ソースコードの文法情報を利用したソースコード要約生成

宮崎 広夢　　鶴岡 慶雅
東京大学 工学部電子情報工学科
{miyazaki,tsuruoka}@logos.t.u-tokyo.ac.jp

1 はじめに

現代社会の高度な情報化に伴い、プログラミングの重要性はますます高まっている。ここで言うプログラミングとは主にソースコードを記述するコーディングを意味する。プログラミングの義務教育への導入も検討されているなど、人々がプログラミングをする機会は増加傾向にある。それゆえ、プログラミングに際して他人の書いたソースコードを読む機会も増加している。しかし、一般に他人の書いたソースコードを読み解くコストは非常に高い。ソースコードそれ自体はある目的を達成するための手段に過ぎず、その書き方は個人差があり、必ずしも可読性が高いとは言い難いためである。コメントやドキュメントの充実によりソースコードをより分かりやすいものにすることは可能だが、それ自体にも決して小さくないコストが伴う。

さて、近年、ニューラルネットワークの登場に伴い、文章の要約や言語間翻訳などといった言語処理に関連する研究が盛んに行われている。様々な言語を対象にして、様々なアプローチで言語処理を行う試みが多くなされている。その一環として、プログラミングのソースコードを対象とした研究も行われている。ソースコードの特徴としては、自然言語とは大きく異なった文法規則を持っていることや、質の良いデータを大量に収集するのが自然言語よりも困難であることなどが挙げられる。

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。本研究で対象としているソースコードの要約 [4] や、自然言語で表された指示からソースコードを生成するソースコード生成 [5, 9]、ソースコードのバグ検出 [1, 3] などが挙げられる。

上述した背景を踏まえ、本研究で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する。具体的には、従来手法で提案されたモデルに対して改良を行い、要約性能の改善を図る。従来手法 [4] では、ソースコードが持ち得る構造的・意味的な情報などを利用していないという問題がある。本研究では、ソースコードをエンコーディングする際にそれらの情報を利用することを目指す。

表 1: ソースコードとそれに対応する要約文

```
public int TextWidth(string text) {
    TextBlock t = new TextBlock();
    t.Text = text ;
    return (int)Math.Ceiling(t.ActualWidth);
}
```

Compute the actual textwidth inside a textblock

2 関連研究

2.1 ソースコードの要約

本研究ではソースコードの要約を目標としているが、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、ソースの文とターゲットの文で扱われる言語は同じである。場合によっては語彙も共通であることも多い。一方、ソースコードの要約は、入力であるソースはプログラミング言語であり、出力は英語である。そのため、要約だけでなく翻訳の要素も含まれる。具体例を表 1 に示す。上段は C#のソースコードであり、下段がそれに対応する要約文である。

2.2 CODE-NN

Iyer ら [4] 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 Atttention [6] を利用した図 1 のようなモデルである。CODE-NN は、エンコーダとデコーダから構成されているが、一般的なエンコーダ・デコーダモデル [2]

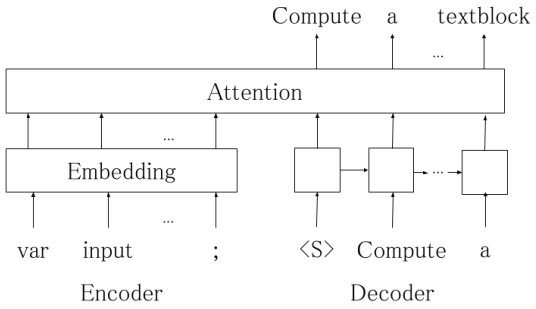

図 1: CODE-NN

とは異なる。エンコーダ側は入力のソースコードを Embedding (入力単語を密なベクトルで表現したもの) のベクトル列に直すだけであり、LSTM を利用しない。デコーダはエンコーダによる出力、すなわち前述した Embedding 列を受け取り、Attention でのみ利用する。デコーダ内の各ブロックは LSTM を表しており、一般的な Attention 付きエンコーダ・デコーダモデルにおけるデコーダと同様に機能する。最終的に出力する単語の確率分布は、Attention を利用して得られた context vector c_t と、デコーダの LSTM からの出力 h_t を用いて、式 (1) で得られる。ここで、W、W_1、W_2 はそれぞれ重みであり、y_t は出力語彙次元のサイズを持つベクトルである。また、損失関数には Negative Log Likelihood (NLL) を用いており、デコーダの出力と正答を比較して損失を計算し、誤差逆伝播法を用いて学習を行う。

