Understanding Code using Natural Language & Graph Neural Networks

Miltos Allamanis
Microsoft Research Cambridge

Joint work with Marc Brockschmidt, Alex Gaunt, Alex Polozov, Patrick Fernandes, Mahmoud Khademi
Deep Program Understanding

- Deep Learning
  - Understands images/language/speech
  - Finds patterns in noisy data
  - Requires many samples
  - Handling structured data is hard

- DPU
  - Interpretable
  - Generalisation verifiable
  - Manual effort
  - Limited to specialists

Program Structure
Source Code and Natural Language
Code Autocompletion

Text text = new Text(parent, SWT.NONE);

class Text {
  setText(String string): void - Text - 51%
  getText(): String - Text - 12%
  setEnabled(boolean enabled): void - Control - 11%
  setText(String string): void - Text - 86%
  addModifyListener(ModifyListener listener): void - Text - 31%
  setFont(Font font): void - Control - 8%
}

Press 'Space' to show Adaptive Template Proposals (Code Recommenders)

http://www.eclipse.org/recommenders/

https://visualstudio.microsoft.com/services/intellicode/
## Argument Swapping

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Original argument</th>
<th>Correct argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>responseTTLDuration</td>
<td>frequencyCapDuration</td>
<td>responseTTLDuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>responseTTLDuration</td>
<td>frequencyCapDuration</td>
</tr>
<tr>
<td>List&lt;A&gt;</td>
<td>slotResponse</td>
<td>slotResponse</td>
<td>slotResponse</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Method call</th>
<th>New value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Builder</td>
<td>builder</td>
<td>builder</td>
<td>builder</td>
</tr>
<tr>
<td>boolean</td>
<td>isTransposed</td>
<td>isTransposed</td>
<td>isTransposed</td>
</tr>
<tr>
<td>int</td>
<td>startColumnIndex</td>
<td>a.getStartColumnIndex()</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>endColumnIndex</td>
<td>a.getEndColumnIndex()</td>
<td>rows.size()</td>
</tr>
<tr>
<td>int</td>
<td>startRow</td>
<td></td>
<td>a.getStartColumnIndex()</td>
</tr>
<tr>
<td>int</td>
<td>endRow</td>
<td>rows.size()</td>
<td>a.getEndColumnIndex()</td>
</tr>
</tbody>
</table>

---

Rice et al. 2017 “Detecting Argument Selection Defects”
Inferring Type Refinements

Conceptual Types

"a password"

"a JSON string"

Defined Types

string password;

string data = Json.Load();

Latent; we don’t observe in the conceptual types.

Defined explicitly by the programmer.

Dash et al. 2018 “RefiNym: Using Names to Refine Types”
Variable Misuse

Allamanis et al. “Learning to Represent Programs with Graphs”. 2018
Research in ML+Code

- Infer latent intent
- Ambiguous information

https://ml4code.github.io

A Survey of Machine Learning for Big Code and Naturalness

MILTIAKIS ALLAMANIS, Microsoft Research
EARL T. BARR, University College London
PREMKUMAR DEVANBU, University of California, Davis
CHARLES SUTTON, University of Edinburgh and The Alan Turing Institute

Research at the intersection of machine learning, programming languages, and software engineering has recently taken important steps in proposing learnable probabilistic models of source code that exploit code’s abundance of patterns. In this article, we survey this work. We contrast programming languages against natural languages and discuss how these similarities and differences drive the design of probabilistic models. We present a taxonomy based on the underlying design principles of each model and use it to navigate the literature. Then, we review how researchers have adapted these models to application areas and discuss cross-cutting and application-specific challenges and opportunities.
- Graph Neural Networks
- Applications to Source Code
- Applications to NLP
Graph Neural Networks

and Neural Message Passing
Graph Neural Networks

Li et al (2015). *Gated Graph Sequence Neural Networks.*

Neural Message Passing

Current Neighbor States

Prepare “Message”

