プログラム言語の構文情報を用いた LSTM によるコメントの自動生成

大西 朔永 [1]　　行本 典弘 [2]　　椎名 広光 [3]
岡山理科大学 総合情報学部 情報科学科 [1,2,3]

i16i014os@ous.jp[1], i16i087yf@ous.jp[2], shiina@mis.ous.ac.jp[3]

1　はじめに

近年，AI やビックデータなど IT 技術の発展に伴い，プログラミングに関心が集まっている．これらの影響により，プログラミングの学習者が増加傾向にある．

プログラミングの学習では，プログラミング言語の文法の学習だけでなく，プログラムに必要な処理とその手順を理解することが重要である．

プログラムのソースコードからコメント（手順）を自動生成することで，プログラムの理解の補助になることを目的としている．プログラムコードの一行単位ではなく，ブロック単位のコメント生成やプログラムの仕様書の作成を最終目的としている．

関連するプログラムのコメント生成に関する研究として，LSTM[1] によるソースコードとコメントのペアを用いたプログラムコメントの自動生成，外部情報を利用したプログラムコメント自動生成 [2] がある．ソースコードからのコメント生成の精度向上を目的とし，外部情報として変数の情報を問題文から取り出し，コメントの生成時に利用している．また，構文情報を用いたコメント生成の先行研究としては，Java プログラムの構文を辿るコメント生成 [3] が提案されている．

本研究では，構文情報を用いて一行単位のプログラムコメント生成を行っている．ソースコードの一行に対応する構文情報とコメントのペアをニューラルネットワークの LSTM を用いた Encoder-Decoder 翻訳モデル [4] により学習し，新規ソースコードのコメント生成を行っている．学習データとして，岡山理科大学総合情報学部情報科学科の講義で使用された C 言語のプログラムを用いている．

プログラムのソースコードのコメント生成には，ソースコードの行を跨いだ依存関係が強く関わると考えられる．本研究ではコード間の関係を深く取得するために，プログラムのソースコードの構文木を入力として，事前に Word2Vec[5] を用いて分散表現を構築し，その

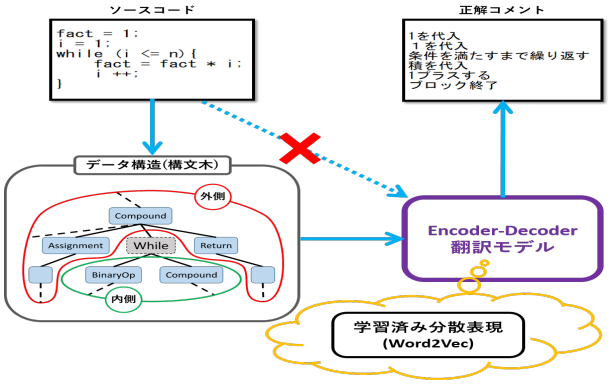

図 1: 構文木を用いたコメント生成の概要

依存関係を捉えた分散表現を Encoder 側の LSTM への入力に用いることで，依存関係を捉えたコメントの生成を試みている．また，依存関係を Word2Vec で捉えることが有効性を検証するために，乱数による初期値を分散表現に用いた場合 (構文情報による初期分散表現) についても実験している．

2　LSTM による構文情報からのコメント生成

Encoder-Decoder 翻訳モデルを利用する場合，入力と出力の単語列を学習に利用する．2 つの表面的な文字列の組を直接的に利用し，プログラムのソースコードからそれに対応するコメントを生成する場合でも，ソースコードのトークン列とコメントのペアを学習するのが通例と考えられる．

一方，本研究では，Encoder 側の LSTM の入力をソースコードのトークン列から構文木のノード列に変更している．ソースコードの 1 行に当たるノード列を入力し，そのソースコードに対応するコメントを正解コメントとし，学習を行った．

