LSTMを用いたソースコード内の演算子推定手法

舟山 優
京都工芸繊維大学 大学院工芸科学研究科
博士前期課程 情報工学専攻
y-funayama@se.is.kit.ac.jp

水野 修
京都工芸繊維大学 情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

統合開発環境（*IDE*）には関数名や変数名の候補を表示するなどの補完機能が備わっているものが多い．このような機能は，素早くソースコードを記述したり，不具合の混入を抑制したりするなど，ソフトウェア開発の生産性を向上させている．このような生産性に関連する研究は数多く行われている．

本研究ではソースコード中の演算子に関する有用な情報をユーザに提示することを目的とする．有用な情報の例としては，演算子の補完機能や，ソースコード中の不適切な演算子の検出が考えられる．ソースコード中の不適切な演算子とは，例えば ”`if(a > 5)`” とするべきところを ”`if(a >= 5)`” としてしまった場合などである．目的達成のための第一歩として，本研究ではソースコード中のある箇所の演算子の種類が不明な場合に，その箇所に最も当てはまる演算子を推定する手法を提案した．*Java* で記述されたソースコードをトークンの並びに変換し，それをもとに欠落した演算子を推定する *LSTM* を用いた機械学習モデルを作成した．*LSTM* を用いたモデルを *3* つ作成し，それらのモデルを用いた手法と欠落した演算子をランダムに推定する手法で実験し，考察を行った．実験の結果，最大で約 *72%*の正解率を得られた．この結果から，ソースコードには欠落した演算子の特定に有用な特徴が含まれており，*LSTM* を用いることでそれらの特徴を学習できると考えられる．

1. はじめに

統合開発環境（IDE）には補完機能が備わっているものが多い．補完機能とは，文字列を入力しているときに次に連なる字句を推測して候補を提示する機能である．この機能は，開発時にソースコードを記述する速度を向上させたり誤字を減らしたりすることなどを目的として導入される．

ユーザに対して有用な情報を提示することに関して様々な研究が行われている．例えば，Yang ら [1] は，プログラムを修正した際にその修正した要素と類似したものを抽出し，次に修正する可能性がある部分を強調して表示するツール SimilarHighlight を作成した．このツールを使用することで，プログラム開発の生産性が向上したり，レビューをする際に簡単に類似の要素を見つけることができるようになった．また，山本ら [2] は，ユーザが開発作業を中断することなく，必要に応じてソースコードを再利用できる手法を提案した．過去のソフトウェアに含まれる既存のソースコードをコーパス化してデータベースを作成し，ユーザが書いている途中のソースコードの後に続くソースコードとして，過去のソフトウェアの中で確率が高かったものを提示する．

本研究では，ソースコード中の演算子に関する有用な情報をユーザに提示することを目的とする．最終的にはソースコード中の任意の箇所を推定し提示することを目指すが，その第一歩としてまずは推定の対象を演算子のみに限定し，ソースコード中のある箇所の演算子の種類が不明な場合に，その箇所に最も当てはまる演算子を推定する手法を提案する．「演算子の種類が不明なソースコード中のある箇所」を，これ以降便宜的に「空欄」と呼称する．将来的には，演算子の補完機能の開発や，ソースコード中の不適切な演算子の検出などに貢献できると考えられる．

演算子を推定する方法として，本研究では Long short-term memory（LSTM）を用いる．LSTM は自然言語の

分野などで利用されている．本研究で作成した機械学習モデルは，ソースコードの一部をトークン列へ変換したものを LSTM への入力とすることで空欄に当てはまる演算子を推定する．作成したモデルを用いて，空欄より前の部分のソースコードを入力とした場合，空欄より後の部分のソースコードを入力とした場合，その両方を入力とした場合，ソースコードを入力とせず出現頻度で重み付けされた確率に基づいてランダムに推定した場合の 4 つの実験を行い，比較した．

2. 研究の目的

本研究の目的は，ソースコードに関する有用な情報をユーザに提示するため，ソースコード中の任意の箇所を推定し提示することの第一歩として，ファイルに含まれるソースコードの情報をもとに，ソースコード中のある箇所の演算子の種類が不明とされた場合に，その箇所に最も当てはまる演算子を LSTM を用いて推定することである．

