Summarizing Git Commits and GitHub Pull Requests Using Sequence to Sequence Neural Attention Models

Ali Zaidi

Abstract

Every day millions of developers and programmers push commits to GitHub to ensure their projects are version controlled, reproducible, and remotely accessible. There are nearly 20 million public repositories (collections of source code in the form of projects) on GitHub today, and over 16 million unique users. Users are able to commit additions or changes to their own repositories, as well as the repositories of others. Being able to summarize the key changes a new edit will have on an existing repository can therefore be highly useful to ease and accelerate collaboration. In this work, we use sequence to sequence neural attention models to learn and subsequently generate commit and pull request messages from the actual diffs (changes) in the many git commits made by users. The primary task of this project is code summarization, with the added difficulty that the input sequence is programming language in the form of source code, and the output sequence is natural language (NL). Our model is shown to provide useful messages for a family of programming languages, and provides a useful automatic messaging and summarization service that could be embedded or added to existing version control systems.

1 Introduction

GitHub has become the de facto home for many open-source and collaborative software development projects. As of March 2017, there are over 16 million users working and collaborating on nearly 20 million public repositories on GitHub. This torrential amount of activity on GitHub means there is a huge potential for natural language models to better enable productivity and communication amongst users. In this work, we analyze a large collection of GitHub commit and pull request messages in tandem with the actual source code differentials (diffs) to create a code summarization model amongst the lines of abstract summarization in natural language articles or documents. We take inspiration from [1] and state-of-the-art neural translation models using sequence to sequence neural attention algorithms to encode source code differentials into summarized natural language representations.

GitHub is built on `git`, a distributed version control system where each member of a project has their own local copy of the entire repository (and it's history). After making edits to a fork of the repository, developers can merge changes back with a remote master repository on GitHub by pushing their commits (changes) back to their forked copy of the repository on GitHub with messages for each commit, and subsequently issuing a pull request to merge those changes into the master repository (note, if the user is the owner of the remote repository, she may directly push her commits back to the master remote repository). Pull requests and commit messages provide a transparent mechanism for everyone associated with a project to review the committed changes, observe any conflicts with other pieces of the software, comment, and decide whether or not to merge. For large repositories, involving many edits throughout the software, useful messages when issuing commits and pull requests are crucial to ensure everyone on the project is aware of the changes associated with a commit and can make an informed decision about whether to merge or ask for additional modifications.

1.1 Background and Related Work

There has been a flurry of activity around translation models using sequence to sequence learners. The primary objective in neural machine translation is to use a neural network to estimate a distribution over sequences in the target language conditioned on a given sequence in the source language (see the cartoon figure 2 for an example of this architecture). Such systems are typically designed in reference to an encoder-decoder architecture ([7]), where the source network, or the encoder, encodes the source sequence into a representation that is fed to a decoder network to generate the appropriate target sequence. Such architectures, especially those that include attention mechanisms in the decoder network as in [1], thereby allowing it to decide which portions of the source sentence to pay attention to, have shown to have state of the art success as translation models. Recently, these architectures have also been used in the generation of abstractive summaries, [6], allowing the model to generate condensed representations of the source document, as well as for program synthesis and code summarization in [4] and [5]. We focus on the latter task in this project.

2 Methodology

2.1 GitHub Data and REST API

The data that was used for this project is from the *ghtorrent* project [3], which is publicly available under a Creative Commons license. The *ghtorrent* project provides a mirror of the REST API provided by GitHub for tracking public data about GitHub repositories and users. In this project we utilize the data provided through the monthly MySQL database dumps on the downloads page of the website. This dump contains 21 tables, each corresponding to a separate view of the data available through GitHub's API. A schematic view of the relations between these tables is available on their website. While the commits, projects, and pull requests tables provide information on which repository and branch that commit was submitted to, as well as the user that submitted that commit, they do not actually include the source differential between the commit and the previous head branch (state prior to commit). However, we are able to use find the URL associated with the actual commit using these tables, which we can subsequently query against the GitHub REST API to pull in the actual source code in the commit diff (the API returns a JSON structure we can parse to locate the relevant "files".

2.2 Exploratory Data Analysis

The data is too large to process in-memory on a single machine (250 GBs for the tables, and an additional 90 GBs returned from the API of diffs), so we put the tables in Azure Storage account and used an Azure HDInsight Spark cluster to merge and clean the data using Spark SQL [8]. The merged dataset consists of 130 MM pull requests and their commits.