Summarize Received Information

Next Node State

Current Node State
Gated GNNs

\[ x = \sum_{n' \in \text{neig}(n)} E_{\tau(n' \rightarrow n)} h_{t-1}^{n'} \]

Graph Neural Networks: Message Passing
GNNs: Synchronous Message Passing (All-to-All)
GGNs: Asynchronous Message Passing

Define Schedule
Send Messages
GGNs: Asynchronous Message Passing
Graph Neural Networks: Output

- node selection
- node classification
- graph classification

Li et al (2015). Gated Graph Sequence Neural Networks.

https://github.com/Microsoft/gated-graph-neural-network-samples
Understanding & Generating Source Code

...with graph neural networks.
int SumPositive(int[] arr, int lim) {
    int sum = 0;
    for (int i = 0; i < lim; i++)
        if (arr[i] > 0)
            sum += arr[i];
    return sum;
}
Assert.NotNull(clazz);
Programs as Graphs: Data Flow

\[(x, y) = \text{Foo}();\]

while \((x > 0)\)

\[x = x + y;\]
```c
int SumPositive(int[] arr, int lim) {
    int sum = 0;
    for (int i = 0; i < lim; ++i)
        if (arr[i] > 0)
            sum += arr[i];
    return sum;
}
```

~900 nodes/graph  ~8k edges/graph
Initial Node Representations

Label: outFilePrefix
Type: string

out, file, prefix → Embed → Average → Concat

string, object, ... → Embed → Max Pool
Variable Misuse Task

```csharp
var clazz = classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first = classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(first);

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

Possible type-correct options: clazz, first

⚠️ Not easy to catch with static analysis tools.
Graph Representation for Variable Misuse

```csharp
var clazz = classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.IsNotNull(clazz);

var first = classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.IsNotNull(first);

