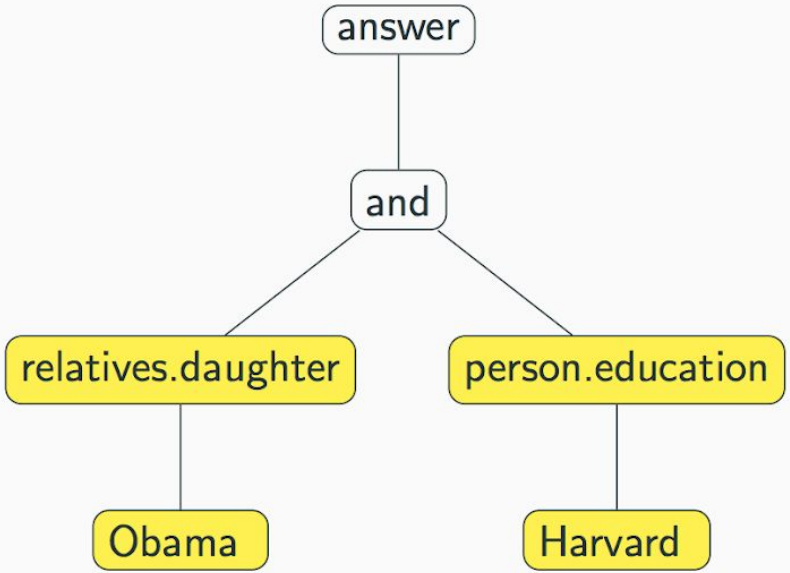
Oracle action sequence

NT(answer), NT(and), NT(relation), TER(entity), REDUCE, **NT(relation)**

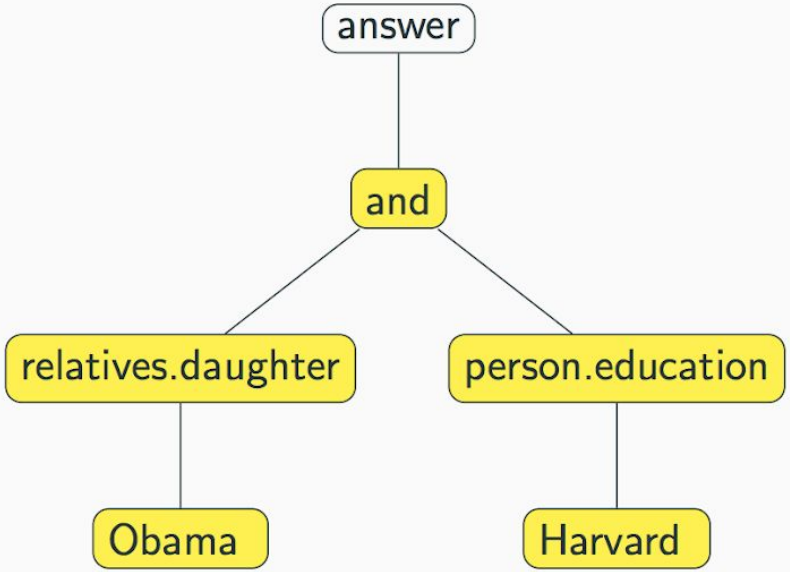


Oracle action sequence

NT(answer), NT(and), NT(relation), TER(entity), REDUCE, NT(relation), **TER(entity)**