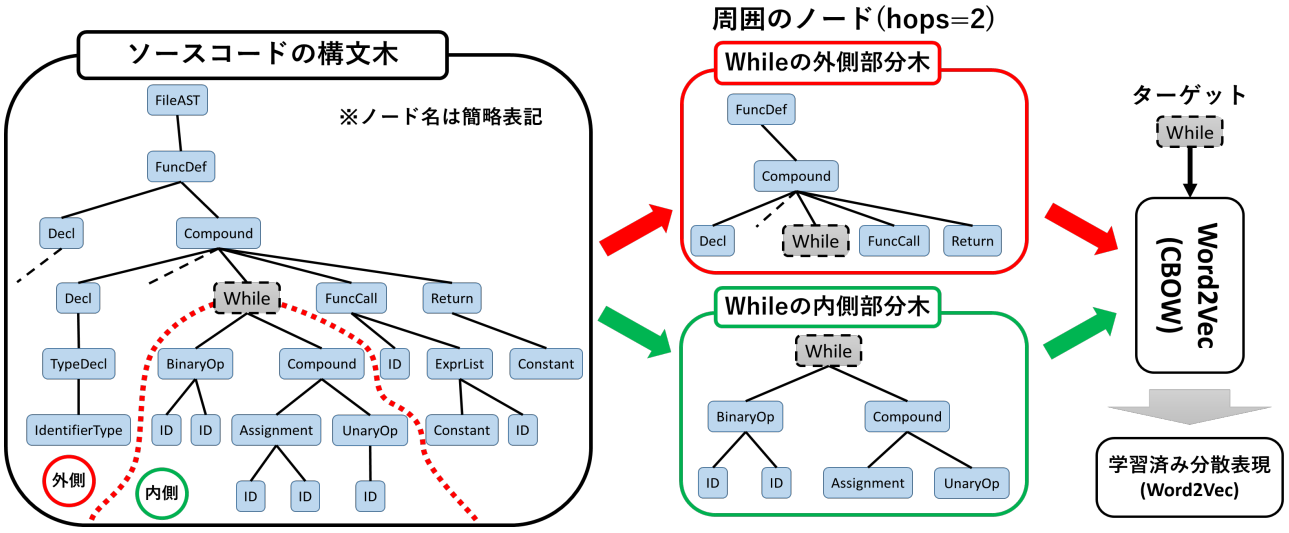

図 2: Word2Vec による構文情報の分散表現の構築

2.1 pycparser(構文解析器) によるソースコードの構文情報の生成

C 言語のソースコードの構文解析をするために，Python のライブラリである pycparser[6] を用いた．生成された構文木 (構文情報) を辿ることで，ノードの内側と外側の構文情報を得ることができる．pycparser を用いて生成した総和を求めるソースコードの構文木を図 2 に示す．図 2 の “While” の左部分木が条件文，右部分木はブロック内の処理に対応する．

2.2 Word2Vec による構文情報の分散表現の構築

2.2.1 構文情報の取得

本研究では，Word2Vec の CBOW モデルを用いて，生成された構文木から各ノードの分散表現の構築を行う．通常，CBOW モデルは Wikipedia などの自然言語のコーパスを用い，コーパス中の各単語を中心として前後 n 単語から中心の単語を推論するタスクを学習する．

それに対して，本研究では，ソースコードから生成された構文木を用い，構文木の各ノードを中心として内側部分木と外側部分木から中心のノードを推論するタスクに変更している．最大の特徴は，通常の CBOW モデルにおける前後 n 単語を，中心のノードまでのエッジ数が $hops$ 以下のノードから成る内側部分木と外側部分木に変更していることである．なお，$hops = 2$ の場合の構文木の内側部分木と外側部分木の例を図 2 に示す．

2.2.2 Word2Vec の学習による分散表現の構築

2.2.1 において述べた処理ノードの内側部分木と外側部分木の構文情報と処理ノードを Word2Vec の CBOW モデルに入力し，学習を行った．ソースコードにおける単語の近さではなく，構文木におけるノードの近さ ($hops$) を考慮することで，プログラムの構造を捉えた分散表現の構築を行う．

2.3 Encoder-Decoder 翻訳モデルによるコメント生成

ソースコードの一行に対応する構文木を辿って得られるノード列とコメントのペアをニューラルネットワークの LSTM を用いた Encoder-Decoder 翻訳モデルで学習させた．Encoder-Decoder 翻訳モデルでは，Encoder 側が入力された構文木のノード列を分散表現に変換し，Decoder 側ではその分散表現をもとに入力された正解コメントを生成できるように学習を進めていく．そして，学習した Encoder-Decoder 翻訳モデルを用いて，新規ソースコードのコメント生成を行う．