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

Possible type-correct options: clazz, first
Graph Representation for Variable Misuse

```csharp
var clazz = classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first = classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(first);
Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

**Goal:** make the representation of SLOT as close as possible to the representation of the correct candidate node

\[ f(h_T^{SLOT}, h_T^{first}) \gg f(h_T^{SLOT}, h_T^{clazz}) \]
<table>
<thead>
<tr>
<th>Name</th>
<th>Git SHA</th>
<th>kLOCs</th>
<th>Slots</th>
<th>Vars</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akka.NET</td>
<td>719335a1</td>
<td>240</td>
<td>51.3k</td>
<td>51.2k</td>
<td>Actor-based Concurrent &amp; Distributed Framework</td>
</tr>
<tr>
<td>AutoMapper</td>
<td>2ca7c2b5</td>
<td>46</td>
<td>3.7k</td>
<td>10.7k</td>
<td>Object-to-Object Mapping Library</td>
</tr>
<tr>
<td>BenchmarkDotNet</td>
<td>1670ca34</td>
<td>28</td>
<td>5.1k</td>
<td>6.1k</td>
<td>Benchmarking Library</td>
</tr>
<tr>
<td>BotBuilder</td>
<td>190117c3</td>
<td>44</td>
<td>6.4k</td>
<td>8.7k</td>
<td>SDK for Building Bots</td>
</tr>
<tr>
<td>choco</td>
<td>93985688</td>
<td>36</td>
<td>3.8k</td>
<td>5.2k</td>
<td>Windows Package Manager</td>
</tr>
<tr>
<td>commandline</td>
<td>09677b16</td>
<td>11</td>
<td>1.1k</td>
<td>2.3k</td>
<td>Command Line Parser</td>
</tr>
<tr>
<td>CommonMark.NETDev</td>
<td>f3d54530</td>
<td>14</td>
<td>2.6k</td>
<td>1.4k</td>
<td>Markdown Parser</td>
</tr>
<tr>
<td>Dapper</td>
<td>931c700d</td>
<td>18</td>
<td>3.3k</td>
<td>4.7k</td>
<td>Object Mapper Library</td>
</tr>
<tr>
<td>EntityFramework</td>
<td>fa0b7ec8</td>
<td>263</td>
<td>33.4k</td>
<td>39.3k</td>
<td>Object-Relational Mapper</td>
</tr>
<tr>
<td>Hangfire</td>
<td>ffc4912f</td>
<td>33</td>
<td>3.6k</td>
<td>6.1k</td>
<td>Background Job Processing Library</td>
</tr>
<tr>
<td>Humanizer</td>
<td>cc11a77e</td>
<td>27</td>
<td>2.4k</td>
<td>4.4k</td>
<td>String Manipulation and Formatting</td>
</tr>
<tr>
<td>Lean</td>
<td>f574bdf7</td>
<td>190</td>
<td>26.4k</td>
<td>28.3k</td>
<td>Algorithmic Trading Engine</td>
</tr>
<tr>
<td>Nancy</td>
<td>72e16f14</td>
<td>70</td>
<td>7.5k</td>
<td>15.7</td>
<td>HTTP Service Framework</td>
</tr>
<tr>
<td>Newtonsoft.Json</td>
<td>6057d9b8</td>
<td>123</td>
<td>14.9k</td>
<td>16.1k</td>
<td>JSON Library</td>
</tr>
<tr>
<td>Ninject</td>
<td>7006297f</td>
<td>13</td>
<td>0.7k</td>
<td>2.1k</td>
<td>Code Injection Library</td>
</tr>
<tr>
<td>NLog</td>
<td>643e326a</td>
<td>75</td>
<td>8.3k</td>
<td>11.0k</td>
<td>Logging Library</td>
</tr>
<tr>
<td>Opserver</td>
<td>51b032e7</td>
<td>24</td>
<td>3.7k</td>
<td>4.5k</td>
<td>Monitoring System</td>
</tr>
<tr>
<td>OptiKey</td>
<td>7d35c718</td>
<td>34</td>
<td>6.1k</td>
<td>3.9k</td>
<td>Assistive On-Screen Keyboard</td>
</tr>
<tr>
<td>Orleans</td>
<td>e0d6a150</td>
<td>300</td>
<td>30.7k</td>
<td>35.6k</td>
<td>Distributed Virtual Actor Model</td>
</tr>
<tr>
<td>Polly</td>
<td>0afdbc32</td>
<td>32</td>
<td>3.8k</td>
<td>9.1k</td>
<td>Resilience &amp; Transient Fault Handling Library</td>
</tr>
<tr>
<td>quartznet</td>
