Neural Generation of Source Code for Program Synthesis

Kensen Shi (kensens@stanford.edu)

Motivation

Some existing program synthesizers consider many "obviously bad" candidate programs:

```
String uppercase(String str) {
  int var1 = 0;
  String var2 = "";
  str = var2;
  return "";
}
```

Objective: generate more natural candidate programs (methods) for the synthesizer

Task Setup

Input: types in method signature
Output: sequence of tokens

Datasets:

1. *GitHub*: ~10,000 methods scraped
2. *Synthesizer*: ~500 solutions plus ~3,000 "helpful" methods, with weights, for 90 different tasks. Train/dev/test split by task.
3. *Solutions*: subset of only the ~500 solutions

Variable names are canonicalized (e.g., `arg1`, `var2`). Some types of tokens are grouped, e.g., `230` becomes `<NumberLit>`. Vocab size of 100.

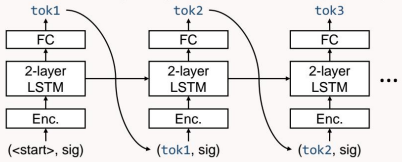
Example training pair:

- Input: `long <Class>`
- Output: `return ( arg1 == null ) ? <NumberLit> : arg1 . <Field> ;`

Model

Architecture:

(previous token, function signature)
→ 64-dimensional encodings of tokens
→ 2-layer LSTM (512 hidden units per layer)
→ FC softmax layer (outputs token probs)



Loss Function: negative weighted LL of the dataset, normalized by the sequence length

$$\left(-\sum_{i=1}^{m} \frac{w^{(i)}}{|y^{(i)}|} \sum_{j=1}^{|y^{(i)}|} \log \hat{p}\left(y^{(i)\langle j \rangle}\right) \right) \Big/ \sum_{i=1}^{m} w^{(i)}$$

Transfer Learning:

- *GitHub* dataset is large but doesn't contain many interesting control structures
- *Synthesizer* dataset is small but is exactly the "style" of code we want to generate
- Transfer from *GitHub* to *Synthesizer*

Results: loss (acc.) on *Syn.* & *Sol.* datasets

Model	*Syn*-Train	*Syn*-Test	*Sol*-Test
GitHub	1.4 (69%)	1.3 (71%)	1.3 (72.4%)
Synthesizer	0.5 (84%)	1.1 (68%)	1.1 (69.4%)
Solutions	0.9 (73%)	1.3 (61%)	1.3 (62.6%)
Transfer-*Syn*	0.3 (90%)	**1.0 (74%)**	**0.9 (75.1%)**
Transfer-*Sol*	0.5 (86%)	1.1 (73%)	1.0 (74.6%)

Analysis

Example Generated Programs:

Signature: `<Class>[] <Class> <Class>[]`

```
for (int i1 = <NumberLit>;
     i1 <Ineq> arg2.<Field>; i1++) {
  arg2[i1] = arg1.<Method>(arg2);
}
return arg2;
```

Signature: `int Object String`

```
if (arg1 == null) { arg2 = <NumberLit>; }
return <Class>.<Method>(arg1.<Method>());
```

Signature: `List String[] String[]`

```
ArrayList var1 = new ArrayList();
for (String elem1 : arg1) {
  for (String elem1 : arg2) { ...
```

Main Conclusions:

- Transfer learning results in the best models
- Training on full *Synthesizer* dataset boosts performance even when tested on *Solutions*
- The model generates natural-looking code
- The generated code doesn't always compile. Most common errors:
  - Not understanding types
  - Incorrect variable names
  - Extraneous or unmatched parens/braces
- Token encodings help the model generalize

Future Work:

- Generate a tree instead of a sequence
- Force the model to follow language rules by only sampling from allowable options at each step, possibly with beam search