$$y_t = \mathrm{Softmax}(W \tanh(W_1 h_t + W_2 c_t)) \qquad (1)$$

2.3 Tree-LSTM

LSTM の 1 種として Tree-LSTM [8] がある。Tree-LSTM は木構造に対して LSTM を利用することができるモデルである。

図 2 に Tree-LSTM モデルの構造を簡単に表す。各ブロックは LSTM ブロックを表しており、x_i はその LSTM ブロックに対する入力を表している。図 2 にあるように、各ブロックは自らの隠れ層の状態 h とセル c を自分の親のブロックに伝播する。伝搬の順は深さ優先探索の順に行う。親であるブロックは子のブロックからそれらを受け取り、また演算を行い自らの親に同様にして伝播する。これにより、木構造をそのまま

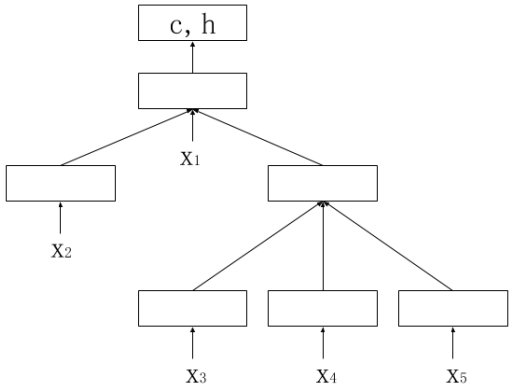

図 2: Tree-LSTM

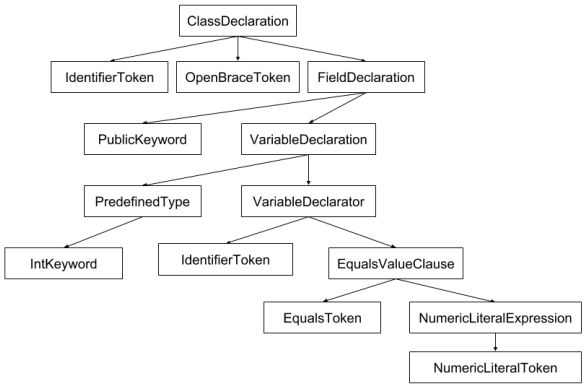

図 3: AST

LSTM を用いたニューラルネットワークとして表現することができる。

3 提案手法

本研究では、CODE-NN に対してソースコードが持ち得る文法情報を盛り込むことで、精度の向上を図る。ここでいう文法情報とは、具体的にはソースコードの持つ階層構造性や各トークンの持つ意味属性などを指し示す。手続き型言語のソースコードは木構造で表現することができ、特に AST (Abstract Syntax Tree) を利用して抽象木を構成することができる。AST の一例を図 3 に示す。AST はソースコードのトークンや宣言などを抽象化したノードで表し、その階層構造を木構造で表現したものである。抽象化の際に各トークンの文脈中の意味属性を表したタグを割り当てることができ、トークンの意味情報も取得できる。また、木構

造により構造的情報を得ることもできる。ソースコードのコンパイルの際にも、一度ソースコードを AST にパースしてから各種の解釈を行うことが多い。

本研究では、CODE-NN のエンコーダを Tree-LSTM に変更し、AST を入力とすることを提案する。まず、データセットのソースコード（C#）を roslyn [1] というコード解析ライブラリを用いて AST に変換し、その AST を Tree-LSTM に入力することで、文法情報をエンコーディングする。roslyn は C#のコンパイラにも利用されており、ソースコードを AST にパースすることができる。Tree-LSTM の root（最も親のノード）の隠れ層およびセルの値をデコーダに渡すとともに、Tree-LSTM 内の全てのノードの隠れ層の値を Attention で重み付けして利用する。これにより、前述したようなソースコードの文法情報を有効に活用したエンコーディングが行える。