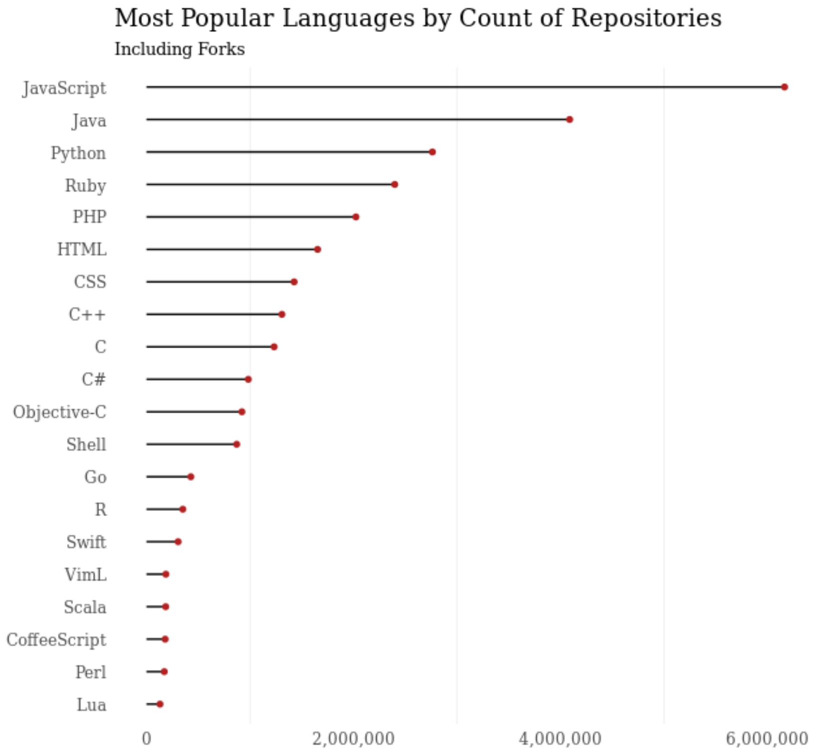
In order to obtain the most relevant pieces of the code differential data, we parsed each commit and only extract the source code diff (changes). We appended the term `<DEL>` for deletions and `<ADD>` for additions.

The majority (at least 60%) of GitHub projects are in a dormant state. We filtered to those repositories that had at least one committer and a separate reviewer in the last six months, as well as commit and PR messages with at least 20 characters. We also filtered to programming languages that share a similar "functional" syntax: Python, Ruby, Java, Javascript, R, Scala and C#. The most popular languages (see Figure 1) on GitHub include HTML and other markup languages, which have sufficiently different coding syntax than the functional languages above and would require a different tokenization, and so were deliberately excluded from this analysis.

One of the key elements for the success of our translation model is a useful embedding of source code and it's related commit message / pull request thread. If there is little information between these two sources, our encoder will not be able to summarize the code diffs into something that can be unwrapped into meaningful statements by the decoder. In order to bootstrap our algorithm, we tokenize the source code into tokens containing function names, operators, and strip out comments and whitespace. Furthermore, we replace all infrequent terms with `<unk>`.

The most challenging aspect of our code diffs are their length. Some of our code lengths are greater than 1000 tokens, which would be very hard to summarize even in NNLMs with significant memory

Figure 1:



capacity. Instead, we break up our tokens by sub-modules (i.e., loops, classes or functions), and re-use the same message across modules, or by a maximum of 100 tokens.

2.3 Sequence to Sequence Attention Models

We use a generative attention-based neural network to model the conditional distribution of a natural language summary conditioned on a code diff. Let U_C denote the set of all code diffs and U_N denote our set of all natural language summaries. For a training corpus with J code diffs and natural language summary pairs $(c_j, n_j)_{j=1}^{J}$, $c_j \in U_c$, $n_j \in U_N$, our task is generate a natural language sentence that maximizes some scoring function $s \in (U_C \times U_N \to \mathbb{R})$:

$$n^{\star} = \mathrm{argmax}_n s\,(c, n)\,. \tag{1}$$

Formally, we represent a natural language summary as a sequence of one-hot vectors $\boldsymbol{n}_1, \ldots, \boldsymbol{n}_\ell \in \{0,1\}^{|N|}$, where N is the summaries' vocabulary. Our model computes the probability of n from (1) as a product of the conditional next-word probabilities:

$$s\,(c, n) = \prod_{i=1}^{\ell} p\,(n_i|n_1, \ldots, n_{i-1})\,,$$

with

$$\text{with } p\,(n_i|n_1, \ldots, n_{i1}) \propto \boldsymbol{W} \tanh\left(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{t}_i\right), \tag{2}$$

and where $\boldsymbol{W} \in \mathbb{R}^{|N| \times H}$ and $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{H \times H}$, with H denoting the dimensionality of the summaries embedding matrix. $\boldsymbol{t}_i$ serves as the contribution from the attention model on the source code, $\boldsymbol{h}_i$ representing the hidden state of the LSTM cell at the current time step, which is computed based on the previous LSTM cell state $\boldsymbol{m}_{i-1}$ and the previous LSTM hidden state $\boldsymbol{h}_{i-1}$ as

$$\boldsymbol{m}_i; \boldsymbol{h}_i = f\left(\boldsymbol{n}_{i-1} \boldsymbol{E}, \boldsymbol{m}_{i-1}, \boldsymbol{h}_{i-1}; \theta\right),$$

