A Grammar-Based Structural CNN Decoder for Code Generation

Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, Lu Zhang

Peking University
AdeptMind
INTRODUCTION

- Generating code from natural language description.
  - Open the file, F1 → f = open('F1', 'r')

- Automatically code generation is beneficial in various scenarios.
  - Similar code snippets can be generated from another.
  - It takes a long time for a programmer to learn a new implement.
INTRODUCTION

- Previous works with neural network are all based on RNN or LSTM.
  - Researchers [1, 2, 3] have proposed several approach based on AST using LSTM.

- A program is much larger than a natural language sentence and that RNNs suffer from the long dependency problem [4].
  - A program is made up of a large number of AST nodes.

Researchers are showing growing interest in using the CNN as the decoder.

- QANet [1], a CNN encoder-decoder, achieves a significant improvement in SQuAD dataset for question answering.

Can we use CNN for code generation?

---

Background: Learning to Synthesize

- **CNN produces a classifier of a fixed number of categories**
  - Cannot be applied to program generation: the category is infinite or extremely large

- **Yingfei Xiong, Bo Wang, Guirong Fu, Linfei Zang. Learning to Synthesize. GI'18.**
  - Decompose a program generation problem into a series of classification problems
  - Guided by the grammar of the program
  - Presented at APLAS-NIER 2017
Decompose into classification problems

- The key step of generation is to predict the grammar rule, which will be applied to expand the AST.

- The probability of an entire code is decomposed as

\[
p(\text{program}) = \prod_{n=1}^{N} p(r_n | r_1, \ldots, r_{n-1})
\]
The prediction is mainly based on three types of information:

- the context information (e.g. the natural language description)
- the partial AST that has been generated.
- the position of the node to be expanded
OVERVIEW

Context

Partial AST

Position

Information Type

CNN Modules

CNN

Tree-based CNN

Pre-order CNN

Tree-path CNN

Pooling & Attention

Max pooling

Attentive pooling

Max pooling

Attentive pooling

Max pooling

Attentive pooling

Prediction

MLP

softmax

Embedding
The context of our model is a piece of description.

We first tokenize the context, and obtain a sequence of tokens.

Then, a set of convolutional layers are applied.

- We adopt shortcut connections every other layer parallel to linear transformation, as in ResNet [1].

OVERVIEW
TO ENCODE THE PARTIAL AST

- The partial AST is decided by the rules predicted
  - A sequence of rules can be directly encoded by a CNN

- However, it is difficult for the CNN to learn the structure of the AST
  - A tree-based CNN is further applied to capture the structure information
TO CAPTURE THE STRUCTURAL INFORMATION

- We split each node into two nodes.

(a) Tree-based convolution[1]

- A local feature detector of a fixed depth, sliding over a tree to extract structural feature.
- We put a placeholder to indicate where the next grammar rule is applied.

(b) Pre-order convolution

OVERVIEW
In order to tell the position where the next grammar rule is applied, we design a tree-path CNN to catch this information.

We extract the path from the root to the node to expand.
The scope name is often important for code prediction

- Class name
- Method name

The current local scope name is also encoded
Moreover, we also design several components for code generation.

- The CNN for predicted.
- The attentive pooling.
Our main experiment is based on an established benchmark dataset, HearthStone (HS) [1].

The dataset comprises 665 different cards of the HearthStone game.

We use StrAcc (exact match), Acc+ and BLEU-4 score as metrics.

- Acc+ is a human-adjusted accuracy.

EXPERIMENT: HEARTHSTONE

- Our model is compared with previous state-of-the-art results.

- Our model outperforms all previous results.

- We have designed reasonable components of the neural architecture, suited to the code generation task.

- Ablation tests to analyze the contribution of each component.

