MATCHING DESCRIPTIONS WITH CODE FRAGMENTS BY LEARNING DEEP JOINT EMBEDDINGS

Research Proposal
Divyansh Manocha

ABSTRACT

Numerous software engineering tasks, including re-factoring, detecting bugs or understanding complicated APIs, significantly benefit from finding semantically similar or equivalent fragments of code. Not being able to do this is not only an impediment for development that requires system-specific knowledge, but also for future software maintainability and evolution. This proposal suggests two solutions: (1) search using descriptions and (2) generating comments from code, which can also serve as augmented data. The former may be achieved by learning a joint embedding, applied to the application of searching for code using natural language descriptions, it aims to reduce system-wide duplication. For the latter, a novel variational recurrent neural machine translation model is proposed, inspired by the recent advancements in neural variational inference and generative latent variable models for language processing.

1 CONTEXT

With the rise of large scale open code collaborations within systems such as companies or research communities, a major issue that has largely not been addressed yet is the ability to assist a programmer through reusability of existing code. Recently (Koschke & Bazrafshan, 2016) conducted a large clone-rate survey from 7,800 open-source C/C++ projects, finding 44% of the projects have at least one identical (type 1) clone fragment. 80% have type 2 or more (see below for definition). The clone-rate was found to be $\sim 12\%$ for type 2.

Code duplication presents challenges in software maintenance, where they require similar bug fixes or refactors to multiple locations. To be able to tackle this problem, it is essential to be able to search for and identify semantically similar code. In the cases where APIs developed are complicated, searching for keywords in the system is the only option present to developers, which is often very time consuming and seldom achieves desired results. This is a different approach to the problem than analysing the system to then identify and remove duplicates, primarily through visual techniques as described by Rieger et al. (2004).

The problem of identifying duplication is very different to presenting aggregate statistics for similarity, as is used in plagiarism detection tools. State of the art algorithms that do not rely on deep learning largely focus on syntactic similarity, as semantics is a very difficult problem. These can be categorised into the following:

- Strings: Code fragments used as contiguous sequence of strings
- Tokens: Lexer converts fragments into token sequences, where a similar sub-sequence can be identified (Kamiya et al., 2002) (Li et al., 2004)
- Tree: Using an abstract syntax tree (AST) representation of the fragment, parsed sub-trees are identified (Baxter et al., 2004)

Generally, the accepted definition for categories of clone types is as defined by (Roy et al., 2009). **Type 1**: Fragments identical except for variations in whitespace, comments or layouts. **Type 2**: Identical fragments except for variations in the names of user-defined identifiers (e.g. variables, class, methods, etc.). **Type 3**: In addition to the differences in Type 2, the fragments have modified statements added, modified or deleted. Therefore they are still syntactically similar, and semantically identical. **Type 4**: Semantic similarity that has the same functionality, but is syntactically dissimilar. Examples of each are shown in Appendix A.

Deep learning techniques should particularly outperform traditional statistical methods for type 4, which is still an active area of research. Recent advancements in multimodal learning, particularly in the context of translation, has led to significant improvements in Natural Language Processing (NLP) . Hindle et al. (2012) suggested that source code can be treated as a natural language with semantics just as rich as that found in text.

Embeddings map discrete tokens to real-valued vectors, thereby translating data in a relatively low dimension space to higher dimensional vectors representation. This has become a very popular technique for machine learning, especially on a larger corpus (dataset). The idea behind joint embeddings is to project two separate sources in the same higher-dimensional vector space (e.g. text comments with the code snippets), learnt through a neural net or log-linear model architecture Wang et al. (2018). This also allows searching based on the semantics of the input, rather than its syntax, using similarity measures.

A common approach to a problem of this nature requires an embedding that does not assume each word represents the document from the corpus equally, as with *Word2Vec*. Information retrieval using TF-IDF (term frequency-inverse document frequency) penalises common words by assigning them lower weights whilst giving importance to words that are less frequent in the entire corpus but frequent in a few documents Sparck Jones (1988). These are commonly applied at word level, N-gram level or character level. From here topics that contain the best information in the collection are identified using techniques such as LDA (Linear Discriminant Analysis).

2 Relevant Literature

There have been several recent advances in text processing and generation recently. Most notably, (Radford et al., 2019) (Radford et al., 2018) use supervised fine tuning on an unsupervised model based on the *Transformer* architecture (Vaswani et al., 2017) to achieve state of the art performance. Unlike other models that use recurrence and convolutions, this is solely based on attention mechanisms and achieved state of the art quality in translation and document generation. (Miao et al., 2016) presented a neural variational framework that combines a continuous document representation with a generative bag of words model, inspired by variational auto-encoders (Kingma et al., 2014). This framework is able to learn to model the posterior instead of just analytic approximations, and can therefore use an inference network such as RNN or CNN to learn non-linear distributions through backpropagation.