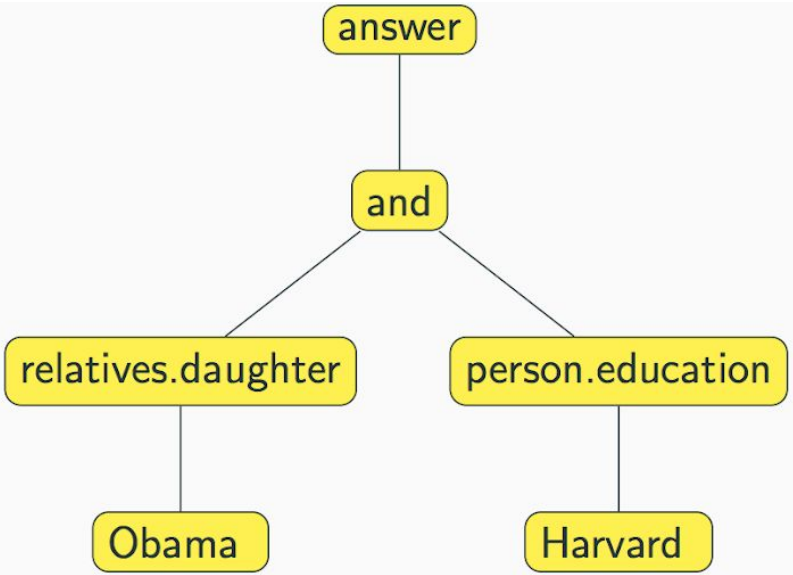


Oracle action sequence
NT(answer), NT(and), NT(relation), TER(entity), REDUCE, NT(relation), TER(entity), **REDUCE**



Oracle action sequence

NT(answer), NT(and), NT(relation),
TER(entity), REDUCE, NT(relation),
TER(entity), REDUCE, **REDUCE**



Oracle action sequence

NT(answer), NT(and), NT(relation), TER(entity), REDUCE, NT(relation), TER(entity), REDUCE, REDUCE, **REDUCE**

- The model generates the ungrounded representation U conditioned on utterance x by recursively calling one of the above three actions.

- U is defined by a sequence of actions (**a**) and a sequence of term choices (**u**)

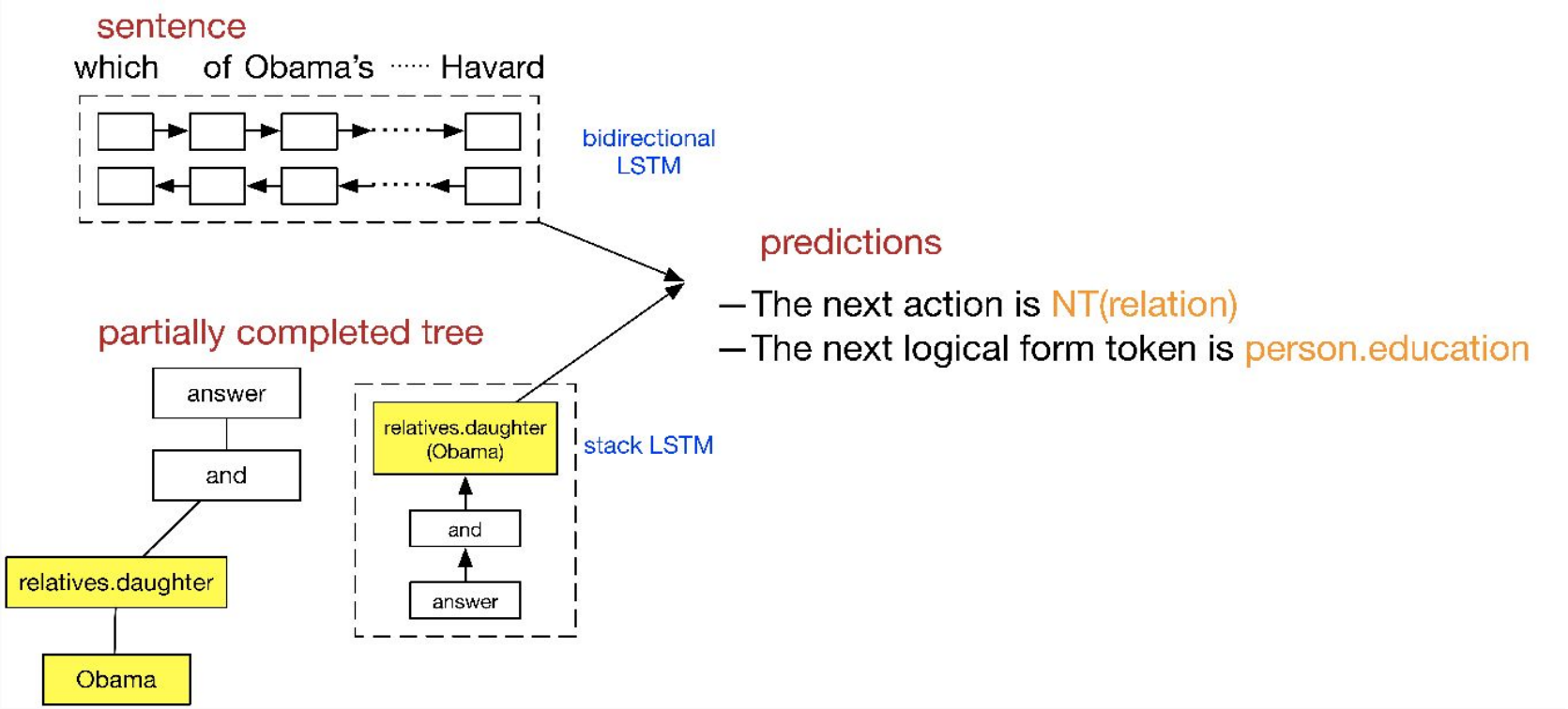
$$
\begin{aligned}
p(U|x) &= p(a, u|x) \qquad (1) \\
&= \prod_{t=1}^{T} p(a_t|a_{<t}, x) p(u_t|a_{<t}, x)^{\mathbb{I}(a_t \neq \text{RED})}
\end{aligned}
$$

