Understanding Source Code with Deep Learning

Miltos Allamanis

FOSDEM 2019

miltos1

https://miltos.allamanis.com

Microsoft Research Cambridge
Source Code is Bimodal
Code Autocompletion

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

https://visualstudio.microsoft.com/services/intelllicode/
Predicting Types

Predicting Program Properties from Code
V. Raychev, M. Vechev, A. Krause. 2015
http://jsnice.org/

Deep Learning Type Inference
V. Hellendoorn, C. Bird, E.T. Barr, M. Allamanis. 2018

DeepTyper

let \( a = 1 \); let \( b = a + 1 \);
Inferring Type Refinements

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

**Conceptual Types**

"a password" → `string password;`

"a JSON string" → `string data = Json.Load();`

**Defined Types**

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

Defined explicitly by the programmer.
## 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>Duration</td>
<td>frequencyCapDuration</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>Builder</th>
<th>builder</th>
<th>builder</th>
<th>builder</th>
</tr>
</thead>
<tbody>
<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>a.getStartColumnIndex()</td>
<td>0</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”
Infer latent intent
Ambiguous information
Learned heuristics

https://ml4code.github.io

A Survey of Machine Learning for Big Code and Naturalness

MILTIADIS 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.
Detecting Variable Misuse Bugs

By representing source code as graphs
Target 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.
```csharp
var newBounds = new List<Rect>(bounds.Count);
foreach (var b in bounds)
{
    double x1 = b.Left - padding.Left;
    double x2 = b.Right + padding.Right;
    if (x1 < x2)
    {
        double y1 = b.TextTop - padding.Top;
        double y2 = b.TextBottom + padding.Bottom;
        newBounds.Add(new Rect(x1, y1, x2 - x1, y2 - y1));
    }
}
return newBounds;
}
```

// Set up the initial rectangle

```csharp
public static Rect GetInitialRect()
{
    if (rectangle == null)
        return new Rect(0, 0, 0, 0);
    return rectangle;
}
```
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);
(x, y) = Foo();
while (x > 0)
    x = x + y;
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
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);
```

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

\[(h_T^{SLOT})^T h_T^{first} \gg (h_T^{SLOT})^T h_T^{clazz}\]
Vector Space Representations

Local representation

Distributed representation
Graph Neural Networks

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

Graph Neural Networks: Message Passing

Current Node Representation $h_{t-1}^n$

Input Messages

Combining

Update

Next Representation
Graph Neural Networks: Message Passing

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

\[ h_t^n = \text{GRU}(h_{t-1}^n, x) \]

Graph Neural Networks: Unrolling
Graph Neural Networks: Unrolling

Graph Neural Networks: Unrolling

- node selection
- node classification
- graph classification

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

https://github.com/Microsoft/gated-graph-neural-network-samples
Quantitative Results – Variable Misuse

<table>
<thead>
<tr>
<th></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, σ= 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)
bool TryFindGlobalDirectivesFile(string baseDirectory, string fullPath, out string path) {
    baseDirectory = baseDirectory.TrimEnd(Path.DirectorySeparatorChar);
    var directivesDirectory = Path.GetDirectoryName(baseDirectory)
        .TrimEnd(Path.DirectorySeparatorChar);
    while (directivesDirectory != null && directivesDirectory.Length >= baseDirectory.Length) {
        path = Path.Combine(directivesDirectory, GlobalDirectivesFileName);
        if (File.Exists(path)) return true;

        directivesDirectory = Path.GetDirectoryName(directivesDirectory)
            .TrimEnd(Path.DirectorySeparatorChar);
    }
    path = null;
    return false;
}
Learning Signals

- Given dataset \( \{(x_1, y_0), \ldots, (x_N, y_N)\} \)
- Minimize Loss \( \mathcal{L}(\theta) = \frac{1}{N} \sum_i L(f_\theta(x_i), y_i) \)
Source Code is Bimodal

Programs as Graphs

```c
int sumPositive(int[] arr, int limit) {
    int sum = 0;
    for (int i = 0; i < limit; i++)
        if (arr[i] > 0)
            sum += arr[i];
    return sum;
}
```

~500 nodes/graph  ~8k edges/graph

Graph Neural Networks: Message Passing

\[ x = \sum_{t' \in \text{neigh}(x)} e_{(t', t)} h_{t', t} + b_t(t', t) \]

Research in ML + Code

- Infer latent intent
- Ambiguous information
- Learned heuristics

https://ml4code.github.io

~500 nodes/graph  ~8k edges/graph

Programs as Graphs

Graph Neural Networks: Message Passing

\[ x = \sum_{t' \in \text{neigh}(x)} e_{(t', t)} h_{t', t} + b_t(t', t) \]

[Li et al. (2019), Global graph sequence neural networks.

Microsoft

@miltos1

miltos.allamanis.com