機械学習手法を用いた 動的型付け言語のコード補完に向けて

松永 智將　千葉 滋

本論文では, 機械学習手法を用いて動的型付け言語のコード補完を行う方法について検討する. 多くの統合開発環境やソースコードエディタには, 生産性の向上のためにコード補完機能が組み込まれている. コード補完をする際, 静的型付け言語の場合はレシーバーの型情報をメソッド名の提案に利用することができる. しかし, 動的型付け言語は単純にこのような型情報を利用することは出来ない. そこで, LSTM を利用した言語モデルを用いてメソッドチェイン内のメソッド名の補完を行う.

In this work-in-progress paper, we discuss code completion based on machine learning for dynamically typed languages. Code completion is adopted on major IDEs and source code editors for improving developers' productivity. Although we can use type information for suggesting methods in statically typed languages, we cannot directly adopt this approach for dynamically typed languages. This paper presents code completion for method-call chains by using a language model based on LSTM.

1 はじめに

コード補完機能は, ソフトウェア開発の生産性向上のために重要な機能である. 現在主流の統合開発環境 (IDE) やソースコードエディタの多くには, コード補完機能が組み込まれている. コード補完機能を利用することで, プログラマーはコーディング時間を短縮することや記述ミスを減らすことができる.

ソフトウェアライブラリの記述方法のひとつとしてメソッドチェインスタイルがある. メソッドチェインスタイルは, メソッドの返り値を直接レシーバーとして次のメソッドを呼び出すような記述方法である. 例として, Java 言語で MySQL のライブラリをメソッドチェインスタイルで記述すると以下のようになる.

```
queryBuilder.select("name").from("users
    ").where("id = 1").build();
```

動的型付け言語の場合, レシーバーの型が静的に決定できないため, 補完時に型情報を利用することができない. しかし, 我々はメソッドチェイン上のメソッド列であれば, 次に現れるメソッドを予測し, より正確なメソッド補完を実現できるのではないかと考える.

そこで本研究では, RNN の言語モデルを利用することでメソッドチェインのコード補完を行う. また, 実際に言語モデルを用いたコード補完を IDE やエディタに実装するために何が必要であるかを議論する.

以降の節は次のように構成される: 第 2 節で動的型付け言語のコード補完における問題点について述べる. 第 3 節では, RNN を用いたメソッドチェイン部分のコード補完手法の検証と評価を行う. また, 獲得したモデルをコード補完機能に応用するために考慮すべきことについて議論する. 第 4 節では本研究の関連研究について述べる. 第 5 節では本論文のまとめと今後の課題について述べる.

Towards Code Completion for Dynamically Typed Programming Language using Machine Learning

Tomomasa Matsunaga, 東京大学大学院情報理工学系研究科創造情報学専攻, Graduate School Information and Science Technology, The University of Tokyo.

2 背景

コード補完の提案内容の中でも重要なものに「呼び出し可能なメソッド名の提案」がある．InteliJ [9] などの IDE では, プログラマーがメソッド呼び出しを記述する際にコード補完が利用できる．具体的には, レシーバーとドットを記述した時点で呼び出し可能なメソッド名が複数提案される．このとき, 呼び出し可能なメソッドはレシーバーの型情報をもとに決定される．プログラマーは提案されたメソッド名の中から目的のものを選択することでコードの一部を補完することができる．これにより, プログラマーには以下のような利点がある．

i). レシーバーの型やメソッド名を覚えていなくてもメソッド呼び出しを記述できる

ii). 間違ったメソッド呼び出しをコンパイル前に防ぐことができる

一般的に, メソッドを記述する機会は多いためメソッド名の提案は開発効率に大きな影響を与える．

一方, 近年, メソッドチェインスタイルの API をもったライブラリが増えている．このようなライブラリでは, 静的型付け言語であれば戻り値の型を工夫することでコード補完の効果を高めることができる [3, 12]. 例として, MySQL をドメイン特化言語として提供するようなライブラリを作成することを考える．FROM 句が複数記述されると MySQL の文法として正しくないため, 以下のようなメソッド呼び出しはプロトコル違反であるといえる．

```
queryBuilder.select("name").from("users
    ").from("posts").build();
```

このような誤った API の利用はエラーとして検出するべきである．通常, 呼び出せるメソッドの種類ごとに返り値を切り替えることで API の利用方法を制限する．つまり, from メソッドは from メソッドを持たないオブジェクトを返すことで上記のプロトコル違反を防ぐことができる．

```
// 返り値は from メソッドを持たない
queryBuilder.select("name").from("users
    ");

// 型エラーが発生
queryBuilder.select("name").from("users
    ").from("posts");
```

