Vol.2018-SE-200 No.3
2018/12/3

Stack Overflow投稿を用いた深層学習による自動バグ修正にむけて

高橋 裕太[1,a)]　佐藤 亮介[1,b)]　亀井 靖高[1,c)]　鵜林 尚靖[1,d)]

概要：ソフトウェア工学の分野において自動でバグを修正する研究が盛んに行われている．我々はQ&Aサイトである Stack Overflow に開発者によるバグ修正の知見が集まっていると考えた．本研究では Stack Overflow の投稿データを用いて deepfix と呼ばれる機械学習によるバグ修正手法に適用を試み，適用するにあたってどのような課題があるかを明らかにする．我々は Stack Overflow の投稿データから質問と回答に含まれるソースコードのコードクローンを検出することにより，バグ修正前/修正後のソースコードのペアを作成した．これを学習データセットとして deepfix という研究で用いられた sequence-to-sequence ニューラルネットワークモデルの学習を行った．実際に android に関する SO 投稿 984,533 件を用いて実験を行い，3,924 件のデータセットを作成することができた．また作成したデータセットのうち 903 件を用いて学習を行ったところ，自動的にバグを修正し得る出力は得られなかった．結果よりなぜ期待する学習ができなかったのかの原因を議論を行い，データセット数の少なさや学習する語彙数の多さなどの課題を挙げた．

キーワード：Stack Overflow，Automatic Software Repair，Sequence to sequence

1. はじめに

ソフトウェア開発において，バグ修正は時間を要し開発者にとって大きな負担である．デバッグ作業がソフトウェアの開発コスト全体の 50 %を占めるとも言われている [1]．そのためバグ修正のコストを削減することを目的に，自動バグ修正の研究が盛んに行われている．Luca らによる Automatic Software Repair に関するサーベイ論文 [2] によると年々自動バグ修正に関する論文の投稿が増えており研究者の関心が集まっていることが分かる．

我々は自動バグ修正に Q&A サイトである Stack Overflow の投稿を利用できないかと考える．開発者はバグを修正する際に Stack Overflow を参照することが多くある．Stack Overflow には 2018 年現在 1,600,000 件を超える投稿があり，バグ修正に役立つ記事が多く存在する．そうした背景から Stack Overflow をバグ修正に役立てる研究が行われている．Chen らの研究では Stack Overflow の質問投稿と回答投稿に含まれるコードと，OSS に蓄積されたソースコードとの間のコードクローンを検出することによりバグを検出する手法を提案している [3]．Liu らの研究では Stack Overflow から修正テンプレートを作成する手法を提案している [4]．このように Stack Overflow に着目した研究は多く，バグ修正を手助けする情報が多く含まれていることが期待できる．

我々は Stack Overflow の性質として，質問者と回答者がそれぞれソースコードを記述しているものが多くある点に注目した．あるバグについて困っている開発者の投稿に対して，質問者のソースコードはバグを含むソースコード，回答者のソースコードはそのバグを修正したものであると仮定できる．このようなバグ修正の情報が含まれるコードのペアを使うことにより，自動バグ修正が行えないかと考える．

Stack Overflow 投稿を利用した自動バグ修正のためのアプローチとして，Rahul らが提案した deepfix[5] と呼ばれる sequence-to-sequence ニューラルネットを用いた学習手法を用いる．Rahul らは初歩的な C 言語プログラムの構文的エラーによるバグを対象に，sequence-to-sequence ニューラルネットによる学習を行いた自動バグ修正の手法を提案した．sequence-to-sequence ニューラルネットモデルは文字列を入力に取り，文字列を出力するネットワークモデルであり，主に翻訳など自然言語の学習に用いられ

[1] 九州大学
Kyushu University
[a)] takahashi@posl.ait.kyushu-u.ac.jp
[b)] sato@ait.kyushu-u.ac.jp
[c)] kamei@ait.kyushu-u.ac.jp
[d)] ubayashi@ait.kyushu-u.ac.jp

る．Rahul らはバグを含むソースコード/修正行のソースコードをそれぞれトークン化し学習を行なっている．我々はこの手法を用いて，Stack Overflow から抽出したソースコードをもとに学習を試みる．Stack Overflow に存在しているバグ修正の情報には，論理的エラーに関するものも含まれており，本研究により deepfix での学習手法が論理的エラーに適用できるかどうかを検証することができる．

