ISSN (e): 2250 – 3005 || Volume, 14 || Issue, 4|| Jul. - Aug. – 2024 ||
International Journal of Computational Engineering Research (IJCER)

Software Engineering Training in Several Languages

Biswajit Tripathy[1] Subhadra Biswal[2] Snehalata Parida[3]
[1,2,3] Dept. of CSE, Einstein Academy of Technology and Management, Bhubaneswar

Abstract
Nowadays, a lot of software engineering activities may be automated with the help of well-trained machine-learning models that use a lot of data from open-source software. This strategy has been used to a number of SE problems, with performance progressively improving over the past few years thanks to improved models and training techniques. Training benefits from an increasing amount of diverse, well-labelled, and clean data; nevertheless, creating high-quality datasets is difficult and time-consuming. Increasing the amount and variety of clean, tagged data can be applied in a broad range of situations. Labelled data may be scarcer for certain languages (like Ruby) and more concentrated in certain application domains (like JavaScript) for other languages.

I. INTRODUCTION

Researchers in the NLP area have reported that multilingual training is beneficial for low-resource language [16, 23]. Several papers show that multilingual-trained models show better performance and are more practical to deploy [9]. However, this is observed in two situations: 1) for low-resource languages and 2) when the languages are related. We find that programs in different languages solving the same problem use more similar identifiers; furthermore, different languages sometimes have similar keywords and operators. High-capacity deep learning models are capable of learning inter lingual shared semantic representation between languages. Moreover, with tasks like summarization, or method naming, we are dealing with a simplified, many-to-one setting: translating multiple source languages to a single target language), which is believed to be easier than multi-way task [20]. We begin by introducing the code summarization task, which we use to motivate multilingual training.

Developers often rely heavily on comments, to gain a quick (even if approximate) understanding of the specification and design of code they are working on. An actual example of a comment is shown in Figure 1. Such comments help a developer gain a quick mental preview of what the proximate code does, and how it might go about it; this helps the developer know what to look for in the code. Knowing that such comments are useful to others (or even later to oneself) incentivizes developers to create comments that explain the code; however, the resulting redundancy (viz., code that does something, and some nearby English text that describes just what the code does), with the same concept expressed in two languages results in a bit of extra work for the original coder. This extra work, of creating aligned comments explaining the code, can be fruitfully viewed [21] as a task related to natural language translation (NLT) (e.g., translating English to German). The mature & powerful technology of NLT becomes applicable for comment synthesis; ML approaches developed for the former can be used for the latter. An effective comment synthesizer could help developers: by saving them the trouble of writing comments; and perhaps even be used on-demand in the IDE to create descriptions of selected bits of code.

II. BACKGROUND & MOTIVATION

We now present some motivating evidence suggesting the value of multilingual training data for deep-learning applications to software tasks. We begin the argument focused on code summarization. Deep learning models have been widely applied to code summarization, with papers reporting substantial gains in performance over recent years [1, 2, 7, 18, 19]. We focus here on what information in the code ML models leverage for summarization (while we use summarization to motivate the approach, we evaluate later on 3 different tasks). Does every token in the program under consideration matter, for the code summarization task? Or, are the function and variable names used in the programs most important? Since identifiers carry much information about the program, this may be a reasonable assumption. Considering the content words2 in the example in Figure 1 there are four major terms (i.e., Returns, text content, node, and descendants) used in the summary. The first 3 directly occur as tokens or sub tokens in the code. Though the word “descendants" is missing in the program, high-capacity neural models like BERT [17] can learn to statistically connect, e.g., "descendant" with the identifier sub token “child”. This suggests that, perhaps, comments are recoverable primarily from identifiers. If this is so, and identifiers matter more for comments than the exact syntax of the programming language, that may actually be very good news indeed. If developers choose identifiers in the same way across different languages (viz., problem-dependent, rather than language dependent) perhaps we can improve the diversity and quality of dataset by pooling training set across may languages. Pooled data sets may allow us to finetune using multilingual data, and improve performance, especially for low-resource languages (e.g., Ruby and JavaScript from CodeXGLUE). Since this is a core theoretical background for work, we start off with two basic research questions to empirically gauge the possibility and promise of multilingual fine-tuning.

The Models

For our study of multilingual training, we adopt the BERT, or “foundation model” paradigm. Foundation models [13, 15, 17] have two stages: i) unsupervised pre-training with corpora at vast scale and ii) fine-tuning with a smaller volume of supervised data for the actual task. Foundation models currently hold state-of-the-art performance for a great many NLP tasks. BERT [17] style models have also been adapted for code, pre-trained on a huge, multilingual, corpora, and made available: CodeBERT and GraphCodeBERT are both freely available: both source code and pre-trained model parameters. While these models for code have thus far generally been fine-tuned monolingually, they provide an excellent platform for training experiments like ours, to measure the gains of multilingual fine-tuning. CodeBERT&GraphCodeBERT use a multi-layer bidirectional Transformer-based architecture, and it is exactly as same as the RoBERTa , with 125M parameters; we explain them further below. Pre-training The CodeBERT [18] dataset, has two parts: a matchedpairs part with 2.1M pairs of function and associated comment (NLPL pairs) and 6.4M samples with just code. The code includes several programming languages. It was created by Hussain et al. . CodeBERT model is pre-trained with two objectives (i.e., Masked Language Modeling and Replaced Token Detection) on both parts.

