Using Recurrent Neural Networks for Decompilation

Deborah S. Katz
Carnegie Mellon University

Jason Ruchti and Eric Schulte
Grammatech, Inc.

SANER 2018
We want better decompilation.
Approach:
We use a model based on recurrent neural networks to translate from binary machine code to source code.
Decompilation is Translating Binary Code to Source Code

00 00 09 00 d3 12 00 00 70 33 00 00
00 00 00 00 00 00 09 00 d3 12 00 00
78 33 00 00 00 00 00 00 00 00 00 09 00
d3 12 00 00

00000000 00000000 00001001 00000000
00010010 00000000 00000000 01110000
00110011 00000000 00000000 00000000
...
Decompilation is Translating Binary Code to Source Code

```c
00 00 09 00 d3 12 00 00 70 33 00 00
00 00 00 00 00 00 00 09 00 d3 12 00 00
78 33 00 00 00 00 00 00 00 00 00 09 00
d3 12 00 00

g_return_if_fail(screen_info != NULL);
```
Source Code is More Useful to Humans than Binary

• Human-Readable
• More analysis tools available for source
• Decompilation does not always produce the most useful output
  – Can leave in compiler artifacts, such as:
    • GOTOs
    • Stack pushes for function calls
• Newer techniques rely on compiler details
  – Very specific to individual compilers/languages
• Existing tools are expensive and often unavailable
Decompile is a Translation Problem

• On some level decompilation is translating:
  – Machine-level binary code to higher-level source or intermediate code

• Look to the techniques for translating other equivalent sequences
Key insight:
To we can translate from **binary** to **source** in the same way we can **translate** natural languages.
What You Need to Know About Neural Networks (Not Much)

- **Encoder-Decoder Model**
  - Available off-the-shelf with TensorFlow*
  - Designed for translating sequences
- Adapt to translate compiled machine code to higher-level source
- Train model, then use for decompilation

* https://www.tensorflow.org/
Overview
Creation of Parallel Corpuses

- Used a customized version of Clang to obtain a database of snippets of source code and the equivalent machine/binary code
  - Under certain compiler settings
- We obtained the corpus by compiling many open-source RPM packages
  - 1,151,013 paired snippets of source and binary
The Encoder-Decoder Model Operates on Sequences of Integers

• We train the model on the paired snippets:
  – Machine code and the equivalent source code
  – Each snippet is represented as a sequence of integers

• For example:

  Binary: 00 00 09 00 d3 12 00 00 70 33 00 00 00 00 00 00 00 00 09 00 d3 12 00 00
  Binary tokenization: 4 4 80 4 198 136 4 4 118 173 4 4 4 4 4 4 4 80 4 198 136 4 4
Tokenize Binary and Source Into Useful Units

- Lex source into language-appropriate tokens
  - Keep most popular variable names
- Normalize others
- Assign integers based on frequency in the corpus

<table>
<thead>
<tr>
<th>Integer</th>
<th>Source Token</th>
<th>Integer</th>
<th>Source Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>(</td>
<td>15</td>
<td>}</td>
</tr>
<tr>
<td>5</td>
<td>)</td>
<td>16</td>
<td>*</td>
</tr>
<tr>
<td>6</td>
<td>;</td>
<td>17</td>
<td>if</td>
</tr>
<tr>
<td>7</td>
<td>,</td>
<td>18</td>
<td>var_3</td>
</tr>
<tr>
<td>8</td>
<td>var_0</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>function</td>
<td>20</td>
<td>“string”</td>
</tr>
<tr>
<td>10</td>
<td>=</td>
<td>21</td>
<td>]</td>
</tr>
<tr>
<td>11</td>
<td>var_1</td>
<td>22</td>
<td>[</td>
</tr>
<tr>
<td>12</td>
<td>-&gt;</td>
<td>23</td>
<td>var_4</td>
</tr>
<tr>
<td>13</td>
<td>var_2</td>
<td>24</td>
<td>.</td>
</tr>
<tr>
<td>14</td>
<td>{</td>
<td>25</td>
<td>1</td>
</tr>
</tbody>
</table>
Our Trained Model Takes Binary Machine Code as Input

INPUT:
00 00 09 00 d3 12 00 00
70 33 00 00 00 00 00 00
00 00 09 00 d3 12 00 00
78 33 00 00 00 00 00 00
00 00 09 00 d3 12 00 00
Our Trained Model Turns Binary Machine Code Into Tokens

**INPUT:**
00 00 09 00 d3 12 00 00
70 33 00 00 00 00 00 00
00 00 09 00 d3 12 00 00
78 33 00 00 00 00 00 00
00 00 09 00 d3 12 00 00

**Binary Tokenization:**
4 4 80 4 198 136 4 4
118 173 4 4 4 4 4 4 4 4 4
80 4 198 136 4 4 78 173
4 4 4 4 4 4 4 4 80 4
198 136 4 4