Encoder 側の LSTM の入力としてはソースコードに対応する構文木を行きがけ順で辿ったノード列である．

ここで，Encoder 側の LSTM へ入力する各ノードの分散表現に 2.2 で述べた Word2Vec により構築した

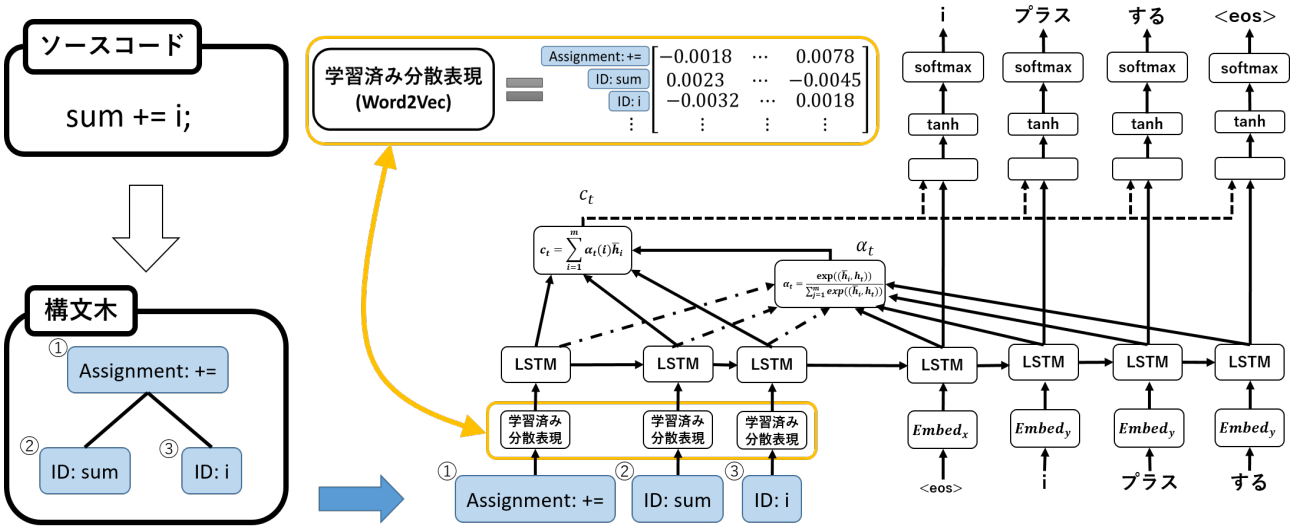

図 3: 学習済み分散表現を用いた Encoder-Decoder 翻訳モデル

分散表現を用いている．構文木におけるノードの近さ (*hops*) を考慮した学習済み分散表現を利用することにより，プログラムの構造を捉えたコメント生成が可能かを実験している．また，依存関係を Word2Vec で捉えることが有効かを検証するために，各ノードの分散表現を Word2Vec で学習せずに，乱数による初期値を分散表現に用いた場合 (構文情報による初期分散表現) も評価している．

表 1: 生成コメントの全体評価

プログラム	ソースコードのトークンモデル	構文情報による初期分散表現	構文情報による学習済み分散表現
P1	4.45	4.58	5.56
P2	3.30	4.03	3.82
P3	4.12	4.83	3.79
P4	4.58	4.17	4.50
P5	4.39	4.77	4.52
平均	4.17	4.48	4.44

3 構文情報を利用した LSTM による生成コメントの評価

3.1 実験環境

学習データとして，大学の情報科学科の講義で使用された C 言語のプログラム 30 個を用いた．評価したモデルは，ベースラインとしてソースコードによる分散表現のモデル，構文情報による分散表現のモデル，構文情報による学習済み分散表現のモデルの 3 種類である．生成コメントの評価は，オープンテストにより，評価している．評価方式については，5 種類のプログラムを対象に，6 人の大学 4 年生による 1 から 6 の 6 段階での評価を行った．

3.2 生成コメントの評価

6 人による評価の平均を表 1 に示す．評価結果としては，ソースコードのトークンを利用したモデルと比較すると構文情報を用いたモデル双方とも評価の平均が高くなっている．Encoder 側の LSTM の入力には，ソースコードに対応する構文木を行きがけ順に辿ったノード列を用いたため，辿り方が評価に影響を与えた可能性が考えられる．また，学習済み分散表現の使用の有無を見ると，使用しない方が微差ではあるが，0.04 高いという結果となっている．構文情報による学習済み分散表現の評価が学習済み分散表現を使用しない場合と比べて，1.04 低いプログラムも存在する．しかし，評価値の差があまり開いていないこと，学習済み分散表現を用いた方が評価が高いプログラムがあることから，構文情報の分散表現の構築方法の工夫により，評価の向上がみられる可能性がある．