Stack Overflow 投稿から学習データセットを作成するにあたり，ソースコードの抽出方法が課題になる．Stack Overflow の質問投稿と回答投稿の双方にソースコードが含まれていたとしても，必ずしも後者が前者を修正したものであるとは限らない．そこで質問ソースコードと回答ソースコード間のコードクローンを検出する．コードクローンが見つかれば少なくとも質問ソースコードに変更を加えた (と見なせる) 回答ソースコードを検出することができる．

我々は deepfix の手法を Stack Overflow から作成したデータセットに置き換えて，自動バグ修正の可能性について議論するため以下の Research Question を立てた．

RQ1．Stack Overflow からどのくらいバグ修正情報を含むソースコードを抽出できるか

我々が提案する Stack Overflow からのソースコード抽出手法によって，どのくらいの数のソースコードのペアが抽出できるかの評価を行う．また抽出したソースコードのペアがどの程度正確にバグ修正情報を取得できているかの評価も必要になる．

RQ2．抽出したソースコードから作成したデータセットにより学習を行ったモデルで，バグ修正が可能か

sequence-to-sequence モデルによる自動バグ修正は既存研究において C 言語の構文エラーを対象に行われている．我々が今回対象とする Stack Overflow のソースコードには論理的エラーも含まれており，sequence-to-sequence モデルの学習による自動バグ修正が論理的エラーに範囲を広げても適用できるのかという議論を行うためにこの RQ を設定した．

我々は Java のソースコードを対象に 2008/08/07〜2017/03/13 までの期間に投稿された android に関する Stack Overflow の記事 984,533 件に対して，提案するソースコードの抽出法を適用した．抽出に成功した 3,024 件のうち，コードの単語数が学習に適した長さのもの 1,003 件にふるい分け学習データセットとし，sequence-to-sequence モデルの学習を行なった．どの程度学習が行えたかを確認するために，データセットのうち 100 件をテストデータとし，学習済みモデルからの出力と実際の回答ソースコードを目視で比較した．

結果として正解と同じ操作をしたものは 100 件中 1 件も存在せず，また構文的に誤りのあるコードを出力をしたものは 30 件あった．

良い結果が得られなかった原因として，訓練とテストの問題空間が異なること，学習する語彙数の多さ，データセット数の少なさが考えられた．また得られた結果の妥当性への脅威として，Stack Overflow からバグ修正の情報のみを完全に抽出できていない可能性が挙げられた．

実験で得られた結果を元に，Stack Overflow 投稿を用いて既存の自動バグ修正手法に適用したときなぜ上手くいかなかったか，今後どのようにすれば改善するのかという議論が行うことができた．

以降，第 2 章では関連研究について紹介する．第 3 章では我々が今回用いるアプローチについて述べる．第 4 章では実験とその結果について述べる．第 5 章では今後の課題について述べる．第 6 章では本稿のまとめを述べる．

2. 関連研究

2.1 自動バグ修正

ソフトウェア開発においてバグ修正は非常にコストの高い工程である．そうした背景からバグを自動で修正する研究は盛んに行われている．自動バグ修正は欠陥箇所を特定し，修正の生成・適用を繰り返しテストケースをパスさせる，という過程から成り立つ．修正の生成の方法は原始的な変更による修正とテンプレートを用いた修正に分けられる [2]．原始的な変更による修正はプログラム中の演算子を変換することで修正を試みる手法で，代表的なものに遺伝的アルゴリズムを用いる GenProg[6] や GenProg を派生させた RSRepair[7] などが上げられる．テンプレートを用いた修正はあらかじめ定義してある修正テンプレートを用いることでバグ修正を試みる手法で，人が作成したパッチから抽出したパターンを用いて修正を行う PAR[8] といった手法がある．

本研究で用いる sequence-to-sequence によるバグ修正は，モデルに修正パターンの情報を蓄積することになるためテンプレートを用いた修正に分類することができる．

