A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes

Pablo Loyola, Edison Marrese-Taylor and Yutaka Matsuo
Graduate School of Engineering
The University of Tokyo
Tokyo, Japan
{pablo,emarrese,matsuo}@weblab.t.u-tokyo.ac.jp

Abstract

We propose a model to automatically describe changes introduced in the source code of a program using natural language. Our method receives as input a set of code commits, which contains both the modifications and message introduced by an user. These two modalities are used to train an encoder-decoder architecture. We evaluated our approach on twelve real world open source projects from four different programming languages. Quantitative and qualitative results showed that the proposed approach can generate feasible and semantically sound descriptions not only in standard in-project settings, but also in a cross-project setting.

1 Introduction

Source code, while conceived as a set of structured and sequential instructions, inherently reflects human intent: it encodes the way we command a machine to perform a task. In that sense, it is expected that it follows to some extent the same distributional regularities that a proper natural language manifests (Hindle et al., 2012). Moreover, the unambiguous nature of source code, comprised in plain and human-readable format, allows an indirect way of communication between developers, a phenomenon boosted in recent years given the current software development paradigm, where billions of lines code are written in a distributed and asynchronous way (Gousios et al., 2014).

The scale and complexity of software systems these days has naturally led to explore automated ways to support developers' code comprehension (Letovsky, 1987) from a linguistic perspective. One of these attempts is automatic summarization, which aims to generate a compact representation of the source code in a portion of natural language (Haiduc et al., 2010).

While existing code summarization methods are able to provide relevant insights about the purpose and functional features of the code, their scope is inherently static. In contrast, software development can be seen as a sequence of incremental changes, intended to either generate a new functionality or to repair an existing one. Source code changes are critical for understanding program evolution, which motivated us to explore if it is possible to extend the notion of summarization to encode code changes into natural language representations, i.e., develop a model able to *explain* a source code level modification. With this, we envision a tool for developers that is able to *i)* ease the comprehension of the dynamics of the system, which could be useful for debugging and repairing purposes and *ii)* automate the documentation of source code changes.

To this end, we rely on the concept of code commit, the standard contribution procedure implemented in modern subversion systems (Gousios et al., 2014), which provides both the actual change and a short explanatory paragraph. Our model consists of an encoder-decoder architecture which is trained on a set of triples conformed by the version of a system before and after the change, along with the comment. Given the high heterogeneity of the modalities involved, we rely on an attention mechanism to efficiently learn the parts of the sequences that are more expressive and have more explanatory power.

We performed an empirical study on twelve real world software systems, from which we obtained the commit activity to evaluate our model. Our experiments explored in-project and cross-project scenarios, and our results showed that the proposed model is able to generate semantically sound descriptions.

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 287–292
Vancouver, Canada, July 30 - August 4, 2017. ©2017 Association for Computational Linguistics
https://doi.org/10.18653/v1/P17-2045

2 Related Work

The use natural language processing to support software engineering tasks has increased consistently over the years, mainly in terms of source code search, traceability and program feature location (Panichella et al., 2013; Asuncion et al., 2010).

The emergence of unifying paradigms that explicitly relate programming and natural languages in distributional terms (Hindle et al., 2012) and the availability of large corpus mainly from open source software opened the door for the use of language modeling for several tasks (Raychev et al., 2015). Examples of this are approaches for learning program representations (Mou et al., 2016), bug localization (Huo et al.), API suggestion (Gu et al., 2016) and code completion (Raychev et al., 2014).

Source code summarization has received special attention, ranging from the use of information retrieval techniques to the addition of physiological features such as eye tracking (Rodeghero et al., 2014). In recent years several representation learning approaches have been proposed, such as (Allamanis et al., 2016), where the authors employ a convolutional architecture embedded inside an attention mechanism to learn an efficient mapping between source code tokens and natural language keywords.