**Source Tokenization**

Source-3 <-> Source-2 <-> Source-1

DEC-3 <-> DEC-2 <-> DEC-1

ENC-1 <-> ENC-2 <-> ENC-3

**Binary Tokenization**

INPUT:
00 00 09 00 d3 12 00 00
70 33 00 00 00 00 00 00
00 00 09 00 d3 12 00 00
78 33 00 00 00 00 00 00
00 00 09 00 d3 12 00 00
Our Trained Model Translates Binary Tokens to Source Tokens

Source Tokenization:
111 4 8 42 31 5 6

Predictions

Binary Tokenization:
4 4 80 4 198 136 4 4
118 173 4 4 4 4 4 4 4 4
80 4 198 136 4 4 78 173
4 4 4 4 4 4 4 4 80 4
198 136 4 4
Our Trained Model Translates Binary Tokens to Source Tokens

Source Tokenization:
111 4 8 42 31 5 6

Binary Tokenization:
4 4 80 4 198 136 4 4
118 173 4 4 4 4 4 4 4 4
80 4 198 136 4 4 80 4
198 136 4 4

(Actual tokenizations are reversed to allow the model to build context)
Our Trained Model **Turns Source Token Sequences Into Source Code**

Source Tokenization:

Predicted Source Code:
```c
return_if_fail( var_0 != var_NULL );
```

Source Tokenization:

Binary Tokenization:

```
111 4 8 42 31 5 6
```
Evaluating Accuracy and Usefulness

- We evaluate on a metric based on edit distance (lower is better)
  - Evaluation on recovery of exact token sequences
  - Evaluation on recovery of the correct types of tokens
- Ideally, we would like to do a user study to evaluate the usefulness of the translations
Research Questions:
RQ1: How long do we have to train for useful translations?
RQ2: How effective is our technique at translating machine code binary to C source code?
RQ1: The Effect of Additional Training Levels Out

![Graph showing the effect of additional training levels on edit distance. The graph plots the edit distance (lower is better) against the number of training pairs (in thousands). Two lines are shown: one for C token types and another for C token values. The graph indicates a decrease in edit distance as the number of training pairs increases.]
RQ2: Usefulness by Edit Distance

<table>
<thead>
<tr>
<th></th>
<th>Maximum Number of C Tokens Per Snippet</th>
<th>Mean Edit Distance</th>
<th>Mean Edit Distance on Token Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>All C Source</td>
<td></td>
<td>0.70</td>
<td>0.52</td>
</tr>
<tr>
<td>Small snippets</td>
<td>5</td>
<td>0.65</td>
<td>0.56</td>
</tr>
<tr>
<td>Small-medium</td>
<td>9</td>
<td>0.67</td>
<td>0.45</td>
</tr>
<tr>
<td>Medium</td>
<td>17</td>
<td>0.72</td>
<td>0.52</td>
</tr>
<tr>
<td>Large</td>
<td>88</td>
<td>0.75</td>
<td>0.55</td>
</tr>
</tbody>
</table>
Example: Recovery of Function Call, Function Name, and Variable Name

• Ground Truth:
  
g_return_if_fail(screen_info != NULL);

• Translation:
  
g_return_if_fail( var_0 != var_NULL );

• Edit distance: 0.29
Example: Recovery of Function Call, Function Name, and Variable Name

• Ground Truth:
  \texttt{g\_return\_if\_fail(screen\_info \neq \textsc{NULL});}

• Translation:
  \texttt{g\_return\_if\_fail( \texttt{var\_0} \neq \texttt{var\_NULL} );}

• Edit distance: 0.29
Example: Recovery of Function Call, Function Name, and Variable Name

- Ground Truth:
  \[ g\_return\_if\_fail(screen\_info \neq \text{NULL}); \]

- Translation:
  \[ g\_return\_if\_fail(\ var\_0 \neq \var\_NULL\ ); \]

- Edit distance: 0.29
Example: Recovery of the General Structure of a Statement

- Ground Truth:
  \[ \text{itr} \rightarrow \text{e} = \text{h} \rightarrow \text{table}[i]; \]

- Translation:
  \[ \text{var}_0 \rightarrow \text{var}_1 = \text{var}_2 \rightarrow \text{var}_3; \]

- Edit distance: 0.64
  - Misses variable names and array index
Example: Recovery of an if statement

• Ground Truth:

```java
if (ts) {
    adjusted_timespec[0] = timespec[0];
    adjusted_timespec[1] = timespec[1];
    adjustment_needed = validate_timespec(ts);
}
```

• Translation:

```java
if ( var_0 ) {
    function( var_1 , var_0->var_2 );
}
```

• Edit distance: 0.79
Example: Recovery of a for loop

- **Ground Truth:**
  ```c
  for (node = tree->head; node; node = next) {
    next = node->next;
    avl_free_node(tree, node);
  }
  ```

- **Translation:**
  ```c
  for (var_0 = var_1 ) var_0 != var_NULL ; var_0 = var_0->var_2 {
    function(var_0->var_3);
  }
  ```

- **Edit distance:** 0.66
Technique Advantages

- Language-independence
- Recovers semantic knowledge about programs
Summary

itr->e = h->table[i];

var_0->var_1 = var_2->var_3;