Sketches for Automatic Coding

Solar-Lezama et al, Murali et al

Presented by Dimitri Leggas and Jeevan Farias

April 18, 2019
Automatic Coding

- Neural Program Induction
- Neural Program Synthesis
I/O Pair Examples

\[
[2, 3, 4, 5, 6] \rightarrow [2, 4, 6] \\
[5, 8, 3, 2, 2, 1, 12] \rightarrow [8, 2, 2, 12]
\]
I/O Pair Examples

\[ [1, 2, 3] \rightarrow X \]
\[ [4, 5, 6] \rightarrow ? \]
Related Work

Learning Simple Algorithms From Examples (Zaremba et al, 2015)
Related Work

Neural Random Access Machines (Kurach et al, 2015)
Related Work

DeepCoder (Balog et al, 2016)

a ← [int]  
b ← FILTER (<0) a  
c ← MAP (×4) b  
d ← SORT c  
e ← REVERSE d

An input-output example:

Input:  
[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]

Output:  
[-12, -20, -32, -36, -68]
Related Work

RobustFill (Devlin et al, 2017)
Motivation for Program Generation

- Implicit programs
- Learning over source code
- Specificity of domain
- Natural language specification
Definitions

- Program Sketch
- Domain Specific Language
Problem Overviews

- Neural Sketch Learning for Conditional Program Generation
- Learning to Infer Program Sketches
Problem Formulation

Learn over program sketches using a probabilistic encoder-decoder, conditioned on labels, to generate source code in AML.
Goal

Create a model that can generate source code from some 'spec'

Learn a function $g$

For test case $(X, \text{Prog})$, $g(X) = \text{Prog}'$
Example 1

\[ X_{Types} = \{ \text{FileWriter} \} \]
\[ X_{Calls} = \{ \text{write} \} \]
\[ X_{Keys} = \emptyset \]

BufferedWriter bw;
FileWriter fw;
try {
    fw = new FileWriter($String, $boolean);
    bw = new BufferedWriter(fw);
    bw.newLine();
    bw.write($String);
    bw.flush();
    bw.close();
} catch (IOException _e) {
}
Example 1a

```java
String s;
BufferedReader br;
FileReader fr;
try {
    fr = new FileReader($String);
    br = new BufferedReader(fr);
    while ((s = br.readLine()) != null) {
        br.close();
    }
} catch (FileNotFoundException _e) {
    _e.printStackTrace();
} catch (IOException _e) {
    _e.printStackTrace();
}
```
Example 1a

A diagram illustrating a program sketch for conditional program generation. The sketch includes nodes for Try, Catch, While, BR.new, BR.readLine, BR.close, FNFException, IOException, and printStackTrace. The edges show the flow of control and data through the sketch.
Example 2a

Label $X = (X_{\text{Calls}}, X_{\text{Types}}, X_{\text{Keys}})$

$$X = (\{\text{readLine}\}, \emptyset, \emptyset)$$

String s;
BufferedReader br;
FileReader fr;
try {
    fr = new FileReader($\text{String}$);
    br = new BufferedReader(fr);
    while ((s = br.readLine()) != null) {}
    br.close();
} catch (FileNotFoundException _e) {
} catch (IOException _e) {
}

(a)

Solution?

$X = (\{\text{readline}\}, \{\text{FileReader}\}, \emptyset)$

(b)

String s;
BufferedReader br;
InputStreamReader isr;
try {
    isr = new InputStreamReader($\text{InputStream}$);
    br = new BufferedReader(isr);
    while ((s = br.readLine()) != null) {}
} catch (IOException _e) {
}
Conditional Program Generation

- Functional equivalence
- Maximize the expected value that $g(X)$ and some $Prog$ belong to the same equivalence relation
- $E[I((g(X), Prog) \in Eqv)]$
- BAYOU
Technical Approach

- $P(\text{Prog}|X, \theta)$
- $\theta^* = \arg \max_\theta \sum_i \log P(\text{Prog}_i|X_i, \theta)$
- $g(X) = \arg \max_{\text{Prog}} P(\text{Prog}|X, \theta^*)$
Abstraction

- Define abstraction function \( \alpha : \mathbb{P} \rightarrow \mathbb{Y} \)
- \( \text{sat}(Y) \) if \( \alpha^{-1}(Y) \neq \emptyset \) aka...
- \( P(\text{Prog} \mid Y) \neq 0 \iff Y = \alpha(\text{Prog}) \)
Motivation
Neural Sketch Learning for Conditional Program Generation
Learning to Infer Program Sketches

