ソースコード要約において識別子の与える影響の調査

宮崎 広夢
東京大学大学院 情報理工学系研究科
miyazaki@logos.t.u-tokyo.ac.jp

1 はじめに

国内では義務教育への導入も決定し、人々がプログラミングをする機会は増加傾向にある。一般に、プログラミングをする際に他人のソースコードを読んだり、ライブラリを使用する機会は多い [9]。しかし、対象のソースコードが何をやっているのか、コメントやドキュメントなく一読して把握することは困難である場合も多い。

近年、様々な自然言語を対象にして言語処理を行う試みが多くなされており、その一環としてプログラムのソースコードを対象とした研究も行われている。ソースコードを対象とした研究にも自然言語と同様に様々なものが存在しているが、その 1 つにソースコードの要約（以下、ソースコード要約）がある。ソースコード要約とは、与えられたソースコードについてそれが何を行うものかを表す自然言語の要約を与えることを目標とした分野である。いくつかの手法が提案されているが、まだ十分な成果が出ているとは言い難く非常に困難なタスクである。

本研究ではニューラルネットワークを用いたソースコード要約において、既存のデータセットに含まれる識別子の性質および重要性について調査を行なった。

2 関連研究

2.1 ソースコード要約

ソースコード要約は、入力にソースコードを受け取り、出力にソースコードが何をやっているかを表現した自然言語を出力するタスクである。具体例を 1 に示す。上段が Java のソースコードを表しており、下段がそれに対応する要約文である。ソースコードの要約は一般的な文章要約タスクとは異なる。一般的な文章要約タスクでは、入力文と出力文で扱われる言語は同じである。一方、ソースコードの要約は、入力はプログラミング言語であり、出力は自然言語であるため、要約だけでなく翻訳の要素も含まれている。

表 1: ソースコードとそれに対応する要約文

```
public Matrix copy() {
    Matrix X=new Matrix(m,n);
    double[][] C=X.getArray();
    for (int i=0; i < m; i++) {
        for (int j=0; j < n; j++) {
            C[i][j]=A[i][j];
        }
    }
    return X;
}
```

Make a deep copy of a matrix

今日、ソースコード要約を行う様々な手法が提案されているが、大きく分けて以下の 3 つに分類される。

2.1.1 ルールベースの手法

最も古くから研究されている手法として、ルールベースの手法がある。これは、入力から出力を得る際のルールを予め人手で定めておき、それに基づき要約を行うものである。メソッド名や変数名などを抽出し特定のテンプレートに当てはめて要約文を出力する手法がいくつか提案されている。

Sridhara ら [11] は Java のメソッドを要約するルールベースの手法を提案した。Sridhara らは、ソースコード要約を、ソースコードから必要な部分を抽出する Content Selection と、Content Selection で抽出されたソースコードの一部から実際に要約文を出力する Text Generation の 2 つのタスクに分割した。その上で、それぞれのタスクについて経験則からなる知見を元にルールを定めることで、要約を実現した。

また、Msie'deen[1] らはソースコード内のトークン

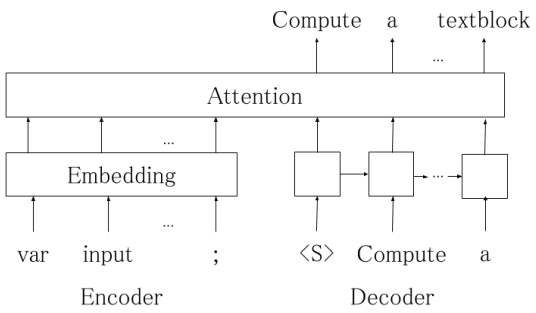

図 1: CODE-NN[5]

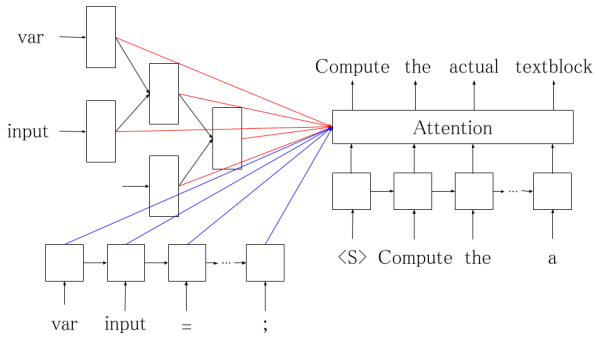

図 2: Wan らの提案したモデル [13]

の依存関係に着目してルールを定める手法を提案した。Sridhara と同じく Java のメソッドの要約を出力することを目指しているが、Sridhara らがメソッド単位でしかソースコードを扱っていなかった一方で、Msie'deen らはクラス単位でソースコードを扱うことで、クラス内のメソッドやプロパティ間の関係性も明示的に利用した。

