DEEPCODER:
LEARNING TO WRITE PROGRAMS

Balog et. al. ICLR 2017

Presented By: Azad
CS 294 - 159
March 13, 2019
Outline

- Brief Overview: Program Synthesis
  - Components and Challenges
- Which does Deep Coder actually solve??
- Proposed Approach: Learning Inductive Program Synthesis (LIPS)
- Key Results
- Main Contribution
- Limitations
- Discussion
Brief Overview: Program Synthesis

Counter Example Guided Inductive Synthesis

Synthesis

- Typically search techniques are employed for the synthesis
  - Enumerative Search
    - Enumerate programs: typically from smaller to larger
    - Which order do we enumerate the programs? DeepCoder !!!
  - Stochastic Search
    - Search landscape not smooth

```python
t ← [int]
p ← [int]
c ← MAP (-1) t
d ← MAP (-1) p
e ← ZIPWITH (+) c d
f ← MINIMUM e
```
Proposed Approach: Learning Inductive Program Synthesis (LIPS)

- DSL and Attributes
- Data Generation
- Machine Learning Model
- Search
Proposed Approach: Learning Inductive Program Synthesis (LIPS)

- DSL and Attributes
- Data Generation
- Machine Learning Model
- Search

- Expressive Enough so that it actually can solve the problem
- Restrictive Enough to limit search space
Proposed Approach: Learning Inductive Program Synthesis (LIPS)

- DSL and Attributes
- Data Generation
- Machine Learning Model
- Search

- Attribute vector
  - Maps the program to a vector
  - Works as a link between the ML component and the search component
Proposed Approach: Learning Inductive Program Synthesis (LIPS)

- DSL and Attributes
- Data Generation
- Machine Learning Model
- Search

- Should be feasible to generate a large dataset
- Set of Programs in the DSL
- Attribute Vectors
- Set of I/O examples
Proposed Approach: Learning Inductive Program Synthesis (LIPS)

- DSL and Attributes
- Data Generation
- Machine Learning Model
- Search

- Learns a distribution over attribute vectors
- \( P(\text{Attribute Vectors} \mid \text{Set of I/O examples}) \)
Proposed Approach: Learning Inductive Program Synthesis (LIPS)

- DSL and Attributes
- Data Generation
- Machine Learning Model
- Search

- \( P(\text{Attribute Vectors} | \text{Set of examples}) \)
- Guided by the distribution learned by the model
Deep Coder: DSL

\[
\begin{align*}
t &\leftarrow [\text{int}] \\
p &\leftarrow [\text{int}] \\
c &\leftarrow \text{MAP} \ (-1) \ t \\
d &\leftarrow \text{MAP} \ (-1) \ p \\
e &\leftarrow \text{ZIPWITH} \ (+) \ c \ d \\
f &\leftarrow \text{MINIMUM} \ e \\
x &\leftarrow [\text{int}] \\
y &\leftarrow [\text{int}] \\
c &\leftarrow \text{SORT} \ x \\
d &\leftarrow \text{SORT} \ y \\
e &\leftarrow \text{REVERSE} \ d \\
f &\leftarrow \text{ZIPWITH} \ (*) \ d \ e \\
g &\leftarrow \text{SUM} \ f
\end{align*}
\]

First-order functions: HEAD, LAST, TAKE, DROP, ACCESS, MINIMUM, MAXIMUM, REVERSE, SORT, SUM
Higher-order Functions: MAP, FILTER, COUNT, ZIPWITH, SCANL1
Lambdas: (+1), (-1), (*2), (/2), (*(-1)), (**2), (*3), (/3), (*4), (/4) - map, (+), (-), (*), MIN, MAX - ZIPWITH, SCANL1
Predicates (>0), (<0), (%2==0), (%2==1)
Deep Coder: Attribute Vector

