Guided Code Synthesis using Deep Neural Networks

Carol V. Alexandru
Software Evolution and Architecture Lab
University of Zurich, Switzerland
alexandru@ifi.uzh.ch

ABSTRACT

Can we teach computers how to program? Recent advances in neural network research reveal that certain neural networks are able not only to learn the syntax, grammar and semantics of arbitrary character sequences, but also synthesize new samples 'in the style of' the original training data. We explore the adaptation of these techniques to code classification, comprehension and completion.

CCS Concepts

•**Software and its engineering** → **Code generation;**

Keywords

Deep Learning, Code Classification, Code Synthesis

1. THE CHALLENGE

Although writing software is a creative process, wherein parts of an implementation may always require a 'human touch', repetition is commonplace in software development. Tasks such as reading a CSV file or comparing two dates are too uncommon for memorization and too unwieldy for traditional auto-completion or templating, yet developers can easily find existing solutions online: on question answering sites such as StackOverflow, in forums and mailing list discussions, or in existing code on platforms such as GitHub. Humans can manually navigate these resources to seek help, but beyond search-based tools and platform-specific systematic approaches, *e.g.*, displaying StackOverflow discussions in an IDE [7], we currently lack effective means to leverage the bulk knowledge contained in these resources.

Recent neural network research has yielded models that can learn not only to recognize and classify noisy data, *e.g.*, objects in an image or natural language in an audio signal, but also generate new, synthetic samples that resemble real-world data [4, 5, 13]. Sutskever *et al.* [10] demonstrate that a deep *recurrent neural network* (RNN) trained on 1GB of text from Wikipedia is able to synthesize a stream of text which contains few grammatical errors, comprises semantically coherent sentences and paragraphs and correctly models long-running relationships (like opening and closing braces). The model views data at the character level and has no explicit knowledge of grammar or vocabulary, both of which it learns purely by example. The same model can be trained on source code with similar success: learning solely from raw text, it internalizes the grammar and typical structure of a programming language, gaining the ability to generate new samples indistinguishable from 'real' code at first glance.

Unfortunately, this simple model is of limited practical use, as the programs it produces are more or less random at a higher semantic level. *We hypothesize that by improving the initial data extraction, augmenting the model with contextual information and post-processing the synthetic code, this promising approach can be made useful.* As a practical manifestation, we propose the idea of Guided Code Synthesis, an interactive process by which the developer guides a machine-trained model to produce the desired source code without manually accessing resources outside of the IDE. Hence, we formulate our primary research question:

RQ* How can character-level deep neural networks aid the synthesis of source code for practical applications?

2. PROPOSED RESEARCH

In traditional code completion, code fragments are either given one at a time, *e.g.*, providing potential method names after writing a dot, or as templates, *e.g.*, providing a skeleton `for` loop to be filled in by the developer. We envision an approach where the developer can view, cycle through, and modify large chunks of likely completions depending not only on preceding code but also on other contextual information such as keywords from comments, tags, and manually provided search terms. If powered by a rich model and implemented in a responsive, easy-to-use fashion, this approach would allow the developer to 'home in' on the right solution without having to resort to browsing online artifacts, as the model can provide multiple likely, customized completions.

To train such a model, several questions need to be addressed:

RQ1 How can we identify, classify and parse relevant code examples from noisy web resources (*e.g.*, StackOverflow) to build a canonicalized training set?

RQ2 What is a suitable neural network architecture to learn from both source code and contextual data to generate code samples of high quality and relevancy?

RQ3 How should the developer interact with the model while developing in an IDE?

FSE'16, November 13–18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4218-6/16/11...
http://dx.doi.org/10.1145/2950290.2983951

Noisy code classification and parsing.

To train our model on code from mixed-content sources such as StackOverflow, we need the ability to (i) isolate snippets from other content, (ii) classify snippets by their language and possibly their usage context, and (iii) parse snippets, even if they are noisy or incomplete. Regrettably, `code` tags are used liberally on StackOverflow and tags are generally unreliable. For example, a `code` block in a post tagged with 'Java' may contain a segment of a `pom.xml` or simply a stack trace. And while a content type is easily determined given a complete file, classifying a short snippet is hard without using complex heuristics. Furthermore, actual code snippets are predominantly incomplete programs and may contain noise such as ellipses or pseudocode placeholders. Traditional grammar-based parser are generally unable to parse isolated snippets, especially if they contain noise. RNN have successfully been used for text classification [14] and RNN sequence-to-sequence translation [11] should allow us to parse even noisy character sequences into partial abstract syntax trees (ASTs). The snippet→type and snippet→AST pairs needed for training can be obtained by parsing complete sources from GitHub using regular parsers and then extracting snippets and their corresponding AST segments afterwards. To make the parser more robust, synthetic noise can be added to the snippets used in training.

Neural network architectures for code synthesis.