2.2 Sequence-to-sequence ニューラルネットワークモデル

sequence-to-sequence ニューラルネットワークモデル [9] は，エンコーダとデコーダと呼ばれる二つの Recurrent Neural Network(RNN) から成るモデルである．エンコーダはシーケンスを入力に取り単一のベクトルを出力し，デコーダがエンコーダが出力したベクトルを読み，出力シーケンスを生成する．sequence-to-sequence モデルは長さ n のシーケンスを入力に受け取ると，長さ m のシーケンスを出力することができる．そのため文章の翻訳などに応用されている．本研究で目的とする自動バグ修正というコンテキストにおいての入出力は，入力がバグの含むソースコード，出力が修正後のソースコード (もしくは修正のための操作) に対応させることができる．

また sequence-to-sequence モデルはシーケンスが長くな

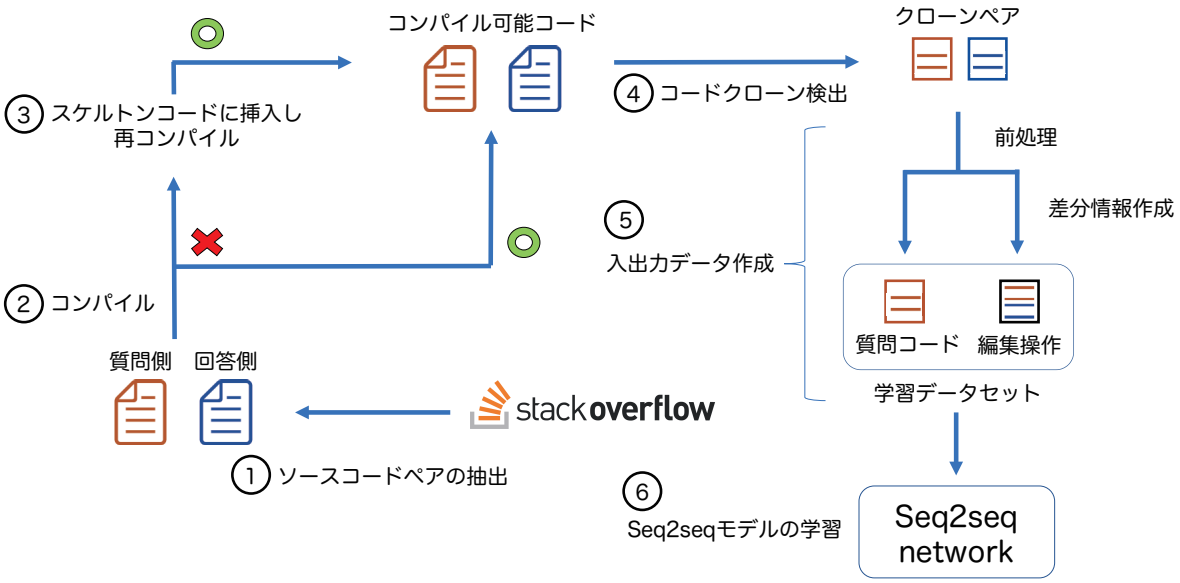

図 1 アプローチ全体の流れ

ると，始めの方に入力されたデータがエンコーダが作成する特徴ベクトルに反映されにくくなるという特徴がある．この課題を解決するために attention モデル [10] と呼ばれるメカニズムが提案されている．attention はデコーダでのシーケンス予測時に，入力シーケンスのどの部分に注目をするかを考慮する手法である．これにより文章翻訳における指示語などの前に出現した単語を示す言葉などを対応させることができる．ソースコードの学習においては，} といったスコープの終わりを示すトークンを，前に出現した { に対応させることができる，といった効果が期待できる．

2.3 Deepfix

Rahul らは deepfix と呼ばれる sequence-to-sequence ニューラルネットと attention を用いた C 言語の自動バグ修正手法を提案している [5]．彼らは 93 個の C 言語のプログラミングタスクに対して，プログラミング入門コースの学生が作成したプログラムにこの手法を適用している．6,971 件のエラープログラムのうち，1,881(27%) 件の修正に成功し，1,338(19%) 件を部分的に修正した．deepfix ではプログラム中に複数のエラーが含まれている場合でも，繰り返し修正を行うことで全てのエラーの修復を可能にする．

また彼らは正しいプログラムからエラーを含むプログラムを 9,230 件生成し，これらに対しても修正を適用している．結果として 5,185(56%) 件の修正に成功している．