本研究では以下に示す研究設問について検証を行う．

RQ1: 空欄より前の情報から空欄に入る演算子をどの程度の精度で推定できるか．

RQ2: 空欄より後の情報から空欄に入る演算子をどの程度の精度で推定できるか．

RQ3: 空欄の前後の情報から空欄に入る演算子をどの程度の精度で推定できるか．

RQ4: LSTM に与える情報によって推定した結果に差異はあるか．

3. 準備

3.1. 演算子

本研究では Java で記述されたソースコードを対象とする．Java には数多くの演算子があるが，その全てを扱うと問題が複雑になるため，二項演算子である

`+`，`-`，`*`，`/`，`%`，`<`，`>`，`<=`，`>=`，`==`，`!=`，`&&`，`||`

の 13 種の演算子を実験対象とした．

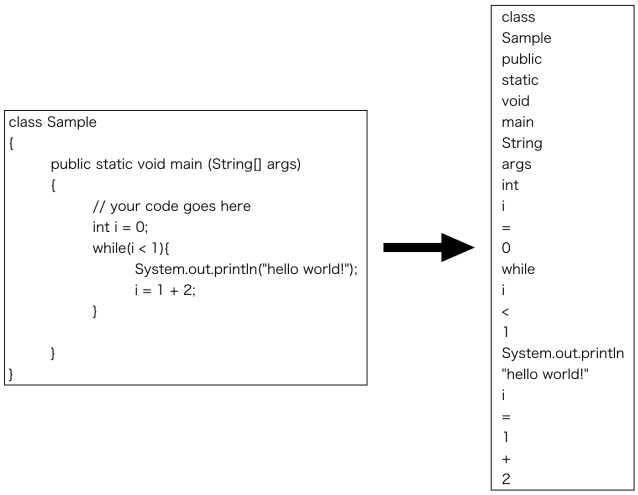

図 1. トークナイザによる変換例

3.2. トークナイズ

トークナイズ（tokenize）とは，ソースコードをトークンの並びに変換することである．トークンとはキーワードや識別子，演算子などを指す．トークナイズを行うプログラム等のことをトークナイザと呼ぶ．Java で記述されたソースコードに対して本研究で用いたトークナイザを使用した例を図 1 に示す．このトークナイザはコメントと括弧，区切り文字 ';' を無視しつつ，先頭から順にトークンを抽出する．

3.3. 深層学習

深層学習とは機械学習の一種であり，人間の脳神経回路を模倣して考案されたニューラルネットワークの層を増やし，階層を深めたアルゴリズムのことである．階層を深くすることで複雑なパターンのデータも学習することができ，モデルの表現力を向上させることができる．

3.3.1. Keras

Keras [3] とは，Python で記述されたニューラルネットワークを扱うライブラリである．TensorFlow などの上で実行できる．Keras は迅速な実験を可能にすることに重点を置いて開発されており，学習コストが低く手軽に使用できることから，事業や研究で幅広く利用されている．本研究では Keras を用いて学習モデルを作成した．

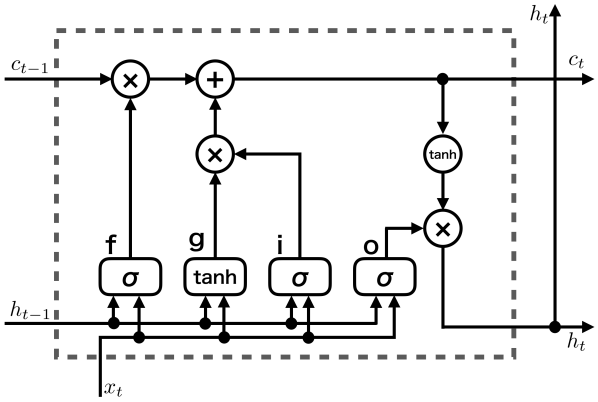

図 2. LSTM の計算グラフ

3.3.2. LSTM 層

Long short-term memory（LSTM）層は，受け取った入力から隠れ状態を計算し結果を出力する層である．再帰型ニューラルネットワーク（RNN）と同じく，出力された隠れ状態を自身への入力とすることで再帰的な計算を行う．