where $\mathbb{I}$ is an indicator function.

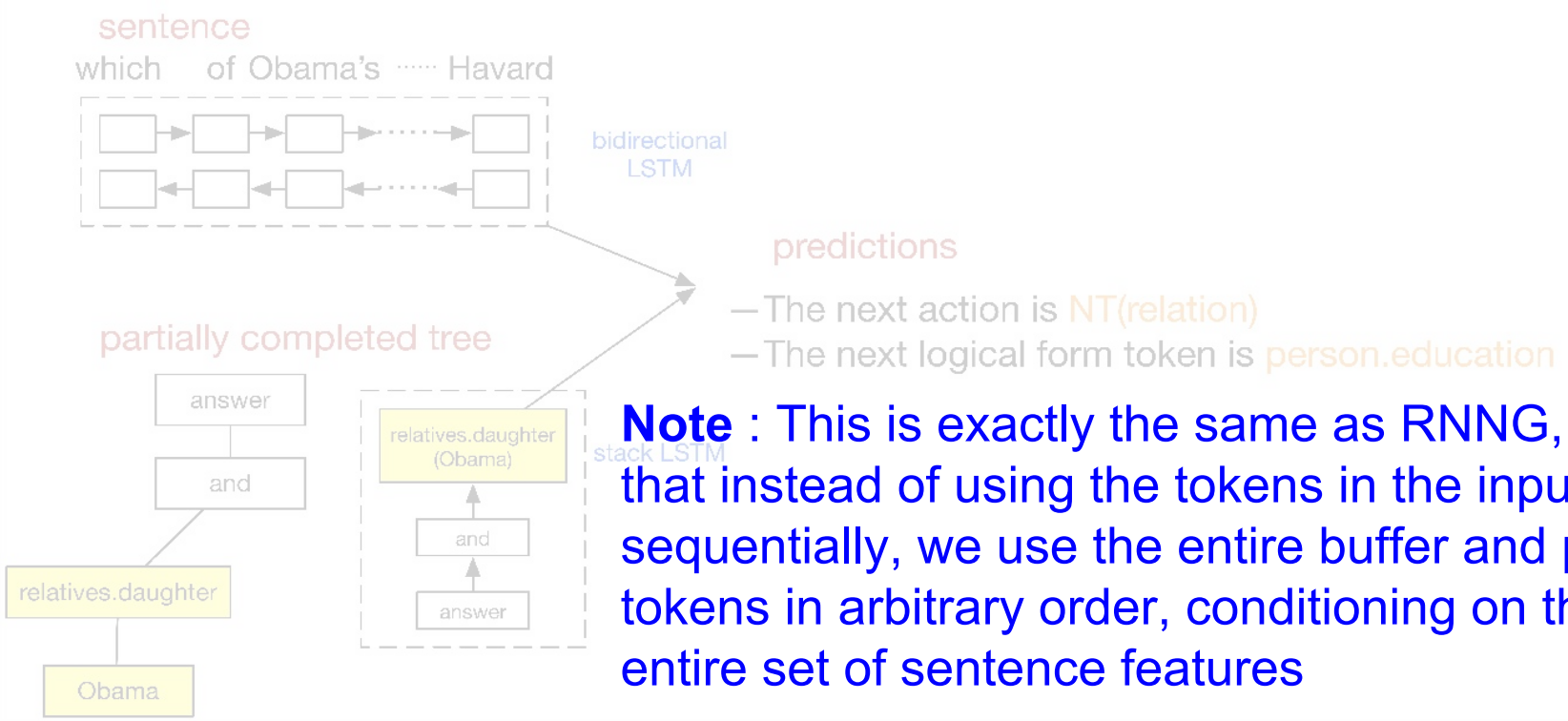
- The actions (a) and logical tokens (u) are predicted by encoding :
  -Input buffer (b) with a **bidirectional LSTM** (encodes sentence context)
  -Output stack (s) with a **stack-LSTM** (encodes generation history)

