Carnegie Mellon
School of Computer Science

Language Grounding to Vision and Control

Executable Semantic Parsing

Katerina Fragkiadaki

What is ESP?

Goal: Building systems capable of ``*Language understanding*”

…the system must produce, given context c, an appropriate action a upon receiving an input utterance x by human, e.g., (x: a query c: a knowledge base a: answer) or (x: a command, c: the robot’s environment a: an action sequence to be executed by the robot)

What is ESP?

Goal: Building systems capable of ``*Language understanding*”

…the system must produce, given context c, an appropriate action a upon receiving an input utterance x by human, e.g., (x: a query c: a knowledge base a: answer) or (x: a command, c: the robot’s environment a: an action sequence to be executed by the robot)

ESP = Mapping of Language into Logical Form

Logical Form = A program which if executed will yield the desired action (behavior), e.g., an answer, an item retrieval, mobile phone operation invocation etc.

Thus ESP central to language understanding

History

First ESP systems were based on hand crafted rules, e.g., SHRDLU, could both answer questions and perform actions in a toy blocks world environment, such as: Find a block which is taller than the one you are holding and put it into the box."

T. Winograd. Understanding Natural Language. Academic Press, 1972.

History

First ESP systems were based on hand crafted rules, e.g., SHRDLU, could both answer questions and perform actions in a toy blocks world environment, such as: Find a block which is taller than the one you are holding and put it into the box.”

T. Winograd. Understanding Natural Language. Academic Press, 1972.

Statistical semantic parsing. In 1990, the statistical paradigm prevailed: collect pairs of (input utterances, desired logical forms) and train a statistical model for ESP.

L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classication with probabilistic categorial grammars.

M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming.

Y. W. Wong and R. J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus.

History

First ESP systems were based on hand crafted rules, e.g., SHRDLU, could both answer questions and perform actions in a toy blocks world environment, such as: Find a block which is taller than the one you are holding and put it into the box."

T. Winograd. Understanding Natural Language. Academic Press, 1972.

Statistical semantic parsing. In 1990, the statistical paradigm prevailed: collect pairs of (input utterances, desired logical forms) and train a statistical model for ESP.

L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classication with probabilistic categorial grammars.

M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming.

Y. W. Wong and R. J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus.

Weaker supervision. Instead of pairs of (input utterances, desired logical forms) use pairs of (input utterances, answers)

History

First ESP systems were based on hand crafted rules, e.g., SHRDLU, could both answer questions and perform actions in a toy blocks world environment, such as: Find a block which is taller than the one you are holding and put it into the box."

T. Winograd. Understanding Natural Language. Academic Press, 1972.

Statistical semantic parsing. In 1990, the statistical paradigm prevailed: collect pairs of (input utterances, desired logical forms) and train a statistical model for ESP.

L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classication with probabilistic categorial grammars.

M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming.

Y. W. Wong and R. J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus.

Weaker supervision. Instead of pairs of (input utterances, desired logical forms) use pairs of (queries, answers).

Scaling up. From small domains for question answering to large scale knowledge bases such as Freebase, Wikidata etc.

Freebase

Freebase Deleted Triples

We also provide a dump of triples that have been deleted from Freebase over time. This is a one-time dump through March 2013. In the future, we might consider providing periodic updates of recently deleted triples, but at the moment we have no specific timeframe for doing so, and are only providing this one-time dump.

The dump is distributed as a .tar.gz file (2.1Gb compressed, 7.7Gb uncompressed). It contains 63,036,271 deleted triples in 20 files (there is no particular meaning to the individual files, it is just easier to manipulate several smaller files than one huge file).

Thanks to Chun How Tan and John Giannandrea for making this data release possible.

Total triples: 63 million

Updated: June 9, 2013

Data Format: CSV

License: CC-BY

2 GB gzip

8 GB uncompressed

DOWNLOAD

The data format is essentially CSV with one important caveat. The object field may contain any characters, including commas (as well as any other reasonable delimiters you could think of). However, all the other fields are guaranteed not to contain commas, so the data can still be parsed unambiguously.

The columns in the dataset are defined as:

- creation_timestamp (Unix epoch time in milliseconds)
- creator
- deletion_timestamp (Unix epoch time in milliseconds)
- deletor
- subject (MID)
- predicate (MID)
- object (MID/Literal)
- language_code

Google's Knowledge Graph

Knowledge Graph

From Wikipedia, the free encyclopedia

This article is about Google's specific implementation of knowledge graph technology. For knowledge engine technology in general, see Knowledge engine.

The **Knowledge Graph** is a knowledge base used by Google to enhance its search engine's search results with semantic-search information gathered from a wide variety of sources. Knowledge Graph display was added to Google's search engine in 2012, starting in the United States, having been announced on May 16, 2012.[1] It uses a graph database to provide structured and detailed information about the topic in addition to a list of links to other sites. The goal is that users would be able to use this information to resolve their query without having to navigate to other sites and assemble the information themselves.[2] The short summary provided in the knowledge graph is often used as a spoken answer in Google Assistant searches.[3]

According to some news websites, the implementation of Google's Knowledge Graph has played a role in the page view decline of various language versions of Wikipedia.[4][5][6][7] As of the end of 2016, knowledge graph holds over 70 billion facts.[8]

Knowledge Graph data about Thomas Jefferson displayed on Google Web Search, as of January 2015.

Contents [hide]

History [edit]

Prior to Knowledge Graph and Linked Data, Semantic Link Network (SLN) method was used for creating a self-organised semantic networking approach to render knowledge. The theory was published in 2004[9] and updated in 2012.[10] According to Google, information in the Knowledge Graph is derived from many sources, including the *CIA World Factbook*, Wikidata, and Wikipedia.[1] The feature is similar in intent to answer engines such as Wolfram Alpha and efforts such as Linked Data and DBpedia. As of 2012, its semantic network contained over 570 million objects and more than 18 billion facts about and relationships between different objects that are used to understand the meaning of the keywords entered for the search.[11][12]

