Evaluation of Transformers for Code Summarization

Ashwin Shenolikar
Virginia Tech
Falls Church, Virginia, USA
ash07@vt.edu

Prachi Pundir
Virginia Tech
Blacksburg, USA
prachipundir@vt.edu

Abstract

Under the software maintenance and development, the program code is the key to success. High quality comments will help you better understand the program. However, when there is not enough clarity in the event of analyzing another's code, there can be delays in the software lifecycle. An overview of automatic code summarization is proposed to solve these problems. Source code summarization is a new technology for automatically generating a brief description of your code. Current summarization techniques work by selecting a subset of statements and keywords from your code and including information from those statements and keywords in the summary. We explore a new kind of mechanism and architecture in natural language processing - transformers to be applied to this task and see how the model holds up against other existing methods of doing code summarization

Keywords: Code Summarization, Neural Network, Transformers, Encoder-Decoders

ACM Reference Format:
Ashwin Shenolikar and Prachi Pundir. 2018. Evaluation of Transformers for Code Summarization. In *Proceedings of Make sure to enter text (CS 5814 Project Proposal).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

With the rapid development of scale and complexity Software system developers spend about 59% of their time in program understanding[1]. Source code summarization is the task of generating a high-level natural language description for a segment of programming language code. Automatic code summarization helpful in for the software development process as it reduces the manual task.
Creating a readable summary that describes the functionality of the program is known as an overview of the source code. This task is a learning code representation by modeling a pair of relationships between CODE tokens to capture their long range dependencies[6]. To learn the code display to summarize, you use a self-starting mechanism to explore the transformer model and indicate that the long range dependency is valid. In this project, we show that despite the approach, it is easy to exaggerate state technology from a significant end.

CS 5814 Project Proposal, April ,2022, Virginia Tech

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 Related Work

The field of Deep Learning in Software Engineering is currently very active as far as research work goes, and we have investigated related work to better inform us with our project.

In this paper[2], the authors have used a LSTM architecture with the attention mechanism in order to summarize code snippets in C as well as SQL Queries. Their model 'CODE-NN' has been created using data from StackOverflow, which is a popular open website for programming related questions. While it is widely used, the data from the website is also noisy, and a portion of the work is dedicated to data cleaning, which is not as much of an issue for our dataset.
Wasi Uddin Ahmad et al.[3] in their paper, have worked to create a result similar to what we are aiming to accomplish, wherein they used a similar model with some changes to describe code snippets and achieved results comparable to the then state-of-the-art techniques to perform the same task.
LeClair et al.[4] have used incorporated a graph with neural networks to improve upon the then existing models for code summarization. They use the classification of Graph Neural Networks as follows:(1)Recurrent Graph Neural Networks (RecGNNs),(2)Convolutional Graph Neural Networks (ConvGNNs),(3) Graph Autoencoders (GAEs),(4) Spatial-temporal Graph Neural Networks (STGNNs). However, their main focus is on the ConvGNNs, with which they used 'aggregation' techniques to learn representations.
Shido et al. [5] have used an Extended Tree-LSTM architecture to perform code summarization in an attempt to solve a problem with using ASTs by using a Multi-way Tree-LSTM.

3 Proposal Plan

Our plan comprises of three main steps: 1. Finding relevant dataset(s), develop a deep learning model based on the latest advances in the field of Natural Language Processing, and

finally, set evaluation parameters which will help us accurately judge the model's performance. If time permits, we will evaluate the performance of this model with respect to other popular and/or historical architectures, such as base Long-Short Term Memory(LSTMs) or Encoder-Decoder models(which use LSTM), which were used to judge which model performs best, and by how much.

3.1 Data

Our primary source of data comes from the CodeSearchNet Dataset[6], which is a collection of about 2 million samples of code and corresponding comment which acts as the label. This dataset has the programming languages Python, Java, JavaScript, Ruby, Go and PHP, out of which we will initially focus on the first 2, i.e., Java and Python. In general, the code blocks are functions performing some task and have been collected from various open source libraries.
Apart from the massive CodeSearchNet dataset, we have also searched two other datasets that we considered. These are the MUSE Corpus[7] and the Funcom[8] datasets, which have Java code with comments, but otherwise do not cover many other languages. In case of the Java language, we may attempt to use these datasets for validation and testing purposes.

