Neural Symbolic Language Understanding

Ni Lao
11.12.2017

Everything presented here is publicly available.
The opinions stated here are my own, not those of Google.
Plan

● Language & control
  ○ *Neural Symbolic Machines*: Semantic Parsing on Freebase with Weak Supervision
    ■ Chen Liang, Jonathan Berant, Quoc Le, Kenneth Forbus, Ni Lao

● Knowledge & scalability
  ○ Learning to Organize Knowledge with An *N-Gram Machine*
    ■ Fan Yang, Jiazhong Nie, William Cohen, Ni Lao
Language & Reasoning

- Language was primarily invented for reasoning
- Communication comes later
1) **Natural languages** are programming languages to **control** human behavior

2) For machines and human to understand each other, they just need **translation** models trained with **control theory**
Semantic Parsing

- Question answering with structured data (KG / tables / personal data)
- Voice to action
- Personal assistant
- Entertainment

[Berant+ 2013]
[Liang 2013]
Question Answering with Knowledge Base

Largest city in US?

GO
(Hop V1 CityIn)
(Argmax V2 Population)
RETURN

NYC

0. Paraphrase
Many ways to ask the same question, e.g.,
“What was the date that Minnesota became a state?”
“When was the state Minnesota created?”

1. Compositionality

2. Large Search Space
E.g., Freebase:
23K predicates,
82M entities,
417M triplets

3. Optimization
Reinforcement Learning is known
to be hard given sparse reward
WebQuestionsSP Dataset

- 5,810 questions Google Suggest API & Amazon MTurk
- Remove invalid QA pairs
- 3,098 training examples, 1,639 testing examples remaining
- Open-domain, and contains grammatical error
- Multiple entities as answer => macro-averaged F1

• What do Michelle Obama do for a living? writer, lawyer
• What character did Natalie Portman play in Star Wars? Padme Amidala
• What currency do you use in Costa Rica? Costa Rican colon
• What did Obama study in school? political science
• What killed Sammy Davis Jr? throat cancer

[Berant et al, 2013; Yih et al, 2016]
(Scalable) Neural Program Induction

- Impressive works to show NN can learn addition and sorting, but...

- The learned operations are not as scalable and precise.

- Why not use existing modules that are scalable, precise and interpretable?

[Reed & Freitas 2015]

[Zaremba & Sutskever 2016]
Neural Symbolic Machines

Weak supervision
Manager

Neural
Programmer

Symbolic
Computer

Knowledge Base
Predefined Functions

Here's another shovel full of assignments.

Abstract
Scalable
Precise
Non-differentiable
Contributions

- Simple Seq2Seq model is not enough

1. Compositionality
2. Large Search Space
3. Optimization

1. Key-Variable Memory
2. Beam search + Code Assistance
3. Augmented REINFORCE

E.g., Freebase:
- 23K predicates,
- 82M entities,
- 417M triplets

Reinforcement Learning is known to be hard given sparse reward
Key-Variable Memory for Compositionality

- A linearised bottom-up derivation of the recursive program
**Key-Variable Memory: Save Intermediate Value**

<table>
<thead>
<tr>
<th>Key (Embedding)</th>
<th>Variable (Symbol)</th>
<th>Value (Data in Computer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_0 )</td>
<td>R0</td>
<td>m.USA</td>
</tr>
<tr>
<td>( V_1 )</td>
<td>R1</td>
<td>([m.SF, m.NYC, ...])</td>
</tr>
</tbody>
</table>

Expression is finished.