More recently, (Iyer et al., 2016) proposed a encoder-decoder model that learns to summarize from Stackoverflow data, which contains snippet of code along with descriptions. Both approaches share the use of attention mechanisms (Bahdanau et al., 2014) to overcome the natural disparity between the modalities when finding relevant token alignments. Although we also use an attention mechanism, we differ from them in the sense we are targeting the changes in the code rather than the description of a file.

In terms of specifically working on code change summarization, Cortés-Coy et al. (2014); Linares-Vásquez et al. (2015) propose a method based on a set of rules that considers the type and impact of the changes, and (Buse and Weimer, 2010) combines summarization with symbolic execution. To the best of our knowledge, our approach represents the first attempt to generate natural language descriptions from code changes without the use of hand-crafted features, a desirable setting given the heterogeneity of the data involved.

3 Proposed Model

Our model assumes the existence of T versions of a given project $\{v_1, \ldots, v_T\}$. Given a pair of consecutive versions (v_{t-1}, v_t), we define the tuple (C_t, N_t), where $C_t = \Delta_{t-1}^{t}(v)$ represents a code snippet associated to changes over v in time t and N_t represents its corresponding natural language (NL) description. Let $\mathbb{C}$ be the set of all source code snippets and $\mathbb{N}$ be the set of all descriptions in NL. We consider a training corpus with T code snippets and summary pairs (C_t, N_t), $1 \leq t \leq T$, $C_t \in \mathbb{C}$, $N_t \in \mathbb{N}$. Then, for a given code snippet $C_k \in \mathbb{C}$, the goal of our model is to produce the most likely NL description $N^{\star}$.

Concretely, similarly to (Iyer et al., 2016), we use an attention-augmented encoder-decoder architecture. The encoder can be seen as a lookup layer, which simply reads through the source input sequence and returns the embedded tokens. The decoder is a RNN that reads this representation and generates NL words one at a time based on its current hidden state and guided by a global attention model (Luong et al., 2015). We model the probability of a description as a product of the conditional next-word probabilities. More formally, for each NL token $n_i \in N_t$ we define,

$$h_i = f(n_{i-1}E, h_{i-1}) \quad (1)$$

$$p(n_i|n_1, ..., n_{i-1}) \propto W \tanh(W_1 h_i + W_2 a_i) \quad (2)$$

where E is the embedding matrix for NL tokens, $\propto$ denotes a softmax operation, h_i represents the hidden state and a_i is the contribution from the attention model on the source code. W, W_1 and W_2 are trainable combination matrices. The decoder repeats the recurrence until a fixed number of words or a special *END* token is generated. The attention contribution a_i is defined as $a_i = \sum_{j=1}^{k} \alpha_{i,j} \cdot c_j F$, where $c_j \in C_t$ is a source code token, F is the source code token embedding matrix and $\alpha_{i,j}$ is:

$$\alpha_{i,j} = \frac{\exp\left(h_i^{\top} c_j F\right)}{\sum_{c_j \in C_t} \exp\left(h_i^{\top} c_j F\right)} \quad (3)$$

We use a dropout-regularized LSTM cell for the decoder (Zaremba et al., 2015) and also add dropout at the NL embeddings and at the output softmax layer, to prevent over-fitting. We added special *START* and *END* tokens to our training sequences and replaced all tokens and output words occurring less than 2 and 3 times, respectively,

with a special *UNK* token. We set the maximum code and NL length to be 100 tokens. For decoding, we approximate $N^{\star}$ by performing a beam search on the space of all possible summaries using the model output, with a beam size of 10 and a maximum summary length of 20 words.

To evaluate the quality of our generated descriptions we use both METEOR (Lavie and Agarwal, 2007) and sentence level BLEU-4 (Papineni et al., 2002). Since the training objective does not directly optimize for these scores, we compute METEOR on our validation set after every epoch and save the intermediate model that gives the maximum score as the final model. For evaluation on our test set we used the BLEU-4 score.

