様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成　３０　年　　６　月　２０　日現在

機関番号：　３２６６５
研究種目：　基盤研究(C)（一般）
研究期間：　2015～2017
課題番号：　１５Ｋ００１０８

研究課題名（和文）ソースコードコーパスを利用したソフトウェア開発支援手法

研究課題名（英文）Software development support using source code corpus

研究代表者
山本　哲男（YAMAMOTO, Tetsuo）

日本大学・工学部・准教授

研究者番号：４０３８８１２９

交付決定額（研究期間全体）：（直接経費）　　3,600,000円

研究成果の概要（和文）：ソースコードを記述していく際，開発者は効率よくプログラムを作成するために既存のソースコードの再利用やライブラリを活用して開発を行う．そこで，本研究では既存のソースコードに記述されているメソッド呼び出し文の順序に着目し，メソッド呼び出し文を補完する手法について提案した．本手法では，回帰結合ニューラルネットワークを利用し，次に現れるであろうメソッド呼び出し文を予測する．さらに，10プロジェクトのオープンソースソフトウェアを用いて補完候補の精度を計測した．実験の結果，典型的なサンプルソースコードの補完においては，38%の精度で補完候補の一位に必要なメソッド呼び出し文が現れることが確認できた．

研究成果の概要（英文）：Developers reuse existing source code or use libraries to develop effectively. In this study, we focus on the order of method invocation statements in existing source code and propose to suggest method invocation statements. This study proposed an approach to suggest method invocation statements using recurrent neural network. I have implemented the approach and conducted experiments to measure an accuracy with 10 open source software projects. I have investigated various parameters of recurrent neural network. This evaluation has shown that our approach is 38% accuracy in API code suggestion, it can correctly suggest the API with top 1 candidate.

研究分野：　ソフトウェア工学

キーワード：　コード補完　コード推薦　RNN

１．研究開始当初の背景

近年，多くのライブラリやフレームワークが開発されており，開発者は効率よくプログラムを作成するために，それらのライブラリやフレームワークを活用して開発を行うことが頻繁にある．例えば，Android アプリケーションにおいて，Android API を利用する割合は，多いアプリケーションで 42%にもなるという報告がある．

そこで，多くの統合開発環境では，作成中のソースコードに対してコード推薦の機能を提供し，調べる手間や入力の手間を省く機能が存在する．これらの機能を用いると，ソースコード中の変数やクラス名などに対して，それらのメソッド一覧を提示することができる．

しかし，メソッドのコード補完では，クラス内だけの情報しか扱わない，候補が多い場合は分かりにくいといった問題点が存在する．さらに，どのクラスを利用すべきか分からなければ，メソッドのコード補完自体が利用できない．ライブラリのクラス名等が分からなければ，自ら処理を書く必要や一覧表などからクラスやメソッドを調べる時間が必要になり，手間がかかることになる．また，扱うクラスやメソッドが分かったとしても，メソッド呼び出し文を一文書けば完成するということは少なく，他の関連するメソッドや他クラスも利用してソースコードを組み上げていく必要がある．つまり，ソースコードを記述していく際には，API の呼び出し順序や次に必要になるであろうクラスやメソッドを知っておく必要がある．

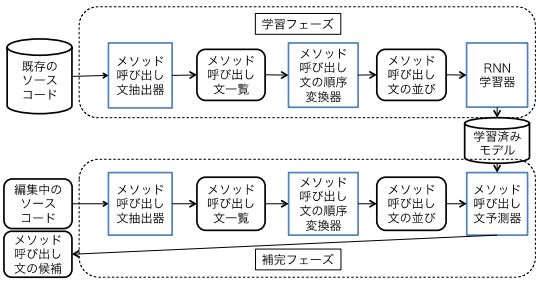
２．研究の目的

そこで，既に記述されたソースコードや開発者が記述中のソースコードから情報を集めて解析し，適切な API やソースコードを推薦する仕組みに関する研究が多く存在する．

これらは，既存のソースコードの情報を利用することで，新規開発する開発者の求めているコード片や API を推薦する．また，API の呼び出し順を情報として利用し推薦する手法も存在する．本研究では二つの API 呼び出し間の関係を情報として利用して，メソッド呼び出し文自体の推薦を行う手法を提案してきており，ニューラルネットワークを利用し，次に現れるであろうメソッド呼び出し文を予測し，候補を表示する手法を構築する．ニューラルネットワークにはリカレントニューラルネットワーク（recurrent neural network）を利用し，既存のソースコードを学習させる．さらに，提案する手法を実装し，サンプルソースコードを用いて実験を行う．

３．研究の方法

本手法を利用したソースコード補完の流れを図 1 に示す．処理の流れは学習フェーズと，作成しているソースコードに対する補完フェーズに分けられる．本手法の最も重要な考



