Bimodal Program Analysis ... in the Wild

Marc Brockschmidt (and Miltos Allamanis and many more)
Target: Variable Names

• Variable names are means of communication between SDE ⇒ natural language!

• Variable names are summaries of program semantics ⇒ static analysis!
Related Work

- Allamanis et al.: n-gram & LBL models for method / class / variable naming

- JSNice (Raychev et al.): Conditional random fields for names / types
Modern Times: 2017

- Input: Source code (fragments) around variable
- Output: Name

- Idea: Cast as translating code to name, use neural NLP toolbox:
  - Deep sequence to sequence architecture
  - Attention
  - Copying for out-of-vocabulary
Modern Times are Complex

Input comes in many forms:
• Tokens around variable uses:
  
  ```
  var ??? = user.GetData();
  ```

• Types implemented by variable:

  ```
  InputStream ??? = FooBar();
  ```

• Formal parameters it's matched to:

  ```
  foo(???);
  ...
  void foo(int userNum) { ... }
  ```

⇒ Model needs to combine different information sources
Model: Encoder

Context RNN

Types RNN

ParName RNN

Aggregate
## Evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain LSTM Decoder</td>
<td>44%</td>
<td>38%</td>
<td>59%</td>
<td>50%</td>
</tr>
<tr>
<td>+ Attention</td>
<td>43%</td>
<td>42%</td>
<td>58%</td>
<td>55%</td>
</tr>
<tr>
<td>+ Copying</td>
<td>47%</td>
<td>44%</td>
<td>64%</td>
<td>55%</td>
</tr>
</tbody>
</table>

[on EntityFramework & RavenDB (medium/large C# OSS projects)]
Results – Group I: Be specific

```csharp
} catch (ErrorResponseException e)
{
    if (e.StatusCode == HttpStatusCode.NotFound)
    {
        var text = e.ResponseString;
        if (text.Contains("maxQueryString"))
            throw new ErrorResponseException(e, text);
    }
}

if (HandleException(responseException))
    return null;
```
Results – Group I: Be specific

```csharp
var documentStores = new Dictionary<string, IDocumentStore>
{
    {"one", new DocumentStore{Url = "http://localhost:8078"}},
    {"two", new DocumentStore{Url = "http://localhost:8077"}},
    {"tri", new DocumentStore{Url = "http://localhost:8076"}},
};

foreach (var documentStore in documentStores)
{
}

var shardStrategy = new ShardStrategy(documentStores);
```

<table>
<thead>
<tr>
<th>Model</th>
<th>Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suggestions</td>
<td>shards (75.71%), strategy</td>
</tr>
</tbody>
</table>
Results – Group II: Be descriptive

```csharp
if (fieldsToFetch.FetchAllStoredFields)
{
    var fields = new HashSet<string>(document.GetFields().Select(x => x.Name));
    fields.Remove(Constants.ReduceKeyFieldName);
    var documentFromFields = new RavenJObject();
    AddFieldsToDocument(document, fields, documentFromFields);
}
```

<table>
<thead>
<tr>
<th>Model</th>
<th>Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suggestions</td>
<td>fieldNames (64.59%)</td>
</tr>
</tbody>
</table>
Results – Group II: Be descriptive

```java
public BoundParameter Parameter(ParameterSymbol p)
{
    return new BoundParameter(Syntax, p, p.Type) { WasCompilerGenerated = true; };
}
```

<table>
<thead>
<tr>
<th>Model</th>
<th>Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Suggestions parameter (67.45%),</td>
</tr>
</tbody>
</table>

receiver, FieldSymbol f)

public
{...
Models? What are they good for?

Deep Learning models are awesome!
1. ... but need so much data!
2. ... and also need to learn project-specific customs!
3. ... and need a GPU for training (and during inference?)
4. ... inference too slow for live feedback anyway
5. ... so when do we show feedback to users?
Training in the Wild

Most projects (start) small
  ⇒ Pre-train on corpus of projects, adapt to local project

Technically:
• Leave parts of decoder vocabulary unused in pre-training
• Fill up with project-specific data
• Retrain all parameters on project data

Consequences:
• Pre-training is expensive (lots of data, many epochs required)
• Adaptation is cheap (little data, few epochs required)
Training and Inference in the Wild

Deep RNN models require training, are big, slow on CPU
⇒ Training and inference as a service

Technically:
• RESTful API, selecting model & providing variable contexts
• Server trains, stores and runs models (no deployment troubles!)
• Clients extract relevant data and do UI

Consequences:
• Deploying new models becomes easy
• Clients are thin, can be integrated into everything
UX in the Wild

Idea: Warn if certain that name should differ from used
Analysis requires some context, too slow/expensive for live warnings
⇒ Do in CI, PR or on request

Questions:
• How to use user feedback (action/no action)?
• How to avoid repeating warnings?
• How to decide how aggressively to show warnings?
Conclusions

The good:
• Neural NLP toolbox can work for source code as well
• Effective models possible, can catch interesting issues

The bad:
• Deep Learning models hard to deploy, relatively expensive to train
• Infrastructure needs to be re-invented all the time

The ugly:
• UX aspects not studied, research focused on headline numbers
• Trade-off between model complexity & performance unclear