

Slice-Based Code Change Representation Learning

Fengyi Zhang, Bihuan Chen, Yufei Zhao, Xin Peng

発表者 花田

2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)

背景

➤ ソースコード

新機能の追加，バグの修正，性能の向上，リファクタリングによって継続的に進化

⇒ コード変更は，ソフトウェア開発やメンテナンスの中核

➤ ソフトウェア開発や保守作業を行う場合

⇒ 開発者がコード変更を理解する必要^[44]

※ コード量が多く，負担が大きい



深層学習でモデルを構築

[44]Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, “How do software engineers understand code changes? anexploratory study in industry,”in FSE, 2012, pp. 1–11.

背景

深層学習をベースとするアプローチ

⇒ コード変更表現学習が重要

- 複数のソフトウェア工学タスクを解決するために提案^[4, 18, 34, 38, 48]

複数のタスクを解決するためのアプローチ,
特定のタスクを解決するためのアプローチの2種類

< 問題点 >

- コード変更をAST（抽象構文木）として表現するため、制御やデータ依存に関するコード変更の学習があまりできない
- ノイズの多いコード変更を考慮しないことがほとんどで、解決したタスクの精度に影響

[4] R. Cabrera Lozoya, A. Baumann, A. Sabetta, and M. Bezzi, "Commit2vec: Learning distributed representations of code changes," SN Computer Science, vol. 2, no. 3, pp. 1–16, 2021.

[18] T. Hoang, H. J. Kang, D. Lo, and J. Lawall, "Cc2vec: Distributed representations of code changes," in ICSE, 2020, p. 518–529.

[34] L. Y. Nie, C. Gao, Z. Zhong, W. Lam, Y. Liu, and Z. Xu, "Coregen: Contextualized code representation learning for commit message generation," Neurocomputing, vol. 459, pp. 97–107, 2021.

[38] S. A. Qureshi, S. Mehta, R. Bhagwan, and R. Kumar, "Assessing the effectiveness of syntactic structure to learn code edit representations," arXiv preprint arXiv:2106.06110, 2021.

[48] M. Tufano, J. Pantiuchina, C. Watson, G. Bavota, and D. Poshyvanyk, "On learning meaningful code changes via neural machine translation," in ICSE, 2019, pp. 25–36.

目的

既存のコード変更表現学習の問題を克服するため

スライスベースのコード変更表現アプローチを提案

変更されたコードに制御，データ依存性を持つ未変更コードを
考慮

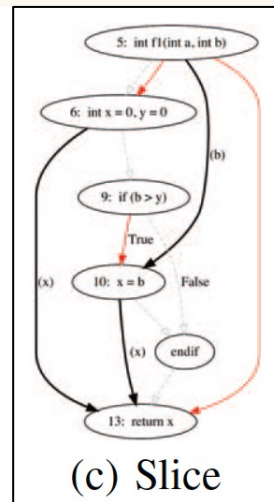
スライス

⇒ プログラム内のある文のある変数(スライス基準)に影響を与える全ての文の集合

➤ プログラムスライスを利用することでプログラムの参照範囲を限定

```
public int f1 (int a, int b)
{
    int x = 0, y = 0;
    while (a > 0) {
        y = a++;
        if (b > y) {
            x = b;
        }
    }
    return x;
}
```

(a) Code



(c) Slice

PDG

Step1.プログラムの各文で定義, 参照される変数を抽出

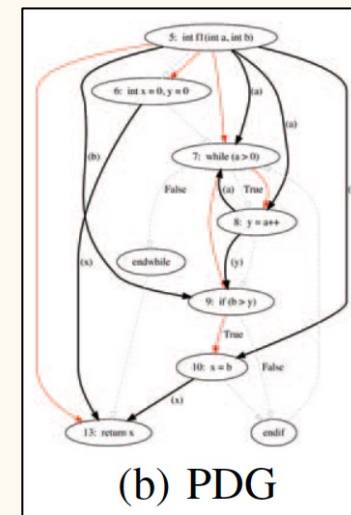
Step2.プログラム文間のデータ依存関係, 制御依存関係を計算

Step3.依存関係解析結果を元に手続きごとに手続き依存グラフを作成

Step4.プログラム中の全ての手続き依存グラフに手続き間の依存情報を組み合わせる



プログラム依存グラフ(PDG)