Abstraction Function

\[
\begin{align*}
\alpha(\text{skip}) &= \text{skip} \\
\alpha(\text{call } Sexp_0, a(Sexp_1, \ldots, Sexp_k)) &= \text{call } \tau_0.a(\tau_1, \ldots, \tau_k) \text{ where } \tau_i \text{ is the type of } Sexp_i \\
\alpha(\text{Prog}_1; \text{Prog}_2) &= \alpha(\text{Prog}_1); \alpha(\text{Prog}_2) \\
\alpha(\text{let } x = Sexp_0, a(Sexp_1, \ldots, Sexp_k)) &= \text{call } \tau_0.a(\tau_1, \ldots, \tau_k) \text{ where } \tau_i \text{ is the type of } Sexp_i \\
\alpha(\text{if } \text{Exp} \text{ then } \text{Prog}_1 \text{ else } \text{Prog}_2) &= \text{if } \alpha(\text{Exp}) \text{ then } \alpha(\text{Prog}_1) \text{ else } \alpha(\text{Prog}_2) \\
\alpha(\text{while } \text{Exp} \text{ do } \text{Prog}) &= \text{while } \alpha(\text{Cond}) \text{ do } \alpha(\text{Prog}) \\
\alpha(\text{try } \text{Prog} \text{ catch}(x_1) \text{Prog}_1 \ldots \text{catch}(x_k) \text{Prog}_k) &= \text{try } \alpha(\text{Prog}) \text{ catch}(\tau_1) \alpha(\text{Prog}_1) \ldots \text{catch}(\tau_k) \alpha(\text{Prog}_k) \text{ where } \tau_i \text{ is the type of } x_i \\
\alpha(\text{Exp}) &= \text{[ ] if Exp is a constant or variable name} \\
\alpha(Sexp_0, a(Sexp_1, \ldots, Sexp_k)) &= [\tau_0.a(\tau_1, \ldots, \tau_k)] \text{ where } \tau_i \text{ is the type of } Sexp_i \\
\alpha(\text{let } x = \text{Call : Exp}_1) &= \text{append}(\alpha(\text{Call}), \alpha(\text{Exp}_1))
\end{align*}
\]
Grammar for Sketches

\[
\begin{align*}
Y &::= \text{skip} | \text{call } \text{Cexp} | Y_1; Y_2 | \\
& \quad \text{if } \text{Cseq} \text{ then } Y_1 \text{ else } Y_2 | \\
& \quad \text{while } \text{Cseq} \text{ do } Y_1 | \text{try } Y_1 \text{ Catch} \\
\text{Cexp} &::= \tau_0.a(\tau_1, \ldots, \tau_k) \\
\text{Cseq} &::= \text{List of Cexp} \\
\text{Catch} &::= \text{catch}(\tau_1) Y_1 \ldots \text{catch}(\tau_k) Y_k
\end{align*}
\]
Encoder-Decoder

\[ P(Y|X, \theta) = \int_{Z \in \mathbb{R}^m} P(Z|X, \theta)P(Y|Z, \theta)dZ \]
Encoder

- Convert each label (ex. $X_{\text{Calls},i}$) to one-hot vector representation
- Assume $h$ hidden units
- Define an encoder function, ex:
  $$f(X_{\text{Calls},i}) = \tanh((W_h \cdot X_{\text{Calls},i} + b_h) \cdot W_d + b_d)$$
- $W_h \in \mathbb{R}^{|\text{Calls}| \times h}$, $b_h \in \mathbb{R}^h$, $W_d + b_d \in \mathbb{R}^d$
Decoder

- Task: generate sketch $Y$ by sampling from the space of $P(Y|Z)$
- $Z$ is a real vector-valued latent variable
- Start with the root node pair $(root, child)$
- Depth first tree exploration
Decoder (cont)

1. (try, c), (FR.new(String), s), (BR.new(FR), s), (while, c), (BR.readLine(), c), (skip, ·)
2. (try, c), (FR.new(String), s), (BR.new(FR), s), (while, s), (BR.close(), ·)
3. (try, s), (catch, c), (FileNotFoundException, c), (T.printStackTrace(), ·)
4. (try, s), (catch, s), (catch, c), (IOException, c), (T.printStackTrace(), ·)

String s;
BufferedReader br;
FileReader fr;
try {
    fr = new FileReader($String);
    br = new BufferedReader(fr);
    while (((s = br.readLine())) != null) {} 
    br.close();
} catch (FileNotFoundException _e) {
    _e.printStackTrace();
} catch (IOException _e) {
    _e.printStackTrace();
}
Concretization

- Type directed, stochastic search
- Given sketch $Y$, perform random walk of space of partially concretized sketches
- Follows distribution of $P(Prog | Y)$
- Ex. $x_1.a(x_2); \tau_1.b(\tau_2)$
- Defined set of neighbors for each state
- Prioritize simple programs
Experiments