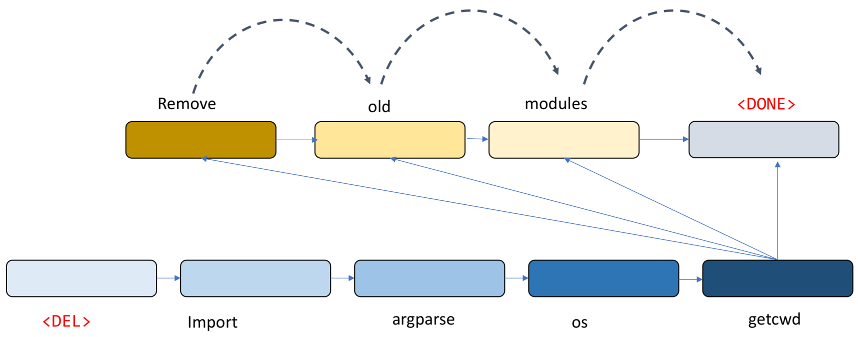


Figure 2: Representation of Encoder-Decoder Model

and finally, where $\boldsymbol{E} \in \mathbb{R}^{|N| \times H}$ is a word embedding matrix for the summaries, and f is computed using LSTM cell specifications and parameters θ. Hence, we are parameterizing the conditional distribution of our language model as a seq2seq neural network that uses LSTM as it's attention mechanism.

The generation of each word is guided by a global attention model, which computes a weighted sum of the embeddings of the code differences tokens based on the current LSTM state. For a code diff c, we represent it as a sequence of one-hot vectors$\boldsymbol{c}_1, \ldots \boldsymbol{c}_k \in \{0,1\}^{|C|}$ for each source code token, where C is the vocabulary of all tokens in our source code diffs. Our attention model computes

$$\boldsymbol{t}_i = \sum_{j=1}^{k} \alpha_{i,j} \cdot \boldsymbol{c}_j \boldsymbol{F},$$

where $\boldsymbol{F} \in \mathbb{R}^{|C| \times H}$ is a token embedding matrix and each activation $\alpha_{i,j}$ is computed by softmax of the LSTM hidden state $\boldsymbol{h}_i$ and the corresponding token embedding of the code diff $\boldsymbol{c}_j$:

$$\alpha_{i,j} = \frac{\exp\left(\boldsymbol{h}_i^\top \boldsymbol{c}_j \boldsymbol{F}\right)}{\sum_{j=1}^{k} \exp\left(\boldsymbol{h}_i^\top \boldsymbol{c}_j \boldsymbol{F}\right)}.$$

3 Training

3.1 Initialization and Decoding

We train our NNLM (2) using Adam stochastic gradient descent with mini batch sizes set to 25, and regularization specified through dropout (see Chapters 7 and 8 in [2]). We did not get a chance to try add additional LSTM layers or use GRU layers instead of LSTMs in this project, but hope to do so in future iterations.

Upon training our models and providing a code diff c, the optimal summary involves searching through the n-space in the score function's support. We approximate this value of $n^\star$ by performing beam search on the space of all possible summaries using the model output.

3.2 Model Evaluation

Our models take 8 hours to train 100 epochs on Azure NC VMs with Tesla K80 GPUs. The learning rate in Figure 3 does seem to show that our network parameters have found a local minima. In Figure 4 we visualize the relative magnitudes of the attention weights $\alpha_{i,j}$ for an example R code diff while generating it's summary. The darker regions indicate higher weights. The Figure shows that our network is able to align key summary words to informative tokens from the code diff, even those that span multiple tokens (such as `rowSums` aligning to axis, row and sums)!