<table>
<thead>
<tr>
<th>Model</th>
<th>StrAcc</th>
<th>Acc+</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPN (Ling et al. 2016)</td>
<td>6.1</td>
<td>–</td>
<td>67.1</td>
</tr>
<tr>
<td>SEQU2TREE (Dong and Lapata 2016)</td>
<td>1.5</td>
<td>–</td>
<td>53.4</td>
</tr>
<tr>
<td>SNM (Yin and Neubig 2017)</td>
<td>16.2</td>
<td>~18.2</td>
<td>75.8</td>
</tr>
<tr>
<td>ASN (Rabinovich, Stern, and Klein 2017)</td>
<td>18.2</td>
<td>–</td>
<td>77.6</td>
</tr>
<tr>
<td>ASN+SUPATT (Rabinovich, Stern, and Klein 2017)</td>
<td>22.7</td>
<td>–</td>
<td>79.2</td>
</tr>
<tr>
<td><strong>Our system</strong></td>
<td><strong>27.3</strong></td>
<td><strong>30.3</strong></td>
<td><strong>79.6</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line #</th>
<th>Model Variant</th>
<th>Acc+</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Full model</td>
<td>30.3</td>
<td>79.6</td>
</tr>
<tr>
<td>2</td>
<td>Pre-order CNN → LSTM</td>
<td>21.2</td>
<td>78.8</td>
</tr>
<tr>
<td>3</td>
<td>– Predicted rule CNN</td>
<td>24.2</td>
<td>79.2</td>
</tr>
<tr>
<td>4</td>
<td>– Pre-order CNN</td>
<td>25.8</td>
<td>80.4</td>
</tr>
<tr>
<td>5</td>
<td>– Tree-based CNN</td>
<td>25.8</td>
<td>79.4</td>
</tr>
<tr>
<td>6</td>
<td>– Tree-path CNN</td>
<td>28.8</td>
<td><strong>80.4</strong></td>
</tr>
<tr>
<td>7</td>
<td>– Attentive pooling</td>
<td>24.2</td>
<td>79.3</td>
</tr>
<tr>
<td>8</td>
<td>– Scope name</td>
<td>25.8</td>
<td>78.6</td>
</tr>
</tbody>
</table>
EXPERIMENT: HEARTHSTONE

The code we successfully generated.

Our model used a different argument name, but implements a correct functionality.
EXPERIMENT: SEMANTIC PARSING

- Semantic parsing aims to generate logical forms given a natural language description.

**Input description:** list airport in ci0

**Output λ-calculus:**

\[
\text{lambda } \$0 \ e \ ( \ \text{and} \ ( \ \text{airport} \ \$0 \ )

( \ \text{loc:t} \ \$0 \ \text{ci0} \ ))
\]

- We evaluated our model on two semantic parsing datasets (ATIS and JOBS) used in Dong and Lapata (2016) [1] with Accuracy.

EXPERIMENT: SEMANTIC PARSING

- The logic form for semantic parsing is usually short, containing only 1/4–1/3 tokens as in HS.

<table>
<thead>
<tr>
<th></th>
<th>ATIS</th>
<th></th>
<th>JOBS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System</td>
<td>Accuracy</td>
<td>System</td>
<td>Accuracy</td>
</tr>
<tr>
<td></td>
<td>ZH15</td>
<td>84.2</td>
<td>ZH15</td>
<td>85.0</td>
</tr>
<tr>
<td></td>
<td>ZC07</td>
<td>84.6</td>
<td>PEK03</td>
<td>88.0</td>
</tr>
<tr>
<td>Traditional</td>
<td>91.3</td>
<td></td>
<td>LJK13</td>
<td>90.7</td>
</tr>
<tr>
<td>Neural</td>
<td>SEQ2TREE</td>
<td>84.6</td>
<td>SEQ2TREE</td>
<td>90.0</td>
</tr>
<tr>
<td></td>
<td>ASN</td>
<td>85.3</td>
<td>ASN</td>
<td>91.4</td>
</tr>
<tr>
<td></td>
<td>ASN-SUPATT</td>
<td>85.9</td>
<td>ASN-SUPATT</td>
<td>92.9</td>
</tr>
<tr>
<td></td>
<td>Our System</td>
<td>85.0</td>
<td>Our System</td>
<td>89.3</td>
</tr>
</tbody>
</table>

- Neural models are generally worse than the WKZ14 system (based on CCG parser).
- Our model achieves results similar to the state-of-the-art neural models.
CONCLUSION

- We propose a grammar-based structural CNN for code generation.

- Our model makes use of the abstract syntax tree (AST) of a program, and generates code by predicting the grammar rules.

- We address the problem that traditional RNN-based approaches may not be suitable to program generation.
Thank you!

A Grammar-Based Structural CNN Decoder for Code Generation