sparseTransformer model（スパース変換モデル）

⇒ Self-attentionを複数のAttentionに分解することで計算を効率化したTransformer

Transformer

エンコーダとデコーダをAttentionというモデルのみのネットワークアーキテクチャ（深層学習モデル）

Attention

データの中の予測に役立つものに対し重みづけ

Self-attention

入力と出力が同じ言葉で構成されているTransformerで用いられるAttention

提案手法の詳細

- 変更前後のコードのプログラム依存グラフ (PDG) 削除された (追加された) コードに
- 変更前後のコードを, 階層構造を持つスライス, 制御フローパス, ステートメント, コードトークンの列として表現
- 表現に基づき, コード変更に対する事前学習済みスパース変換モデル



CCS2VECを提案

データセット

Javaプロジェクトからコミットを収集

- (1)500以上のスターとフォークがあるプロジェクト
- (2)1,000以上のコミットがあるプロジェクト
- (3)50以上のバグ修正コミットがあるプロジェクト

234Kのコミットと38Kのバグ修正コミットを持つJavaプロジェクト109件を収集

データセットを7：1：2

トレーニング，検証，テストセットに分割(original Data set)

ボットとトリビアコミットメッセージを除去した(Cleaned Data Set)

コミットメッセージ比較

提案モデルが作成したコミットメッセージの品質を比較するため

複数のタスクに対する最新モデルである→cc2vec

cc2vecと比較するため調整を行った提案モデル⇒ccs2vec-SIM

特定のタスクに対する最新モデルである→FIRA

FIRAと比較するため調整を行った提案モデル⇒ ccs2vec-TRAN

TABLE I: Results on Commit Message Generation

Approach	Original Data Set	Cleaned Data Set
CC2VEC	32.2	16.2
FIRA	37.6	20.5
CCS2VEC-SIM	40.3	24.7
CCS2VEC-TRAN	42.4	26.2

generation, and JITLINE [36] and LAPREDICT [60] for the task of just-in-time defect prediction.

※生成されたコミットメッセージの品質を評価するためにBLEU-4スコアを採用

コミットメッセージ比較

CCS2VECは、コミットメッセージ生成のタスクにおいて、BLEU-4スコアで、

CC2VECを少なくとも25.2%↑

FIRAを少なくとも12.8%↑

➤CCS2VECが従来のアプローチと比較して、より意味を考慮した表現を生成

TABLE I: Results on Commit Message Generation

Approach	Original Data Set	Cleaned Data Set
CC2VEC	32.2	16.2
FIRA	37.6	20.5
CCS2VEC-SIM	40.3	24.7
CCS2VEC-TRAN	42.4	26.2

generation, and JITLINE [36] and LAPREDICT [60] for the task of just-in-time defect prediction.

リサーチクエスチョン

事前学習したモデルの有効性を評価するため

RQ1: CCS2VECのダウンストリームタスクにおける有効性は、最先端のアプローチと比較してどの程度なのか

- ダウンストリームタスク（コミットメッセージ生成，バグフィックスコミット特定，ジャストインタイム欠陥予測）

RQ2: CCS2VECの各コンポーネントの有効性への貢献度はどの程度なのか

RQ3: CCS2VECの有効性は設定可能なパラメータに対してどの程度の感度を持つのか

リサーチクエスチョン結果

RQ1

最先端のタスクアプローチPATCHNET ,LPU-SVMを用いる

提案モデル(ccs2Vec)とcc2vec を比較

結果

⇒提案モデルが既存のモデルを上回る

➤ ccs2Vecの有効性を示す.

TABLE II: Results on Bug-Fix Commit Identification				
Approach	Acc.	Prec.	Rec.	F1
CC2VEC + PATCHNET	78.2	77.9	72.1	74.9
CC2VEC + LPU-SVM	65.1	64.5	62.6	63.4
CCS2VEC + PATCHNET	76.5	78.4	78.2	78.3
CCS2VEC + LPU-SVM	69.1	71.9	70.2	71.5

モデルの性能比較

結論

スライスベースのコード変更表現を開発

+

データとコントロールの依存関係を考慮したアプローチ

事前トレーニング済みのparseTransformer modelも開発

ダウンストリームタスクで事前学習したモデルを微調整する実験

CCS2VECが最先端のCC2VECよりも改善されることが実証