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

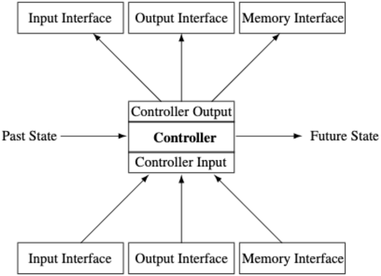
$$[2, 3, 4, 5, 6] \rightarrow [2, 4, 6]$$
$$[5, 8, 3, 2, 2, 1, 12] \rightarrow [8, 2, 2, 12]$$

I/O Pair Examples

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

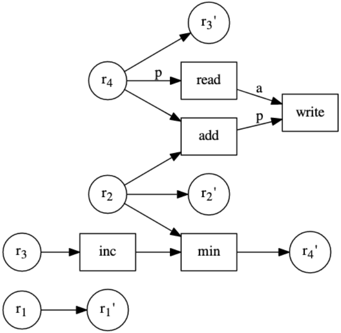
Related Work

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



Related Work

Neural Random Access Machines (Kurach et al, 2015)



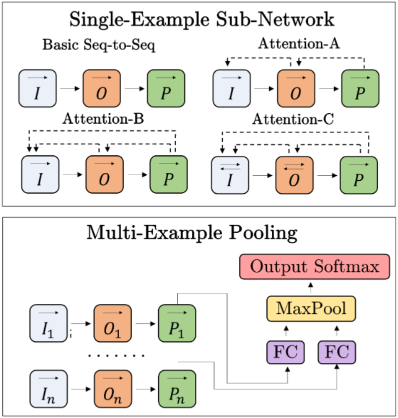
Related Work

DeepCoder (Balog et al, 2016)

a ← [int]	**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]

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 $(X, Prog)$, $g(X) = Prog'$

Example 1

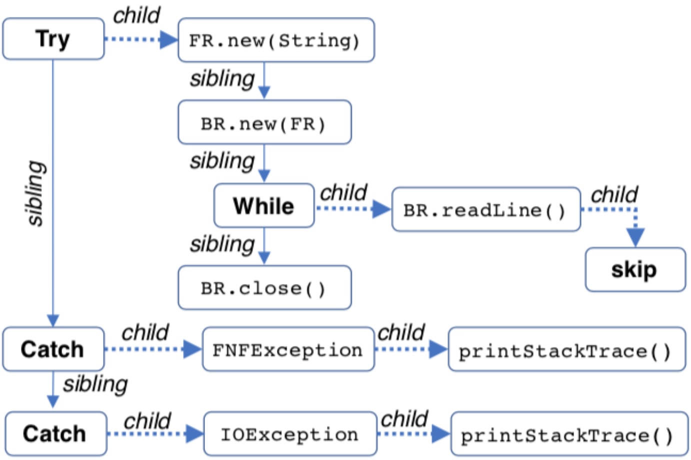
$X_{Types} = \{\texttt{FileWriter}\}$
$X_{Calls} = \{\texttt{write}\}$
$X_{Keys} = \emptyset$

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

Example 1a

```
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


Try
child
FR.new(String)
sibling
BR.new(FR)
sibling
While
child
BR.readLine()
child
skip
sibling
BR.close()
sibling
Catch
child
FNFException
child
printStackTrace()
sibling
Catch
child
IOException
child
printStackTrace()

Example 2a

Label $X = (X_{Calls}, X_{Types}, X_{Keys})$
$X = (\{\texttt{readLine}\}, \emptyset, \emptyset)$