LSTM の中身を計算グラフを用いて表したものを図 2 に示す．この図は時刻 t のときの LSTM の状態を表しており，入力 $\boldsymbol{x}_t$ と，一つ前の時刻 $t-1$ の後述する記憶セル $\boldsymbol{c}_{t-1}$ と隠れ状態 $\boldsymbol{h}_{t-1}$ を受け取り，$\boldsymbol{c}_t$ と $\boldsymbol{h}_t$ を出力する．また，図中の'σ' はシグモイド関数を表している．

LSTM は RNN とよく似ているが，LSTM には記憶セル $\boldsymbol{c}$ と呼ばれるものがある．LSTM 層から出力された記憶セルは自分自身のみへの入力として使用される．記憶セルの計算にはゲートという機能を使用する．ゲートはデータの流れをコントロールするための機能で，0 以上 1 以下の実数で表される．0 はデータを全く流さない閉じた状態を表し，1 はデータを全て流す全開の状態を表す．

ゲートには，忘却ゲート，入力ゲート，出力ゲートの 3 種類がある．

忘却ゲート

図 2 の'f' に相当し，次の式で表される．

$$\boldsymbol{f} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^f + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^f + \boldsymbol{b}^f) \tag{1}$$

式中の $\boldsymbol{W}_x^f$ は入力 $\boldsymbol{x}_t$ に対する重みであり，$\boldsymbol{W}_h^f$ は一つ前の隠れ状態 $\boldsymbol{h}_{t-1}$ に対する重みである．$\boldsymbol{b}^f$ はバイアスを表す．

このゲートは，一つ前の記憶 $\boldsymbol{c}_{t-1}$ から不要な情報を捨てる働きをする．

入力ゲート

図 2 の'i' に相当し，次の式で表される．

$$\boldsymbol{i} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^i + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^i + \boldsymbol{b}^i) \tag{2}$$

LSTM には前述した忘却ゲートによる不要な情報を捨てる機能だけでなく，新しい情報を得るための機能がある（図 2 の'g'）．これは次の式で表される．

$$\boldsymbol{g} = \tanh(\boldsymbol{x}_t \boldsymbol{W}_x^g + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^g + \boldsymbol{b}^g) \tag{3}$$

この機能に対して制御を行うゲートを入力ゲートと呼ぶ．入力ゲートは追加される新しい情報にどれだけの価値があるのかを判断する．

出力ゲート

図 2 の'o' に相当し，次の式で表される．

$$\boldsymbol{o} = \tanh(\boldsymbol{x}_t \boldsymbol{W}_x^o + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^o + \boldsymbol{b}^o) \tag{4}$$

隠れ状態 $\boldsymbol{h}_t$ は $\boldsymbol{c}_t$ に tanh 関数を適用することで求められるが，その際に出力ゲートは $\tanh(\boldsymbol{c}_t)$ の各要素が次の時刻の隠れ状態としてどれだけ重要であるかを調整する．

これらのゲートを用いて記憶セル $\boldsymbol{c}_t$ と隠れ状態 $\boldsymbol{h}_t$ はアダマール積（$\circ$）を用いて次の式で表される．

$$\boldsymbol{c}_t = \boldsymbol{f} \circ \boldsymbol{c}_{t-1} + \boldsymbol{g} \circ \boldsymbol{i} \tag{5}$$

$$\boldsymbol{h}_t = \boldsymbol{o} \circ \tanh(\boldsymbol{c}_t) \tag{6}$$

ゲートと記憶セルによって，RNN を使用した場合にしばしば問題となる勾配消失を抑制することができる．

LSTM では過去の情報が記憶されるため，過去のデータと現在のデータに関連性があるようなデータ，例えば時系列データを扱う際によく用いられる．文章も単語の並びに関連性があると考えると，時系列データの一種とみなせる．