彼らの提案手法はプログラミング言語に依存するものではないため，本研究で対象とする Java プログラムにも適用可能である．また彼らは C 言語の構文的エラーのみを対象としていたが，論理的エラーに対しても学習のフェイズはそのまま適用できる．そこで我々は自動バグ修正のモデルの学習に deepfix の学習フェイズの手法を用いる．本研究により論理的エラーに対しても deepfix の手法によりバグ修正し得るかを議論することができる．

3. アプローチ

我々は Stack Overflow 投稿には開発者による論理的エラーを含むバグ修正の知見が多く存在していると仮定し，自動バグ修正のための学習データとして用いることはできないかと考えた．Rahul らの研究では sequence-to-sequence ネットワークモデルを用いた構文エラーに対する自動バグ修正手法が行われており，27%のエラープログラムの修正に成功している [5]．

以上の理由から本研究では Stack Overflow 投稿から質問コードと対応する回答ソースコードを抽出し，sequence-to-sequence ネットワークを用いて学習させる手法を用いる．全体のアプローチの流れを図 1 に示す．なお今回の対象とする言語は Java 言語とする．

3.1 Stack Overflow 投稿の抽出

本節では Stack Overflow 投稿中の文章から，どのようにバグ修正情報を含んだソースコードの抽出を行うかを説明する．

ソースコードペアの抽出 (図 1 の①)：Stack Overflow の投稿は質問投稿と回答投稿から成っており，また本文は自然言語とソースコードで構成されている．ソースコードを抜き出すために，<code>タグを利用する．<code>タグは記事作成者がソースコードを読みやすいレイアウトにするため利用するタグである．本手法では<code>タグに囲ま

```
public class Main {
    public static void main(String args[]) {
        /*insert here*/
    }
}
```

図 2 テンプレートコード

れている文章をソースコードとみなす．質問投稿とその質問に対する回答投稿に含まれるソースコードを抜き出し，ペアを作成する．なお質問投稿一つに対して複数の回答投稿が存在する場合もあるため，ソースコードを含む回答それぞれにつきペアを作成する．

コードクローンの検出：質問投稿と回答投稿からソースコードをそれぞれ抽出しただけでは，後者が前者に変更を加えたソースコードかどうかは分からない．そこで作成したペア全てに対し，コードクローン (type3) の検出を試みる．type3 は文の挿入，削除，変更が行われたコードクローンであり [11]，差分を含む二つのソースコード間でクローンを検出できる．よって type3 コードクローンが検出されれば，二つのソースコードは類似していてかつ差分により互いに変換可能できる．見つかったクローンペアの質問投稿側をバグ修正前のソースコード，回答投稿側をバグ修正後のソースコードと仮定する．

type3 コードクローンの検出を行うためには，ソースコードがコンパイル可能である必要がある．しかし Stack Overflow 投稿に含まれるソースコードはメソッドのみが記述されている場合等，コンパイル可能なソースコードであるとは限らない．そこでまず抽出したソースコードペア全てに対して，コンパイルを試みる (図 1 の②)．その後コンパイル不可能なソースコードに対して，テンプレートコードに挿入することによりコンパイルを試みる (図 1 の③)．図 2 に今回用いるテンプレートコードを示す．

コンパイルが行えたソースコードに対して，コードクローンの検出を行う (図 1 の④)．type3 コードクローンの検出には scorpio[12] を用いる．なお今回は最低 4 行一致しているコードクローンを検出する．一致行数が少なすぎる場合，メソッドの return 部とその次の行の } など，一般的なコードに対してクローンを検出してしまうため，この値を設定した．この段階でコンパイル可能なもののみがふるい分けられるため，投稿中の<code>タグ内に今回対象とする Java 言語以外のソースコードが含まれていたとしても問題なく抽出が行える．