The lack of standard benchmarks for our model makes it a bit difficult to assess model performance in comparison to other models. Nonetheless, we are able to create a simple performance by looking

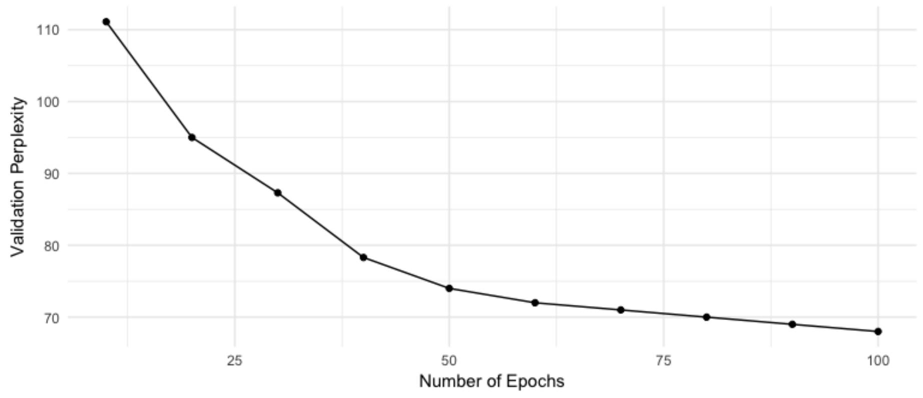


Figure 3: Learning Rate and Perplexity

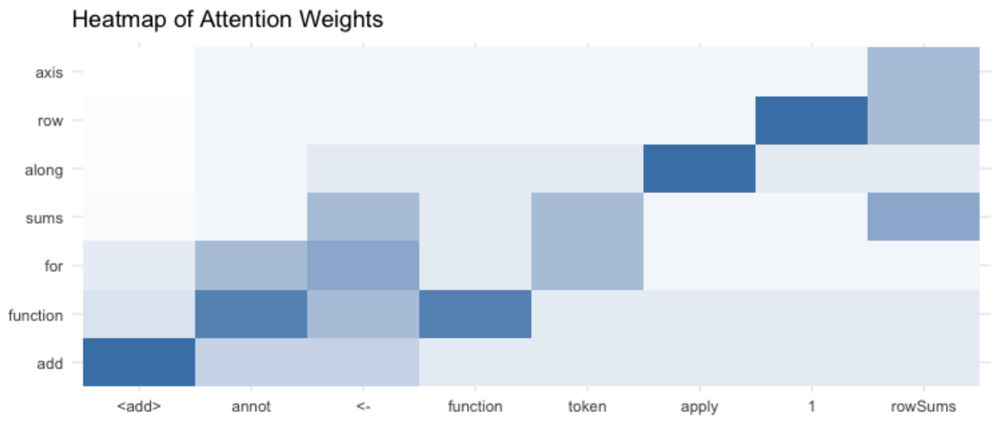


Figure 4: Heat map of Attention Activations

at the bag of words feature vectors of all the natural language descriptions from our model and compare them to the test comments and pull request threads using cosine-similarity. This allows us to calculate BLEU scores to get average n-gram precision. It should be noted, however, that our numbers will be biased downwards due to the fact that we split long code diffs into smaller tokenizations. Our results are summarized in Table 1, with the test scores shown first, and validation scores shown in parentheses. The results are interesting, showing that the model performs better on R, Python and Ruby than C#, Java and Scala. This might be due to the fact that the former are dynamically typed languages, and therefore have less boilerplate associated with type references that could make it easier to make summaries.

Language	***BLEU-4***
Ruby	20.3 (18.7)
Python	21.2 (17.4)
R	23.2 (23.0)
Javascript	19.3 (17.7)
Java	13.5 (11.7)
Scala	14.8 (12.4)
C#	13.3 (13.9)

Table 1: BLEU Scores by Language

4 Conclusion

In this work, we trained a sequence to sequence neural attention model for summarizing source code commits and pull requests. A future direction might be to add tags for commits, such as "bug-fix". In particular, it might be possible to couple the commits with the issues, and find commits that are related to specific resolutions of outstanding issues. Another interesting use case might be code-retrieval, where the user could provide natural language summaries and obtain scored retrieval rankings of source code that corresponds to that query. In this case, rather than search through the natural language space $n^\star \in U_N$ for the sentence that maximizes the score in (1), we may instead want to retrieve the highest scoring code chunk $c^\star \in U_C$ that maximizes the score function:

$$c^\star = \text{argmax}_c s\left(c, n\right).$$

Such a model would allow the user to describe a desired piece of code (or repository of code) in natural language, and the model would retrieve the closest match to the description in terms of the code embeddings. However, in this case we may want to generalize beyond beam search we might be more limited in the context of the large feature space provided by code diffs (this is my hunch at least). We save these directions for future projects!

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR Proceedings*, 2015. 1, 2

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 4

[3] Georgios Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press. 2

[4] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 2073–2083, 2016. 2

[5] Xi Victoria Lin, Chenglong Wang, Deric Pang, Kevin Vu, Luke Zettlemoyer, and Michael D. Ernst. Program synthesis from natural language using recurrent neural networks. Technical Report UW-CSE-17-03-01, University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, March 2017. 2

[6] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015. 2

[7] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 2

[8] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association. 2