<td>b33e6f86</td>
<td>49</td>
<td>9.6k</td>
<td>9.8k</td>
<td>Scheduler</td>
</tr>
<tr>
<td>raven.dbDev</td>
<td>55230922</td>
<td>647</td>
<td>78.0k</td>
<td>82.7k</td>
<td>Document Database</td>
</tr>
<tr>
<td>RestSharp</td>
<td>70de357b</td>
<td>20</td>
<td>4.0k</td>
<td>4.5k</td>
<td>REST and HTTP API Client Library</td>
</tr>
<tr>
<td>Rx.NET</td>
<td>2d146fe5</td>
<td>180</td>
<td>14.0k</td>
<td>21.9k</td>
<td>Reactive Language Extensions</td>
</tr>
<tr>
<td>scripts</td>
<td>f3cc8bcb</td>
<td>18</td>
<td>2.7k</td>
<td>4.3k</td>
<td>C# Text Editor</td>
</tr>
<tr>
<td>ServiceStack</td>
<td>6d59da75</td>
<td>231</td>
<td>38.0k</td>
<td>46.2k</td>
<td>Web Framework</td>
</tr>
<tr>
<td>ShareX</td>
<td>718dd711</td>
<td>125</td>
<td>22.3k</td>
<td>18.1k</td>
<td>Sharing Application</td>
</tr>
<tr>
<td>SignalR</td>
<td>fa88089e</td>
<td>53</td>
<td>6.5k</td>
<td>10.5k</td>
<td>Push Notification Framework</td>
</tr>
<tr>
<td>Wox</td>
<td>cda66272</td>
<td>13</td>
<td>2.0k</td>
<td>2.1k</td>
<td>Application Launcher</td>
</tr>
</tbody>
</table>
Quantitative Results – Variable Misuse

<table>
<thead>
<tr>
<th>Accuracy (%)</th>
<th>BiGRU</th>
<th>BiGRU+Dataflow</th>
<th>GGNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seen Projects</td>
<td>50.0</td>
<td>73.7</td>
<td>85.5</td>
</tr>
</tbody>
</table>

Seen Projects: 24 F/OSS C# projects (2060 kLOC): Used for train and test

3.8 type-correct alternative variables per slot (median 3, $\sigma = 2.6$)
Quantitative Results – Variable Misuse

<table>
<thead>
<tr>
<th>Accuracy (%)</th>
<th>BiGRU</th>
<th>BiGRU+Dataflow</th>
<th>GGNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seen Projects</td>
<td>50.0</td>
<td>73.7</td>
<td>85.5</td>
</tr>
<tr>
<td>Unseen Projects</td>
<td>28.9</td>
<td>60.2</td>
<td>78.2</td>
</tr>
</tbody>
</table>

Seen Projects: 24 F/OSS C# projects (2060 kLOC): Used for train and test
Unseen Projects: 3 F/OSS C# projects (228 kLOC): Used only for test
3.8 type-correct alternative variables per slot (median 3, σ= 2.6)
Based on this repo's code patterns, did you intend to use 'newDocument' (confidence 92%) rather than 'existingDocument' (confidence 7%) here? Review is recommended by Research bot's Variable Misuse analysis.
Returns the sum of the positive numbers in an array

Fernandes et al. 2019 “Neural Structured Summarization”
Graph to Sequence Model
## Quantitative Results

<table>
<thead>
<tr>
<th>C# Documentation</th>
<th>F1</th>
<th>ROUGE-2</th>
<th>ROUGE-L</th>
</tr>
</thead>
<tbody>
<tr>
<td>biRNN -&gt; RNN</td>
<td>35.2</td>
<td>20.8</td>
<td>36.7</td>
</tr>
<tr>
<td>GNN -&gt; RNN</td>
<td>38.9</td>
<td>25.6</td>
<td>37.1</td>
</tr>
<tr>
<td>biRNN + GNN -&gt; RNN</td>
<td>45.4</td>
<td>28.3</td>
<td>41.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java Method Naming</th>
<th>F1</th>
<th>ROUGE-2</th>
<th>ROUGE-L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alon et al. *</td>
<td>43.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>biRNN -&gt; RNN</td>
<td>35.8</td>
<td>17.9</td>
<td>39.7</td>
</tr>
<tr>
<td>biRNN + GNN -&gt; RNN</td>
<td>44.7</td>
<td>21.1</td>
<td>43.1</td>
</tr>
</tbody>
</table>

Expression Completion Task

