Neural Inference of API Functions from Input–Output Examples

Rohan Bavishi, Caroline Lemieux, Neel Kant, Roy Fox, Koushik Sen, Ion Stoica
Introduction

- Discovering what APIs to use can be time difficult and time-consuming
- Speed of creation of new APIs outpaces the completeness, clarity, and even correctness of the documentation
- **Program synthesis** is the process of automatically generating a program conforming to a higher-level specification
- Goal is the automating the process of finding the correct API given a set of input-output values
Challenges

- For a language with **n functions**, taking an average of **m argument** values, the **number of sequential programs** of length **k** grows as \((nm)^k\)
- Existing approaches work on small subsets of problems or **Domain Specific Languages**
- Identify the actual function and its arguments, which may have interactions
- Exhaustive search is feasible for determining arguments but not functions
- Use a hybrid approach with exhaustive search for arguments and a neural inference mechanism to predict the functions
Methodology

Map a given I/O example to a pandas function which performs the transformation specified by the example

Steps:

1. Preprocessing I/O examples into a graph
2. Feeding these examples into a trainable neural network which learns a high-dimensional representation for each node of the graph,
3. Pooling to output of the neural network and applying softmax to select a pandas function.
4. Use exhaustive search to find the correct arguments
Graph Abstraction

The operation used in an I/O example is often captured by the relationships amongst the elements, rather than the concrete data itself.
Nodes

- Every data cell in the input and output DataFrame is represented as a single node
- Multiple levels of column names or row indices appear as additional nodes
- Node is labeled with a type tuple (data type, is input)

Edges

- Edges to represent the relationships between nodes in input and output
- **Equality edges** are between any nodes with the same value
- **Adjacency edges** represent the basic structural characteristics of the DataFrames
- **Indexing edges** are between a column name (resp. row index) and all the data nodes that belong to that column
Gated Graph Neural Networks

Graph Neural Networks map graphs to outputs via two steps:

1. **Propagation** step that computes node representations for each node
2. Compute **output model** that maps from node representations and corresponding labels to an output

**Gated Graph Neural Networks**: GNN with recurrent unit that stores node state and uses backpropagation through time in order to compute gradient
Network

- **Edge** $e$ is a 3-tuple $(v_s, v_t, t_e)$ where $v_s$ and $v_t$ are the source and target nodes and $t_e$ is the type of the edge.
- Every node $v$ has a corresponding **state vector**
- Information is propagated using **message passing** across $k$ rounds
- For each node, the **incoming messages** are **aggregated**
- The new node state vector for the next round is computed using **recurrent unit**
- **Element-wise sum-pool** the node state vectors into a graph state vector $h$.
- Use a multi-layer perceptron with one hidden layer, and apply softmax to produce a probability distribution over the target classes
Accuracy Results

Accuracy is computed using (1) synthesized validation set and (2) I/O examples taken from real-world sources.

Table 1: Accuracy in predicting the ground-truth or a correct function for I/O examples.

<table>
<thead>
<tr>
<th>Ground-Truth</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top-1</td>
</tr>
<tr>
<td>Validation</td>
<td>65%</td>
</tr>
<tr>
<td>Test</td>
<td>59%</td>
</tr>
<tr>
<td>Clean Test</td>
<td>66%</td>
</tr>
</tbody>
</table>

Table 2: Effect of graph abstraction features on Top-1 validation accuracy.

<table>
<thead>
<tr>
<th>Control</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Node Features</td>
<td>57%</td>
</tr>
<tr>
<td>No Edge Features</td>
<td>63%</td>
</tr>
<tr>
<td>No Structural Edges</td>
<td>61%</td>
</tr>
<tr>
<td>No Equality Edges</td>
<td>46%</td>
</tr>
</tbody>
</table>
Thoughts

Pros:

● Encoding I/O pairs as a graph
● Flexible compared to existing approaches

Doubts:

● Limited to single function programs
● Scalability and performance in real world data
● Does not consider parameter selection