```
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) {
} catch (IOException _e) {
}
```

(a)

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

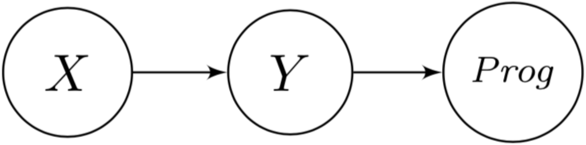
(b)

Solution?
$X = (\{\texttt{readline}\}, \{\texttt{FileReader}\}, \emptyset)$

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

Techincal Approach



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

Abstraction

- Define abstraction function $\alpha : \mathbb{P} \rightarrow \mathbb{Y}$
- $sat(Y)$ if $\alpha^{-1}(Y) \neq \emptyset$ aka...
- $P(Prog|Y) \neq 0 \iff Y = \alpha(Prog)$

Abstraction Function

$$
\begin{array}{rcl}
\alpha(\textbf{skip}) & = & \textbf{skip} \\
\alpha(\textbf{call}\ \mathsf{Sexp}_0.a(\mathsf{Sexp}_1,\ldots,\mathsf{Sexp}_k)) & = & \textbf{call}\ \tau_0.a(\tau_1,\ldots,\tau_k)\ \text{ where } \tau_i \text{ is the type of } \mathsf{Sexp}_i \\
\alpha(\mathsf{Prog}_1;\mathsf{Prog}_2) & = & \alpha(\mathsf{Prog}_1);\alpha(\mathsf{Prog}_2) \\
\alpha(\textbf{let}\ x = \mathsf{Sexp}_0.a(\mathsf{Sexp}_1,\ldots,\mathsf{Sexp}_k)) & = & \textbf{call}\ \tau_0.a(\tau_1,\ldots,\tau_k)\ \text{ where } \tau_i \text{ is the type of } \mathsf{Sexp}_i \\
\alpha(\textbf{if}\ \mathsf{Exp}\ \textbf{then}\ \mathsf{Prog}_1\ \textbf{else}\ \mathsf{Prog}_2) & = & \textbf{if}\ \alpha(\mathsf{Exp})\ \textbf{then}\ \alpha(\mathsf{Prog}_1)\ \textbf{else}\ \alpha(\mathsf{Prog}_2) \\
\alpha(\textbf{while}\ \mathsf{Exp}\ \textbf{do}\ \mathsf{Prog}) & = & \textbf{while}\ \alpha(\mathsf{Cond})\ \textbf{do}\ \alpha(\mathsf{Prog}) \\
\alpha(\textbf{try}\ \mathsf{Prog}\ \textbf{catch}(x_1)\ \mathsf{Prog}_1\ \ldots\ \textbf{catch}(x_k)\ \mathsf{Prog}_k) & = & \textbf{try}\ \alpha(\mathsf{Prog}) \\
 & & \quad \textbf{catch}(\tau_1)\ \alpha(\mathsf{Prog}_1)\ldots\ \textbf{catch}(\tau_k)\ \alpha(\mathsf{Prog}_k) \\
 & & \text{where } \tau_i \text{ is the type of } x_i \\
\alpha(\mathsf{Exp}) & = & [\,]\ \text{if Exp is a constant or variable name} \\
\alpha(\mathsf{Sexp}_0.a(\mathsf{Sexp}_1,\ldots,\mathsf{Sexp}_k)) & = & [\tau_0.a(\tau_1,\ldots,\tau_k)]\ \text{ where } \tau_i \text{ is the type of } \mathsf{Sexp}_i \\
\alpha(\textbf{let}\ x = \mathsf{Call} : \mathsf{Exp}_1) & = & \mathit{append}(\alpha(\mathsf{Call}),\alpha(\mathsf{Exp}_1))
\end{array}
$$

Grammar for Sketches

$$
\begin{array}{lcl}
\mathsf{Y} & ::= & \textbf{skip} \mid \textbf{call}\ \mathsf{Cexp} \mid \mathsf{Y}_1 ; \mathsf{Y}_2 \mid \\
 & & \textbf{if}\ \mathsf{Cseq}\ \textbf{then}\ \mathsf{Y}_1\ \textbf{else}\ \mathsf{Y}_2 \mid \\
 & & \textbf{while}\ \mathsf{Cseq}\ \textbf{do}\ \mathsf{Y}_1 \mid \textbf{try}\ \mathsf{Y}_1\ \mathsf{Catch} \\
\mathsf{Cexp} & ::= & \tau_0.a(\tau_1, \ldots, \tau_k) \\
\mathsf{Cseq} & ::= & \text{List of Cexp} \\
\mathsf{Catch} & ::= & \textbf{catch}(\tau_1)\ \mathsf{Y}_1\ \ldots\ \textbf{catch}(\tau_k)\ \mathsf{Y}_k
\end{array}
$$

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. $X_{Calls,i}$) to one-hot vector representation
- Assume h hidden units
- Define an encoder function, ex: $f(X_{Calls,i}) = \tanh((W_h \cdot X'_{Calls,i} + b_h) \cdot W_d + b_d)$
- $W_h \in \mathbb{R}^{|Calls| \times h}$, $\boldsymbol{b}_h \in \mathbb{R}^h$, $W_d + \boldsymbol{b}_d \in \mathbb{R}^d$

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**, $\cdot$)
2. (**try**, c), (`FR.new(String)`, s), (`BR.new(FR)`, s), (**while**, s), (`BR.close()`, $\cdot$)
3. (**try**, s), (**catch**, c), (`FNFException`, c), (`T.printStackTrace()`, $\cdot$)
4. (**try**, s), (**catch**, s), (**catch**, c), (`IOException`, c), (`T.printStackTrace()`, $\cdot$)