Using a mixture of experts architecture of bi-directional RNNs [5] is a plausible candidate for our model. Contextual information (such as keywords provided by the user or gathered from comments and the IDE itself) could be used to give a strong bias towards selecting an expert. However, determining the right model is subject to research and experimentation. It has been shown that model averaging generally improves predictions [6, 12], so in the end we may combine different architectures at the output stage.

IDE integration.

In order to evaluate our approach in a practical setting, we intend to integrate it into an IDE. The plugin will utilize a pre-trained model, but it may learn from local code as well, possibly with a higher bias. To generate samples, the model will be given a number of previous characters at the current cursor position and contextual information as required. We envision that the generated code fragments will vary in length depending on the confidence of the model, and that it should be possible to cycle through multiple candidate completions. Furthermore, it may be useful to post-process the completions to check for sanity and to replace important elements, *e.g.*, variable names, on the fly.

The ultimate goal is a code completion system that internalizes the vast knowledge of internet resources such as StackOverflow. It should be able to continuously offer likely multi-token predictions while the developer is writing code, thus reducing context switches and increasing productivity.

3. EVALUATION

The performance of the snippet classifier and parser can be expressed in basic statistical terms. The abilities of our neural network architecture for producing source code and the suitability of the IDE integration can be measured on two levels: (i) by comparing the model's ability to predict the next n characters compared to other approaches like [3, 1], as done by White *et al.* [12], and (ii) by performing a three-way user study comparing the time needed by developers to solve certain programming tasks using our tool, using other auto-completion tools, and using web resources only.

4. EXPECTED CONTRIBUTIONS

Our research makes the following contributions:

- A tool for isolating and classifying snippets by their language and possibly their usage context.
- A sequence-to-sequence learner for parsing noisy snippets into partial ASTs.
- A neural network architecture learning from code and context to predict code fragments.
- An IDE plugin utilizing our model to provide multi-token auto-completion capabilities.

5. RELATED WORK

White *et al.* found that recurrent neural networks significantly outperform n-grams for doing code suggestions [12]. They reiterate that given the amount of unstructured data available to software engineering researchers, state-of-the-art approaches such as recurrent neural networks can, even in simple configurations, outperform existing solutions for common software engineering problems.

Raychev *et al.* extract method calls from large codebases to do a limited form of auto-completion for API usage, where gaps in existing source code are filled with appropriate tokens [9]. They use a combination of n-grams, which exceed at capturing short-term relationships, and an RNN, which is better at taking into account long-term relationships. They also made JSNice, a JavaScript deobfuscator which uses conditional random fields to give joint predictions of program properties [8]. It can rename obfuscated variables, annotate their types, and determine the return types of functions. The tool piqued interest among developers, hitting 30'000 downloads one week after its initial release.

Corley *et al.* use document vectors (DVs), another specialized instance of deep neural networks that aims to strengthen the relationship between a token and its context, for the purpose of feature location [2]. In a comparison, the DVs outperform classical LDA based approaches and as an added advantage, DVs can be trained much more quickly.

6. PROGRESS AND OUTLOOK

We applied the approach by Sutskever *et al.*, training an RNN on 10MB of Java source code to generate synthetic samples. Even though the dataset is tiny, the generated samples convincingly resemble real Java code. This supports our strategy of training multiple expert models, since each one needs fairly little training data. We also collected 7GB worth of code from 825 GitHub projects, used cloc[1] to detect the content type for every file, extracted 3000 snippets randomly for each of 24 programming languages and trained an RNN to detect the correct type with 91% accuracy. Moving on, we will build the training set for the neural parser and start experimenting with different neural network architectures for context-sensitive code completion.

[1]https://github.com/AlDanial/cloc

7. REFERENCES

[1] M. Bruch, M. Monperrus, and M. Mezini. Learning from examples to improve code completion systems. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 213–222, New York, NY, USA, 2009. ACM.

[2] C. S. Corley, K. Damevski, and N. A. Kraft. Exploring the use of deep learning for feature location. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 556–560, Sept 2015.

[3] C. Franks, Z. Tu, P. Devanbu, and V. Hellendoorn. Cacheca: A cache language model based code suggestion tool. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 705–708, May 2015.

[4] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. *CoRR*, abs/1505.07376, 2015.

[5] A. Graves, N. Jaitly, and A. r. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278, Dec 2013.

[6] H. K. H. Lee. Model selection for neural network classification. *Journal of Classification*, 18:227–243, 2001.

[7] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 102–111, New York, NY, USA, 2014. ACM.

[8] V. Raychev, M. Vechev, and A. Krause. Predicting program properties from "big code". In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 111–124, New York, NY, USA, 2015. ACM.

[9] V. Raychev, M. Vechev, and E. Yahav. Code completion with statistical language models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 419–428, New York, NY, USA, 2014. ACM.

[10] I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1017–1024, New York, NY, USA, June 2011. ACM.

[11] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[12] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk. Toward deep learning software repositories. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 334–345, Piscataway, NJ, USA, 2015. IEEE Press.

[13] H. Zen, A. Senior, and M. Schuster. Statistical parametric speech synthesis using deep neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 7962–7966, 2013.

[14] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.