- At each time step, the model uses the concatenated representation to predict an action and then a logical token

- The actions (a) and logical tokens (u) are predicted by encoding :
  -Input buffer (b) with a **bidirectional LSTM** (encodes sentence context)
  -Output stack (s) with a **stack-LSTM** (encodes generation history)



- The actions (a) and logical tokens (u) are predicted by encoding :
  -Input buffer (b) with a **bidirectional LSTM** (encodes sentence context)
  -Output stack (s) with a **stack-LSTM** (encodes generation history)

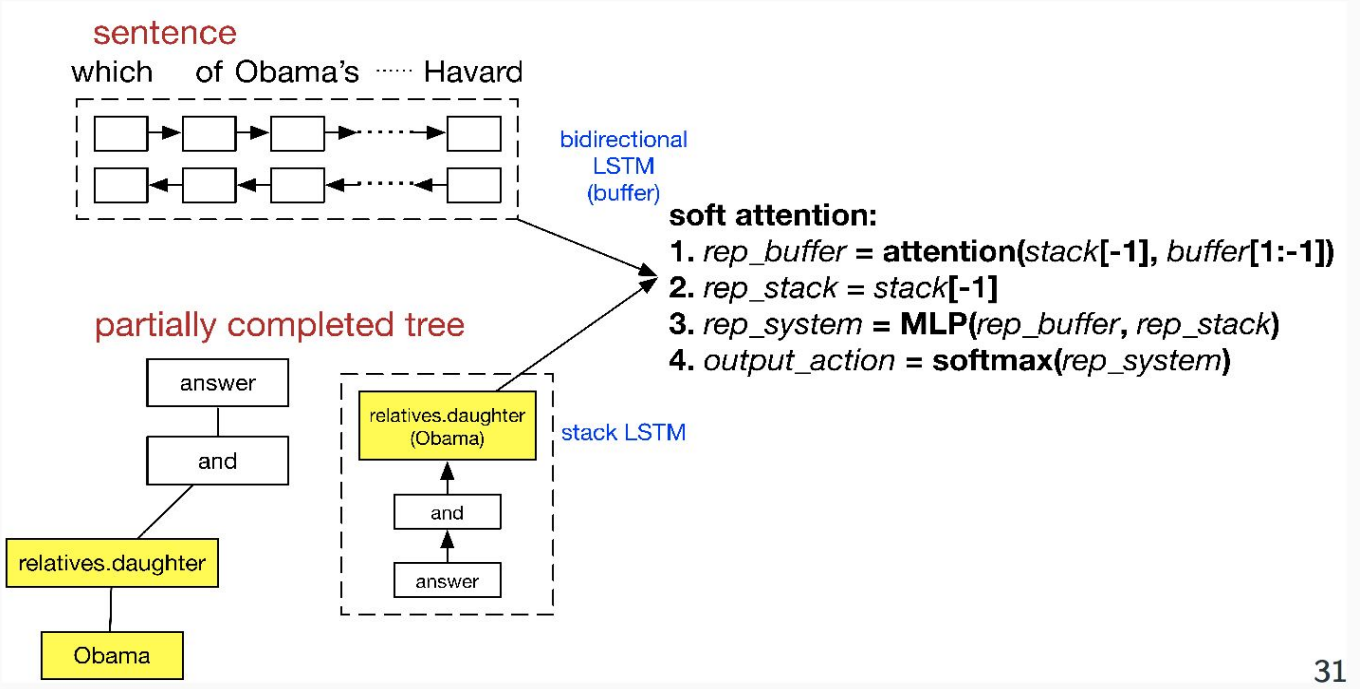


Note : This is exactly the same as RNNG, except that instead of using the tokens in the input buffer sequentially, we use the entire buffer and pick tokens in arbitrary order, conditioning on the entire set of sentence features

Predicting the next action (a_t)

$$p(a_t|a_{<t}, x) \propto \exp(W_a \cdot e_t)$$

$e_t = b_t \mid s_t$



Predicting the next logical term (u_t)

When a_t is NT or TER, an ungrounded term u_t needs to be chosen from the candidate list depending on the specific placeholder x.

select a domain-general tern $p(u_t^{\text{GENERAL}}|a_{<t}, x) \propto \exp(W_p \cdot e_t)$

select a natural language ter $p(u_t^{\text{NL}}|a_{<t}, x) \propto \exp(s_t)$

Model: Generating grounded representation (G)

Since ungrounded structures are **isomorphic** to the target meaning representation -- converting U to G becomes a simple **lexical mapping problem**

- To map u_t to g_t , we compute the conditional probability of g_t given u_t with a bi-linear neural network

$$p(g_t|u_t) \propto \exp \vec{u_t} \cdot W_{ug} \cdot \vec{g_t}^\top$$