こうしてコードクローンが見つかったソースコードのペアを学習データセットの候補とする．

3.2 データセットの作成とモデルの学習

学習を行うために前節で作成したデータセットの候補から入出力データを作成する (図 1 の⑤)．

```
0 ~ public class addContacts extends ListActivity {
1 ~ @ Override public void onCreate ( Bundle Bundle_Id_1 ) {
2 ~ super . onCreate ( savedInstanceState ) ;
3 ~ setContentView ( R . layout . add_contacts ) ;
4 ~ Uri Uri_Id_1 = Uri . parse ( "StringLiteral" ) ;
5 ~ Cursor Cursor_Id_1 = managedQuery ( Uri_Id_1 , null , null , null , null ) ;
6 ~ String [ ] NoType_Id_1 = new String [ ] { ContactsContract . Contacts .
DISPLAY_NAME , ContactsContract . Contacts . _ID } ;
7 ~ int [ ] NoType_Id_2 = new int [ ] { R . id . contactName , R . id . contactID } ;
8 ~ SimpleCursorAdapter SimpleCursorAdapter_Id_1 = new SimpleCursorAdapter
( this , R . layout . add_contacts , c , columns , views ) ;
9 ~ this . setListAdapter ( SimpleCursorAdapter_Id_1 ) ;
10 ~ }
11 ~ }<end>
```

図 3 入力データの例

```
<add> getListView().setChoiceMode
(ListView.CHOICE_MODE_MULTIPLE);
<before> this.setListAdapter(SimpleCursorAdapter_Id_1);
<end>
<rm> setContentView(R.layout.add_contacts);
<before> super.onCreate(savedInstanceState);
<end>
```

図 4 出力データの例

前処理：質問側回答側のソースコードそれぞれに対して，変数名などを一般化するために型情報を保持して id を割り振る．例えば

ListView foobar;

といった変数宣言があった場合，

ListView ListView_Id_1;

というように変数名を一般化する．

入力データの作成：前処理を行なった質問側のソースコードに対して，各行の先頭に行番号を追加する．その後単語単位でソースコードを区切りトークン化する．この時スペースや改行，コメントなどは削除する．得られたトークン列の最後に列の終端を意味する<end>タグを挿入する．入力データの具体例を図 3 に示す．図 3 は「Listview and SimpleCursorAdapter」というタイトルで質問された投稿 [*1] のソースコードから入力データを作成したものである．

出力データの作成：回答側のソースコードから編集操作を作成する．編集操作は「バグを修正するためにどのような編集を行なったか」という差分情報である．編集操作の定義は以下とする．

- ソースコードを追加する操作を
  <add> ステートメント A
  <before> ステートメント B
  と記述する．これはステートメント B の前の行にステートメント A を挿入するという操作を意味する．
- ソースコードを削除する操作を

[*1] https://stackoverflow.com/questions/13995673/listview-and-simplecursoradapter

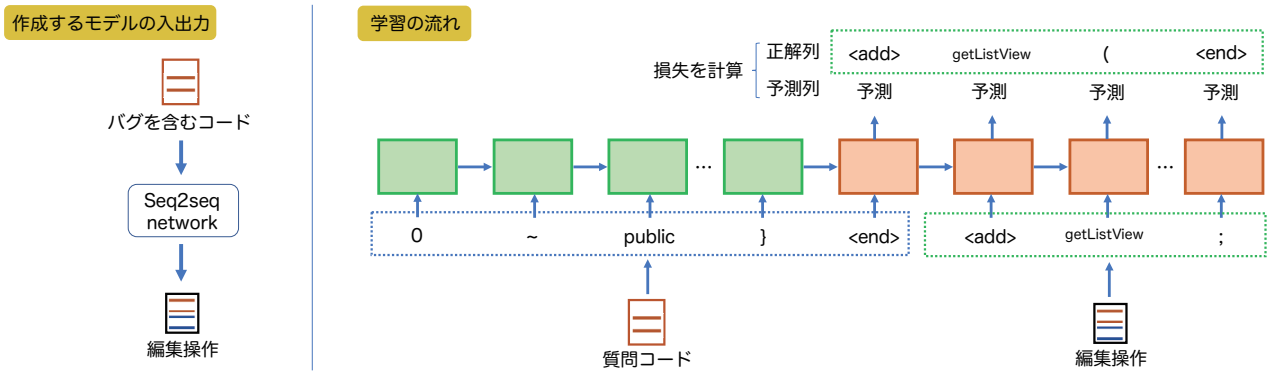

図 5 作成する学習モデルと学習の流れ

表 1 Stack Overflow に投稿されたタグごとの投稿件数の順位 (2018/09/02 時点)

タグ	投稿件数
javascript	1,675,052
java	1,452,980
c#	1,239,428
php	1,223,820
android	1,131,094

<rm> ステートメント A
<before> ステートメント B
と記述する．これはステートメント B の前の行のステートメント A を削除するという操作を意味する．