One-Hot Vector of the functions and lambdas

First-order functions: HEAD, LAST, TAKE, DROP, ACCESS, MINIMUM, MAXIMUM, REVERSE, SORT, SUM
Higher-order Functions: MAP, FILTER, COUNT, ZIPWITH, SCANL1
Lambdas: (+1), (-1), (*2), (/2), (*(-1)), (**2), (*3), (/3), (/4) - map, (+), (-), (*), MIN, MAX - ZIPWITH, SCANL1
Predicates (>0), (<0), (%2==0), (%2==1)
Deep Coder: Data Generation

- Synthetic Program Generation
  - Pruning

- Run programs to generate inputs from outputs

| a ← [int]       | An input-output example:     |
| b ← FILTER (<0) a | Input:                      |
| c ← MAP (*4) b   | [-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11] |
| d ← SORT c       | Output:                     |
| e ← REVERSE d    | [-12, -20, -32, -36, -68]   |
Deep Coder: ML Model

Attribute Predictions

Final Activations

Pooled

Hiddens 3

Hiddens 2

Hiddens 1

State Embeddings

Program State

Inputs 1

Outputs 1

... Inputs 5

Outputs 5

14 sigmoids

256 sigmoids
Deep Coder: Search Component

- Depth-first search ~ $3 \times 10^6$ programs per second with caching
- “Sort and add” enumeration
- Sketch
- $\lambda^2$
## Key Results

<table>
<thead>
<tr>
<th>Timeout needed to solve</th>
<th>DFS</th>
<th>Enumeration</th>
<th>$\chi^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>DeepCoder</td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td>163s</td>
<td>24s</td>
<td>463s</td>
</tr>
<tr>
<td>40%</td>
<td>2887s</td>
<td>514s</td>
<td>48s</td>
</tr>
<tr>
<td>60%</td>
<td>6832s</td>
<td>2654s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8181s</td>
<td>9s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$&gt;10^4s$</td>
<td>264s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$&gt;10^4s$</td>
<td>4640s</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>48s</td>
</tr>
<tr>
<td>Speedup</td>
<td>6.8×</td>
<td>5.6×</td>
<td>907×</td>
</tr>
<tr>
<td></td>
<td>2.6×</td>
<td></td>
<td>$&gt;37\times$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$&gt;2\times$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.6×</td>
</tr>
</tbody>
</table>

- Baseline: Simple Prior as Function Probabilities
- Training Programs: length 1 to 4
- 100: Test Programs length 5 (Search Space in the order of $10^{10}$)
Key Results

- RNN Encoder and Decoder: Beam search was used to explore likely programs predicted by the RNN

- Solution comparable with the other techniques when searching for programs of lengths $T \leq 2$
  - where the search space size is very small (on the order of $10^3$).
Other Results

Programming:
- **DFS**: using neural network
- **DFS**: using prior order
- **L2**: Sort and add using neural network
- **L2**: Sort and add in prior order
- **Enumeration**: Sort and add using neural network
- **Enumeration**: Sort and add in prior order
- **Beam search**
- **Sketch**: Sort and add using neural network
- **Sketch**: Sort and add in prior order
Main Contribution & Impact

Neural-Guided Search

- Using weak supervision to guide-search in the program space
- Can be used as a component of the existing framework
Possible Improvements/Limitations

- Very restricted problem domain

- Learns distribution over input data
  - Can not utilizes/condition on partial/intermediate programs as it generates

- Does not learn anything about function order/dependency

- No notion of how good the found program is/ no sense of program ranking
  - The problem is under-specified
  - There may be more than one program that may conform the I/O
    - The returned program may not be the one the user wants

- Learning Longer Programs with Loops/ Conditionals
Discussion

- End-to-end Deep Program Synthesis vs. Neural-Guided Search
- Deep Learning + Stochastic Search ??
- Using Natural Language as specification and encode ?
- Explainability & Program Synthesis
- Detecting wrong predictions and back-track ?
Some Useful Papers/Blog Posts

● Program Synthesis in 2017-18 by Alex Polozov: https://alexpolozov.com/blog/program-synthesis-2018/


● Open Review: https://openreview.net/forum?id=ByldLrqlx

● Neuro-Symbolic Program Synthesis. Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, Pushmeet Kohli
  ○ End-to-end Synthesis, RNN based