表 1 の P1 に対するコメントの生成例と評価を表 2 に示す．4 行目では，ソースコードによる分散表現のモデルよりも，構文情報を用いたモデル双方とも評価が高くなっている．また，7 行目では，ソースコードによる分散表現のモデルは，代入しか取れておらず，構文情報による分散表現のモデルは，数値 1 しか取れ

表 2: コメントの生成例と評価 (プログラム例 P1)

行	プログラム	ソースコードの トークンモデル	評価	構文情報による 初期分散表現	評価	構文情報による 学習済み分散表現	評価
1	int main(void) {	main 関数の宣言	5.83	main 関数の宣言	5.83	main 関数の宣言	5.83
2	int i, n;	整数型変数を宣言	5.83	整数型変数を宣言	5.83	整数型変数を宣言	5.83
3	int fact;	整数型変数を宣言	5.83	整数型変数を宣言	5.83	整数型変数を宣言	5.83
4	printf(“ n= “);	=	2.00	n=を表示	5.50	n=を表示	5.50
5	scanf (“ %d ”, &n);	数値の入力処理	5.83	入力の処理	4.67	数値の入力処理	5.83
6	fact = 1;	1 を代入	5.83	1 を代入	5.83	1 を代入	5.83
7	i = 1;	0 を代入	2.33	1 プラスする	3.67	1 を代入	5.83
8	while(i <= n){	条件を満たすまで繰り返す	5.83	条件を満たすまで繰り返す	5.83	条件を満たすまで繰り返す	5.83
9	fact = fact * i;	積を代入	5.83	積を代入	5.83	積を代入	5.83
10	i ++;	0	1.00	<eos>	1.00	i プラスする	3.50
11	}	整数型変数を宣言	1.50	そうでない場合	1.33	ブロック終了	5.83
12	printf(“ n!=%d\n ”, fact);	n を表示	3.00	n!=fact を表示	5.83	n !=fact を代入	4.67
13	return(0);	返り値 0 を返す	5.83	返り値 0 を返す	5.83	返り値 0 を返す	5.83
14	}	ブロック終了	5.83	そうでない場合	1.00	ブロック終了	5.83
平均			4.45		4.58		5.56

ていないが，構文情報による学習済み分散表現のモデルは，Word2Vec で構文情報を事前学習したことにより，その両方とも含んだコメントをとることができている．

4 まとめと今後の課題

本研究では，ソースコードとコメントのペアではなく，ソースコードの構文木とコメントのペアを対訳データとして学習させ，プログラムに必要な処理とその手順を理解するための手順生成を行った．また，Word2Vec を用いて，構文情報の分散表現を構築し，その分散表現を Encoder 側の LSTM への入力とすることで，プログラムの構造を捉えたコメントの生成を試みた．ソースコードを直接 Encoder 側の LSTM に入力するよりも構文情報を生成し入力した方が良いことが確認できた．また，学習済み分散表現の使用の有無には，大きな差が見られなかった．

今後は，構文情報の分散表現を用いて，構造上の類似性を測り，似ているプログラムを参照し，コメント生成に応用する．プログラムのブロック単位のコメント生成や仕様書の生成が課題である．また，構文情報の分散表現の構築方法においても，入力の順序関係を考慮しない点や内側と外側の違いをあまり考慮できていない点など課題がある．

参考文献

[1] Greff, K. et al., “LSTM: A Search Space Odyssey”, IEEE Transactions on Neural Networks and Learning Systems, Vol. 28, Issue10, pp. 2222–2232, 2017.

[2] Akiyoshi Takahashi, Hiromitsu Shiina, Nobyuki Kobayashi, “Automatic Generation of Program Comments based on Problem Statements for Computational Thinking”, IIAI International Conference on Advanced Applied Informatics 2019, pp.629-634, 2019.

[3] Xing Hu, Ge Li, Xin Xia, David Lo, Zhi Jin, “Deep Code Comment Generation”, ICPC, pp. 200-210, 2018.

[4] Minh-Thang Luong, Hieu Pham, Christopher D. Manning, “Effective Approaches to Attention-based Neural Machine Translation”, EMNLP, pp. 1412-1421, 2015.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, “Efficient estimation of word representations in vector space ”, arXiv preprint arXiv:1301.3781, pp. 1-12, 2013.

[6] Eli Bendersky, “GitHub – eliben/pycparser: Complete C99 parser in pure Python”, https://github.com/eliben/pycparser, 2019-07-15 参照.