Semantic parsing beyond QA

VQA

Identifying objects in the image from referential expressions

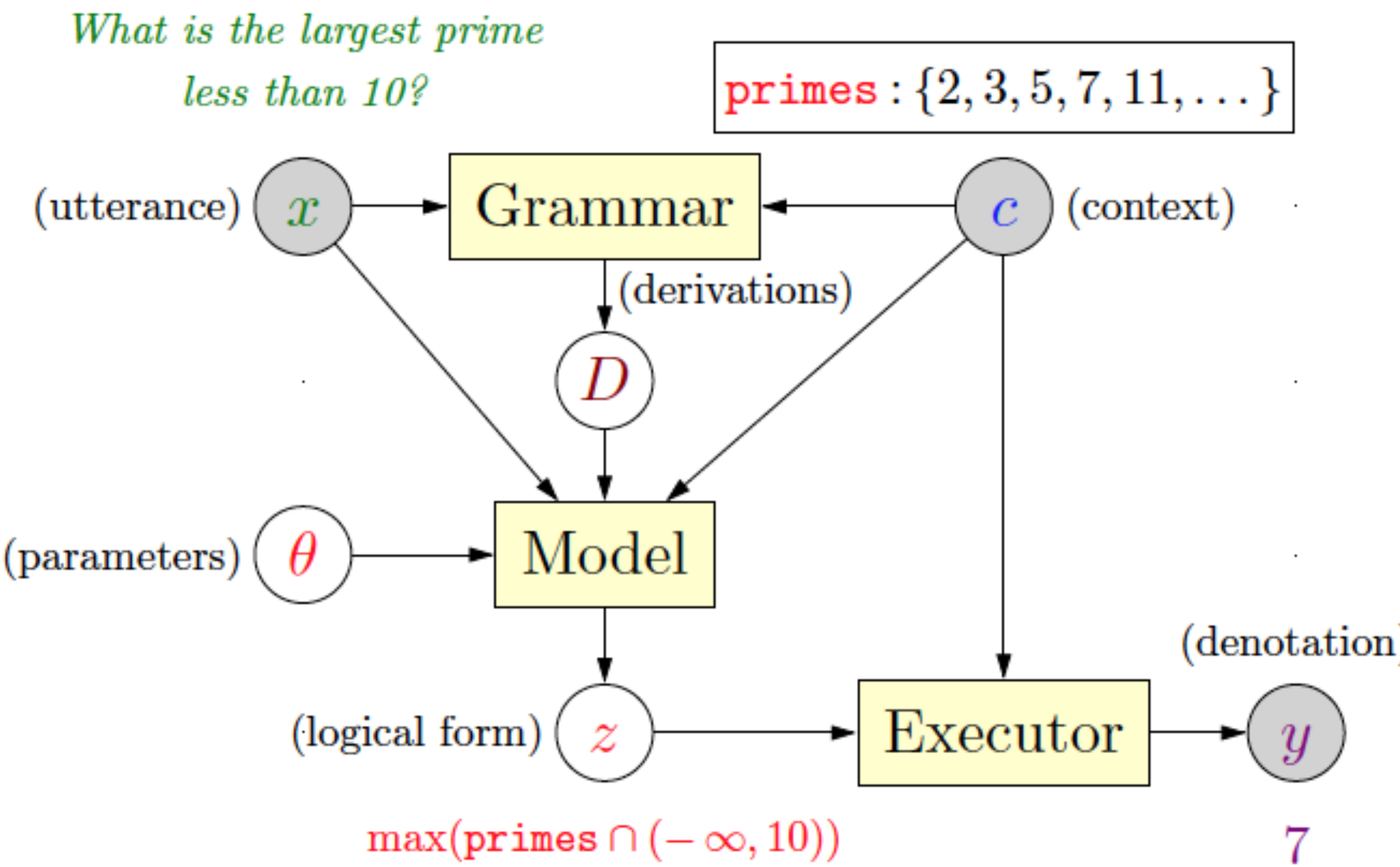
Robot navigation/manipulation

Mobile phone operations

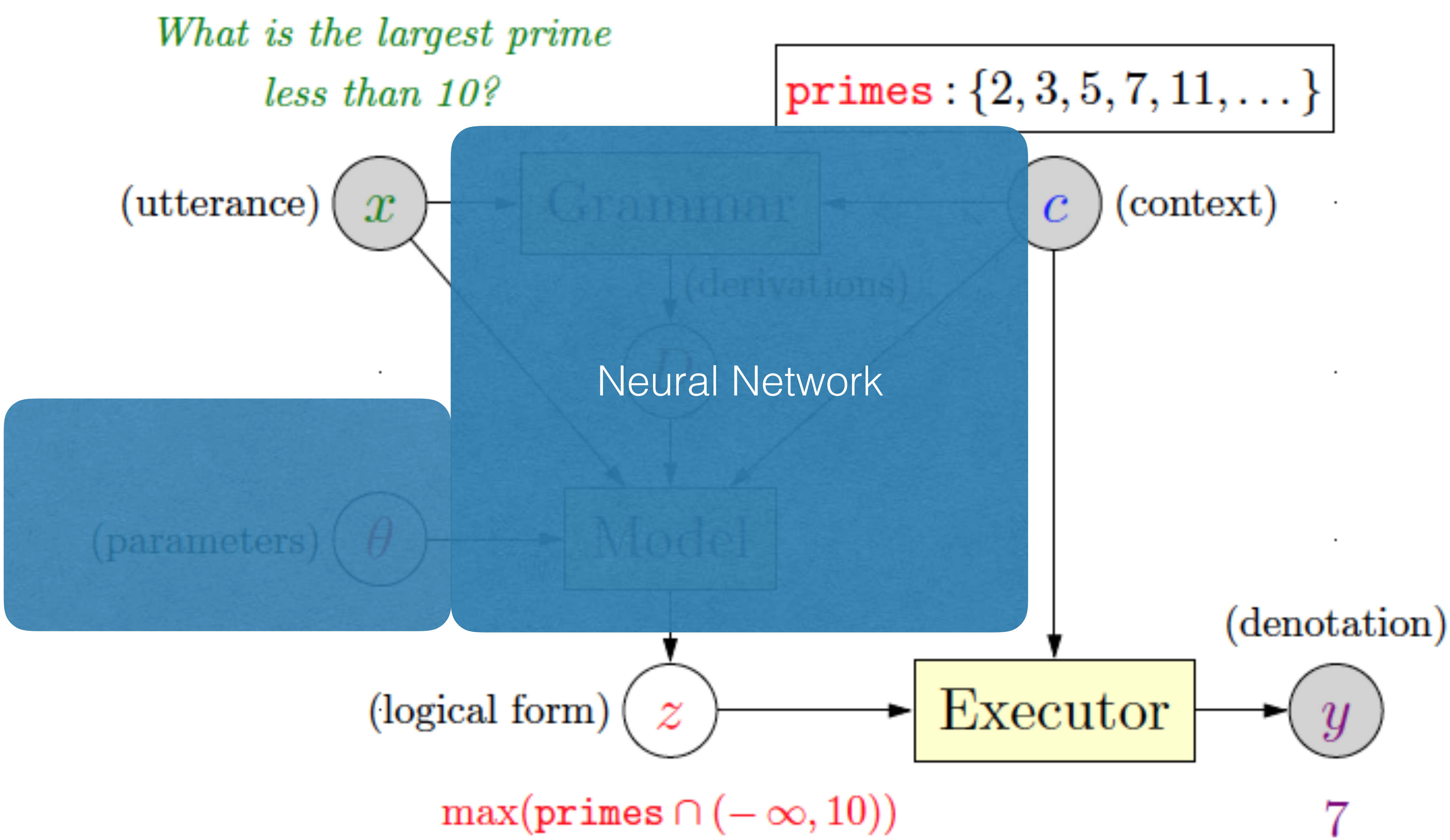
flight booking

Etc.