4 Empirical Study

Data and pre-processing: We captured historical data from twelve open source projects hosted on Github based on their popularity and maturity, selecting 3 projects for each of the following languages: *python*, *java*, *javascript* and *c++*. For each project, we downloaded diff files and metadata of the full commit history. Diff files encode per-line differences between two files or sets of files in a standard format, allowing us to recover source code changes in each commit at the line level. On the other hand, medatada allows us to recover information such as the author and message of each commit.

The extracted commit messages were processed using the Penn Treebank tokenizer (Marcus et al., 1993), which nicely deals with punctuation and other text marks. To obtain a source code representation of each commit, we parsed the diff files and used a lexer (Brandl, 2016) to tokenize their contents in a per-line fashion allowing us to maximize the amount of source code recovered from the diff files. Data and source code available[1].

Experimental Setup: Given the flat structure of the diff file, source code in contiguous lines might not necessarily correspond to originally neighboring code lines. Moreover, they might come from different files in the project. To deal with this issue, we first worked only with those commits that modify a single file in the project; we call this the *atomicity* assumption. By using only *atomic* commits we reduced our training data by an average of roughly 50%, but in exchange we made sure all the extracted code lines came from the same file. At the same time, we expect to maximize the likelihood of observing a direct relation between the commit message and the lines altered.

[1] http://github.com/epochx/commitgen

Language	Project	Full	Atomic	Added	Rem.
python	Theano	24,200	65.40%	11.43%	2,83%
	keras	2,855	66.02%	11.07%	3,01%
	youtube-dl	13,968	74.49%	11.52%	2,59%
javascript	node	15,811	53.17%	11.87%	3,21%
	angular	6,204	32.90%	5.59%	1,72%
	react	7,806	53.29%	12.67%	2,72%
c++	opencv	20,480	50.08%	8.83%	1,66%
	CNTK	10,792	38.36%	6.00%	2,23%
	bitcoin	12,596	48.11%	9.84%	2,56%
java	CoreNLP	9,149	42.77%	7.84%	1,98%
	elasticsearch	25,764	43.77%	9.02%	2,61%
	guava	3,821	38.63%	8.90%	2,64%
Average		12,787	50.58%	9.55%	2,48%

Table 1: Summary of our collected data.

We then relaxed our *atomicity* assumption and experimented with the *full* commit history. Given our maximum sequence length constrain of 100 tokens, we only observed an average of 1,97% extra data on each project. Since source code lines may come from different files, we added a delimiting token *NEW_FILE* when corresponding.

We were also interested in studying the performance of the model in a cross-project setting. Given the additional challenges that this involves, we designed a more controlled experiment. Starting from the *atomic* dataset, we selected commits that only add or only remove code lines, conforming a derived dataset that we call *uni-action*. We chose the *python* language to maximize the available data. See Table 1.

Results and Discussion: We begin by training our model on the *atomic* dataset. As baseline we used MOSES (Koehn et al., 2007) which although is designed as a phrase-based machine translation system, was previously used by Iyer et al. (2016) to generate text from source code. Concretely, we treated the tokenized code snippet as the source language and the NL description as the target. We trained a 3-gram language model using KenLM (Heafield et al., 2013) and used mGiza to obtain alignments. For validation, we use minimum error rate training (Bertoldi et al., 2009; Och, 2003) in our validation set.

As Table 3 shows, our model trained on *atomic* data outperforms the baseline in all but one project with an average gain of 5 BLEU points. In particular, we observe bigger gains for java projects such as *CoreNLP* and *guava*. We hypothesize this is because program differences in Java tend to be longer than the rest. While this impacts on training time, at the same time it allows the model to

work with a larger vocabulary space. On the other hand, our model performs similarly to MOSES for the *node* and slightly worse for the *youtube-dl*. A detailed inspection of the NL messages for *node* showed that many of them exhibit a fixed pattern in their structure. We believe this rigidity restrains the generation capabilities of the decoder, making it more prone to memorization.