Outline

- Introduction
- Problem Formulation
- Model
- **Training Objective**
- Experimental Results
- Key takeaways

Training objective

Two cases :

- When the target meaning representation (G) is available
- When only denotations (y) are available (will not focus on this)

Training objective : When G is known

Goal : Maximize the likelihood of the grounded meaning representation p(**G**|**x**) over all training examples.

p(**G**|**x**) = p (**a, g**| **x**) = p (**a**|**x**) p (**g**|**x**)

Where **a** = action term sequence, **g** = grounded term sequence

Training objective : When G is known

$$\mathcal{L}_G = \mathcal{L}_a + \mathcal{L}_g$$

$$\mathcal{L}_a = \sum_{x \in \mathcal{T}} \log p(a|x) = \sum_{x \in \mathcal{T}} \sum_{t=1}^{n} \log p(a_t|x)$$

$$\mathcal{L}_g = \sum_{x \in \mathcal{T}} \sum_{u} [p(u|x) \log p(g|u, x)]$$

$$= \sum_{x \in \mathcal{T}} \sum_{u} \left[p(u|x) \sum_{t=1}^{k} \log p(g_t|u_t) \right]$$

$\mathcal{L}_g$ is lower bound of *log p(**g**|**x**)*

L_G optimtized by a method described in Lieu et al.

Outline

- Introduction
- Problem Formulation
- Model
- Training Objective
- **Experimental Results**
- Key takeaways

Experiments: Datasets used

1. **GeoQuery** - 880 questions and database queries about US geography
2. **Spades** - 93,319 questions derived from CLUEWEB09 sentences
3. **WebQuestions** - 5,810 question-answer pairs (real questions asked by people on the Web)
4. **GraphQuestions** - contains 5,166 question-answer pairs created by showing Freebase graph queries to Amazon Mechanical Turk workers and asking them to paraphrase them into natural language.