えは，既存のソースコードの中のあるメソッド呼び出し文の並びから次に存在するであろうメソッド呼び出し文の割合を計算し，ソースコード補完をする事である．そのために，事前に既存のソースコードを学習させ，次に存在するであろうメソッド呼び出し文の割合を計算しておく必要がある．

学習フェーズでは，最初にソースコードを解析しすべての呼び出し文を抽出する．その後，メソッド単位で呼び出し文を順番に並べる．この際，後の学習でメソッド呼び出し文を単語として処理するために，メソッド呼び出し文を変換ルールに従った文字列に変換する．さらに，メソッドの区切りを特殊な文字列として挿入する．学習フェーズの最後として，作成したメソッド呼び出し文の並びを文書とみなして，RNN 学習器に入力する．モデル構築するために RNNLM を利用する．

補完フェーズでは，開発者が補完させたいソースコード断片を基に処理を始める．ソースコード断片中から補完したい場所の直前のメソッド呼び出し文を抽出し，学習フェーズで利用した変換ルールに従い，文字列に変換する．そして，予測器と学習済みのモデルを利用してソースコード補完の候補を取得し，開発者に提示する．

４．研究成果

提案手法が実用に耐えうるかを評価するために実験を行った．実用的な速度で実現可能かを検証するためのパフォーマンス評価について記述し，その後，本手法の有効性の評価について考察する．

(1) Adnroid-6.0 SDK に含まれるサンプルソースコード 1,531 ファイルを対象にモデルの構築を行った．その中に計 4,561 メソッド存在した（メソッド呼び出し文が 1 個以下のメソッドは除外）．すべてのメソッドを変換ルールに基づき変換した後の文書ファイルには 33,487 単語（メソッド呼び出し文）存在し，語彙数にすると 6,231 となった．

RNN 学習器は利用して作成し，文書ファイルを NVIDIA GeForce GT 755M 1GB memory GPU で学習させた結果，構築時間に 47 分 43 秒かかり，学習済みモデルのサイズは約 1.9MB になった．その際，隠れ層のユニット数 M を 40，epoch 数を 50，truncated BPTT の長さを 35 としてモデル構築を行った．

次に，大きなアプリケーションの構築に関す

る実験を行った．既存のソースコードとして，2010 年 5 月 22 日にチェックアウトした Eclipse のソースコードを用いた．Java ファイルの総数は 60,265 個，総行数（空行，コメント行を含む）は 10,083,198 である．これらには，Eclipse 本体のソースコードはもちろん，テスト用のソースコードも含まれる．メソッド呼び出し文抽出に 20 分 38 秒にかかり，277,959 メソッド存在した．すべてのメソッドを変換ルールに基づき変換した後の文書ファイルには 2,667,924 単語（メソッド呼び出し文）存在し，語彙数にすると 360,726 となった．隠れ層のユニット数 M を 40，epoch 数を 5，truncated BPTT の長さを 35 としてモデル構築を行った．文書ファイルを NVIDIA GeForce GTX 670 2GB memory GPU で学習させた結果，構築時間に 16 時間 55 分かかり，学習済みモデルのサイズは約 104MB になった．その際，隠れ層のユニット数 M を 40，epoch 数を 10，truncated BPTT の長さを 35 としてモデル構築を行った．

(2) あるサンプルコードを例としてモデルを適用した．適用したモデルは(1) で作成した Android のサンプルで作成したモデルである．結果を表 1 と表 2 と表 3 に示す．それぞれの表はサンプルソースコードの 4 行目の直後，5 行目の直後，8 行目の直後でソースコード補完を実施した際の候補一覧を示している．
表 1 の 8 位に次に現れるメソッド呼び出し文が候補として現れる．表 2 の 2 位に次に現れるメソッド呼び出し文が候補として現れる．表 3 の 3 位に次に現れるメソッド呼び出し文が候補として現れる．ただし，6 行目の後で補完候補を取得した場合は，次に現れるメソッド呼び出し文は 10 位以内には存在しなかった．サンプルソースコードを試した限り，おおむね 10 位以内に候補が出現している．ただし，すべての補完候補において上位に適切な呼び出し文の候補が現れる結果が得られていない．

(3) 手法の有効性を評価するために，以下の手順で実験を実施する．まず，10 個のプロジェクトを訓練用のプロジェクトと評価用のプロジェクトの 2 種類に分類する．分類後，訓練用のプロジェクトを用いてモデルを構築する．そして，構築したモデルを利用して，評価用のプロジェクトのソースコードが補完可能かを評価する．
次に評価方法について説明する．評価用のプロジェクトのソースコードのすべてのメソッドを，ソースコード解析器とメソッド呼び出し文抽出器を用いて，メソッド呼び出し文の並びに変換する．あるメソッドのメソッド呼び出し文の並びが，a_1，a_2，...，a_k，だったとする．まず，a_$まで入力されていたとして，a_2 が補完候補の何位に出現する