Result

Execution

Computer
Key-Variable Memory: Reuse Intermediate Value

<table>
<thead>
<tr>
<th>Key (Embedding)</th>
<th>Variable (Symbol)</th>
<th>Value (Data in Computer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V₀</td>
<td>R₀</td>
<td>m.USA</td>
</tr>
<tr>
<td>V₁</td>
<td>R₁</td>
<td>[m.SF, m.NYC, ...]</td>
</tr>
</tbody>
</table>

**Diagram:**

- **Softmax**
- **Neural**
- **Symbolic**

**Diagram Elements:**

- **CityIn**
- **Argmax**
- **Computer**
Code Assistance: Prune Search Space

Pen and paper

IDE
Code Assistance: Syntactic Constraint

Decoder Vocab

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_0$</td>
<td>R0</td>
</tr>
<tr>
<td>$V_1$</td>
<td>R1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$E_0$</td>
<td>Hop</td>
</tr>
<tr>
<td>$E_1$</td>
<td>Argmax</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$P_0$</td>
<td>CityIn</td>
</tr>
<tr>
<td>$P_1$</td>
<td>BornIn</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Variables: <10
- Functions: <10
- Predicates: 23K
Last token is ‘(’, so has to output a function name next.

Decoder Vocab

<table>
<thead>
<tr>
<th>V0</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>R1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E0</th>
<th>Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>Argmax</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P0</th>
<th>CityIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>BornIn</td>
</tr>
</tbody>
</table>

- Variables: <10
- Functions: <10
- Predicates: 23K
Code Assistance: Semantic Constraint

Decoder Vocab

<table>
<thead>
<tr>
<th>V_0</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_1</td>
<td>R1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>E_0</td>
<td>Hop</td>
</tr>
<tr>
<td>E_1</td>
<td>Argmax</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>P_0</td>
<td>CityIn</td>
</tr>
<tr>
<td>P_1</td>
<td>BornIn</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Variables: <10
Functions: <10
Predicates: 23K
Given definition of \( \text{Hop} \), need to output a predicate that is connected to \( R_2 \) (\( m.\text{USA} \)).
Policy Gradient (REINFORCE)

- **MLE** optimizes log likelihood of approximate gold programs
  \[ J^{ML}(\theta) = \sum_q \log P(a_{0:T}^{best}(q)|q, \theta) \]

- **RL** optimizes the expected reward under a stochastic policy \( P \)
  \[ J^{RL}(\theta) = \sum_q \mathbb{E}_{P(a_{0:T}|q, \theta)}[R(q, a_{0:T})] \]

- The gradient is almost the same as that for MLE except for a weight \( P(R-B) \)
  \[ \nabla_\theta J^{RL}(\theta) = \sum_q \sum_{a_{0:T}} P(a_{0:T}|q, \theta)[R(q, a_{0:T}) - B(q)] \nabla_\theta \log P(a_{0:T}|q, \theta) \]

- The **baseline** does not change the solution but improves convergences, e.g.,
  \[ B(q) = \sum_{a_{0:T}} P(a_{0:T}|q, \theta)R(q, a_{0:T}) \]

[Williams 1992]
REINFORCE Training

1. High variance
   Requires a lot of (expensive) samples

2. Cold start problem
   Without supervised pretraining, the gradients at the beginning

\[ \nabla_{\theta} J^{RL}(\theta) = \sum_{q} \sum_{a_{0:T}} P(a_{0:T}|q, \theta) [R(q, a_{0:T}) - B(q)] \nabla_{\theta} \log P(a_{0:T}|q, \theta) \]
Iterative Maximum Likelihood Training (Hard EM)

1. Spurious program
   Mistake PlaceOfBirth for PlaceOfDeath.

2. Lack of negative examples
   Mistake SiblingsOf for ParentsOf.

\[ J_{ML}(\theta) = \sum_q \log P(a_{0:T}^{\text{best}}(q)|q, \theta) \]
Augmented REINFORCE

1. Reduce variance at the cost of bias
2. Mix in approximate gold programs to bootstrap and stabilize training

Beam search

Top k in beam

(1 - \alpha)

\alpha

Approximate Gold Programs

Updated Model

Policy gradient update
Distributed Architecture

- 200 actors, 100 KG servers, 1 learner
New State-of-the-Art on WebQuestionsSP

- First end-to-end neural network to achieve SOTA on semantic parsing with weak supervision over large knowledge base
- The performance is approaching SOTA with full supervision