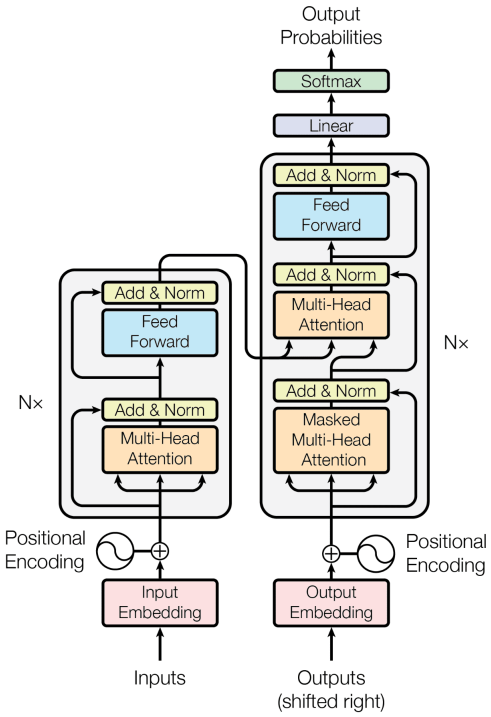
3.2 Model

In natural language processing, currently some of the most state-of-the-art technology is the transformer. A transformer[9] is basically a version the encoder-decoder model, and hence is good for Sequence to Sequence tasks.The most important addition to the classical encoder-decoder model which makes the transformer perform better is the attention mechanism. Using the attention mechanism gives the model more than just the literal 'translation' of the input; it also gives it context. With attention, the transformer can estimate which words in the input are also more relevant to the current word being processed. Also, it does not use any recurrent networks due to the presence of its attention mechanism. While the implementation of a transformer is the primary goal, we also want to compare performances of LSTM and/or Encoder-Decoder models without transformers. Provided that the implementation of the transformer yields a satisfactory result, we aim to implement these models to do a comparative analysis.

3.3 Model Evaluation

For the evaluation of the model, we plan on using the classical machine learning metrics of precision, recall, F-1 score, training accuracy and loss, validation accuracy and loss, and as it is a Text generation problem, the BLEU Score. In Natural Language tasks, the BiLingual Evaluation Understudy (BLEU) score is a very important metric which is used to calculate how precise or qualitative of an output the model has calculated compared to the reference or given labels.

Figure 1. Transformer Model[4]



The BLEU score is always a value between 0 and 1, inclusive, where a higher value relates to higher overlap between the model output and true values.

4 Expected Outcomes

At the conclusion of the project, we expect to have created a Deep Learning model which can produce string outputs which describe the function of input code blocks with an acceptable and reliable accuracy and BLEU scores. We also hope to be able to describe, by comparison, the performance of other architectures used for the task of code summarization compared to the main method that we are using, so that we can see how the performances of these models stack comparatively.
A learning outcome of this project is to be able to learn in detail about encoder-decoder models, transformers and in general, the workflow of any Deep Learning task in the subdomain of Natural Language Processing. Additionally, we also aim to better understand how Software Engineering-specific tasks can be improved with the integration of machine learning and deep learning models, which can increase readability and accuracy of existing source codes.

References

[1] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code,"

in 2010 17th Working Conference on Reverse Engineering. IEEE, 2010, pp. 35–44.

[2] Iyer, Srinivasan Konstas, Ioannis Cheung, Alvin Zettlemoyer, Luke. (2016). Summarizing Source Code using a Neural Attention Model. 2073-2083. 10.18653/v1/P16-1195.

[3] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, Kai-Wei Chang,"A Transformer-based Approach for Source Code Summarization" https://doi.org/10.48550/arXiv.2005.00653

[4] Alexander LeClair, Sakib Haque, Lingfei Wu, Collin McMillan,"Improved Code Summarization via a Graph Neural Network" https://doi.org/10.48550/arXiv.2004.02843

[5] Yusuke Shido, Yasuaki Kobayashi, Akihiro Yamamoto, Atsushi Miyamoto, Tadayuki Matsumura,"Automatic Source Code Summarization with Extended Tree-LSTM"arXiv:1906.08094v2

[6] CodeSearchNet Challenge, https://github.com/github/CodeSearchNet

[7] MUSE Corpus, https://github.com/facebookresearch/MUSE

[8] LeClair, A., McMillan, C., "Recommendations for Datasets for Source Code Summarization", in Proc. of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19), Short Research Paper Track, Minneapolis, USA, June 2-7, 2019.

[9] Vaswani, Ashish Shazeer, Noam Parmar, Niki Uszkoreit, Jakob Jones, Llion Gomez, Aidan Kaiser, Lukasz Polosukhin, Illia. (2017)https://arxiv.org/pdf/2004.02843.pdf