表 1 `android.view.LayoutInflater#inflate` の後の呼び出し文候補
Table 1 Method invocation statement candidates after `android.view.LayoutInflater#inflate`

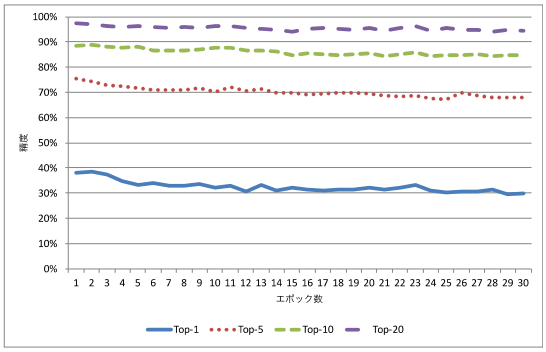
順位	呼び出し文の候補	割合
1	java.lang.String#equals	0.116
2	android.util.Log#isLoggable	0.097
3	android.view.LayoutInflater#inflate	0.075
4	android.content.Intent#<init>	0.060
5	android.widget.TextView#setText	0.032
6	java.util.ArrayList#<init>	0.028
7	GoogleApiClient.Builder#addApi	0.027
8	android.view.View#findViewById	0.025
9	<eom>	0.021

表 2 `android.view.View#findViewById` の後の呼び出し文候補
Table 2 Method invocation statement candidates after `android.view.View#findViewById`

順位	呼び出し文の候補	割合
1	android.view.View#findViewById	0.716
2	android.widget.TextView#setText	0.117
3	android.widget.Button#setOnClickListener	0.016
4	com.example.android.slidingtabscolors.ViewPager#setAdapter	0.010
5	com.example.android.xyztouristattractions.ui.DetailActivity#launch	0.008
6	com.example.android.slidingtabsbasic.ViewPager#setAdapter	0.007
7	android.opengl.GLSurfaceView#setEGLContextClientVersion	0.007
8	android.app.AlertDialog.Builder#<init>	0.006
9	android.view.View#setTag	0.005
10	com.example.android.xmladapters.Adapters.CursorBinder#bind	0.005

表 3 `android.view.View#findViewById` の後の呼び出し文候補
Table 3 Method invocation statement candidates after `android.view.View#findViewById`

順位	呼び出し文の候補	割合
1	android.view.View#getTag	0.339
2	android.view.View#findViewById	0.107
3	android.widget.Button#setOnClickListener	0.088
4	NotificationCompat#<init>	0.052
5	android.text.Editable#toString	0.035
6	java.lang.Integer#toHexString	0.034
7	com.example.android.fingerprintdialog.FingerprintUiHelper#isFingerprintAuthAvailable	0.034
8	android.widget.EditText#setOnEditorActionListener	0.023
9	android.widget.Switch#setChecked	0.018
10	android.widget.EditText#setText	0.016



か計測する．同様に，a_$まで入力されていたとして，a_3 が補完候補の何位に出現するか計測する．最終的に a_k-1 まで計測を実施する．k 個の呼び出し文が並んでいた時は，k-$回の試行を実施することになる．もちろん，a_k-1 までの入力があったときは，a_1 から a_k-1 のすべてが入力されていると考え，順番にモデルに入力する．その後，最終的な出力を a_k の候補とみなして計測する．そして，この計測を評価用プロジェクトのすべてのメソッドについて行う．今回の実験では，補完候補のメソッド呼び出し文の取得する際，クラス名は分かっているものとして計測を行う．これは，通常の統合開発環境と同様の補完と似ている．
android-23 を評価用プロジェクトに，それ以外の 9 つのプロジェクトを訓練用プロジェクトとして計測した結果を図 2 に示す．
隠れ層を 2 層にし，それぞれのユニット数 M を 512 としたネットワークを作成し，epoch 数を 1 回から 30 回までの 30 通りの学習モデルを作成して計測を行った．図 2 の横軸は epoch 数である．Top-k は補完候補の k 番目までに候補が含まれている割合を示している．Top-1 の epoch 数が 2 の時に 38%という割合になっている．android-23 のソースコード中に 8482 カ所補完する箇所が存在し，3249 カ所で補完候補に一位に正解候補が現れたことを示している．
図 2 の epoch 数と精度の関係を見ると，epoch 数を増加させても精度は変わらない，もしくは下がる傾向にある．訓練させすぎることで訓練用データに過剰に適合し，評価プロジェクトから外れる傾向にあることが分かる．訓練データ自体に同じような API の並びが多数あると思われることから，学習モデルを構築する際に epoch 数を多くする必要はないと考えられる．3 割以上の確率で一位に候補が出現することが分かる．また 5 位以内を考えると，7 割になることが分かる．補完候補をすべての箇所で計測したことを考えると，高い精度で補完表示されていると考えられる．