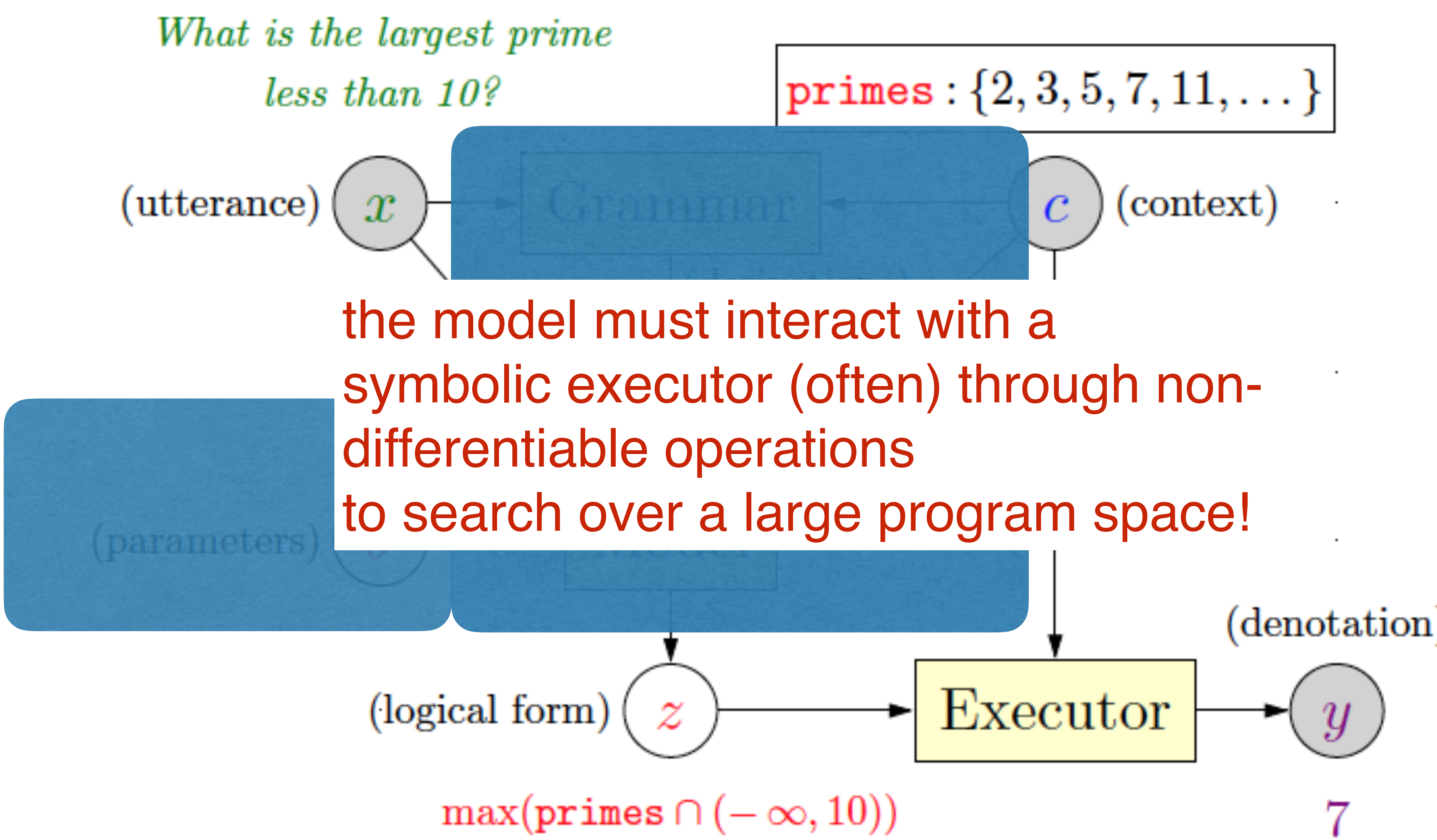
ESP before 2012



ESP nowadays



ESP nowadays



the model must interact with a symbolic executor (often) through non-differentiable operations
to search over a large program space!

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

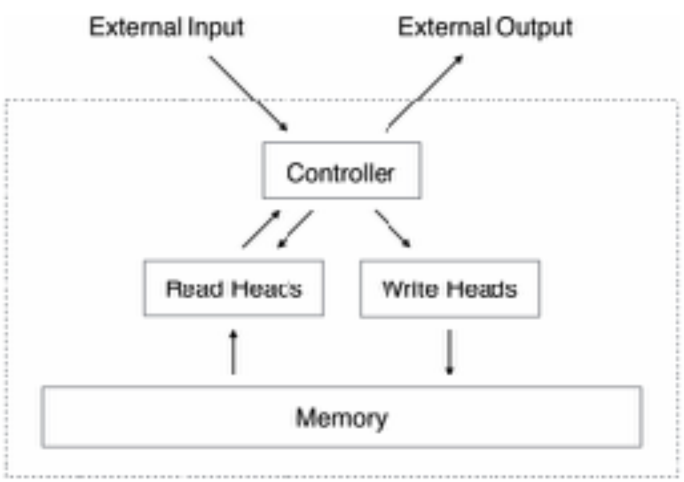
Similarities to Neural Turing machines

- They use input/output examples to learn programs, that if executed will generate the desired output, e.g., in NTM, copying of a string
- They both use differentiable memory

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

Differences to Neural Turing machines

- NTM use differentiable low-level actions such as, read/write in memory, read from inout and write in the designated output



- NSM use high-level non-differentiable (symbolic) actions, such as *filter.* They are pre-coded and NSM just learn to encode them by using their corresponding token and arguments (programmability through language)

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

Learning SP from annotated input/output pairs, without annotated logical form.