<table>
<thead>
<tr>
<th>Model</th>
<th>Avg. Prec. @1</th>
<th>Avg. Rec. @1</th>
<th>Avg. F1@1</th>
<th>Acc. @1</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAGG</td>
<td>67.3</td>
<td>73.1</td>
<td>66.8</td>
<td>58.8</td>
</tr>
<tr>
<td>NSM – our model</td>
<td>70.8</td>
<td>76.0</td>
<td>69.0</td>
<td>59.5</td>
</tr>
<tr>
<td>STAGG (full supervision)</td>
<td>70.9</td>
<td>80.3</td>
<td>71.7</td>
<td>63.9</td>
</tr>
</tbody>
</table>
Augmented REINFORCE

- REINFORCE get stuck at local maxima
- Iterative ML training is not directly optimizing the F1 score
- Augmented REINFORCE obtains the best performances

<table>
<thead>
<tr>
<th>Settings</th>
<th>Train Avg. F1@1</th>
<th>Valid Avg. F1@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterative ML only</td>
<td>68.6</td>
<td>60.1</td>
</tr>
<tr>
<td>REINFORCE only</td>
<td>55.1</td>
<td>47.8</td>
</tr>
<tr>
<td>Augmented REINFORCE</td>
<td>83.0</td>
<td>67.2</td>
</tr>
</tbody>
</table>
Plan

● Language & control
  ○ *Neural Symbolic Machines*: Semantic Parsing on Freebase with Weak Supervision
    ■ Chen Liang, Jonathan Berant, Quoc Le, Kenneth Forbus, Ni Lao

● Knowledge & scalability
  ○ Learning to Organize Knowledge with An *N-Gram Machine*
    ■ Fan Yang, Jiazhong Nie, William Cohen, Ni Lao
Where does knowledge come from?

- The human brain contains roughly 100 billion neurons each capable of making around 1,000 connections.

- Where do we get these 100 TB parameters?

- How many lines of code do I need to write if I want to achieve AI?
The Mind’s Eye

a small machine which can copy large amount of complexity from the world to the brain

the world

a suitable representation
Knowledge and Scalability

- How information should be organized for scalability?

"AS WE MAY THINK"
(1945)

Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and to coin one at random, memex will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.
Scalability of modern search engines

- Can respond to user's requests within a fraction of a second
- But are weak at text understanding and complex reasoning
Scalability of mammal memory

- **Very rapid adaptation** (in just one or a few trials) is necessary for survival
  - E.g., associating taste of food and sickness

- **Need fast responses** based on large amount of knowledge
  - Needs good representation of knowledge

- **However, good representation** can only be learnt gradually
  - During sleeps
  - To prevent interference with established associations

[Garcia+ 1966]
[Wickman 2012]
[Bartol+ 2015]
Complementary Learning Theory

Connections within and among neocortical areas (green) support gradual acquisition of structured knowledge through interleaved learning.

Bidirectional connections (blue) link neocortical representations to the hippocampus/MTL for storage, retrieval, and replay.

Encoder

Rapid learning in connections within hippocampus (red) supports initial learning of arbitrary new information.

Episodic memory

Record & Replay

Observations

Primary sensory and motor cortices
(Scalable) Neural Nets

- Impressive works to show NN can learn addition and sorting, but...

- The learned operations are not as scalable and precise.

- Why not use existing modules that are scalable, precise and interpretable?
Question answering as a simple test bed

- A good semantic representation should support reasoning & scalability

![Diagram showing the process of question answering with text, knowledge store, question, program, answer, expected answer, reward, execute (no learning), generate (learning).]
Facebook bAbI Tasks

- Simulated question answering tasks to test the ability to "understand"
- We introduce a special version ("life-long bAbI"), which has stories of up to 10 million sentences

