Language-Agnostic Representation Learning of Source Code from Structure and Context

Daniel Zügner1, Tobias Kirschstein1, Michele Catasta2, Jure Leskovec2, Stephan Günnewann1

Task: Code Summarization

- Predict a method’s name given its body.
- A common task to evaluate ML models’ performance on code.
- In high-quality code bases, a method’s name summarizes its functionality.
- Labels come ‘for free’, and there is lots of high-quality open-source code.

Example

```python
def [MASK]():
    if lin ("true", "false", "1"):
        return True
    if lin ("false", "true", "0"): return False
raise Exception("Unable to convert string "+ s
```

Relative Distances on the AST

- Can be computed from the AST for any programming language.
- Capture the local and global structure in the AST.
- Language-agnostic design instead of specialized proprietary pipelines.
- Enables training on any programming language, even jointly (see results).
- Efficient binning enables using continuous-valued distances (e.g., PPR).

Code Transformer

**Structure and Context Representation**

Structure and Context are complementary representations of a program.

- Source code as sequence of tokens (referred to as Context)
- Source code as Abstract Syntax Tree (AST, referred to as Structure)

Typical models: LSTMs, Transformers

- For instance, get_model() is far away from the return statement in the Context; it is hard for a model to capture such long-range dependencies.
- In the AST, the two corresponding nodes are only a few hops apart.
- Many previous works only learn from either Structure or Context.
- We propose the Code Transformer, which learns jointly from both.

Ablation study

- Can be computed from the AST for any programming language.
- Capture the local and global structure in the AST.
- Language-agnostic design instead of specialized proprietary pipelines.
- Enables training on any programming language, even jointly (see results).
- Efficient binning enables using continuous-valued distances (e.g., PPR).

Multilanguage embeddings

- We obtain a shared embedding space for multiple programming languages.
- Similar methods tend to be mapped to regions close-by in embedding space.
- We can ‘measure’ similarity of programming languages via their embeddings.

Results

**Table: Code Summarization Results on the CodeSearchNet dataset (F1 score).**

<table>
<thead>
<tr>
<th>Model</th>
<th>Python</th>
<th>Javascript</th>
<th>Ruby</th>
<th>Go</th>
</tr>
</thead>
</table>
| code2seq               | 29.3   | 24.0       | 14.3 | 47.5
| GREAT                  | 33.2   | 28.9       | 23.4 | 48.2|
| Code Transformer       | 35.0   | 32.1       | 27.5 | 51.3|
| code2seq*              | 29.3   | 26.1       | 19.9 | 48.2|
| GREAT*                 | 33.9   | 30.4       | 26.9 | 50.4|
| Mult. + Pretrain       | 36.2   | 33.2       | 31.2 | 53.0|
| Mult. + Pretrain*      | 37.4   | 34.3       | 32.0 | 54.7|

*Adapted by us for multilanguage training

- Our Code Transformer outperforms all baselines in all scenarios.
- Multilanguage training improves results on each individual language.
- This is true for all studied models using the Structure, opening exciting directions for future research. No improvement for Context-only models.
- Strongest gains are on Ruby, the language with fewest training samples.

Multilanguage embeddings

- We obtain a shared embedding space for multiple programming languages.
- Similar methods tend to be mapped to regions close-by in embedding space.
- We can ‘measure’ similarity of programming languages via their embeddings.

Figure: t-SNE visualization of multilanguage embeddings