2.1.2 統計的な手法

Haiduc ら [4] は tf-idf などの統計量を用いてソースコード要約を行う手法を提案した。類似の研究として、小田ら [7] も統計量を用いてソースコードを擬似コードに変換する研究を行なっている。

2.1.3 ニューラルネットワークを用いた手法

Iyer ら [5] はソースコードの要約を行う CODE-NN というモデルを提案した。CODE-NN は、LSTM とアテンション [6] を利用した図 1 のようなモデルである。エンコーダとデコーダからなり、エンコーダは入力のソースコードを Bag of words に変換し、デコーダはエンコーダから受け取った Bag of words を元に要約文を出力する。

一方で Wan ら [13] は、ソースコードのテキストだけでなく、抽象構文木 (AST, Abstract Syntax Tree) を利用する手法を提案した。抽象構文木は、通常の構文木から言語の意味に関係ない情報を取り除き、意味のある情報のみを抽出した抽象度の高い構文木である。Wan らのモデルの構造を、図 2 に示した。Wan らのモデルは、アテンション付きエンコーダ・デコーダモデル [6] の 1 種であるが、エンコーダが 2 種類のエンコーダからなっている。1 つはソースコードのトークン系列をそのまま受け取る通常の RNN エンコーダ、もう 1 つはソースコードの抽象構文木を受け取る Tree-LSTM [12] エンコーダである。これらのエンコーダを組み合わせることで、トークン系列の語順的な情報と、木構造の構造的・意味的な構造をともに利用することができる。

3 提案手法

3.1 ニューラルネットワークを用いた手法の課題

ルールベース・統計的な手法は、ソースコード内に含まれる識別子（変数名、関数名などユーザ定義のもの）の情報がよく利用されている。その際に、特に語彙サイズの制限などを行ってはいない。一方でニューラルネットワークを用いた手法では、全ての語彙を含めてしまうと計算量が膨大になってしまうため、語彙サイズを制限することが多い。具体的には、語彙を制限する際に低頻度語をダミートークン（UNK など）に置き換えることが一般的に行われている。一部の識別子は低頻度語であるため、上述の処理によってダミートークンに置き換えられてしまう。しかし、識別子はそれ自体の予測がタスクになる [2] など、重要な情報を持っていると考えられ、その中でも特に出現頻度の低い識別子は、ソースコードの理解に重要な情報を持っていると考えられる。例えば、tmp や array といった変数名は出現頻度が高い一方で、ソースコードが何をやっているかを考える上で重要な情報を持ち得るとは考えにくい。現状のニューラルネットワークを用いた手法で識別子が有効活用されていないという課題は依然解決されないままである。

表 2: Python データセット

データ数	65,288
トークン数	7,980,473
識別子数	3,164,833
ユニークなトークン数	320,855
ユニークな識別子数	197,266
識別子数 (語彙サイズ 50,000)	2,853,759
ユニークな識別子数 (語彙サイズ 50,000)	42,294

3.2 手法

上述した課題を解決するための準備として、以下の 2 つの調査を行なった。まず、既存のデータセットについて識別子の割合や、語彙サイズを制限した時にどれだけ識別子がダミートークンで置き換えられるかを調査することで、データセットの性質を検証した。次に、既存のモデルについて以下の 3 つの場合における性能の変化を比較した。

1. 先行研究と同様の実験設定で語彙サイズを制限し、一部をダミートークンで置き換えた場合
2. 全ての識別子をダミートークンで置き換えた場合
3. BPE (Byte Pair Encoding) [10] を用いて全てのトークンをサブワード分割することで、ダミートークンが生じないようにした場合

モデルの性能の比較は、BLEU[8] を通じて行った。

4 実験・結果

4.1 実験設定

モデルのパラメータやハイパーパラメータなどの実験設定は既存研究 [5, 13] の設定に則った。また、通常の語彙数は 50,000 に設定し、サブワードの語彙数は 16,000 とした。

4.2 データセットの検証

今回用いたデータセット [3] の特徴を表 2 に示す。このデータセットは GitHub[1]から集められたメソッドとそれに付随するコメントのペアを収集したものである。Wan ら [13] もこのデータセットを用いて実験を行なっている。

Wan らは実験において、語彙サイズを頻度の多いものから 50,000 に限定している。その結果、全体の識別子数としては 10%ほどしかダミートークンに置き換えられていないが、ユニークな識別子で考えると元々の識別子のうち 79%が切り捨てられてしまっている (表 2)。また、それぞれ語彙内に含まれる・除外される識別子を観察したところ、高頻度の識別子は tmp や array など抽象度・一般性が高いのに対して、低頻度の識別子は、resource_update や exam_module_genertor など、より具体的な単語の組み合わせで表されるものが多かった。このことからも、語彙内から除外されてしまう低頻度の識別子は要約に必要な情報を持ち得ると考えられる。