```
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

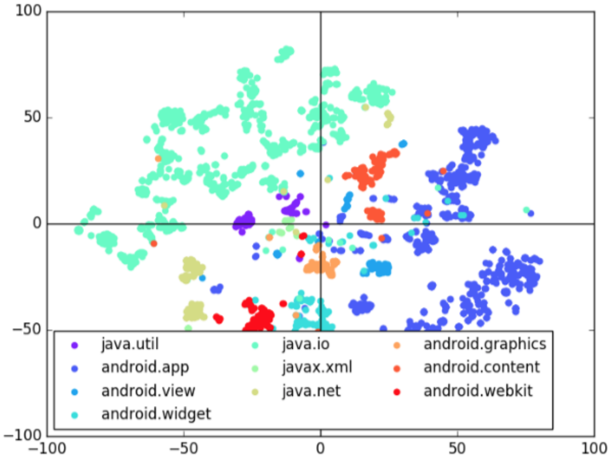
- Type directed, stochastic search
- Given sketch Y, perform random walk of space of *partially concretized sketches*
- Follows distribution of $P(Prog|Y)$
- Ex. $x_1.a(x_2); \tau_1.b(\tau_2)$
- Defined set of neighbors for each state
- Prioritize simple programs

Experiments

	Min	Max	Median	Vocab
X_{Calls}	1	9	2	2584
X_{Types}	1	15	3	1521
X_{Keys}	2	29	8	993
X	4	48	13	5098

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

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

Model	Input Label Observability			
	100%	75%	50%	25%
GED-AML	0.13	0.09	0.07	0.02
GSNN-AML	0.07	0.04	0.03	0.01
GED-Sk	**0.59**	**0.51**	**0.44**	**0.21**
GSNN-Sk	0.57	0.48	0.41	0.18

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

Model	Input Label Observability			
	100%	75%	50%	25%
GED-AML	0.82	0.87	0.89	0.97
GSNN-AML	0.88	0.92	0.93	0.98
GED-Sk	**0.34**	**0.43**	**0.50**	**0.76**
GSNN-Sk	0.36	0.46	0.53	0.78

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

Model	Input Label Observability			
	100%	75%	50%	25%
GED-AML	0.52	0.58	0.61	0.77
GSNN-AML	0.59	0.64	0.68	0.83
GED-Sk	**0.11**	**0.17**	**0.22**	**0.50**
GSNN-Sk	0.13	0.19	0.25	0.52

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

Model	Input Label Observability			
	100%	75%	50%	25%
GED-AML	0.49	0.47	0.46	0.46
GSNN-AML	0.52	0.49	0.49	0.53
GED-Sk	**0.05**	**0.06**	**0.06**	**0.09**
GSNN-Sk	**0.05**	**0.06**	**0.06**	**0.09**

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

Model	Input Label Observability			
	100%	75%	50%	25%
GED-AML	0.31	0.30	0.30	0.34
GSNN-AML	0.32	0.31	0.32	0.39
GED-Sk	**0.03**	**0.03**	**0.03**	0.04
GSNN-Sk	**0.03**	**0.03**	**0.03**	**0.03**

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

Model	Metric				
	M1	M2	M3	M4	M5
GED-AML	0.02	0.97	0.71	0.50	0.37
GSNN-AML	0.01	0.98	0.74	0.51	0.37
GED-Sk	**0.23**	**0.70**	**0.30**	**0.08**	**0.04**
GSNN-Sk	0.20	0.74	0.33	**0.08**	**0.04**

(f) Metrics for 50% obsevability 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
- State-of-the-art performance via self-supervised learning

Formulation

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

$$\forall j : F_i(x_{ij}) = y_{ij}$$

Formulation

- Can we solve the problem quickly?
- The problem becomes:

$$\max \log \mathbb{P}\left[\mathrm{Time}(\mathcal{X}_i \rightarrow F_i) < t\right]$$

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:

$$\max_{\phi} \log \mathbb{P}_{s \sim q_{\phi}(-|\mathcal{X}_i)} \left[\mathrm{Time}(s \rightarrow F_i) < t \right]$$

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:

$$\max_{\phi} \log \mathop{\mathbb{P}}_{\substack{t \sim \mathcal{D}_t \\ s \sim q_\phi(-|\mathcal{X}_i)}} [\mathrm{Time}(s \to F_i) < t]$$

Loss

- Maximize the objective function:

$$obj = \mathop{\mathbb{E}}_{\substack{t \sim \mathcal{D}_t \\ (F, \mathcal{X}) \sim \mathcal{G}}} \log \sum_{s : \mathrm{Time}(s \to F) < t} q_\phi(s|\mathcal{X})$$