これにより, 誤った API の利用をした際に静的型付け言語の場合は型エラーとしてプロトコル違反を検出することができる．また, 動的型付け言語の場合は実行時エラーとしてプロトコル違反が検出される．

静的型付け言語において, メソッドチェイン API はコード補完との相性が良い．レシーバーの型情報からプロトコルに違反しないようなメソッド名のみを提案することができるためである．これにより, プログラマーは補完候補から目的のメソッドを選択していくだけでライブラリを使用することができる．しかし, 動的型付け言語の場合はレシーバーの型を実行時まで決定することができないため, 型情報をもとにしたメソッドの提案を行うことができない．そのため, 動的型付け言語のコード補完を行う場合, 型情報以外の情報を利用して提案するメソッドを決定する必要がある．

型情報に頼らずにコード補完を行うために, 既存のソースコードを用いて機械学習手法を適用する方法が知られている．コード補完では, 既に記述されているトークン列をもとに次に記述できるトークンを提案する．そのため, 既存のトークン列 $t_0 \dots t_{n-1}$ を入力として次に記述されるトークン t_n を予測できればコード補完に応用可能であると考えられる．このような予測を行うためには, 言語モデルが利用できる [7]. 自然言語のみならず, ソースコードを対象としたタスクにおいても言語モデルを用いた研究が行われている [4, 5, 11]. コード補完のタスクでは, 言語モデルを用いて Eclipse 上で Java のコード補完を行う CACHECA などが提案されている [2].

3 メソッドチェイン部分を対象とした RNN を用いたメソッド提案

メソッド名の補完において, 我々は, プログラムからメソッドチェイン部分だけを抜き出して言語モデルを獲得することで, より精度の高いメソッド名の補完が実現できるのではないかと予想している．本研究では, 予備的実験としてメソッドチェインの言語モ

デルを獲得することで JavaScript コードのメソッド補完を行う．そのために, メソッドチェインを構成するメソッド名の列 $m_0 \ldots m_{n-1}$ から次に呼ばれるメソッド名 m_n を予測するような言語モデルの獲得する．具体的には, Long short-term memory を用いてメソッドチェイン部分の言語モデルの学習を行う．また, 獲得したモデルを用いてメソッド提案機能としての性能を評価する．学習データには, GitHub から収集した JavaScript コードから, メソッドチェインを構成するトークン列のみを抜き出して利用する．3.5 節では, コード補完機能としての実用を目指すにあたって考慮すべきことについて議論する．

3.1 データセット

本論文では, 主要な動的型付け言語のひとつである JavaScript を対象としてモデルの学習及び評価を行う．データセットを作成ために, 以下の手順で処理を行った．

i). GitHub から JavaScript ファイルを取得
ii). メソッドチェイン部分のみを抽出
iii). メソッド名の列の作成

以降で各項目の詳細を述べる．

i) **GitHub から JavaScript ファイルを取得**: GitHub から特定のライセンス †1 に従う JavaScript リポジトリを 546 リポジトリ取得した．その後, 取得したリポジトリから拡張子が “min.js” でない JavaScript ファイルを抜き出した．最終的に, 構文解析に成功した 123356 ファイルのみを以降の処理の対象とした．ここで, 構文解析には babel-parser†2 を利用している．

ii) **メソッドチェイン部分のみを抽出**: 各 JavaScript コードからメソッドチェイン部分を抜き出す処理を行う．メソッドチェインは, 途中状態を変数に保持することで分割して記述することができる．例えば, 図 1 のコード例は以下のように記述することで同等の処理が可能である．

```
select("name").from("users").where("age
    >= 20").limit(100);
```

図 1　収集される JavaScript コード例

```
let state1 = select("name");
let state2 = state1.from("users");
let state3 = state2.where("age >= 20");
state3.limit(100);
```

このようなチェインを分割する記述方法は, 可読性や再利用性の観点から度々行われる．我々は, 上記のような複数のメソッド呼び出しは単一のメソッドチェインとして学習データに取り込むべきであると考えた．そこで, JavaScript の静的スコープを考慮して変数代入を追跡することでメソッド呼び出しの連結を行った．そして, 連結されたメソッド呼び出し列が二回以上のメソッド呼び出しから成るものを学習に使用するメソッドチェインとした．

iii) **メソッド名の列の作成**: メソッドチェインを構成する各メソッド呼び出しのメソッド名を順に並べた列を入力列とした．例として, 図 1 のコード例からは以下のようなメソッド名の列が作成される．

```
select , from , where , limit
```