<table>
<thead>
<tr>
<th>Sam walks into the kitchen.</th>
<th>Brian is a lion.</th>
<th>Mary journeyed to the den.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam picks up an apple.</td>
<td>Julius is a lion.</td>
<td>Mary went back to the kitchen.</td>
</tr>
<tr>
<td>Sam walks into the bedroom.</td>
<td>Julius is white.</td>
<td>John journeyed to the bedroom.</td>
</tr>
<tr>
<td>Sam drops the apple.</td>
<td>Bernhard is green.</td>
<td>Mary discarded the milk.</td>
</tr>
<tr>
<td>Q: Where is the apple?</td>
<td>Q: What color is Brian?</td>
<td>Q: Where was the milk before the den?</td>
</tr>
<tr>
<td>A. Bedroom</td>
<td>A. White</td>
<td>A. Hallway</td>
</tr>
</tbody>
</table>

[Weston+ 2015]
The Knowledge Storage

- Each tuple in the KG is an n-gram, which is a list of symbols
- Each tuple is also associated with a timestamp to reason over time

Table 1: Example of probabilistic knowledge storage. Each sentence may be converted to a distribution over multiple tuples, but only the one with the highest probability is shown here.

<table>
<thead>
<tr>
<th>Sentences</th>
<th>Time stamp</th>
<th>Knowledge tuples</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary went to the kitchen.</td>
<td>1</td>
<td>mary to kitchen</td>
<td>0.9</td>
</tr>
<tr>
<td>Mary picked up the milk.</td>
<td>2</td>
<td>mary the milk</td>
<td>0.4</td>
</tr>
<tr>
<td>John went to the bedroom.</td>
<td>3</td>
<td>john to bedroom</td>
<td>0.7</td>
</tr>
<tr>
<td>Mary journeyed to the garden.</td>
<td>4</td>
<td>mary to garden</td>
<td>0.8</td>
</tr>
</tbody>
</table>
The Programs

- A **program** \( C \) is a list of **expressions** \( c_1 \ldots c_N \), where each \( c_i \) is either a special expression **Return** indicating the end of the program, or is of the form \( (F, A_1 \ldots A_L) \).

Table 2: Functions in N-Gram Machines. The knowledge storage on which the programs can execute is \( \Gamma \), and a knowledge tuple \( \Gamma_i \) is represented as \( (i, (\gamma_1, \ldots, \gamma_N)) \). “FR” means from right.

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hop</td>
<td>( v_1 \ldots v_L )</td>
<td>( { \gamma_{L+1} \mid \text{if } (\gamma_1 \ldots \gamma_L) == (v_1, \ldots, v_L), \forall \Gamma \in \Gamma } )</td>
</tr>
<tr>
<td>HopFR</td>
<td>( v_1 \ldots v_L )</td>
<td>( { \gamma_{N-L} \mid \text{if } (\gamma_{N-L+1} \ldots \gamma_N) == (v_L, \ldots, v_1), \forall \Gamma \in \Gamma } )</td>
</tr>
<tr>
<td>Argmax</td>
<td>( v_1 \ldots v_L )</td>
<td>( \text{argmax}<em>i { (\gamma</em>{L+1}, i) \mid \text{if } (\gamma_1 \ldots \gamma_L) == (v_1, \ldots, v_L), \forall \Gamma_i \in \Gamma } )</td>
</tr>
<tr>
<td>ArgmaxFR</td>
<td>( v_1 \ldots v_L )</td>
<td>( \text{argmax}<em>i { (\gamma</em>{N-L}, i) \mid \text{if } (\gamma_{N-L+1} \ldots \gamma_N) == (v_L, \ldots, v_1), \forall \Gamma_i \in \Gamma } )</td>
</tr>
</tbody>
</table>
Example KS & Program

<table>
<thead>
<tr>
<th>Story</th>
<th>Knowledge Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandra journeyed to the hallway.</td>
<td>Sandra journeyed hallway</td>
</tr>
<tr>
<td>John journeyed to the bathroom.</td>
<td>John journeyed bathroom</td>
</tr>
<tr>
<td>Sandra grabbed the football.</td>
<td>Sandra got football</td>
</tr>
<tr>
<td>Daniel travelled to the bedroom.</td>
<td>Daniel journeyed bedroom</td>
</tr>
<tr>
<td>John got the milk.</td>
<td>John got milk</td>
</tr>
<tr>
<td>John dropped the milk.</td>
<td>John got milk</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where is the milk?</td>
<td>ArgmaxFR milk got</td>
</tr>
<tr>
<td></td>
<td>Argmax V1 journeyed</td>
</tr>
</tbody>
</table>
 Seq2Seq components