3.3.3. ドロップアウト層

ドロップアウト層は，学習時にランダムにニューロンを選びそのニューロンを無視することでその先に信号が

伝達するのを止める層である．ドロップアウト層を追加することでランダムな無視が制約となり，過学習を防ぐことができる．過学習とは，学習データに対して特化した学習を行ってしまったために未知のデータに対しては正しい答えを出せないような状態のことである．過学習を防ぐということはニューラルネットワークの汎化性能を向上させるということである．

3.4. 多クラス分類

本研究はソースコード中の空欄に当てはまる演算子を推定するものであり，推定する演算子の候補は 3.1 節で述べたようにようにあらかじめ限定されている．よって，空欄に当てはまる確率が候補の中で最も高いものを選ぶ多クラス分類を行えばよい．

多クラス分類を行うニューラルネットワークでは，ニューラルネットワークが出力するスコアにソフトマックス関数を適用してスコアを確率に変換し，損失関数として交差エントロピー誤差を用いて損失を求めることが多い．ここで，損失関数とはある学習時点における学習結果が正解データとどのくらい異なっているかを示す「損失」を求める関数のことである．学習がうまく行えていると損失は徐々に小さくなり，損失の更新量が一定より小さくなると学習は終了する．

3.4.1. ソフトマックス関数

ソフトマックス関数は，ニューラルネットワークから出力されるスコアを確率に変換する関数であり，次の式で表される．

$$y_k = \frac{\exp(s_k)}{\sum_{i=1}^{n} \exp(s_i)} \tag{7}$$

式（7）はクラスが n 個あるときの k 番目のクラスの出力 y_k を求めている．

ソフトマックス関数を適用した n 個の各出力は 0 以上 1 以下の実数となり，n 個全ての出力を足し合わせると 1.0 になる．このことからソフトマックス関数の出力は確率であるとみなせる．

3.4.2. オプティマイザ

機械学習の分野において，損失関数の値をできるだけ小さくするパラメータの値を見つけることを最適化（optimization）と呼び，その手法のことをオプティマイザ（optimizer）と呼ぶ．タスクによって最適なオプティマイザは異なるが，確率的勾配降下法（SGD）や Adaptive moment estimation（Adam）[4] などがよく利用される．

表 1. 2 クラスの混同行列の例

		予測値	
		Positive	Negative
真値	Positive	TP	FN
	Negative	FP	TN

表 2. 3 クラスの混同行列の例

		予測値		
		A	B	C
真値	A	T_A	F_{BA}	F_{CA}
	B	F_{AB}	T_B	F_{CB}
	C	F_{AC}	F_{BC}	T_C

3.5. 評価尺度

3.5.1. 多クラスの混同行列

説明のため，A，B，C の 3 つのクラスを分類する問題を考える．2 クラスの場合の混同行列は表 1 のようになるが，3 クラスの場合の混同行列は表 2 のようになる．

表 2 の T_A は真値が A で予測値も A であったときの数を表し，F_{AB} は真値は B だが予測値が A であったときの数を表す．以降の説明では，表 2 に基づいて評価尺度を表す．

3.5.2. 正解率 (Accuracy)

正解率とは，正しく予測した割合であり，次の式で表される．

$$Accuracy = \frac{\sum_{i\in\{A,B,C\}} T_i}{\sum_{i\in\{A,B,C\}} T_i + \sum\sum_{i,j\in\{A,B,C\},j\neq i} F_{ij}} \tag{8}$$

正解率は予測の全体的な傾向を把握するには有用であるが，クラスの数の偏りなどに大きく影響を受ける．

3.5.3. 適合率（Precision）

クラス i に対する適合率とは，予測値が i であるもののうち真値が i であったものの割合であり，次の式で表される．

$$Precision_i = \frac{T_i}{T_i + \sum_{j \in \{A,B,C\}, j \neq i} F_{ij}} \tag{9}$$

3.5.4. 再現率（Recall）

クラス i に対する再現率とは，真値が i であるもののうち予測値が i であるものの割合であり，次の式で表される．

$$Recall_i = \frac{T_i}{T_i + \sum_{j \in \{A,B,C\}, j \neq i} F_{ji}} \tag{10}$$