(4) 本手法は RNNLM を利用したコード推薦手法である．ただし，自然言語における段落の扱いを変更して利用している．通常，自然言語の文や段落の順序には意味がある．そこで，RNNLM では区切りを表す特殊文字を利用して文間を区切って学習させている．
一方，クラス内のメソッドの記述順序には決まりがない．そのため，メソッド単位でニューラルネットワークの重みを計算するように変更を加えている．
本実験で最も高い Top-1 精度は 38%であった．一般的に，Top-1 の精度を 100%にするのは困難である．同じような処理を記述する場合でも，組織や開発者でソースコードの書き方が変わるためである．例えば，API の処理の流れとして，open - read - close という処理と open - read - read - close という二種類のソースコードがあった場合を考える．共に，open - read まで記述した場合，推薦してほしい API は，それぞれ，close と read になる．そのため，この二つのソースコードが共に評価用プロジェクトに含まれていた場合，Top-1 精度は 100%にはならない．そのため，Top-5 や Top-10 といった複数の候補の中に必要な API があるかどうかを考えることも重要だと考える．図 2 を見ると，Top-5 で 70%，Top-10 で 80%程度の精度があり，推薦候補の中に必要な API が高い確率で含まれていると考えること出来る．
また，本手法はメソッド内に記述されたメソッド呼び出し文の順序をそのまま利用するため，メソッド内に複数の処理が混在している場合は精度が悪化する場合がある．

(5) 本研究で利用している RNNLM は自然言語処理の分野ですでに提案されており，いくつかの実験を通して評価されているモデルである．そのため，RNNLM 自体は信頼できるモデルであると考えられる．自然言語における単語の並びをメソッド呼び出し文の並びに置き換えて学習させた結果，他のコード推薦手法に比べて高い精度で推薦できる結果となった．ただし，RNNLM の各種パラメーターの値によっては，この結果が大きく変わる可能性がある．
本実験では Android アプリケーションを対象として実験を実施した．そのため，他のドメインのアプリケーションを対象とした場合は精度が異なる傾向になる可能性がある．しかし，本手法はある特定のドメインの API に特化した手法ではなく，一般的に適用できる手法である．ただし，学習済みモデルを作成する際，どのドメインの API 利用例を多く含むソースコードから構築したかによって，推薦精度が変わる可能性がある．学習済みモデルに存在しない API を用いたソースコードを新規に記述する場合は，推薦候補に適切な API が含まれない場合も存在する．そのため，学習済みモデルを構築する際は，新規に開発するソースコード内で利用するであろう API を網羅する学習済みモデルを構築することが望まれる．
今後は，メソッド内の制御構造やオブジェクトに着目した API の呼び出し順序を考慮し，

学習モデルを構築することで精度が向上すると考える.

５．主な発表論文等

〔雑誌論文〕（計　2 件）

1. Tetsuo Yamamoto, Code Suggestion of Method Call Statements using a Source Code Corpus, Proceedings of 2017 24th Asia-Pacific Software Engineering Conference (APSEC2017), pp.666-671, Nanjing, China, Dec 7, 2017. 査読あり
2. 山本哲男, 回帰結合ニューラルネットワークを利用した API 推薦手法, 情報処理学会論文誌 Vol.58, No.4, pp.769-779, 2017. 査読あり

〔学会発表〕（計　3 件）

1. 山本哲男, 分散表現ベクトルを用いたソースコード検索及び分類の検討, 電子情報通信学会技術研究報告, SS2016-71, Vol.116, No.512, pp.67-72, 2017.
2. 山本哲男, 回帰結合ニューラルネットワークを利用した API 推薦手法, 情報処理学会ソフトウェア工学研究会 ソフトウェアエンジニアリングシンポジウム 2016, pp.25-33, 2016.
3. 山本哲男, 機械学習を利用したソースコード補完手法, 電子情報通信学会技術研究報告, SS2015-80, Vol.115, No.508, pp.139-144, 2016.

〔図書〕（計　0 件）

〔産業財産権〕

○出願状況（計　0 件）

○取得状況（計　0 件）

〔その他〕
なし

６．研究組織

(1)研究代表者

山本 哲男（YAMAMOTO Tetsuo）
日本大学・工学部・准教授
研究者番号：40388129