ここで, データセット内で出現回数が少ないメソッド名は未知語としてまとめている．そのため, データ作成時には最小の出現頻度をパラメータとして指定する．

3.2 モデル選択

言語モデルには, n-gram や Reccurent neural network(RNN) [10] を利用したものが存在する．本論文では, RNN の一種である Long Short-Term Memory(LSTM) [8] を言語モデルとして利用する．LSTM を利用した言語モデルは, 品詞タグ付けや機械翻訳などの自然言語処理のいくつかのタスクで良い性能を示している．本研究では, メソッド名の列からその直後に続くメソッド名の尤度を計算できるようなネットワークを学習・評価に使用する．具体的には, 埋め込み層, LSTM 層, 全結合層, そして Log Softmax 層を

†1 MIT, Apache-2.0, MPL-2.0, BSD-2-Clause, BSD-3-Clause, BSD-4-Clause, MS-PL

†2 https://github.com/babel/babel/tree/master/packages/babel-parser

表 1　図 1 から生成される入力データ (入力系列長 2)

入力系列	正解ラベル
select, from	where
from, where	limit

表 2　学習に使用したパラメータ

メソッド名の埋め込み次元数	300
バッチサイズ	64
エポック数	10

順に繋げて構成されたネットワークを用いる.

3.3　モデルの学習

学習時には, メソッド名の列と正解ラベル (直後に続くメソッド名) を入力データとする. このような入力は, 一つのメソッドチェインから複数作成される. 例として, 入力系列長を 2 とした場合に, 図 1 のコード例から生成される入力系列と正解ラベルを表 1 に示す. ここでは, メソッド名の列から指定された長さだけ抜き出すことでデータを作成する処理が, 開始位置をずらしながら順に行われている. また, 学習時に使用したパラメータを表 2 に示す. 損失関数には Negative Log Likelihood を利用し, 最適化アルゴリズムには Adam を利用した.

3.4　評価

本論文では, top k accuracy と mean reciprocal rank(MRR) を評価指標として用いる. top k accuracy は, 提案順位の上位 k 個に正解が含まれている確率を算出する指標である. MRR は正解順位の逆数の平均を算出する指標である. 順位の逆数を求める際に正解が順位リストに含まれない場合は, 逆数ではなく 0 として計算する. top k accuracy と MRR は 0 から 1 の値をとり, 値が 1 に近いほど正しい提案が行えていることを表す.

評価実験を行った結果とモデルの学習・評価に用いたデータ数を表 3 に示す. ここで, L は使用したデータの系列長, F はメソッド名の最小出現頻度を表す. また, MRR を求める際の順位リストは長さ 10 としている. 評価データ数が最小出現頻度ごとに異なるのは, 未知語が正解ラベルとなるようなデータを評価時に取り除いているためである. 実験結果では, 学習の入力系列が短い方が top 5・top 10 共に精度が高くなっている. また, 今回の実験の範囲では最小出現頻度による差異はあまり出ていないようである.

実験結果から, L=2, F=2 の場合は 4 回に 1 回程度正解が 10 位以内に現れることが期待される. この結果と MRR 値が 14.7% ≈ 1/7(平均順位が 7 位) であることから, 予測が正解する場合は順位が高くなっていることが読み取れる.

3.5　議論

3.5.1　異なるコード補完機能との共存

本論文で言及した手法は, メソッドチェイン部分のみを学習することでメソッド名の予測を行うものであった. 通常、コード補完機能にはメソッド名の補完の他にもいくつかの補完パターンが存在する. 例えば, プログラミング言語の文法を利用して if 文や for 文などのコードスニペットを提案するものや, 同じファイル内に記述されている識別子を途中まで書いた時点で補完するものなどがある. これらの機能は, 本研究で取り組んだメソッド名の補完と独立に実装して IDE やソースコードエディタに組み込むことが可能である.

3.5.2　未知語への対応

言語モデルを用いてメソッド提案をする場合, 補完候補にできるメソッド名は学習データ内に出現したもののみである. つまり, 標準ライブラリや利用者の多いライブラリで定義されているメソッド名は補完できるが, 現在のプロジェクト内で定義されたメソッド名などは補完できない.

この問題の解決策のひとつとして, 自然言語の分散表現を利用する方法が考えられる. 一部の例外を除いて, メソッド名は自然言語の単語の組み合わせによって表されている. 例えば, containsKey というメソッド名は contains と key という二つの英単語の組み合わせである. 自然言語として似た文脈を持つ単語同士では, メソッドとしても近い機能を持つことが予想できる. そこで, 自然言語のコーパスで学習された分散表現をメソッド補完に利用できると考えた. 自然言語

表 3　LSTM を用いたメソッド予測の結果と学習・評価に使用したデータ数 (L=入力系列長, F=最小出現頻度)

	Top 5 Accuracy	Top 10 Accuracy	MRR	学習データ数	評価データ数
L=2, F=2	20.0%	26.6%	14.7%	481379	118608
L=2, F=10	20.2%	27.8%	14.6%	481379	115534
L=5, F=2	9.8%	11.9%	7.2%	120332	29642
L=5, F=10	9.3%	11.5%	6.8%	120332	28949

では学習済みの単語の分散表現がいくつか公開されている. 未知のメソッド名と既知のメソッド名の距離をこのような分散表現を利用して求めることで, メソッド補完を拡張することが可能であると考えられる. このような補完対象の拡張は今後の課題である.