Given a knowledge base K, and a question x , produce a program or logical form z that when executed against K generates the right answer y. Let E denote a set of entities (e.g., ABELINCOLN),1 and let P denote a set of properties (e.g., PLACEOFBIRTH). A knowledge base K is a set of assertions or triples (e1; p; e2), such as (ABELINCOLN, PLACEOFBIRTH, HODGENVILLE).

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

Interfacing with the external world (e.g., querying a database). Two choices:

1. Low-level actions, e.g., move to the left, write on the output tape, copy in the register etc., which can be differentiable, e.g., Neural Turing Machines learning to copy a string (remember in NTM all operations, read and write to memory were soft).

2. High-level programming language, where we can call high level functions, but we cannot back propagate through their operations. NSM choose this because it generalize across input of the arguments, e.g., copy can be one operation as opposed to be broken into many many elementary steps: *``we implement operations necessary for semantic parsing with an ordinary programming language instead of trying to learn them with a neural network"*

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE COMPUTER

Each program is comprised of a set of function (F A_1 A_2..) or the Return symbol.

$(\ \mathit{Hop}\ r\ p\ ) \Rightarrow \{e_2 \mid e_1 \in r, (e_1, p, e_2) \in \mathbb{K}\}$
$(\ \mathit{ArgMax}\ r\ p\ ) \Rightarrow \{e_1 \mid e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \geq e\}$
$(\ \mathit{ArgMin}\ r\ p\ ) \Rightarrow \{e_1 \mid e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \leq e\}$
$(\ \mathit{Filter}\ r_1\ r_2\ p\ ) \Rightarrow \{e_1 \mid e_1 \in r_1, \exists e_2 \in r_2 : (e_1, p, e_2) \in \mathbb{K}\}$

Table 1: Interpreter functions. r represents a variable, p a property in Freebase. $\geq$ and $\leq$ are defined on numbers and dates.

The program executor is a LISP interpreter.

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE OUTPUT

Each program is comprised of a set of function (F A_1 A_2..) or the Return symbol.

$(\ \mathit{Hop}\ r\ p\ ) \Rightarrow \{e_2 \mid e_1 \in r, (e_1, p, e_2) \in \mathbb{K}\}$
$(\ \mathit{ArgMax}\ r\ p\ ) \Rightarrow \{e_1 \mid e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \geq e\}$
$(\ \mathit{ArgMin}\ r\ p\ ) \Rightarrow \{e_1 \mid e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \leq e\}$
$(\ \mathit{Filter}\ r_1\ r_2\ p\ ) \Rightarrow \{e_1 \mid e_1 \in r_1, \exists e_2 \in r_2 : (e_1, p, e_2) \in \mathbb{K}\}$

Table 1: Interpreter functions. r represents a variable, p a property in Freebase. $\geq$ and $\leq$ are defined on numbers and dates.

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE OUTPUT

Each program is comprised of a set of function (F A_1 A_2..) or the Return symbol.

$$(\, \mathit{Hop} \; r \; p \,) \Rightarrow \{e_2 | e_1 \in r, (e_1, p, e_2) \in \mathbb{K}\}$$

$$(\, \mathit{ArgMax} \; r \; p \,) \Rightarrow \{e_1 | e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \geq e\}$$

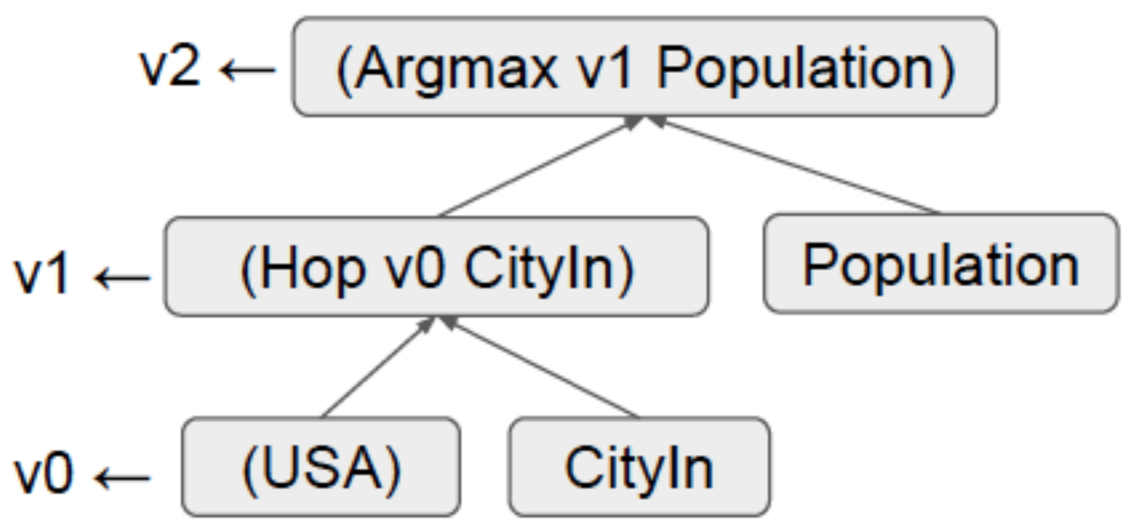
$$(\, \mathit{ArgMin} \; r \; p \,) \Rightarrow \{e_1 | e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \leq e\}$$

$$(\, \mathit{Filter} \; r_1 \; r_2 \; p \,) \Rightarrow \{e_1 | e_1 \in r_1, \exists e_2 \in r_2 : (e_1, p, e_2) \in \mathbb{K}\}$$

Table 1: Interpreter functions. r represents a variable, p a property in Freebase. $\geq$ and $\leq$ are defined on numbers and dates.

x: Largest city in the US $\Rightarrow$ *y: NYC*

Compositionality



Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE OUTPUT

Each program is comprised of a set of function (F A_1 A_2..) or the Return symbol.

$$(\mathit{Hop}\ r\ p\ ) \Rightarrow \{e_2 | e_1 \in r, (e_1, p, e_2) \in \mathbb{K}\}$$

$$(\mathit{ArgMax}\ r\ p\ ) \Rightarrow \{e_1 | e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \geq e\}$$

$$(\mathit{ArgMin}\ r\ p\ ) \Rightarrow \{e_1 | e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \leq e\}$$