Table 2 shows examples of generated descriptions for real changes and their references. Results suggest that our model is able to generate semantically sound descriptions for the changes. We can also visualize the summarizing power of the model, as seen in the *Theano* and *bitcoin* examples. We observe a tendency to choose more general terms over too specific ones meanwhile also avoiding irrelevant words such as numbers or names. Results also suggest the emergence of rephrasing capabilities, specifically in the second example from *Theano*. Finally, our generated descriptions are, in most cases, semantically well correlated to the reference descriptions. We also report not so successful results, such as case of *youtube-dl*, where we can see signs of memorization on the generated descriptions.

Regarding the cross-project setting experiments on *python*, we obtained BLEU scores of 14.6 and 18.9 for only-adding and only-removing instances in the *uni-action* dataset, respectively. We also obtained validation accuracies up to 43.94%, suggesting feasibility in this more challenging scenario. Moreover, as the generated descriptions from the *keras* project in Table 2 show, the model is still able to generate semantically sound descriptions.

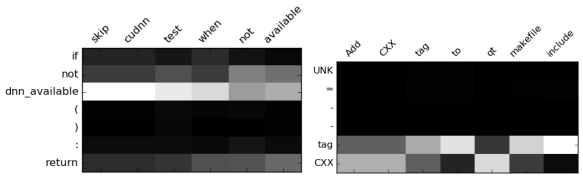

Figure 1: Heatmaps of attention weights $\alpha_{i,j}$.

Despite the small data increase, we also trained our model on *full* datasets as a way to confirm the generative power of our model. In particular, we wanted to test the model is able leverage on *atomic* data to also capture and compress multi-file changes. As shown in Table 3, results in terms of BLEU and validation accuracy manifest reasonable consistency, despite the higher disparity between source code and natural language on this dataset, which means the model was able to learn representations with more compressive power.

	Reference	Generated
keras	Fix image resizing in preprocessing/image	Fixed image preprocessing .
	Fix test flakes	Fix flaky test
Theano	fix crash in the new warning message .	Better warning message .
	remove var not used .	remove not used code .
	Better error msg	better error message .
bitcoin	Merge pull request 4486 45abeb2 Update Debian packaging description for new bitcoin-cli (Johnathan Corgan)	Update Debian packaging description for new bitcoin-cli
	Add two unittest-related files to .gitignore	Add : Minor files to .gitignore
CoreNLP	Add a bunch of verbs which are more likely to be xcomp than vmod	Add a bunch of verbs which are more to be xcomp than vmod
	Add a brief test for optional nodes	make this test do something
youtube-dl	[crunchyroll] Fix uploader and upload date extraction	[crunchyroll] Fix uploader extraction
	[extractor/common] Improve base url construction	[extractor/common] Improve extraction
	[mixcloud] Use unicode_literals	[common] Use unicode_literals
opencv	fixed gcc compilation	fixed compile under linux
	remove unused variables in OCL_PERF_TEST_P ( )	remove unused variable in the module

Table 2: Examples of generated natural language passages v/s original ones taken from the test set.

Soft alignments derived from Figure 1, which shows examples of attention heatmaps, illustrate how the model effectively associates source code tokens with meaningful words.

Dataset	*atomic*			*full*	
	Val. acc	BLEU	Moses	Val. acc	BLEU
Theano	36.81%	9.5	7.1	39.88%	10.9
keras	45.76%	13.7	7.8	59.30%	8.8
youtube-dl	50.84%	16.4	17.5	53.65%	17.7
node	52.46%	7.8	7.7	53.70%	7.2
angular	44.39%	13.9	11.7	45.06%	15.3
react	49.44%	11.4	10.7	48.61%	12.1
opencv	50.77%	11.2	9.0	49.00%	8.4
CNTK	48.88%	17.9	11.8	44.85%	9.3
bitcoin	50.04%	17.9	13.0	55.03%	15.1
CoreNLP	63.20%	28.5	10.1	62.25%	26.7
elasticsearch	36.53%	11.8	5.2	35.98%	6.4
guava	65.52%	29.8	19.5	67.15%	34.3

Table 3: Results on the *atomic* and *full* datasets.