- A **knowledge encoder** that converts sentences to knowledge tuples and defines a distribution $P(\Gamma_i|s_i, s_{i-1}; \theta_{\text{enc}})$
  - The probability of a KS $\Gamma = \{\Gamma_1 \ldots \Gamma_n\}$ is the product of its tuples' probabilities:
    $$P(\Gamma|s; \theta_{\text{enc}}) = \Pi_{\Gamma_i \in \Gamma} P(\Gamma_i|s_i, s_{i-1}; \theta_{\text{enc}})$$

- A **knowledge decoder** that converts tuples back to sentences and defines a distribution $P(s_i|\Gamma_i, s_{i-1}; \theta_{\text{dec}})$ -- it will enable unsupervised training

- A **programmer** that converts questions to programs and defines a distribution $P(C|q, \Gamma; \theta_{\text{prog}})$ -- $\Gamma$ for code assist
Inference

Given an example \((s, q, a)\) from our training set, we would like to maximize the expected reward

\[
O^{QA}(\theta_{\text{enc}}, \theta_{\text{prog}}) = \sum_{\Gamma} \sum_{C} P(\Gamma|s; \theta_{\text{enc}}) P(C|q, \Gamma; \theta_{\text{prog}}) R(\Gamma, C, a),
\]

For gradient estimations we apply beam search instead of MCMC, which has huge variances.

It leads to a **hard search problem**, which we solve by having

1) a stabilized **auto-encoding** objective to bias the encoder to more interesting hypotheses;

\[
O^{AE}(\theta_{\text{enc}}, \theta_{\text{dec}}) = \mathbb{E}_{p(z|x; \theta_{\text{enc}})}[\log p(x|z; \theta_{\text{dec}})] + \sum_{z \in Z^N(x)} \log p(x|z; \theta_{\text{dec}}),
\]

\(Z^N(x): \) all tuples of length \(N\) which only consist of words from \(x\)

2) a **structural tweak** procedure which retrospectively corrects the inconsistency among multiple hypotheses so that reward can be achieved

- While code assist uses the **knowledge storage** to inform the **programmer**,  
  structural tweak adjusts the **knowledge encoder** to cooperate with an uninformed **programmer**.
N-Gram Machines

- Variable
- Function
- Word

Knowledge Storage

- Knowledge Decoder
- Knowledge Encoder

Reconstruction

Reconstruction loss

Story

Task:
- Can you change [variable] to [function]?

Code assist:
- Can be followed by [variable] or [function]

Structure Tweak:
- Can you change [variable] to [function]?

Executor

Program

- Generate (learning)
- Execute (no learning)
- Messages

Question

Programmer

Answer

Expected Answer

Reward
Optimization

- For training stability and tweaking, we augment the training objective with **experience replays**

\[
\nabla_{\theta_{\text{dec}}} O'(\theta) = \sum_{s_i \in \mathcal{S}} \sum_{\Gamma_i} [\beta(\Gamma_i) + P(\Gamma_i | s_i, s_{i-1}; \theta_{\text{enc}})] \nabla_{\theta_{\text{dec}}} \log P(s_i | \Gamma, s_{i-1}; \theta_{\text{dec}}),
\]

\[
\nabla_{\theta_{\text{enc}}} O'(\theta) = \sum_{s_i \in \mathcal{S}} \sum_{\Gamma_i} [P(\Gamma_i | s_i, s_{i-1}; \theta_{\text{enc}}) \log P(s_i | \Gamma_i, s_{i-1}; \theta_{\text{dec}}) + \mathcal{R}(\mathcal{G}'(\Gamma_i)) + \mathcal{R}(\mathcal{G}(\Gamma_i))] \nabla_{\theta_{\text{enc}}} \log P(\Gamma_i | s_i, s_{i-1}; \theta_{\text{enc}}),
\]