```csharp
int methParamCount = 0;
if (paramCount > 0) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInformations(Dummy.Signature, paramCount);
    methParamCount = moduleParamArr.Length;
}
if (false) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInformations(Dummy.Signature,
            paramCount - methParamCount);
}
```
Expression Completion Task

```
int methParamCount = 0;
if (paramCount > 0) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInfo(Dummy.Signature, paramCount);
    methParamCount = moduleParamArr.Length;
}
if (/* TODO */) {
    IParameterTypeInfo[] moduleParamArr =
        GetParamTypeInfo(Dummy.Signature,
                            paramCount - methParamCount);
}
```
Adding Edges During Generation

Variables in Scope

- \( i \)
- \( j \)
Neural Attribute Grammars (NAG)
Information Propagation on Graph Generation

Variables in Scope

Expression

Expression

Expression

-
Information Propagation on Graph Generation

Variables in Scope

Expression 1

Expression 2

Expression 3

Expression 1

Expression 2

Expression 2

Expression 3

T=1

T=2

T=3

T=4

InheritedToSynthesized

Child

Parent

NextSibling

NextToken

LastUse

LastToken
# Filling in the Blanks: Quantitative Results

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>Type-Correct</th>
<th>Match@1</th>
<th>Match @5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq → NAG</td>
<td>8.38</td>
<td>40.4</td>
<td>8.4</td>
<td>15.8</td>
</tr>
<tr>
<td>Graph → Tree</td>
<td>5.37</td>
<td>41.2</td>
<td>19.9</td>
<td>36.8</td>
</tr>
<tr>
<td>Graph → Syntax Networks</td>
<td>3.03</td>
<td>74.7</td>
<td>32.4</td>
<td>48.1</td>
</tr>
<tr>
<td>Graph → Sequentialised Tree</td>
<td>3.48</td>
<td>84.5</td>
<td>36.0</td>
<td>52.7</td>
</tr>
<tr>
<td>Graph → Neural Attr. Gram.</td>
<td>3.07</td>
<td>84.5</td>
<td>38.8</td>
<td>57.0</td>
</tr>
</tbody>
</table>

Training data: 479 C# projects from GitHub
Test data: 114 C# projects from GitHub (~100 000 samples)
Natural Language Processing

...with graph neural networks.
Gunshots were fired at rapper Lil Wayne’s tour bus early Sunday in Atlanta. No one was injured in the shooting, and no arrests have been made, Atlanta Police spokeswoman Elizabeth Espy said. Police are still looking for suspects. Officers were called to a parking lot in Atlanta’s Buckhead neighborhood, Espy said. They arrived at 3:25 a.m. and located two tour buses that had been shot multiple times. The drivers of the buses said the incident occurred on Interstate 285 near Interstate 75, Espy said. Witnesses provided a limited description of the two vehicles suspected to be involved: a “Corvette-style vehicle” and an SUV. Lil Wayne was in Atlanta for a performance at Compound nightclub Saturday night. CNN’s Carma Hassan contributed to this report.

Rapper Lil Wayne not injured after shots fired at his tour bus on an Atlanta interstate, police say. No one has been arrested in the shooting.
## Quantitative Results

<table>
<thead>
<tr>
<th>CNN/DailyMail</th>
<th>ROUGE-1</th>
<th>ROUGE-2</th>
<th>ROUGE-L</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN -&gt; RNN *</td>
<td>31.3</td>
<td>11.8</td>
<td>28.8</td>
</tr>
<tr>
<td>RNN + GNN -&gt; RNN</td>
<td>33.0</td>
<td>13.3</td>
<td>28.3</td>
</tr>
</tbody>
</table>

*See et al. (2015). Get To The Point: Summarization with Pointer-Generator Networks*
## Quantitative Results

<table>
<thead>
<tr>
<th>CNN/DailyMail</th>
<th>ROUGE-1</th>
<th>ROUGE-2</th>
<th>ROUGE-L</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN -&gt; RNN *</td>
<td>31.3</td>
<td>11.8</td>
<td>28.8</td>
</tr>
<tr>
<td>RNN + GNN -&gt; RNN</td>
<td>33.0</td>
<td>13.3</td>
<td>28.3</td>
</tr>
<tr>
<td>RNN -&gt; RNN + pointer *</td>
<td>36.4</td>
<td>15.7</td>
<td>33.4</td>
</tr>
<tr>
<td>RNN + GNN -&gt; RNN + pointer</td>
<td><strong>38.1</strong></td>
<td><strong>16.1</strong></td>
<td>33.2</td>
</tr>
</tbody>
</table>

*See et al. (2017). Get To The Point: Summarization with Pointer-Generator Networks*
NLP with GNNs

Grounded Language

Procedural Text

Semantic Parsing
Source Code & Natural Language

Rich Structure

Reasoning