- 1500 Android apps
- 150,000 methods
- Labels defined by heuristic

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Vocab</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_{Calls}$</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>2584</td>
</tr>
<tr>
<td>$X_{Types}$</td>
<td>1</td>
<td>15</td>
<td>3</td>
<td>1521</td>
</tr>
<tr>
<td>$X_{Keys}$</td>
<td>2</td>
<td>29</td>
<td>8</td>
<td>993</td>
</tr>
<tr>
<td>$X$</td>
<td>4</td>
<td>48</td>
<td>13</td>
<td>5098</td>
</tr>
</tbody>
</table>

Solar-Lezama et al, Murali et al

Sketches for Automatic Coding
t-SNE Plot of Latent Space
Accuracy Metrics

- AST Comparison
- Minimum Jaccard Distance between sets of sequences of API calls
- Minimum Jaccard Distance between the sets of API calls
- Minimum absolute difference between number of statements
- Minimum absolute difference between number of control structures
## Results

(a) M1. Proportion of test programs for which the expected AST appeared in the top-10 results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Input Label Observability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>GED-AML</td>
<td>0.13</td>
</tr>
<tr>
<td>GSNN-AML</td>
<td>0.07</td>
</tr>
<tr>
<td>GED-Sk</td>
<td>0.59</td>
</tr>
<tr>
<td>GSNN-Sk</td>
<td>0.57</td>
</tr>
</tbody>
</table>

(b) M2. Average minimum Jaccard distance on the set of sequences of API methods called in the test program vs the top-10 results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Input Label Observability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>GED-AML</td>
<td>0.82</td>
</tr>
<tr>
<td>GSNN-AML</td>
<td>0.88</td>
</tr>
<tr>
<td>GED-Sk</td>
<td>0.34</td>
</tr>
<tr>
<td>GSNN-Sk</td>
<td>0.36</td>
</tr>
</tbody>
</table>

(c) M3. Average minimum Jaccard distance on the set of API methods called in the test program vs the top-10 results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Input Label Observability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100%</td>
</tr>
<tr>
<td>GED-AML</td>
<td>0.52</td>
</tr>
<tr>
<td>GSNN-AML</td>
<td>0.59</td>
</tr>
<tr>
<td>GED-Sk</td>
<td>0.11</td>
</tr>
<tr>
<td>GSNN-Sk</td>
<td>0.13</td>
</tr>
</tbody>
</table>

(d) M4. Average minimum difference between the number of statements in the test program vs the top-10 results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1</td>
</tr>
<tr>
<td>GED-AML</td>
<td>0.02</td>
</tr>
<tr>
<td>GSNN-AML</td>
<td>0.01</td>
</tr>
<tr>
<td>GED-Sk</td>
<td>0.23</td>
</tr>
<tr>
<td>GSNN-Sk</td>
<td>0.20</td>
</tr>
</tbody>
</table>

(e) M5. Average minimum difference between the number of control structures in the test program vs the top-10 results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1</td>
</tr>
<tr>
<td>GED-AML</td>
<td>0.02</td>
</tr>
<tr>
<td>GSNN-AML</td>
<td>0.01</td>
</tr>
<tr>
<td>GED-Sk</td>
<td>0.23</td>
</tr>
<tr>
<td>GSNN-Sk</td>
<td>0.20</td>
</tr>
</tbody>
</table>

(f) Metrics for 50% observability evaluated only on unseen data.
Learning to Infer Program Sketches

- This paper develops a dynamic system to incorporate pattern recognition and explicit reasoning to solve programming puzzles.
Formulation

- DSL with program space $\mathcal{G}$
- Set of program specifications (specs) containing I/O examples: $\mathcal{X}_i = \{(x_{ij}, y_{ij})\}_{j=1,...,n}$
- We have solved problem $\mathcal{X}_i$ if we find the true program $F_i$ such that

$$\forall j : F_i(x_{ij}) = y_{ij}$$
Can we solve the problem quickly?
The problem becomes:

\[ \max \log \mathbb{P} [\text{Time}(x_i \rightarrow F_i) < t] \]
System

SketchAdapt

- **Sketch Generator**: Proposes set of possible (incomplete) sketches based on a spec
- **Program Synthesizer**: Takes a sketch as a starting point, then performs explicit search to “fill the holes”
Novel Approach

- Define a more general sketch: a valid program tree where any subtree may be replaced with the special token <HOLE>
- This token designates locations in the program tree where pattern recognition is difficult and more explicit search is necessary
- This allows the system to learn how much to rely on each component
Infer Sketches via Self-supervision

- Generator will be parametrized by a RNN, and is trained to assign a high probability to sketches that can be quickly completed.
- We can now reframe the program synthesis problem:

$$\max_{\phi} \log P_{s \sim q_{\phi}(-|x_i)} [\text{Time}(s \to F_i) < t]$$
How to set the time budget?