2.1 Joint Embeddings

Recent embedding methods have been based around maximising correlation between the projected vectors from the two sources in the shared latent space, such as Canonical Correlation Analysis (CCA) (Wang et al., 2015). Several extensions have also been proposed that kernelize the method or apply them in a deep learning framework. Due to its limited scalability, a more common approach is to produce a joint embedding using SGD with a ranking loss. However, this does not outperform CCA.

Kiros et al. (2014) introduced an encoder-decoder pipeline that learns the joint embedding space (images and text) by minimising a pairwise ranking loss and a language model that decodes the representation. This pipeline has already been successful in Neural Machine Translation. This was followed by the very successful RNN based model from Vinyals et al. (2014), adopting a similar model.

(Hu et al., 2018) formulate code comment generation as a neural machine translation task, based on the Sequence-to-Sequence (Seq2Seq) learning framework with attention. Specifically, they implement a two-layered LSTM with 512-dimensional word embeddings. However, they focus on comment generation rather than searching.

There has only been one study, to the best of the authors knowledge, that addresses the same problem: DeepCSGu et al. (2018). In a similar fashion to the afore mentioned research, they project to a higher dimensional space to make queries and source code lexically comparable. An interesting architecture of two deep neural net pipelines was formed where bi-directional LSTMs are used for each split of information appropriate from the code database (method name, api sequence and tokens), embedded with description. Therefore it allows for associative search - not only seeking snippets with matched keywords but also recommends those without matched keywords but are semantically related.

2.2 Neural Machine Translation

Neural machine translation (Cho et al., 2014) (Sutskever et al., 2014), unlike statistical machine translation, aims to tune a single, large neural network for the task of translation. There has been a wealth of literature on NMTs that are trained multilingual systems, through knowledge transfer, implementing multiway translation and multi-source translation. However, most of the literature surrounding machine translation has largely focused on natural languages.

The dominant approach is the encoder-decoders paradigm in which sentences in a corpus are encoded to a fixed-length (or sometimes variable-length) vector and a decoder generates the translations. LSTMs and RNNs are commonly used as the non-linear encoder function defining a hidden state $h_t = f(x_t, h_{t-1}), q(\{h_1, \dots, h_T\} = h_T$. The decoder defines a joint probability over the translation y, decomposing into conditionals where each conditional probability is modelled with a RNN or some de-convolutional neural network. Sampling and Greedy search are commonly used to perform decoding.

Neural variational inference (Kingma & Welling, 2014) approximates the posterior of a generative model, and has been demonstrated to be successful for topic modelling (Miao et al., 2017) - where the models are parametrised with neural networks and trained with variational inference. CNN based models (Gehring et al., 2017) have also achieved good performance, but the current state of the art is the Transformer model (Vaswani et al., 2017) which is fully attention based (Self-Attention with feed-forward connection). This is largely the fundamental architecture of the very successful *GPT* and *BERT* models (Radford et al., 2019) (Radford et al., 2018).

(Zhang et al., 2016) proposed a variational NMT in which they incorporate a latent random variable intended to to represent the underlying semantics of the sentence. (Shah & Barber, 2018) introduce a latent variable that is explicitly designed to learn the semantic meaning, by modelling the probability distribution of the sentence rather than the conditional probability of the target sentence given the current: $p(\mathbf{y}|\mathbf{x}) = \int p_\theta(\mathbf{y}|\mathbf{z}, \mathbf{x}) p_\theta(\mathbf{z}|\mathbf{x}) d\mathbf{z}$. The authors show that their model places higher reliance on the latent variables and achieves superior performance in cases of missing words.

2.3 Neural code clone identification

The first proposal of language models and embeddings for clone detection was proposed by (White et al., 2016), in which they employ an unsupervised *fuse* and *use* technique. They develop a framework that trains for lexical analysis through RNNs using tokens, and syntactic analysis through a recursive auto-encoder.

(Li et al., 2017b) apply a token-based pair by pair method comparison approach using deep neural networks, in which they directly extract features based on different categories of tokens in source code. Therefore it learns patterns from previously trained method pairs.

(Büch & Andrzejak, 2019) employ a supervised AST-based Recursive Neural Network, based on (Jiang et al., 2007), in which tree patterns are recursively aggregated until one characteristic vector root is found that represents the entire tree. A Siamese Network was used to compare the outputs with shared weights.They used Area Under the Curve (AUC) for the Receiver Operating Characteristic (ROC) for evaluation. They also find that pretrained embedding representing nodes in ASTs has been a key to performance improvements.