Experiments: Datasets used

- GeoQuery has utterance-logical form pairs
- Other datasets have utterance-denotation pairs

Experiments: Implementation Details

- **Adam** optimizer for training with an initial learning rate of 0.001, two momentum parameters [0.99, 0.999], and batch size 1
- The dimensions of the word embeddings, LSTM states, entity embeddings and relation embeddings are [50, 100, 100, 100]
- The word embeddings were initialized with **Glove** embeddings
- All other embeddings were randomly initialized

Experiments: **Results**

Authors' method is called SCANNER
(**S**ymboli**C** me**ANiN**g r**E**p**R**esentation)

Models	Accuracy
Zettlemoyer and Collins (2005)	79.3
Zettlemoyer and Collins (2007)	86.1
Kwiatkowksi et al. (2010)	87.9
Kwiatkowski et al. (2011)	88.6
Kwiatkowski et al. (2013)	88.0
Zhao and Huang (2015)	88.9
Liang et al. (2011)	91.1
Dong and Lapata (2016)	84.6
Jia and Liang (2016)	85.0
Jia and Liang (2016) with extra data	89.1
SCANNER	86.7

Table 5: GEOQUERY results.

Models	F1
Berant et al. (2013)	35.7
Yao and Van Durme (2014)	33.0
Berant and Liang (2014)	39.9
Bast and Haussmann (2015)	49.4
Berant and Liang (2015)	49.7
Reddy et al. (2016)	50.3
Bordes et al. (2014)	39.2
Dong et al. (2015)	40.8
Yih et al. (2015)	52.5
Xu et al. (2016)	53.3
Neural Baseline	48.3
SCANNER	49.4

Table 3: WEBQUESTIONS results.

Experiments: **Results**

Models	F1
SEMPRE (Berant et al., 2013)	10.80
PARASEMPRE (Berant and Liang, 2014)	12.79
JACANA (Yao and Van Durme, 2014)	5.08
Neural Baseline	16.24
SCANNER	17.02

Table 4: GRAPHQUESTIONS results. Numbers comparison systems are from Su et al. (2016).

Models	F1
Unsupervised CCG (Bisk et al., 2016)	24.8
Semi-supervised CCG (Bisk et al., 2016)	28.4
Neural baseline	28.6
Supervised CCG (Bisk et al., 2016)	30.9
Rule-based system (Bisk et al., 2016)	31.4
SCANNER	31.5

Table 6: SPADES results.

Experiments: Discussion

- SCANNER achieves state of the art results on Spades and GraphQuestions
- Obtains competitive results on GeoQuery and WebQuestions
- On WebQuestions, it performs on par with the best symbolic systems, despite not having access to any linguistically-informed syntactic structures.

Experiments: Evaluating ungrounded meaning representation

- To evaluate the quality of intermediate representations generated, they compare it to manually created representations on GeoQuery

Metrics	Accuracy
Exact match	79.3
Structure match	89.6
Token match	96.5

Table 7: GEOQUERY evaluation of ungrounded meaning representations. We report accuracy against a manually created gold standard.

Outline

- Introduction
- Problem Formulation
- Model
- Training Objective
- Experimental Results
- **Key takeaways**

Key Takeaways

- A model which jointly learns how to parse natural language semantics and the lexicons that help grounding

Key Takeaways

- A model which jointly learns how to parse natural language semantics and the lexicons that help grounding

- More interpretable than previous neural semantic parsers, as intermediate ungrounded representation is useful to inspect what the model has learned

Key Takeaways

- A model which jointly learns how to parse natural language semantics and the lexicons that help grounding

- More interpretable than previous neural semantic parsers, as intermediate ungrounded representation is useful to inspect what the model has learned

- Model constrained the ungrounded and grounded representations to be isomorphic - sidesteps the challenge of structure mapping, but restricts the expressiveness of the model

Key Takeaways

- A model which jointly learns how to parse natural language semantics and the lexicons that help grounding

- More interpretable than previous neural semantic parsers, as intermediate ungrounded representation is useful to inspect what the model has learned

- Model constrained the ungrounded and grounded representations to be isomorphic - sidesteps the challenge of structure mapping, but restricts the expressiveness of the model

Questions?