- In order to make the system more robust, train it to output sketches that are suitable for a range of timeout budgets.
- Rewrite the previous optimization as:

\[
\max_{\phi} \log \mathbb{P}_{t \sim D_t} \left[ \text{Time}(s \rightarrow F_i) < t \right] \\
\quad \quad \quad s \sim q_{\phi}(-|X_i)
\]
Loss

- Maximize the objective function:

\[
\text{obj} = \mathbb{E}_{t \sim D_t} \log \sum_{(F, \mathcal{X}) \sim G} q_{\phi}(s | \mathcal{X})
\]

- Quickly solve “easy” problems with concrete sketches, but also sample more general sketches for harder problems
Generator Implementation

- The sketch generator is a sequence-to-sequence RNN with attention
- Spec is encoded via LSTM
- Sketch is decoded token-by-token while attending to the spec
Motivation

Neural Sketch Learning for Conditional Program Generation

Learning to Infer Program Sketches

Synthesizer Implementation

- The program synthesizer uses probabilities of primitives appearing in the program in order to induce a PCFG over an incomplete sketch: $p(F|s, \theta)$
- Candidate programs are enumerated in decreasing probability
- The primitive probabilities are provided by a learned recognizer (feed forward MLP ending in softmax)
Architecture

Program spec, $X$ → Neural sketch generator, $q_\phi(-|X)$ → Count > 0 (Map (HOLE)) → Recognizer, $r_\psi(X, s)$ → Enumerator → Count > 0 (Map +1 input) → Full program, $F$
Computing the Loss in Practice

- Note that $\text{Time}(s \rightarrow F) \leq 1/p(F|s, \theta)$
- Bound the objective by

$$
\text{obj} \geq \mathbb{E}_{t \sim D_t} \log \sum_{(F, X') \sim G, s: 1/p(F|s, \theta) < t} q_\phi(s|X')
$$

- Because the generator and synthesizer are highly correlated, sketches that maximize $q_\phi(s|X')$ will minimize $p(F|s, \theta)$. So we can use only the dominating term:

$$
\text{obj}^* = \mathbb{E}_{t \sim D_t} \log q_\phi(s^*|X') \leq \text{obj}
$$
Training

**Algorithm 1** SKETCHADAPT Training

**Require:** Sketch Generator $q_\phi(sket|\mathcal{X})$; Recognizer $r_\psi(\mathcal{X}, sketch)$; Enumerator dist. $p(F|\theta, sketch)$, Base Parameters $\theta_{base}$

*Train Recognizer, $r_\psi$:*

for $F, \mathcal{X}$ in Dataset (or sampled from DSL) do

Sample $t \sim D_t$

sketches, probs $\leftarrow$ list all possible sketches of $F$, with probs given by $p(F|s, \theta_{base})$

$\text{sketch} \leftarrow$ sketch with largest prob s.t. prob $< t$

$\theta \leftarrow r_\psi(\mathcal{X}, \text{sketch})$

grad. step on $\psi$ to maximize $\log p(F|\theta, \text{sketch})$

end for

*Train Sketch Generator, $q_\phi$:*

for $F, \mathcal{X}$ in Dataset (or sampled from DSL) do

Sample $t \sim D_t$

$\theta \leftarrow r_\psi(\mathcal{X})$

sketches, probs $\leftarrow$ list all possible sketches of $F$, with probs given by $p(F|s, \theta)$

$\text{sketch} \leftarrow$ sketch with largest prob s.t. prob $< t$

grad. step on $\phi$ to maximize $\log q_\phi(\text{sketch}|\mathcal{X})$

end for
Results

List Processing: length 3 test programs

List Processing: length 4 test programs

Solar-Lezama et al, Murali et al

Sketches for Automatic Coding
Results

String Editing

% of problems solved

Number of candidates evaluated per problem

10^1 10^2 10^3 10^4

SketchAdapt, beam 100 (ours)
SketchAdapt, beam 50 (ours)
Generator only, beam 100 (RobustFill)
Generator only, beam 50 (RobustFill)
Synthesizer only (Deecoder)
Discussion

- Developed a flexible and robust approach that requires processing less data
- No labels required
- Integrates multiple forms of computation (pattern recognition and search)
Conclusions

- Generalizability
- Evaluation
- Flexibility
- Limitations