編集操作の作成のために，前処理前のコードクローンのペアから差分情報を作成する．差分情報に編集操作の定義に沿ってタグの追加・整形を行う．その後入力データと同様にソースコードを区切りトークン化を行い，トークン列の最後に<end>タグを挿入する．出力データの具体例を図 4 に示す．これは図 3 の抽出元の質問投稿に対して答えた回答投稿のソースコードから抽出・作成したものである．

トークンの長さによるふるい分けとモデルの学習：作成した入出力をトークン列の長さが入力側は 75〜450 の範囲，出力側は〜50 までの範囲のものにふるい分ける．これはトークン列が長すぎたり短すぎるものは学習に不向きと考えるためである．なおこのパラメータは deepfix の手法の実験において Rahul らが用いた数値を採用した．

こうして最終的に残ったものを学習データセットとし，sequence-to-sequence ネットワークモデルを訓練させる (図 1 の⑥)．今回学習させるモデルの入出力と学習の流れを図 5 に示す．入力データを sequence-to-sequence のエンコーダ部に，出力データをデコーダ部に入力して学習を行う．

4. 実験と結果

4.1 実験データセット

今回使用する Stack Overflow の投稿データは Stack Exchange で公開されている投稿データを用いた．データセットの作成で対象とする記事は，2008/08/07〜2017/03/13 までに投稿された android タグを持つ投稿 984,533 件とする．android に関する投稿を用いる理由としては，表 1 に示すように Stack Overflow には android に関する投稿が多く，情報の需要が大きいと考えられるためである．

なお android タグを持つものだけを用いたとしても，コードクローンを検出する段階で Java 言語でないソースコードがふるい落とされるため，Java のソースコードのみを抽出できる．

4.2 RQ1 Stack Overflow からどのくらいバグ修正情報を含むソースコードを抽出できるか

RQ1 を検証するために，提案手法により Stack Overflow 投稿からソースコードペアを抽出する．

アプローチ：Stack Overflow の android に関する投稿 984,533 件に対して 3.1 節で述べたソースコードペアの抽出手法を適用する．

結果と考察：対象の 984,533 件の投稿のうち，コードクローンを検出し作成できたソースコードのペアは 3,924 件であった．割合にすると 0.4%程であり，決して高い割合とは言えない．

> android に関する投稿に我々が提案する Stack Overflow から修正前/修正後のソースコードペアを抽出する手法を適用した結果，984,533 件から 3,924 件のコードペアを抽出することができた．

4.3 RQ2 抽出したソースコードから作成したデータセットにより学習を行ったモデルで，バグ修正が可能か

RQ2 の検証のため，前節で作成したデータセットを用いて sequence-to-sequence モデルの学習を行い，目視によるテストを行うことで提案手法によるバグ修正の可能性について議論する．

アプローチ：対象の投稿から質問ソースコードと回答ソースコードの抽出を行なったのち，3.2 節の手法を適用し学習データセットを作成する．トークン列の長さによるふるい分けを行なった結果，1,003 件の学習データセットが

残った．

1,003 件のうち，100 件を除外しテスト用のデータとした．残りの 903 件を用いてモデルの学習を行い，作成したモデルにテストデータ 100 件を入力し，得られた出力と正解の出力を目視で比較する．比較では作成したモデルによりバグ修正が可能かどうかを議論するため，以下の評価項目を設定した．

(1) 正解の修正と意味的に正しい修正が出力されたか
(2) 修正すべき API を用いた出力がされたか
(3) Java の構文的に正しい出力がされたか

結果と考察：テストデータを用いて結果を目視したところ，以下のような結果になった．

(1) 正解の修正と意味的に正しい出力をしたもの: 0 件
(2) 修正すべき API を用いた出力がされたもの: 3 件
(3) Java の構文的に正しい出力がされたもの: 70 件

作成したモデルによるバグ修正は，100 件のテストデータのうち 1 件も正解の修正と一致する修正を出力したものは無かった．

原因の一つとして問題空間を正しく設定できていなかったことが挙げられる．本実験でのテストデータは，1,003 件の学習データから無作為に 100 件を選んでいる．この際，訓練データとテストデータに出現する語彙のスコープを合わせていない．すなわち訓練で登場しない API を用いる修正がテストに含まれている可能性がある．そのため訓練とテストで問題空間が異なるため良い結果が得られなかったと考えられる．