3.5.5. F 値

F 値（F1 値などとも呼ばれる）とは，適合率と再現率のトレードオフに対して最適解を求めるための尺度であり，適合率と再現率の調和平均で計算される．クラス i に対する F 値は次の式で表される．

$$F_i = \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i} \tag{11}$$

適合率か再現率どちらか一方が大きい値を記録しても良い結果とは言えない．適合率と再現率がどちらも大きい場合に F 値も大きくなる．F 値は各クラスのデータ数に偏りがある場合にも有効な尺度である．

3.5.6. マイクロ平均

上で述べた適合率や再現率，F 値は各クラスごとに計算する方法であり，学習モデル全体の評価を行うにはそれらの平均を取ることなどが考えられる．各クラスごとの指標の平均を求める方法はマクロ平均と呼ばれる．マクロ平均はデータの偏りを考慮していない．

データの偏りを考慮した方法にマイクロ平均というものがある．マイクロ平均はクラスごとに計算するのではなく，多クラスの混同行列を 2 クラスの混同行列に変換して 2 値分類と同様に計算する方法である．表 2 を表 1 のように変換する場合は，

$$\begin{aligned}
TP &= \sum_{i \in \{A,B,C\}} T_i \\
TN_i &= \sum_{\substack{j \in \{A,B,C\} \\ j \neq i}} T_j + \sum_{\substack{j \in \{A,B,C\} \\ j \neq i}} \sum_{\substack{k \in \{A,B,C\} \\ k \neq i}} F_{jk} \\
TN &= \sum_{i \in \{A,B,C\}} TN_i \\
FP &= \sum_{i \in \{A,B,C\}} \sum_{\substack{j \in \{A,B,C\} \\ j \neq i}} F_{ij} \\
FN &= \sum_{i \in \{A,B,C\}} \sum_{\substack{j \in \{A,B,C\} \\ j \neq i}} F_{ji}
\end{aligned} \tag{12}$$

のように計算することで変換できる．式（12）で計算した結果を用いて適合率などを求めることで学習モデル全体の評価を行うことができる．FP と FN が等しくなるため，適合率と再現率は等しくなる．適合率と再現率が等しいとき，式（11）より F 値も適合率および再現率と等しくなる．また，マイクロ平均を用いると正解率も適合率や再現率，F 値と等しい値になる．

4. 提案手法

4.1. 提案手法の概要

提案手法では，ソースコードの情報を入力とする学習モデルを用いて空欄に当てはまる演算子を推定した．LSTM 層などによって構成される 3 種の学習モデルを作成した．

4.2. モデルの定義

空欄より前の部分のソースコードをトークン化したものを t_F，後の部分のソースコードをトークン化したものを t_B とする．

4.2.1. 学習モデル M_F

t_F を用いて空欄に当てはまる演算子を推定する学習モデル M_F を図 3 に示す．バッチサイズは 116 に，埋め込み層の出力の次元数は 100 に，LSTM 層の隠れ状態の数は 128 に，ドロップアウト層のドロップする割合は

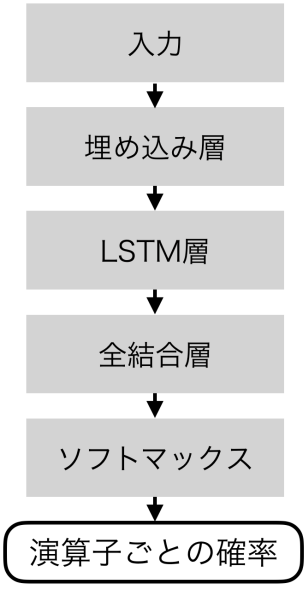

図 3. 学習モデル M_F, M_B の概要図

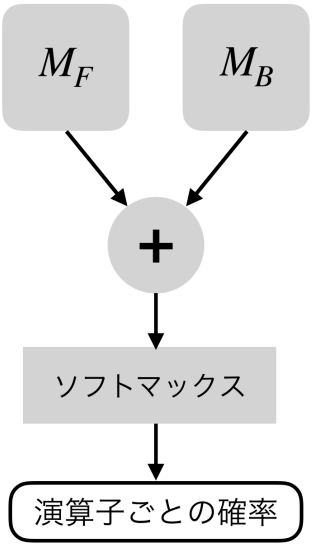

図 4. 学習モデル M_{FB} の概要図

50%に，エポック数は 10 に設定した．オプティマイザには Adam を使用した．また，活性化関数はソフトマックス関数を用い，損失関数は交差エントロピー誤差を用いた．