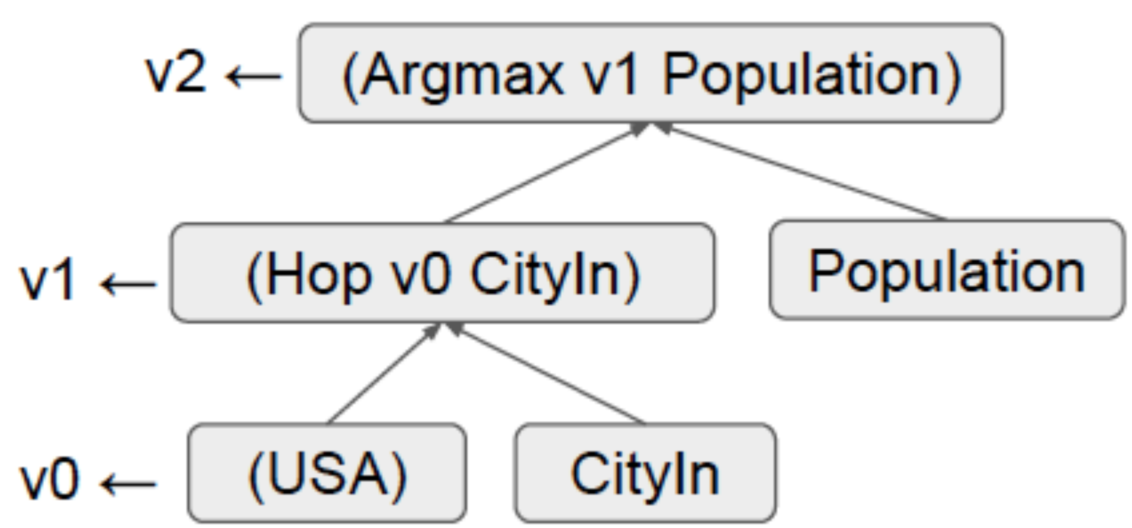
$$(\mathit{Filter}\ r_1\ r_2\ p\ ) \Rightarrow \{e_1 | e_1 \in r_1, \exists e_2 \in r_2 : (e_1, p, e_2) \in \mathbb{K}\}$$

Table 1: Interpreter functions. r represents a variable, p a property in Freebase. $\geq$ and $\leq$ are defined on numbers and dates.

When a function is executed, it returns an entity list that is the expression's denotation in the knowledge base K, and saves it to a new variable.

x: Largest city in the US $\Rightarrow$ *y: NYC*

Compositionality



Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE OUTPUT

Each program is comprised of a set of function (F A_1 A_2..) or the Return symbol.

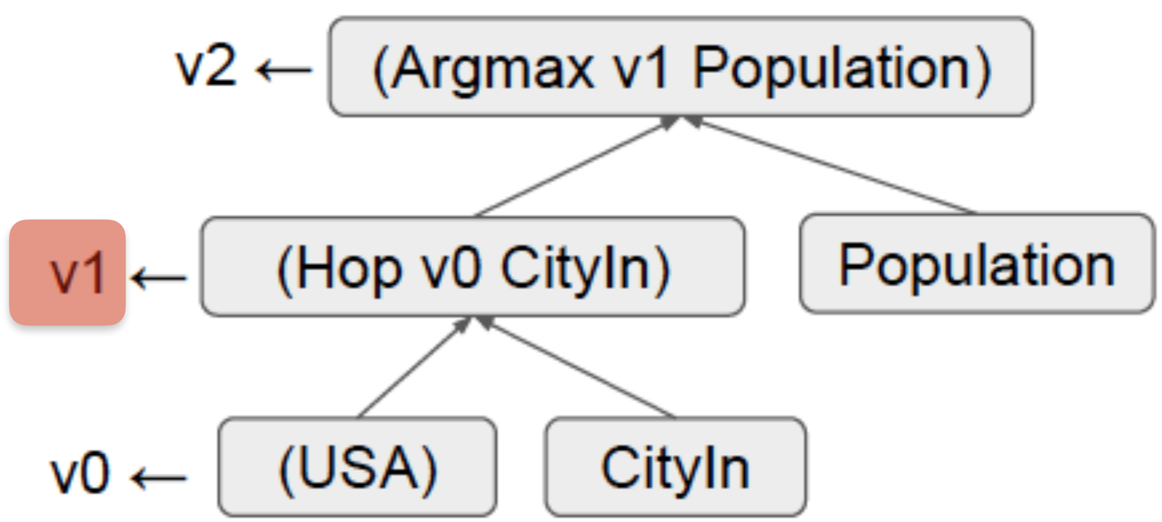
$$
\begin{gathered}
(\ \mathit{Hop}\ r\ p\ ) \Rightarrow \{e_2 | e_1 \in r, (e_1, p, e_2) \in \mathbb{K}\} \\
(\ \mathit{ArgMax}\ r\ p\ ) \Rightarrow \{e_1 | e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \geq e\} \\
(\ \mathit{ArgMin}\ r\ p\ ) \Rightarrow \{e_1 | e_1 \in r, \exists e_2 \in \mathcal{E} : (e_1, p, e_2) \in \mathbb{K}, \forall e : (e_1, p, e) \in \mathbb{K}, e_2 \leq e\} \\
(\ \mathit{Filter}\ r_1\ r_2\ p\ ) \Rightarrow \{e_1 | e_1 \in r_1, \exists e_2 \in r_2 : (e_1, p, e_2) \in \mathbb{K}\}
\end{gathered}
$$

Table 1: Interpreter functions. r represents a variable, p a property in Freebase. $\geq$ and $\leq$ are defined on numbers and dates.

x: Largest city in the US $\Rightarrow$ *y: NYC*

Compositionality

This intermediate variable holds "cities in USA"



Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE COMPUTER

The program executor is a LISP interpreter.

Code assistance: It provides a set of valid tokens at each time step during decoding by checking for syntactic and semantic errors:

- Syntactic: if the previous token is “(”, the next token must be a function name, and if the previous token is “Hop”, the next token must be a variable
- Semantic (run time error): If the previously generated tokens were “(Hop r”, the next available token is restricted to properties that are reachable from entities in r in the KB.

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE PROGRAMMER

The “programmer” needs to map natural language into a program, which is a sequence of tokens that reference operations and values in the “computer”.