また本実験で対象とした Stack Overflow 投稿に含まれる Java のソースコードは，android タグに絞っているものの，多くの種類のクラスやメソッドを含んでおり，学習の際の語彙数が多い．Rahul らが行なった sequence-to-sequence モデルを用いた自動バグ修正では，問題空間は C 言語の初歩的なプログラムに絞っており，出現する語彙も C 言語を構成する演算子に少量の標準関数を加えたものとなっている．Rahul らの実験での語彙数は 102 であるのに対し，本実験での語彙数は 10,000 を超えている．そのため効果的な学習が行えなかった一つの要因と考えられる．

また正解と使用している API が同様の出力も 3 件あったが，全て findViewById というメソッドを用いるものだった．このメソッドは Android アプリケーション開発において，View インスタンスを検索するメソッドであり，頻出するものである．そのため意味的に正しくこのメソッドを選択し出力した可能性よりも，過学習により出力した可能性が高い．

テストデータ 100 件のうち，30 件は構文的に正しくないプログラムを含む出力をした．出力した構文的エラーを含むコードの例を以下に示す．

```
Paint.put(-.PRIORITY_HIGH_ACCURACY);
```

これは学習データセットの少なさに起因すると考えられる．データセット数を増やすことができれば，コンパイル可能なソースコードのみを用いる本手法ではこのような出力は減ると考えられる．

表 2 投稿内容の内訳

投稿内容	件数
バグ修正	45
実装	65
その他	11

表 3 バグ修正に関する投稿 45 件の修正情報

クローン中に全てのバグ修正情報が含まれるか	件数
全て含む	2
部分的に含む	26
含まない	17

Stack Overflow 投稿から作成したデータセットを元に sequence-to-sequence モデルに学習を行った．結果としては自動バグ修正し得るモデルの作成はできなかった．原因として訓練とテストの問題空間の差，学習する語彙数の多さ，データセット数の少なさが考えられる．

5. 今後の課題

5.1 バグ修正情報を抽出する精度

我々は本手法で抽出したデータセットが，自動バグ修正のための学習データとしてふさわしいものだったかどうかを検証することとした．そこで提案手法でコードクローンを検出したソースコードに対して，抽出できたものがバグ修正に関わるものがどの程度存在し，またクローンを検出した部分のみでバグ修正が完結しているものはどの程度存在するかを調査した．今回用いた 984,533 件の android 投稿のうち，無作為に 10,000 件を選び，本手法でコードクローンのペアを作成した．作成できたペアは 121 件であった．それら 121 件の作成元の Stack Overflow 投稿を目視で調査し，投稿の内容がバグ修正に関するものだったかどうかを調べた．その結果を表 2 に示す．

121 件のうちバグ修正に関する投稿 45 件であった．この 45 件対し，検出したコードクローン箇所にバグを修正するためのすべての操作が含まれているかどうかを調べた．その結果を表 3 に示す．

クローン箇所にすべてのバグ修正情報が含まれている件数は 2 件のみとなった．よって無作為に選んだ投稿 10,000 件のうち，最終的に本研究のデータセットとしてふさわしいものは 2 件しかないことが分かった．これは今回の実験では学習の際に役に立たないデータセットが多く含まれている可能性を示している．これらの役に立たないデータセットを機械的にふるいわけるのは難しく，提案手法の限界を示唆している．

5.2 データセットの不足

今回の実験で学習させた件数は 1,000 件に満たず，深層学習のデータセットとしては数が少ない．また Stack Overflow には似たような投稿は削除される仕組み [*2] があり，特定のバグについての修正ノウハウを複数集めにくい．機械学習の分野ではしばしばミュータントを生成することによりデータセットを増やしている．本手法でもミュータントの生成によりデータセット数を増やす余地がある．

Chen らの研究 [3] では Stack Overflow 投稿内のソースコードの抽出に<code>タグに囲まれた区間だけでなく，<pre>タグによって囲まれた区間もソースコードとして抽出している．これはデータセット提供元である Stack Exchange 公式が推薦しているコードフォーマットの手段であり [*3]，<code>タグと同様にソースコード記述されている可能性が高い．よって<pre>にどの程度のソースコードが含まれるかを調査した上で，ソースコードの抽出を行うことでさらに多くのデータセットが作成できると考えられる．