4 実験

4.1 実験設定

Iyer ら [4] は StackOverflow [2] という情報技術共有サービスを利用して独自のデータセットの作成を行った。StackOverflow は、質問者が主にプログラミングに関連した質問を投稿し、他者がそれに回答するというサービスである。特に「特定の動作をするプログラムを組みたい」という質問に対する回答として簡便なソースコード断片が投稿されることが多く、これらを利用することでデータセットを作成した。投稿された質問のタイトルを出力すべき要約文とし、回答内に含まれるソースコードが要約すべきソースであるとした。本実験ではデータセットをトレーニング用と BLEU スコア計算用の 2 つに分割し、トレーニング用のデータ数は 43,066、BLEU スコア計算用のデータ数は 5,397 であった。

本研究では、ベースラインである CODE-NN に加え、エンコーダ側を通常の LSTM にしたもの、および提案手法であるエンコーダ側を Tree-LSTM にしたものの 3 つの実装および要約性能の比較を行った。

[1] https://docs.microsoft.com/ja-jp/dotnet/csharp/roslyn-sdk/

[2] https://stackoverflow.com/

表 2: モデルごとの BLEU スコア

モデル	BLEU スコア
CODE-NN（ベースライン）	0.052
LSTM モデル	0.060
Tree-LSTM モデル	0.056

4.2 評価手法

評価手法は BLEU スコア [7] を利用した。具体的にはテストデータ全体を 1 つの文章とみなしたコーパス BLEU を 4-gram に関してまで計算した。ここで、BLEU スコアの値域は $[0, 1]$ である。

4.3 実験結果

実験結果を表 2 に示す。モデルごとの BLEU スコア計算用データにおける BLEU スコアを比較している。

4.4 考察

ベースラインである CODE-NN に比較して、通常の LSTM を利用したモデルは BLEU スコアが向上している。これは LSTM によってトークンの語順情報を扱うことができるようになったほか、デコーダが初期隠れ層状態としてエンコーダの最後の隠れ層を受け取れたためであると考えられる。

エンコーダを Tree-LSTM に変更したモデルについても、ベースラインのモデルよりは BLEU スコアがわずかに向上しているが、通常の LSTM を利用したモデルよりは低くなった。これにはいくつかの理由が考えられる。

1 つ目は、今回はエンコーダ側で Tree-LSTM を利用したが、それにより Attention の対象とする隠れ層の数が 200〜350 程度と大きくなってしまったことがある。これはソースコードのトークン数は 100 以下のものに限って学習を行なっているが、AST に変換するとトークン数の 2〜3 倍程度に隠れ層の数が増加してしまうためである。これにより結果としてデコーダから見た系列長が長くなってしまい、Attention の効果が発揮されにくくなってしまうという問題がある。

2 つ目は、AST が抽象木であるため、ソースコードのトークンそれ自体が持つ情報が抽象化されてしまうことがある。そのため、要約文生成時に必要な情報が欠落してしまった可能性がある。

5 おわりに

エンコーダに LSTM および Tree-LSTM を利用することで、BLEU スコアの向上を確認できた。しかしながら、そもそも各モデルの BLEU スコアが小さく、また学習時のエポックごとの BLEU スコアの変動も激しいため、この向上に大きな差異があるとは言い切れない。また、本研究で利用したデータセットはデータ数が 43,000 程度と少ないほか、ドメインが大きく偏っているという問題もある。より品質の高いデータセットを大量に集めて再実験をした場合に、結果が変わることは大いにあり得る。ソースコードの要約自体、要約と翻訳の両方の要素を兼ね備えており難しいタスクであることは疑いようがない。データセットの改良とともに、自然言語の生成型要約タスクや機械翻訳タスクにおいて提案されている手法についても応用を考慮する必要がある。

参考文献

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. In ICLR, 2018.

[2] 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 EMNLP, 2014.

[3] Hoa Khanh Dam, Trang Pham, Shien Wee Ng, Truyen Tran, John Grundy, Aditya Ghose, Taeksu Kim, and Chul-Joo Kim. A deep tree-based model for software defect prediction. *arXiv:1802.00921*, 2018.

[4] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. pp. 2073–2083. In ACL, 2016.

[5] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. Latent predictor networks for code generation. 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] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In ACL, 2002.

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

[9] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In ACL, 2017.