where \( \mathcal{R}(\mathcal{G}) = \sum_{\Gamma \in \mathcal{G}} \sum_{C} P(\Gamma | s; \theta_{\text{enc}}) P(C | q, \Gamma; \theta_{\text{prog}}) R(\Gamma, C, a) \) is the total expected reward for a set of valid knowledge stores \( \mathcal{G}, \mathcal{G}(\Gamma_i) \) is the set of knowledge stores which contains the tuple \( \Gamma_i \), and \( \mathcal{G}'(\Gamma_i) \) is the set of knowledge stores which contains the tuple \( \Gamma_i \) through tweaking.

\[
\nabla_{\theta_{\text{prog}}} O'(\theta) = \sum_{\Gamma} \sum_{C} [\alpha I [C \in C^*(s, q)] + P(C | q, \Gamma; \theta_{\text{prog}})] \cdot P(\Gamma | s; \theta_{\text{enc}}) R(\Gamma, C, a) \nabla_{\theta_{\text{prog}}} \log P(C | q, \Gamma; \theta_{\text{prog}}),
\]

where \( C^*(s, q) \) is the experience replay buffer for \( (s, q) \). \( \alpha = 0.1 \) is a constant. During training, the program with the highest weighted reward (i.e. \( P(\Gamma | s; \theta_{\text{enc}}) R(\Gamma, C, a) \)) is added to the replay buffer.

- **optimize by coordinate ascent** – updating three components in alternation with **REINFORCE**
Learning to search

- Structure Space (KBs and programs)
- Model Bias
- Experience
- Replay Buffer

Decoding (beam search)

Training (REINFORCE)

Seq2Seq models

Structure tweaking

Introducing new functions
Results

- The bAbI dataset contains twenty tasks in total. We consider the subset of them that are extractive question answering tasks.

Table 3: Test accuracy on bAbI tasks with auto-encoding (AE) and structure tweak (ST)

<table>
<thead>
<tr>
<th></th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 11</th>
<th>Task 15</th>
<th>Task 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemN2N</td>
<td>1.000</td>
<td>0.830</td>
<td>0.840</td>
<td>1.000</td>
<td>0.440</td>
</tr>
<tr>
<td>QA</td>
<td>0.007</td>
<td>0.027</td>
<td>0.000</td>
<td>0.000</td>
<td>0.098</td>
</tr>
<tr>
<td>QA + AE</td>
<td>0.709</td>
<td>0.551</td>
<td>1.000</td>
<td>0.246</td>
<td>1.000</td>
</tr>
<tr>
<td>QA + AE + ST</td>
<td>1.000</td>
<td>0.853</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Results

- AE is a strong bias towards good representations
- ST helps to achieve consistency, e.g.,
  - "he" vs "john" (coreference)
  - "cat" vs "cats" (singular vs. plural)
  - "go to" vs "journey to" (synonyms)

<table>
<thead>
<tr>
<th>QA</th>
<th>QA + AE</th>
<th>QA + AE + ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>went went went</td>
<td>daniel went office</td>
<td>daniel went office</td>
</tr>
<tr>
<td>mary mary mary</td>
<td>mary back garden</td>
<td>mary went garden</td>
</tr>
<tr>
<td>john john john</td>
<td>john back garden</td>
<td>john went kitchen</td>
</tr>
<tr>
<td>mary mary mary</td>
<td>mary grabbed football</td>
<td>mary got football</td>
</tr>
<tr>
<td>there there there</td>
<td>sandra got apple</td>
<td>sandra got apple</td>
</tr>
<tr>
<td>cats cats cats</td>
<td>cats afraid wolves</td>
<td>cat afraid wolves</td>
</tr>
<tr>
<td>mice mice mice</td>
<td>mice afraid wolves</td>
<td>mouse afraid wolves</td>
</tr>
<tr>
<td>is is cat</td>
<td>gertrude is cat</td>
<td>gertrude is cat</td>
</tr>
</tbody>
</table>
Result