4.2.2. 学習モデル M_B

t_B を用いて空欄に当てはまる演算子を推定する学習モデル M_B を図 3 に示す．バッチサイズなどのパラメータは M_F と同じものを使用する．

M_B は M_F とほぼ同じであるが，M_F と異なり，入力の順を逆順に並び替える．例えば，”`a = b ?? c + d`” というソースコードの場合は，’`d`’，’`+`’，’`c`’ の順で入力される．

4.2.3. 学習モデル M_{FB}

t_F と t_B を用いて空欄に当てはまる演算子を推定する学習モデル M_{FB} を図 4 に示す．バッチサイズなどのパラメータは M_F と同じだが，エポック数は 20 とした．エポック数が異なるのは，損失が収束するまでの学習回数が異なるためである．

図 4 の左側の入力層に入力された t_F は埋め込み層により分散表現へと変換され，左側の LSTM 層に出力される．同様に，右側の入力層に入力された t_B は埋め込み層により分散表現へと変換され，右側の LSTM 層に出力される．その後，左右それぞれで M_F や M_B と同様の計算を行い，出力を加算する．加算した結果にソフトマックス関数を適用し，最終的な演算子ごとの確率から演算子の推定を行う．

5. 実験方法

5.1. データの前処理

5.1.1. 使用するソースコード

実験には Apache [5] のオープンソースソフトウェアプロジェクトである Ant, Camel, Eagle, James の全 Java ファイルのうち，実験対象の 13 種の演算子のうちいずれかを 1 つ以上含んでいるファイル 11,600 個を使用した．この 11,600 個のデータを，リポジトリを区別せずファイル名でソートし先頭から順に，学習データ 6,960 個，検証データ 2,436 個，テストデータ 2,204 個に分けて実験を行なった．学習データはモデルの学習のために用いられ，検証データは学習が 1 エポック終了する毎にモデルの性能を検証するために用いられる．テストデータは学習が終了したモデルの最終的な評価を行うために用いられる．

5.1.2. 前処理の手順

実験に使用する学習データ，検証データ，テストデータに対して行う前処理について述べる．

1. トークナイザを用いて Java ソースコードをトークンの並びに変換する．変換したものをトークンファイルと呼ぶこととする．
2. トークンファイルの中から実験対象の演算子を無作為に 1 つ選び，”??” に置き換える．”??” は空欄を意味する．この時，置き換えた演算子（つまり教師ラベル）を保存しておく．

5.1.3. 実験データに含まれる演算子

実験データに含まれる演算子を表 3 に示す.「全て」は各データのトークンファイルに含まれる全ての演算子の数を,「空欄」は実験のためトークンファイルの演算子一つを空欄に置き換えたときの置き換えられた演算子の数，つまり教師ラベルの数を表している．例えば，’+’ の行の左端の 31,024 は学習データの全トークンファイルに ’+’ が 31,024 個現れることを表している．また，その右の 2,667 は学習データのうち教師ラベルが ’+’ のものが 2,667 個あることを表している．

5.2. RQ1 の実験手順

本実験では，空欄より前のトークン t_F をモデル M_F の入力とし，空欄に当てはまる演算子を推定する．テストデータを用いて推定した結果を混合行列で表した．

5.3. RQ2 の実験手順

本実験では，空欄より後のトークン t_B をモデル M_B の入力とし，空欄に当てはまる演算子を推定する．テストデータを用いて推定した結果を混合行列で表した．

5.4. RQ3 の実験手順

本実験では，空欄より前のトークン t_F と空欄より後のトークン t_B の両方をモデル M_{FB} の入力とし，空欄に当てはまる演算子を推定する．テストデータを用いて推定した結果を混合行列で表した．