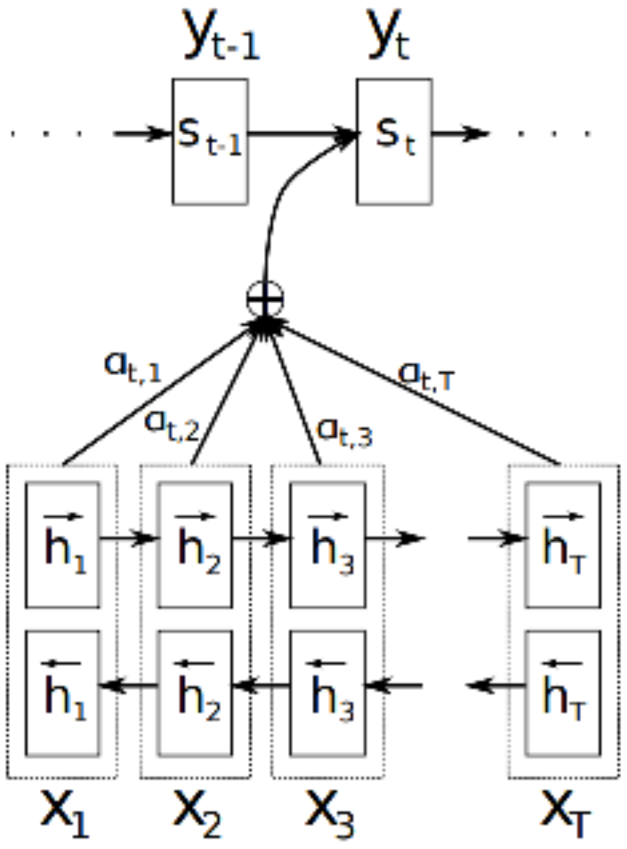
Model: Seq2seq with attention+key/value memory

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE PROGRAMMER

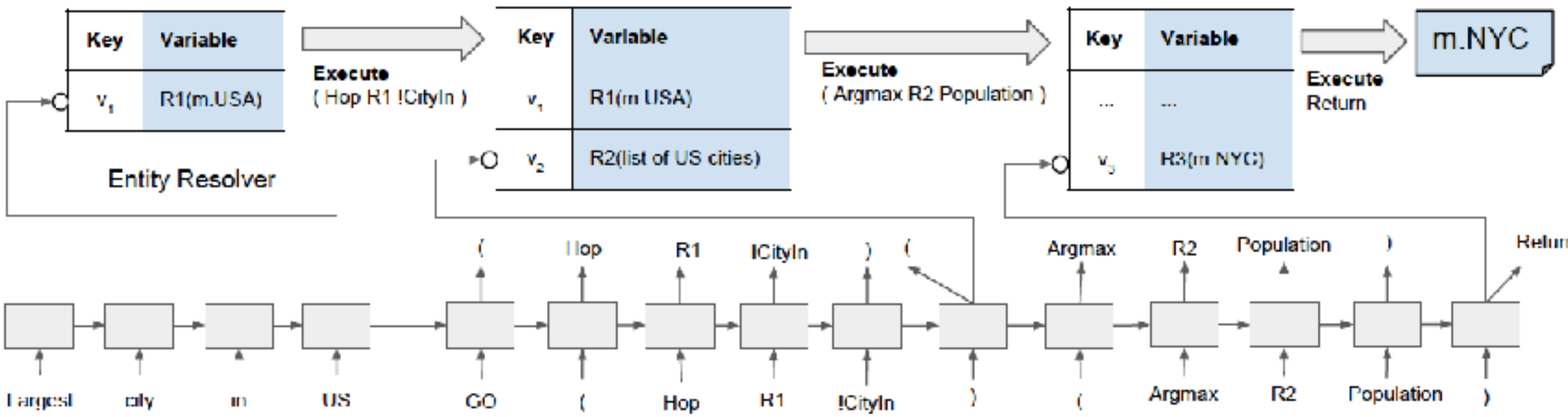
The "programmer" needs to map natural language into a program, which is a sequence of tokens that reference operations and values in the "computer".

Seq2seq with attention



Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE PROGRAMMER



Q: When do we insert something in the memory?

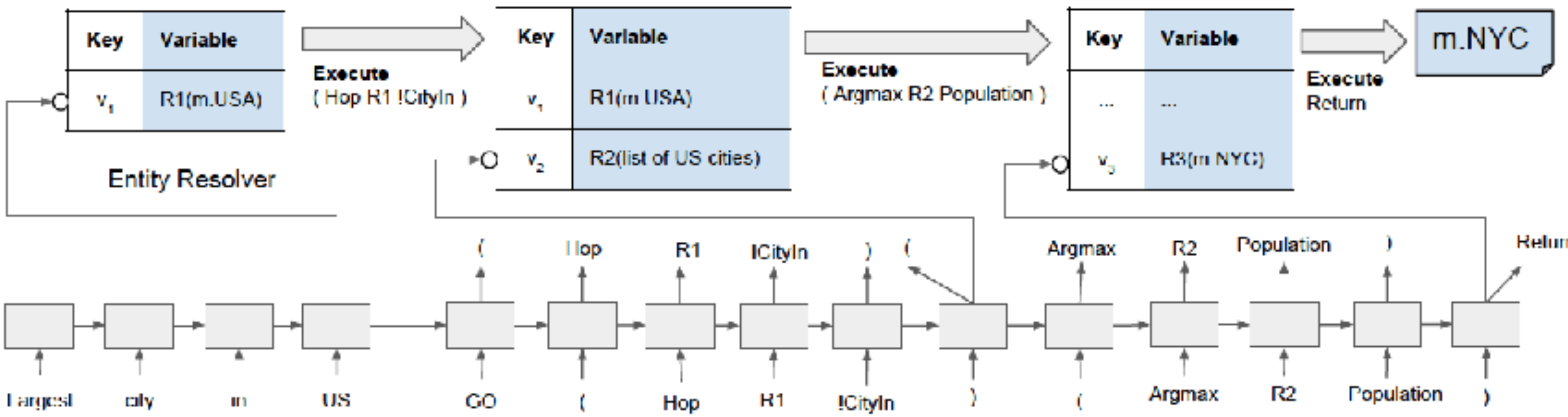
In two cases:

1. When we link text spans to KB entities, e.g. ``US”. Key: avg of hidden state h of the encoder along text span, value: the linked entity
2. During decoding when a full expression is generated (we produce ``)”. Key: the hidden state of the decoder, value: the entity list obtained after executing the expression. Also, the variable name is added into the target vocabulary! In this way the generated program can reference intermediate results.