- Scalability
Thanks!
Generated Programs

- **Question:** “what college did russell wilson go to?”
- **Generated program:**

  (hop v1 /people/person/education)
  (hop v2 /education/education/institution)
  (filter v3 v0 /common/topic/notable_types )

  <EOP>

  In which

  v0 = “College/University” (m.01y2hn1)
  v1 = “Russell Wilson” (m.05c10yf)

- **Distribution of the length of generated programs**

<table>
<thead>
<tr>
<th>#Expressions</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Percentage</strong></td>
<td>0.4%</td>
<td>62.9%</td>
<td>29.8%</td>
<td>6.9%</td>
</tr>
<tr>
<td><strong>F1</strong></td>
<td>0.0</td>
<td>73.5</td>
<td>59.9</td>
<td>70.3</td>
</tr>
</tbody>
</table>
Mandelbrot Set

the nature of complex numbers

\[ z_0 = 0 \]
\[ z_{n+1} = z_n^2 + c \]
\[ c \in M \iff \lim_{n \to \infty} |z_{n+1}| \leq 2 \]
Structure Tweak

- No reward
  - Caused by semantic (i.e. run-time) errors
    - Program: Hop var *cats*
    - Knowledge tuple: *Cat* afraid wolves
  - Propose knowledge tuples that allow the program with high probability to obtain positive reward

- Low expected reward
  - Program with high reward in $P(\text{prog} | q, \text{kg}) \neq$ program with high probability in $P(\text{prog} | q)$
    - “Hop var *journeyed*” vs “Hop var *went*”
  - Propose knowledge tuples that allow program with high probability to obtain high rewards
## Example KS & Program

### Table 8: Task 15 Basic Deduction

<table>
<thead>
<tr>
<th>Story</th>
<th>Knowledge Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sheep are afraid of cats.</td>
<td>Sheep afraid cats</td>
</tr>
<tr>
<td>Cats are afraid of wolves.</td>
<td>Cat afraid wolves</td>
</tr>
<tr>
<td>Jessica is a sheep.</td>
<td>Jessica is sheep</td>
</tr>
<tr>
<td>Mice are afraid of sheep.</td>
<td>Mouse afraid sheep</td>
</tr>
<tr>
<td>Wolves are afraid of mice.</td>
<td>Wolf afraid mice</td>
</tr>
<tr>
<td>Emily is a sheep.</td>
<td>Emily is sheep</td>
</tr>
<tr>
<td>Winona is a wolf.</td>
<td>Winona is wolf</td>
</tr>
<tr>
<td>Gertrude is a mouse.</td>
<td>Gertrude is mouse</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is Emily afraid of?</td>
<td>Hop Emily is</td>
</tr>
<tr>
<td></td>
<td>Hop V1 afraid</td>
</tr>
</tbody>
</table>
Example KS & Program

<table>
<thead>
<tr>
<th>Story</th>
<th>Knowledge Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berhard is a rhino.</td>
<td>Bernhard a rhino</td>
</tr>
<tr>
<td>Lily is a swan.</td>
<td>Lily a swan</td>
</tr>
<tr>
<td>Julius is a swan.</td>
<td>Julius a swan</td>
</tr>
<tr>
<td>Lily is white.</td>
<td>Lily is white</td>
</tr>
<tr>
<td>Greg is a rhino.</td>
<td>Greg a rhino</td>
</tr>
<tr>
<td>Julius is white.</td>
<td>Julius is white</td>
</tr>
<tr>
<td>Brian is a lion.</td>
<td>Brian a lion</td>
</tr>
<tr>
<td>Bernhard is gray.</td>
<td>Bernhard is gray</td>
</tr>
<tr>
<td>Brian is yellow.</td>
<td>Brian is yellow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>What color is Greg?</td>
<td>Hop Greg a</td>
</tr>
<tr>
<td></td>
<td>HopFR V1 a</td>
</tr>
<tr>
<td></td>
<td>Hop V2 is</td>
</tr>
</tbody>
</table>