4　関連研究

Franks らは, Eclipse プラグインとして言語モデルに基づくコード補完を行う CACHECA を提案している [2]. 言語モデルには n-gram を拡張した$-gram という言語モデルを利用し, 既存の Eclipse のコード補完と組み合わせることでコード補完機能を実現している. このような既存の言語モデルに対して, メソッドチェインを対象とした場合の提案精度の比較を行う必要がある.

Visual Studio 上での実際のユーザーの行動に基づいて, コード補完機能を評価する研究も行われている [6]. この研究では, C#リポジトリを対象として学習を行い, Best matching neighbor [1], RNN, n-gram の各モデルについて補完精度を比較している. 得られた知見のひとつとして, 現在のプロジェクト内のファイルなどのローカルな情報を補完に利用することが重要であると述べられている. そのようなローカルな情報を用いた拡張については本研究では未検証である.

Hellendoorn らは TypeScript の型情報を学習データに使用することで JavaScript 上で型の推論を行う手法を提案している [5]. この手法を用いてレシーバーの型を推定することができれば, メソッドの提案に応用することが期待できる. この場合, 未知のメソッド名を提案できないという問題は起こらない. しかし, 学習時に登場しない型のインスタンスをレシーバーとするようなメソッドは提案できないという問題が起こると考えられる.

5　まとめ

本論文では, 言語モデルを用いてメソッドチェイン上でメソッドを予測する手法について検討した. 言語モデルを利用する手法では, レシーバーの型情報を必要としないため動的型付け言語のコード補完にも応用可能である. 今後の課題として, メソッドチェイン部分を抜き出した場合とそうで無い場合の比較を行うことでその差異について議論する必要がある. また, n-gram モデル等の別の言語モデルを用いた場合との比較も行う必要がある. データセット内に現れないメソッド名の補完についても解決方法の検討が必要である.

参 考 文 献

[1] Bruch, M., Monperrus, M., and Mezini, M.: Learning from Examples to Improve Code Completion Systems, *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, New York, NY, USA, ACM, 2009, pp. 213–222.

[2] Franks, C., Tu, Z., Devanbu, P., and Hellendoorn, V.: CACHECA: A Cache Language Model Based Code Suggestion Tool, *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, Piscataway, NJ, USA, IEEE Press, 2015, pp. 705–708.

[3] Gil, Y. and Levy, T.: Formal Language Recognition with the Java Type Checker, *30th European Conference on Object-Oriented Programming (ECOOP 2016)*, Krishnamurthi, S. and Lerner, B. S.(eds.), Leibniz International Proceedings in Informatics (LIPIcs), Vol. 56, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, pp. 10:1–10:27.

[4] Gu, X., Zhang, H., and Kim, S.: Deep Code Search, *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, New York, NY, USA, ACM, 2018, pp. 933–944.

[5] Hellendoorn, V. J., Bird, C., Barr, E. T., and Allamanis, M.: Deep Learning Type Inference, *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, New York, NY, USA, ACM, 2018, pp. 152–162.

[6] Hellendoorn, V. J., Proksch, S., Gall, H. C., and Bacchelli, A.: When Code Completion Fails: A Case Study on Real-world Completions, *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, Piscataway, NJ, USA, IEEE Press, 2019, pp. 960–970.

[7] Hellendoorn, Vincent J. and Devanbu, Premkumar: Are Deep Neural Networks the Best Choice for Modeling Source Code?, *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, New York, NY, USA, ACM, 2017, pp. 763–773.

[8] Hochreiter, S. and Schmidhuber, J.: Long Short-Term Memory, *Neural Comput.*, Vol. 9, No. 8(1997), pp. 1735–1780.

[9] JetBrains s.r.o.: InteliJ IDEA, `https://www.jetbrains.com/idea/`.

[10] Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S.: Recurrent neural network based language model., *INTERSPEECH*, Kobayashi, T., Hirose, K., and Nakamura, S.(eds.), ISCA, 2010, pp. 1045–1048.

[11] Mou, L., Men, R., Li, G., Zhang, L., and Jin, Z.: On End-to-End Program Generation from User Intention by Deep Neural Networks, *CoRR*, Vol. abs/1510.07211(2015).

[12] Nakamaru, T., Ichikawa, K., Yamazaki, T., and Chiba, S.: Silverchain: A Fluent API Generator, *Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, GPCE 2017, New York, NY, USA, ACM, 2017, pp. 199–211.