[1]https://github.com/

表 3: モデルごとの BLEU スコアの変化

	Usual	All-UNK	Subword
CODE-NN	4.6	4.1	4.7
エンコーダ・デコーダ	5.0	4.5	4.7
Wan らのモデル	0.37	0.39	0.53

4.3 既存モデルの性能変化

CODE-NN、LSTM を用いたエンコーダ・デコーダモデル、Wan らのモデルの 3 つについて、性能の変化を調べた。実験結果を表 3 に示す。表 3 中の Usual、All-UNK、Subword はそれぞれ 3.2 節中の実験設定に対応している。

CODE-NN、エンコーダ・デコーダモデルでは、識別子を全てマスクした状態で学習を行うと BLEU の低下が観察された。上記の 2 つのモデルはソースコードをトークンの系列として受け取っており、識別子をダミートークンに置き換えることによる情報の欠落が大きく影響したと考えられる。しかしながら、識別子をサブワード分割して学習を行っても BLEU スコアの変化はほとんど見られなかった。これは、単純なサブワード分割では識別子の情報を上手く扱うことができないことや、語彙内に含まれる低頻度の識別子は直接的に要約の性能に寄与しないことなどが原因として考えられる。一方で、Wan らのモデルはサブワード

分割にすると、性能の向上が見られた。しかし、識別子を全てダミートークンに置き換えても BLEU スコアに大きな変化は見られなかった。Wan らのモデルはソースコードを木構造に変換した抽象構文木も入力として受け取っているため、ダミートークンの置き換えに対してロバストであったと考えられる。

5 おわりに

今回、ソースコード要約における識別子の重要性を踏まえ、データドリブンな手法では識別子の情報の多くが欠落してしまっていることに着目し、実験を行なった。サブワード分割だけでは必ずしも性能の向上に寄与しない結果にはなったが、ソースコードをトークン系列として受け取る CODE-NN やエンコーダ・デコーダモデルなどのモデルでは、識別子名は一定以上の役割を果たしていると考えられる。

一方でソースコード要約に通底する課題として次のような事実が挙げられる。まず、評価手法の観点で言えば、ソースコード要約は BLEU スコアで性能を評価されることが多いが、実際に BLEU スコアで適切な評価ができているとは限らない。特にソースコード要約はその要約が過不足ない情報を持っているかといった点や、正確性、可読性なども評価すべきである。より多様性を考慮することができ、かつ出力文が有用な情報を持っているかどうかを評価する手法を考える必要がある。次にデータセットの観点で言えば、そもそもデータセットの量や質が不十分な側面があり、全体的な精度が低いという課題は解決されていない。最後に、モデルのアーキテクチャの観点で言えば、現在提案されているモデルの多くは抽象構文木を Tree-LSTM などにそのまま入力している。しかしながら、木構造を扱うアーキテクチャとしてこれらのネットワークが必ずしも適切とは言えない。

上述した評価手法、データセットの検討は重要な課題である。加えて、今後の展望としては、Decoder の事前学習を行いより自然言語として妥当な出力が出るようにすることや、木構造を取り扱うより良い構造を提案することなどが考えられる。

参考文献

[1] Ra'Fat Al-Msie'deen and Anas H. Blasi. Supporting software documentation with source code summarization. *arXiv:1901.01186*, 2019.

[2] Uri Alon, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. *arXiv:1808.01400*, 2019.

[3] Antonio Valerio Miceli Barone and Rico Sennrich. A parallel corpus of python functions and documentation strings for automated code documentation and code generation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, 2017.

[4] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. On the use of automated text summarization techniques for summarizing source code. In *Proceedings of the 2010 17th Working Conference on Reverse Engineering*, p. 35–44, 2010.

[5] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In ACL, 2016.

[6] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In EMNLP, 2015.

[7] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Learning to generate pseudo-code from source code using statistical machine translation. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering*, pp. 574–584. IEEE, 2015.

[8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In ACL, 2002.

[9] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional developers comprehend software? In *Proceedings of the 34th International Conference on Software Engineering*, pp. 255–265. IEEE, 2012.

[10] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1715–1725, 2016.

[11] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K Vijay-Shanker. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pp. 43–52. ACM, 2010.

[12] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In ACL/IJCNLP, 2015.

[13] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S. Yu. Improving automatic source code summarization via deep reinforcement learning. *CoRR*, Vol. abs/1811.07234, , 2018.