Final answer returned: the value of the last computed variable

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

THE PROGRAMMER

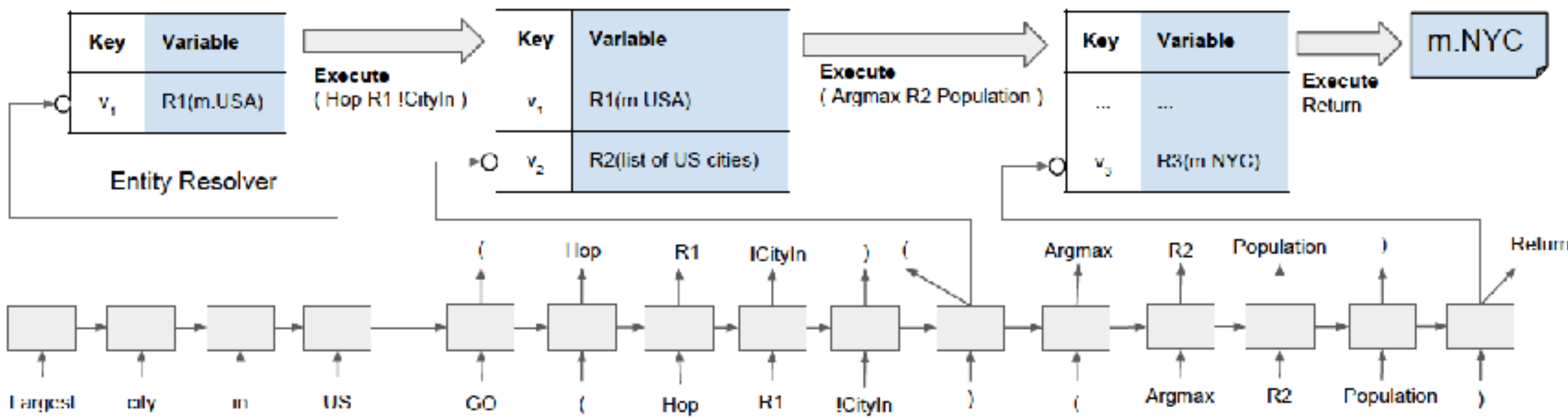


While the key embeddings are differentiable, the values referenced by the variables (lists of entities), stored in the “computer”, are symbolic and non-differentiable.

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

TRAINING

NSM executes non-differentiable operations against a KB, and thus end-to-end backpropagation is not possible. Thus we will train with Reinforcement, trial and error



Reward: sparse in time, whether the final generated answer matches the ground truth

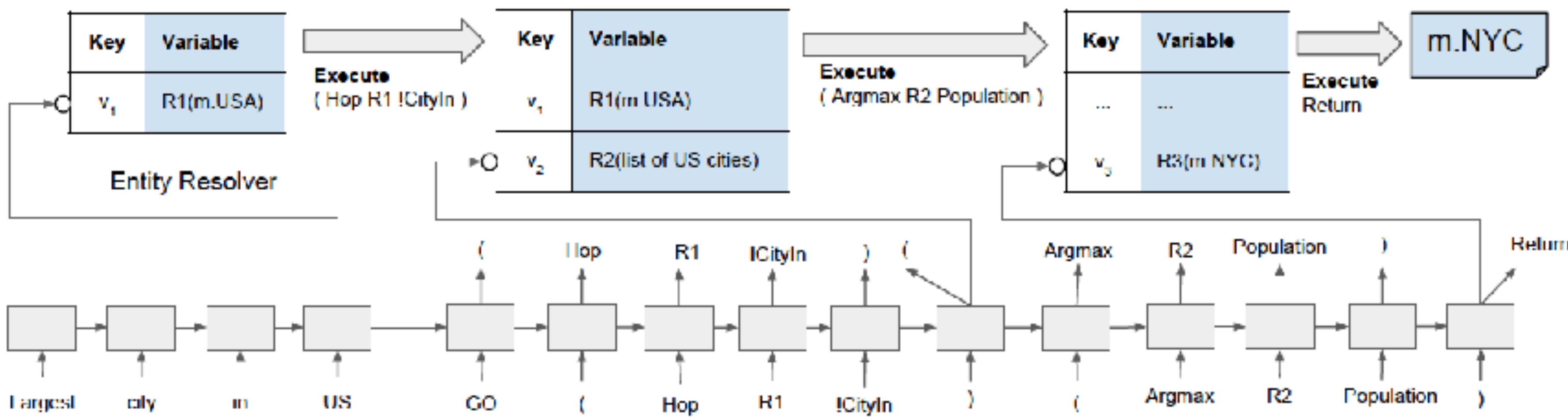
Actions: tokens to generate, remember the interpreter prunes those!

State: The input query and action sequence so far

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

TRAINING

NSM executes non-differentiable operations against a KB, and thus end-to-end backpropagation is not possible. Thus we will train with Reinforcement, trial and error



Instead of sampling token sequences i do:

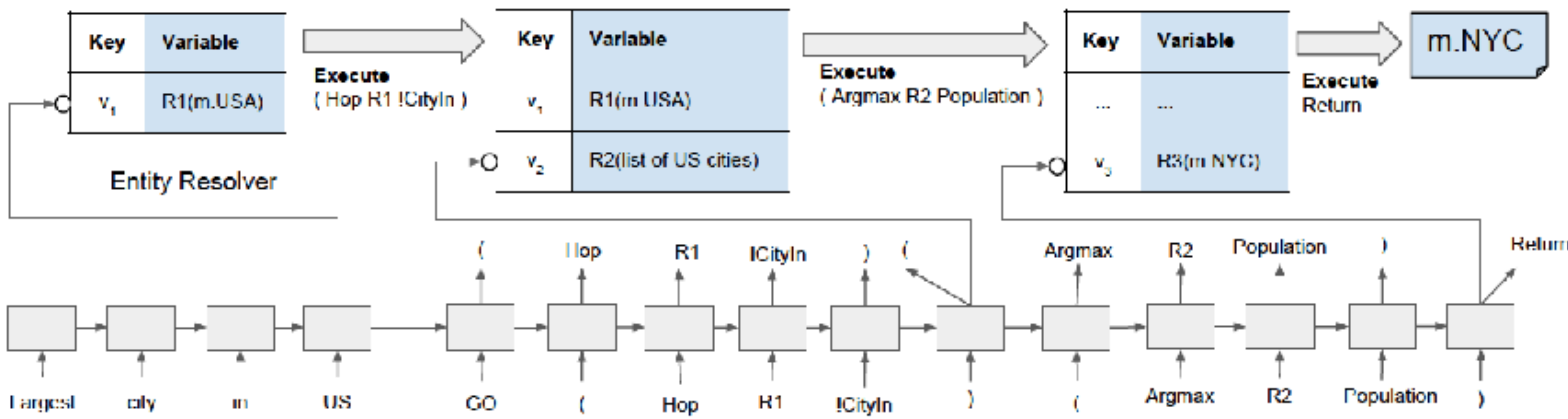
1. Decode with large beam size: generate a large number of valid syntactically programs, which i execute and get rewards
2. Maximum likelihood: I optimize the probability of the highest reward/shortest length program. Regularization!!