5.3 編集操作作成時の差分情報の精度

3.1 節で述べた Stack Overflow 投稿抽出のアプローチにおいて，コードクローンペアから編集操作を作成する際，コード間の差分情報を検出している．本実験では差分検出の実装を筆者が行なっているが，精度の検証は行なっていない．よって差分検出する際に誤りのあるものや，本来検出するはずの差分が検出できていない可能性がある．今後の実験においては既存の差分検出ライブラリを用いることで差分検出の信頼度を上げることを考えている．

5.4 Stack Overflow 投稿データの質

Stack Overflow にはユーザや投稿に対して評価を行うことができ，投稿の質として考慮することができる．本実験では質問や回答，または投稿者の質を考慮していない．そのため質の低い投稿を使用している危険性も存在している．

5.5 自動バグ修正の精度

今回行なった実験では自動バグ修正の精度までは検証するに至っていない．よって実際のプロジェクトに含まれるバグをどの程度修正できるかの実験も今後必要になる．

6. まとめ

本稿では Stack Overflow に集められたバグ修正情報に着目し，deepfix の手法を用いて自動バグ修正を行う学習モデルを作成できないかと考えた．Stack Overflow 投稿からどの学習データセットに用いるためのソースコード差分を抽出する方法の提案と，抽出し作成したデータセットを元に sequence-to-sequence モデルを学習させる実験を行なった．実験の結果として 984,533 件の投稿から 3,924 件のソースコードを抽出することができた．また 900 件ほどのデータセットで機械学習を行なったが，そのままでは自動バグ修正に適応できるとは言えない結果を得た．実験を元に Stack Overflow 投稿を deepfix の手法に適用する際の課題を明らかにした．

謝辞 本研究は，JP26240007，18H04097 による助成を受けた．

[*2] https://stackoverflow.com/help/duplicates
[*3] https://meta.stackexchange.com/questions/22186/how- do-i-format-my-code-blocks

参考文献

[1] Britton, T., Jeng, L., Carver, G., Cheak, P. and Katzenellenbogen, T.: Reversible debugging software, *Judge Bus. School, Univ. Cambridge, Cambridge, UK, Tech. Rep* (2013).
[2] Gazzola, L., Micucci, D. and Mariani, L.: Automatic software repair: A survey, *IEEE Transactions on Software Engineering* (2017).
[3] Chen, F. and Kim, S.: Crowd debugging, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, pp. 320–332 (2015).
[4] Liu, X. and Zhong, H.: Mining stackoverflow for program repair, *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, pp. 118–129 (2018).
[5] Gupta, R., Pal, S., Kanade, A. and Shevade, S.: DeepFix: Fixing Common C Language Errors by Deep Learning, *AAAI*, pp. 1345–1351 (2017).
[6] Le Goues, C., Nguyen, T., Forrest, S. and Weimer, W.: Genprog: A generic method for automatic software repair, *Ieee transactions on software engineering*, Vol. 38, No. 1, p. 54 (2012).
[7] Qi, Y., Mao, X., Lei, Y., Dai, Z. and Wang, C.: The strength of random search on automated program repair, *Proceedings of the 36th International Conference on Software Engineering*, ACM, pp. 254–265 (2014).
[8] Kim, D., Nam, J., Song, J. and Kim, S.: Automatic patch generation learned from human-written patches, *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, pp. 802–811 (2013).
[9] Sutskever, I., Vinyals, O. and Le, Q. V.: Sequence to sequence learning with neural networks, *Advances in neural information processing systems*, pp. 3104–3112 (2014).
[10] Bahdanau, D., Cho, K. and Bengio, Y.: Neural machine translation by jointly learning to align and translate, *arXiv preprint arXiv:1409.0473* (2014).
[11] Bellon, S., Koschke, R., Antoniol, G., Krinke, J. and Merlo, E.: Comparison and evaluation of clone detection tools, *IEEE Transactions on software engineering*, Vol. 33, No. 9 (2007).
[12] 肥後芳樹，楠本真二ほか：プログラム依存グラフを用いたコードクローン検出法の改善と評価，情報処理学会論文誌，Vol. 51, No. 12, pp. 2149–2168 (2010).