III. CONCLUSION

We began this paper with three synergistic observations: First, when solving the same problem, even in different programming languages, programmers are more likely to use similar identifiers (than when solving different problems). Second, identifiers appear to be relatively much more important than syntax markers when training machine-learning models to perform code summarization. Third, we find that quite often a model trained in one programming language achieves surprisingly good performance on a test set in a different language, sometimes even surpassing a model trained on the same language as the test set! Taken together, these findings suggest that pooling data across languages, thus creating multilingual training sets, could improve performance for any language, particularly perhaps languages with limited resources, as has been found in Natural-language processing [16, 23]. We test this theory, using two BERT-style models, Code BERT, and Graph Code BERT, with encouraging results. Foundation models [12] are currently achieving best-in-class performance for a wide range of tasks in both natural language and code. The models work in 2 stages, first “pre-training” to learn statistics of language (or code) construction from very large-scale corpora in a self-supervised fashion, and then using smaller labelled datasets to “fine-tune” for specific tasks. We adopt the multilingual Code XGLUE dataset, and the pre-trained Code BERT and Graph Code BERT models, and study the value of multilingual fine-tuning for a variety of tasks. We find evidence suggesting that multilingual fine-tuning is broadly beneficial in many settings. Our findings suggest that multilingual training could provide added value in broad set of settings, and merits further study. Acknowledgements: This material is based upon work supported by the U.S. National Science Foundation under Grant Nos. 1414172, and 2107592. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Ahmed was also supported by UC Davis College of Engineering Dean’s Distinguished Fellowship.

REFERENCES

[1]. Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, Online, 2655–2668. https://www.aclweb.org/anthology/2021.naaclmain.211

[2]. Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A Transformer-based Approach for Source Code Summarization. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL).

[3]. Toufique Ahmed, Premkumar Devanbu, and Anand Ashok Sawant. 2021. Learning to Find Usage of Library Functions in Optimized Binaries. IEEE Transactions on Software Engineering (2021). https://doi.org/10.1109/TSE.2021.3106572

[4]. Toufique Ahmed, Noah Rose Ledesma, and Premkumar Devanbu. 2021. SYNFIX: Automatically Fixing Syntax Errors using Compiler Diagnostics. arXiv:2104.14671 [cs.SE]

[5]. MiltiadisAllamanis. 2019. The adverse effects of code duplication in machine learning models of code. In Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. 143–153.

[6]. MiltiadisAllamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In International conference on machine learning. PMLR, 2091–2100.

[7]. Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2019. code2seq: Generating Sequences from Structured Representations of Code. In International Conference on Learning Representations. https://openreview.net/forum?id=H1gKYo09tX

[8]. Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. Proceedings of the ACM on Programming Languages 3, POPL (2019), 1–29.

[9]. Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, et al. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. arXiv preprint arXiv:1907.05019 (2019).

[10]. Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization. 65–72.

[11]. Eeshita Biswas, Mehmet Efruz Karabulut, Lori Pollock, and K Vijay-Shanker. 2020. Achieving Reliable Sentiment Analysis in the Software Engineering Domain using BERT. In 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 162–173.

[12]. Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the Opportunities and Risks of Foundation Models. arXiv preprint arXiv:2108.07258 (2021).

[13]. Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020).

[14]. Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. arXiv preprint arXiv:2003.10555 (2020).

[15]. Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. Advances in Neural Information Processing Systems 32 (2019), 7059– 7069.

[16]. Raj Dabre, Chenhui Chu, and Anoop Kunchukuttan. 2020. A survey of multilingual neural machine translation. ACM Computing Surveys (CSUR) 53, 5 (2020), 1–38.

[17]. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).

[18]. Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT: A PreTrained Model for Programming and Natural Languages. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings. 1536–1547.

[19]. Shuzheng Gao, Cuiyun Gao, Yulan He, Jichuan Zeng, Lun Yiu Nie, and Xin Xia. 2021. Code Structure Guided Transformer for Source Code Summarization. arXiv preprint arXiv:2104.09340 (2021).

[20]. Ekaterina Garmash and Christof Monz. 2016. Ensemble learning for multi-source neural machine translation. In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers. 1409–1418.

[21]. David Gros, Hariharan Sezhiyan, Prem Devanbu, and Zhou Yu. 2020. Code to Comment?Translation?: Data, Metrics, Baselining & Evaluation. In 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 746–757.

[22]. Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, LIU Shujie, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. GraphCodeBERT: Pre-training Code Representations with Data Flow. In International Conference on Learning Representations.

[23]. Thanh-Le Ha, Jan Niehues, and Alexander Waibel. 2016. Toward multilingual neural machine translation with universal encoder and decoder. arXiv preprint arXiv:1611.04798 (2016).