If no rewarding program is found for a question, i do not use this question (self pacing)

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

TRAINING

NSM executes non-differentiable operations against a KB, and thus end-to-end backpropagation is not possible. Thus we will train with Reinforcement, trial and error



Instead of sampling token sequences i do:

1. Decode with large beam size: generate a large number of valid syntactically programs, which i execute and get rewards
2. Maximum likelihood: I optimize the probability of the highest reward/shortest length program. Regularization!!
3. I combine ML and REINFORCE by adding the best found programs into its sampling space with a certain probability

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

Augmented Reinforce

Algorithm 1 IML-REINFORCE

Input: question-answer pairs $\mathbb{D} = \{(x_i, y_i)\}$, mix ratio α, reward function $R(\cdot)$, training iterations N_{ML}, N_{RL}, and beam sizes B_{ML}, B_{RL}.
Procedure:
Initialize $C_x^* = \emptyset$ the best program so far for x
Initialize model θ randomly ▷ Iterative ML
for $n = 1$ to N_{ML} **do**
  for (x, y) in D **do**
    $\mathbb{C} \leftarrow$ Decode B_{ML} programs given x
    for j in $1...|\mathbb{C}|$ **do**
      if $R_{x,y}(C_j) > R_{x,y}(C_x^*)$ **then** $C_x^* \leftarrow C_j$
  $\theta \leftarrow$ ML training with $\mathbb{D}_{ML} = \{(x, C_x^*)\}$

This does not directly optimize the accuracy measure we care about (percentage of questions answered correctly)

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

Augmented Reinforce

Algorithm 1 IML-REINFORCE

Input: question-answer pairs $\mathbb{D} = \{(x_i, y_i)\}$, mix ratio α, reward function $R(\cdot)$, training iterations N_{ML}, N_{RL}, and beam sizes B_{ML}, B_{RL}.

Procedure:

Initialize $C^*_x = \emptyset$ the best program so far for x

Initialize model θ randomly ▷ Iterative ML

for $n = 1$ to N_{ML} **do**

    for (x, y) in D **do**

        $\mathbb{C} \leftarrow$ Decode B_{ML} programs given x

        for j in $1 \ldots |\mathbb{C}|$ **do**

            if $R_{x,y}(C_j) > R_{x,y}(C^*_x)$ **then** $C^*_x \leftarrow C_j$

    $\theta \leftarrow$ ML training with $\mathbb{D}_{ML} = \{(x, C^*_x)\}$

Initialize model θ randomly ▷ REINFORCE

for $n = 1$ to N_{RL} **do**

    $\mathbb{D}_{RL} \leftarrow \emptyset$ is the RL training set

    for (x, y) in D **do**

        $\mathbb{C} \leftarrow$ Decode B_{RL} programs from x

        for j in $1 \ldots |\mathbb{C}|$ **do**

            if $R_{x,y}(C_j) > R_{x,y}(C^*_x)$ **then** $C^*_x \leftarrow C_j$

use the good programs found during ML $\mathbb{C} \leftarrow \mathbb{C} \cup \{C^*_x\}$

        for j in $1 \ldots |\mathbb{C}|$ **do**

            $\hat{p}_j \leftarrow (1-\alpha) \cdot \frac{p_j}{\sum_{j'} p_{j'}}$ where $p_j = P_\theta(C_j \mid x)$

            if $C_j = C^*_x$ **then** $\hat{p}_j \leftarrow \hat{p}_j + \alpha$

            $\mathbb{D}_{RL} \leftarrow \mathbb{D}_{RL} \cup \{(x, C_j, \hat{p}_j)\}$

    $\theta \leftarrow$ REINFORCE training with $\mathbb{D}_{RL}$

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

Augmented Reinforce

$$J^{RL}(\theta) = \sum_x \mathbb{E}_{P_\theta(a_{0:T}|x)}[R(x, a_{0:T})],$$

$$\nabla_\theta J^{RL}(\theta) = \sum_x \sum_{a_{0:T}} P_\theta(a_{0:T} \mid x) \cdot [R(x, a_{0:T}) - B(x)] \cdot \nabla_\theta \log P_\theta(a_{0:T} \mid x),$$

$$B(x) = \sum_{a_{0:T}} P_\theta(a_{0:T} \mid x) R(x, a_{0:T})$$

Algorithm 1 IML-REINFORCE

Input: question-answer pairs $\mathbb{D} = \{(x_i, y_i)\}$, mix ratio α, reward function $R(\cdot)$, training iterations N_{ML}, N_{RL}, and beam sizes B_{ML}, B_{RL}.

Procedure:

Initialize $C_x^* = \emptyset$ the best program so far for x

Initialize model θ randomly ▷ Iterative ML

for $n = 1$ to N_{ML} **do**

  for (x, y) in D **do**

    $\mathbb{C} \leftarrow$ Decode B_{ML} programs given x

    for j in $1...|\mathbb{C}|$ **do**

      if $R_{x,y}(C_j) > R_{x,y}(C_x^*)$ **then** $C_x^* \leftarrow C_j$

  $\theta \leftarrow$ ML training with $\mathbb{D}_{ML} = \{(x, C_x^*)\}$

Initialize model θ randomly ▷ REINFORCE

for $n = 1$ to N_{RL} **do**

  $\mathbb{D}_{RL} \leftarrow \emptyset$ is the RL training set

  for (x, y) in D **do**

    $\mathbb{C} \leftarrow$ Decode B_{RL} programs from x

    for j in $1...|\mathbb{C}|$ **do**

      if $R_{x,y}(C_j) > R_{x,y}(C_x^*)$ **then** $C_x^* \leftarrow C_j$

use the good programs found during ML

    $\mathbb{C} \leftarrow \mathbb{C} \cup \{C_x^*\}$

    for j in $1...|\mathbb{C}|$ **do**

      $\hat{p}_j \leftarrow (1-\alpha) \cdot \frac{p_j}{\sum_{j'} p_{j'}}$ where $p_j = P_\theta(C_j \mid x)$

      if $C_j = C_x^*$ **then** $\hat{p}_j \leftarrow \hat{p}_j + \alpha$

      $\mathbb{D}_{RL} \leftarrow \mathbb{D}_{RL} \cup \{(x, C_j, \hat{p}_j)\}$

  $\theta \leftarrow$ REINFORCE training with $\mathbb{D}_{RL}$

Neural Symbolic Machines: Learning Semantic Parsers on Freebase withWeak Supervision

IMPLEMENTATION

Decoders need to constantly query the KB. They used 50 decoders for one encoder

Curriculum learning: Instead of using all training data at once, they start by the short programs, having fewer functions (only hop in the beginning) and predicting only two full expressions