Mila

Google Research

On-the-Fly Adaptation of Source Code Models

Disha Shrivastava*, Hugo Larochelle, Daniel Tarlow

Scan for Paper

Université de Montréal

McGill

Introduction

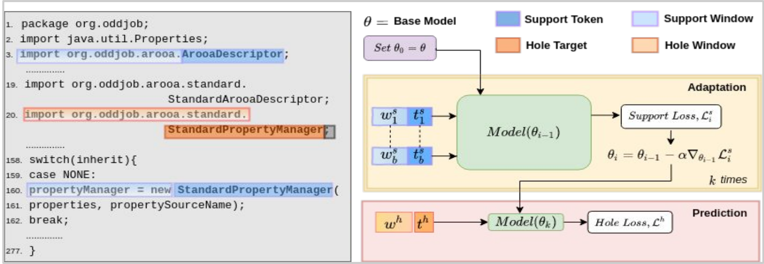
Why Adaptation for Test File?

- **Lots of unseen identifiers.** In the Java GitHub corpus test set, for each project, there is on an average 56.49 original identifiers (not seen in the training set) introduced every thousand lines of code [1]
- **Organization or project-specific coding conventions.** Variable naming conventions (*get_access* vs. getAccess), Data structures/ libraries used (*from google3 import b*)
- **Developer-specific coding preferences.** *for (int i = 0, ...)* vs *for (int j = 0, ...)* , Comments before line or method

Task: Line-level Maintenance

- Blank-out portions of the line following a random cursor in a file and predict the next token (hole target) following the cursor (simulates autocompletion in an IDE)
- Closer to the workflow of a developer compared to a language model (simulates an in-progress edit)

Methodology



Targeted Support Set Adaptation (TSSA-k)

- Targeted Selection of Support tokens (e.g. rare tokens) from anywhere in the file except the blanked-out range.
- Starting from base model parameters θ, perform k steps of adaptation in the inner loop predicting support tokens from support windows and use the adapted parameters to predict the hole target from hole window.

*Corresponding author: dishu.905@gmail.com

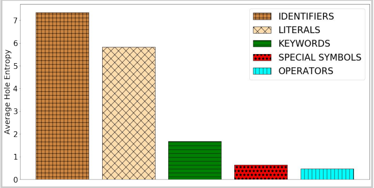
Experiments and Results

Test Performance on Hole Target Prediction

Model	Cross Entropy	MRR@10 (All)(%)	MRR@10 (Identifiers)(%)	Recall@10 (All)(%)	Recall@10 (Identifiers) (%)
Base Model	5.222 ± 0.10	65.20 ± 0.42	24.90 ± 0.64	75.74 ± 0.42	36.20 ± 0.78
Dynamic Evaluation	3.540 ± 0.08	$\mathbf{68.95 \pm 0.41}$	34.44 ± 0.70	80.39 ± 0.39	48.86 ± 0.82
TSSA-1	3.461 ± 0.07	66.94 ± 0.40	35.76 ± 0.70	81.00 ± 0.38	52.04 ± 0.82
TSSA-8	3.383 ± 0.06	67.52 ± 0.40	35.14 ± 0.70	80.65 ± 0.38	50.27 ± 0.82
TSSA-16	$\mathbf{3.240 \pm 0.06}$	68.63 ± 0.40	$\mathbf{36.74 \pm 0.70}$	$\mathbf{81.51 \pm 0.38}$	$\mathbf{52.34 \pm 0.82}$

- **Dataset:** Large-scale Java Github Corpus [1] consisting of 14000 open-source Java projects.
- **Preprocessing:** java-lexing followed by subword tokenization.
- **Model:** seq2seq model with single layer of GRU with 512 hidden dimensions
- **Methods:**
  - Base Model: no adaptation, i.e, directly use θ
  - TSSA-k: adaptation with TSSA with k updates in the inner loop
  - Dynamic Evaluation: Support tokens from context before the hole target [2]
- Set k = avg. # of updates performed by dynamic evaluation = 16 for our test data.

Test Performance across different token-types



Token Type	Base model	TSSA-16	% Improvement
Identifiers	13.16	7.35	44.15
Literals	7.18	5.82	18.94

Conclusions

- TSSA outperforms all baselines including a comparable form of dynamic evaluation, even with half the number of adaptation steps (TSSA-8) or even one step of adaptation (TSSA-1). The latter part is important for downstream autocompletion tasks where latency is critical.
- We improve performance on identifiers and literals by 44% and 19%, respectively as compared to a non-adaptive baseline which results in better performance overall.

References:
[1] Allamanis, M. and Sutton, C. Mining source code repositories at massive scale using language modeling. In Proceedings of the 10th Working Conference on Mining Software Repositories pp. 207–216. IEEE Press, 2013
[2] Karampatsis, R.-M., Babii, H., Robbes, R., Sutton, C., and Janes, A. Big code!= big vocabulary: Open-vocabulary models for source code. arXiv preprint arXiv:2003.07914, 2020