- Quickly solve "easy" problems with concrete sketches, but also sample more general sketches for harder problems

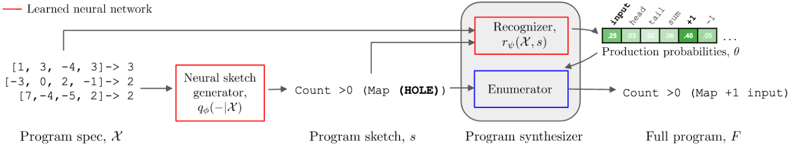
Generator Implementation

- 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

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



Computing the Loss in Practice

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

$$obj \geq \mathop{\mathbb{E}}_{\substack{t \sim \mathcal{D}_t \\ (F,\mathcal{X}) \sim \mathcal{G}}} \log \sum_{s : 1/p(F|s,\theta) < t} q_\phi(s|\mathcal{X})$$

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

$$obj^* = \mathop{\mathbb{E}}_{\substack{t \sim \mathcal{D}_t \\ (F,\mathcal{X}) \sim \mathcal{G}}} \log q_\phi(s^*|\mathcal{X}) \leq obj$$

Training

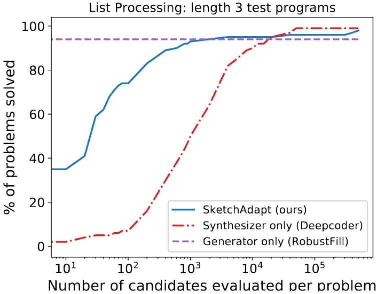
Algorithm 1 SketchAdapt Training

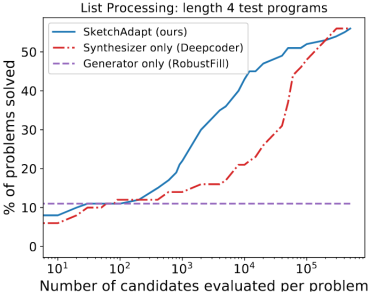
Require: Sketch Generator $q_\phi(sketch|\mathcal{X})$; Recognizer $r_\psi(\mathcal{X}, sketch)$; Enumerator dist. $p(F|\theta, sketch)$, Base Parameters θ_{base}

Train Recognizer, r_ψ:
for $F, \mathcal{X}$ in Dataset (or sampled from DSL) **do**
  Sample $t \sim \mathcal{D}_t$
  $sketches, probs \leftarrow$ list all possible sketches of F, with probs given by $p(F|s, \theta_{base})$
  $sketch \leftarrow$ sketch with largest prob s.t. prob $< t$.
  $\theta \leftarrow r_\psi(\mathcal{X}, sketch)$
  grad. step on ψ to maximize $\log p(F|\theta, sketch)$
end for

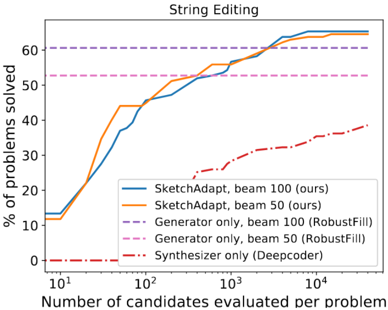
Train Sketch Generator, q_ϕ:
for $F, \mathcal{X}$ in Dataset (or sampled from DSL) **do**
  Sample $t \sim \mathcal{D}_t$
  $\theta \leftarrow r_\psi(\mathcal{X})$
  $sketches, probs \leftarrow$ list all possible sketches of F, with probs given by $p(F|s, \theta)$
  $sketch \leftarrow$ sketch with largest prob s.t. prob $< t$.
  grad. step on ϕ to maximize $\log q_\phi(sketch|\mathcal{X})$
end for

Results


List Processing: length 3 test programs
% of problems solved
100
80
60
40
20
0
10^1
10^2
10^3
10^4
10^5
Number of candidates evaluated per problem
SketchAdapt (ours)
Synthesizer only (Deepcoder)
Generator only (RobustFill)


List Processing: length 4 test programs
% of problems solved
50
40
30
20
10
0
10^1
10^2
10^3
10^4
10^5
Number of candidates evaluated per problem
SketchAdapt (ours)
Synthesizer only (Deepcoder)
Generator only (RobustFill)

Results


String Editing
% of problems solved
0
10
20
30
40
50
60
10^1
10^2
10^3
10^4
Number of candidates evaluated per problem
SketchAdapt, beam 100 (ours)
SketchAdapt, beam 50 (ours)
Generator only, beam 100 (RobustFill)
Generator only, beam 50 (RobustFill)
Synthesizer only (Deepcoder)

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