5 Conclusion and Future work

We proposed an encoder-decoder model for automatically generating natural descriptions from source code changes. We believe our current results suggest that the idea is feasible and, if improved, could represent a contribution for the understanding of software evolution from a linguistic perspective. As future work, we will consider improving the model by allowing feature learning from richer inputs, such as abstract syntax trees and also functional data, such as execution traces.

References

Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. *arXiv preprint arXiv:1602.03001* .

Hazeline U. Asuncion, Arthur U. Asuncion, and Richard N. Taylor. 2010. Software traceability with topic modeling. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*. ACM, New York, NY, USA, ICSE '10, pages 95–104. https://doi.org/10.1145/1806799.1806817.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Nicola Bertoldi, Haddow Barry, and Jean-Baptiste Fouet. 2009. Improved minimum error rate training in moses. *The Prague Bulletin of Mathematical Linguistics* pages 1–11.

Georg Brandl. 2016. Pygments: Python syntax highlighter. `http://pygments.org`.

Raymond P.L. Buse and Westley R. Weimer. 2010. Automatically documenting program changes. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. ACM, New York, NY, USA, ASE '10, pages 33–42. https://doi.org/10.1145/1858996.1859005.

Luis Fernando Cortés-Coy, Mario Linares Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On automatically generating commit messages via summarization of source code changes. In *SCAM*. volume 14, pages 275–284.

Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, pages 345–355.

Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, New York, NY, USA, FSE 2016, pages 631–642. https://doi.org/10.1145/2950290.2950334.

Sonia Haiduc, Jairo Aponte, and Andrian Marcus. 2010. Supporting program comprehension with source code summarization. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, pages 223–226.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. Sofia, Bulgaria, pages 690–696.

Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, pages 837–847.

Xuan Huo, Ming Li, and Zhi-Hua Zhou. ???? Learning unified features from natural and programming languages for locating buggy source code .

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 2073–2083. http://www.aclweb.org/anthology/P16-1195.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*. Association for Computational Linguistics, Prague, Czech Republic, pages 177–180. http://www.aclweb.org/anthology/P07-2045.

Alon Lavie and Abhaya Agarwal. 2007. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Stroudsburg, PA, USA, StatMT '07, pages 228–231. http://dl.acm.org/citation.cfm?id=1626355.1626389.

Stanley Letovsky. 1987. Cognitive processes in program comprehension. *Journal of Systems and software* 7(4):325–339.

Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. Changescribe: A tool for automatically generating commit messages. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, pages 709–712.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1412–1421. http://aclweb.org/anthology/D15-1166.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *Proc. AAAI*. AAAI Press, pages 1287–1293.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Sapporo, Japan, pages 160–167. https://doi.org/10.3115/1075096.1075117.

Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2013. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, Piscataway, NJ, USA, ICSE '13, pages 522–531. http://dl.acm.org/citation.cfm?id=2486788.2486857.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, pages 311–318. https://doi.org/10.3115/1073083.1073135.

Veselin Raychev, Martin Vechev, and Andreas Krause. 2015. Predicting program properties from big code. In *ACM SIGPLAN Notices*. ACM, volume 50, pages 111–124.

Veselin Raychev, Martin Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In *ACM SIGPLAN Notices*. ACM, volume 49, pages 419–428.

Paige Rodeghero, Collin McMillan, Paul W McBurney, Nigel Bosch, and Sidney D'Mello. 2014. Improving automated source code summarization via an eye-tracking study of programmers. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, pages 390–401.

Wojciech Zaremba, Ilya Sutskever, and Vinyals Oriol. 2015. Recurrent neural network regularization. In *Proceedings of the 3rd International Conference on Learning Representations*.