Dividing the current literary work on neural languages into two classes, of explicit models that require restriction of relevant context; and implicit models that attempt to represent the co-occurrence patterns of tokens in a high-dimensional real-valued parameter space. The former primarily rely on n-gram frequency, which make a prediction considering only the previous $n - 1$ tokens and have been proven to be scalable and effective. Deep learning models such as RNNs are commonly used for the latter, where they extract latent representations of text (Mikolov et al., 2011).

3 Research Question

Each questions should result in it's own research task.

One of the fundamental questions outstanding would be *Is it possible to learn embeddings that can be used to effectively search and translate between code and description?* There has been some indication in literature that this is possible (Büch & Andrzejak, 2019) (Li et al., 2017b) Gu et al. (2018) however, the author believes capturing semantic context seems to be a general area for improvement. This is evident from the ranks of the output of the model that often display partially relevant results higher than close to exact ones.

The proposal questions whether extending the work by (Shah & Barber, 2018) to the case of RNNs will obtain a variational recurrent NMT that should theoretically be able to model more complex dependencies between timesteps. There has been an attempt by (Su et al., 2018) inspired by the Variational Recurrent Neural Network as an extension of VAEs, to propose a variational recurrent NMT. However, their approach is based on the work by (Zhang et al., 2016) which, as discussed in Section 2.2, does not model the probability distribution of a sentence.

Does searching for system-wide code duplication of code fragments improve with deep joint embeddings? This will extend the work by (Büch & Andrzejak, 2019) and (Li et al., 2017b), that search on embeddings solely based on code. Similarity measures that perform well in high dimensions should be able to provide a better search mechanism.

Finally, as an extension to the previous questions, *can information from ASTs be embedded in the encoder for generating better comments in the proposed architecture?* Software code has well defined semantics as opposed to natural language, which we can take advantage of when considering embeddings and distance metrics that address the problem. This has previously been employed by (Li et al., 2017a) and (Büch & Andrzejak, 2019), who show that pretrained embedding representing nodes in ASTs and an attention mechanism has been a key to performance improvements.

4 Aims and Objectives

On a high level, the proposal aims to determine whether it is possible to use the same embeddings to generate comments for undocumented code, as well as matching descriptions to functions in programs.

The results of this study should be three folds:

- To contextualise mappings between code and comments/description.
- To be able to search for code using natural language text, and therefore effectively search for duplicate code fragments.
- To infer descriptions for undocumented code fragments.

The research objectives include to be able to search for an existing code fragment, if one exists in the databases, that structurally performs what has been specified by a user in the description comments of an unimplemented piece of code. An interesting application that should also result from this is to generate code comments and/or api documentation for code. In the case of Sphinx [1] documentation, code comments are already required to be well structured, which can be leveraged in experiments.

5 Significance of Research

Code duplication has received increasing attention from industry and reverse-engineering research communities. Due to the poor collaboration (which includes communication through documentation) amongst teams in a large organisation, a lot of effort is wasted into reinventing methods which adversely leads to re-introducing and re-fixing common bugs. Code duplication is also a general problem in machine learning models of source code as it leads to significant bias. The ability to detect and reduce system-wide code duplication should also allow further advancements in machine learning models based on code (Allamanis, 2019).

[1]www.sphinx-doc.org

Although there has been some work related to image captioning and therefore generating comments in a similar manner to that suggested here, there has not been any specifically tailored to addressing code duplication by suggesting from existing source code/api corpuses, based on comments written for an un-implemented method.

6 Future directions of research

- Consider batch scenarios in this proposal, where learning can be expensive but (testing) should not be. Potential future work could explore a continual setting.
- Potential work around contextualised neural code completion. The estimated posterior from this study should serve as a useful starting point.
- A lot of the time multiple languages make up frameworks, e.g. typescript react, config yamls, python, etc. There has been significant work to develop a generic NMT for multiple human languages recently, which can be a further direction of research.

References

Miltiadis Allamanis. 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*, pp. 143–153, 2019.

Ira D Baxter, Christopher Pidgeon, and Michael Mehlich. Dms/spl reg: program transformations for practical scalable software evolution. In *Proceedings. 26th International Conference on Software Engineering*, pp. 625–634. IEEE, 2004.

Lutz Büch and Artur Andrzejak. Learning-based recursive aggregation of abstract syntax trees for code clone detection. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 95–104. IEEE, 2019.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL `https://www.aclweb.org/anthology/D14-1179`.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pp. 1243–1252, 2017.

X. Gu, H. Zhang, and S. Kim. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 933–944, 2018.

Abram Hindle, Earl Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the naturalness of software. *Proceedings - International Conference on Software Engineering*, pp. 837–847, 06 2012. doi: 10.1109/ICSE.2012.6227135.

Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pp. 200–20010. IEEE, 2018.

Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *29th International Conference on Software Engineering (ICSE'07)*, pp. 96–105. IEEE, 2007.

T. Kamiya, S. Kusumoto, and K. Inoue. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002. doi: 10.1109/TSE.2002.1019480.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *stat*, 1050:10, 2014.

Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. *Advances in neural information processing systems*, 27: 3581–3589, 2014.

Ryan Kiros, Ruslan Salakhutdinov, and Richard S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *CoRR*, abs/1411.2539, 2014. URL `http://arxiv.org/abs/1411.2539`.

Rainer Koschke and Saman Bazrafshan. Software-clone rates in open-source programs written in c or c++. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 3, pp. 1–7. IEEE, 2016.

Jian Li, Yue Wang, Michael R Lyu, and Irwin King. Code completion with neural attention and pointer networks. *arXiv preprint arXiv:1711.09573*, 2017a.

Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. Cclearner: A deep learning-based clone detection approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 249–260. IEEE, 2017b.

Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. Cp-miner: A tool for finding copy-paste and related bugs in operating system code. In *OSdi*, volume 4, pp. 289–302, 2004.

Yishu Miao, Lei Yu, and Phil Blunsom. Neural variational inference for text processing. In *International conference on machine learning*, pp. 1727–1736, 2016.

Yishu Miao, Edward Grefenstette, and Phil Blunsom. Discovering discrete latent topics with neural variational inference. In *International Conference on Machine Learning*, pp. 2410–2419, 2017.

Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černockỳ. Strategies for training large scale neural network language models. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, pp. 196–201. IEEE, 2011.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

M. Rieger, S. Ducasse, and M. Lanza. Insights into system-wide code duplication. In *11th Working Conference on Reverse Engineering*, pp. 100–109, 2004.

Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470 – 495, 2009. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2009.02.007. URL `http://www.sciencedirect.com/science/article/pii/S0167642309000367`.

Harshil Shah and David Barber. Generative neural machine translation. *Advances in Neural Information Processing Systems*, 31:1346–1355, 2018.

Karen Sparck Jones. *A Statistical Interpretation of Term Specificity and Its Application in Retrieval*, pp. 132142. Taylor Graham Publishing, GBR, 1988. ISBN 0947568212.

Jinsong Su, Shan Wu, Deyi Xiong, Yaojie Lu, Xianpei Han, and Biao Zhang. Variational recurrent neural machine translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27:3104–3112, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014. URL `http://arxiv.org/abs/1411.4555`.

Guoyin Wang, Chunyuan Li, Wenlin Wang, Yizhe Zhang, Dinghan Shen, Xinyuan Zhang, Ricardo Henao, and Lawrence Carin. Joint embedding of words and labels for text classification. *CoRR*, abs/1805.04174, 2018. URL `http://arxiv.org/abs/1805.04174`.

Liwei Wang, Yin Li, and Svetlana Lazebnik. Learning deep structure-preserving image-text embeddings. *CoRR*, abs/1511.06078, 2015. URL `http://arxiv.org/abs/1511.06078`.

Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. Deep learning code fragments for code clone detection. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 87–98. IEEE, 2016.

Biao Zhang, Deyi Xiong, Jinsong Su, Hong Duan, and Min Zhang. Variational neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 521–530, 2016.

A Appendix A: Listings demonstrating the types of code-cloning

Listing 1: Original code

```
a, b = draws()
if (a == b):
   x = x + 1
else:
   x = y + 1
return func(x,y)
```

Listing 2: Type 1 clone

```
a,b = draws_alt()
if (a == b):
   x = x + 1
else:
   x=y+1
return func(x,y)
```

Listing 3: Type 2 clone

```
m,n = draws_alt()
if (m == n):
   p = p + 1
else:
   q=q+1
return func(p,q)
```

Listing 4: Type 3 clone

```
a, b = draws()
if (a == b):
   x = x + 1
   k = 5
else:
   x = y + 1
   k = 3
func_alt(k)
return func(x,y)
```

Listing 5: Original code

```
i, j=1;
for (i=1; i <= max_value; i++):
  j = j*i
```

Listing 6: Type 4 clone

```
def factorial(n):
   if n == 0:
      return 1
   else:
      return n * factorial(n-1)
```