表 3. 各データの演算子の数

	学習データ		検証データ		テストデータ	
	全て	空欄	全て	空欄	全て	空欄
+	31,024	2,677	9,223	1,003	13,347	835
-	7,107	589	2,048	187	2,140	203
*	1,023	133	358	44	539	90
/	217	16	85	5	134	7
%	122	15	39	4	53	6
<	6,141	734	1,994	255	1,818	216
>	5,667	478	1,789	165	1,699	141
<=	358	36	121	13	126	9
>=	400	30	137	13	201	15
==	8,486	839	2,750	298	2,755	261
!=	10,417	947	3,487	277	3,048	289
&&	4,390	325	1,808	115	1,419	85
||	2,058	141	799	57	727	47

5.5. RQ4 の実験手順

LSTM を用いた 3 つの実験の結果と比較するため，本実験ではソースコード内の演算子出現数にもとづいて重み付けされた確率でランダムに推定するモデル M_R を作成した．確率に重み付けをする際は学習データの教師ラベルの数を元に計算した（詳細な数は表 3 の学習データの空欄列を参照）．例えば，学習データの教師ラベル 6,960 個のうち’+’ が正解のものは 2,677 個あるので，’+’ の重み付けされた確率は約 38%となる．テストデータを用いて推定した結果を混合行列で表した．その後，全モデルの推定結果を F 値を用いて比較した．

6. 実験結果

それぞれの実験の推定結果から得られた混同行列のヒートマップを図 5〜8 に示す．ヒートマップとは行列型の数字データの強弱に色をつけることで行列を見やすくしたものである．数字が大きくなると色が濃くなり，反対に数字が小さくなると色が薄くなる．本実験で作成したヒートマップは混同行列の正規化を行い色付けをしているが，表示されている数字は正規化する前のものである．True label は正解の演算子を表し，Predicted label はモデルが推定した演算子を表す．例えば，図 5

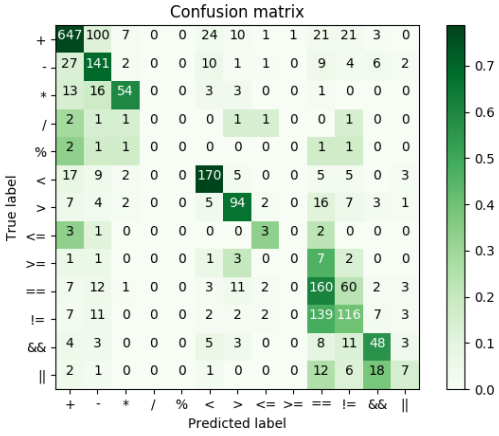

図 5. M_F を用いた実験の結果の混同行列

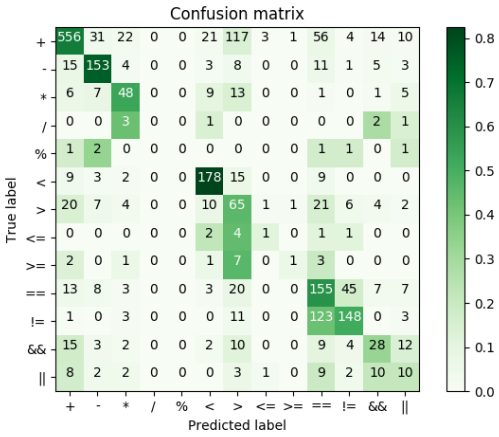

図 6. M_B を用いた実験の結果の混同行列

の 1 行 2 列目の 100 は，空欄に当てはまる正解の演算子は '+' だがモデル M_F が推定した結果が '−' である場合の数が 100 であったことを示している．

各モデルで推定した結果から得られた F 値を表 4 に示す．M_{FB} を用いた推定結果が最も良く，M_R を用いた推定結果が最も悪い結果となった．3.5.6 節で述べたように，マイクロ平均を用いた場合 F 値と正解率は一致するので，表 4 は正解率の表ともいえる．

各モデルで，学習データ全てを用いて学習を行ったときと，テストデータ全てを用いて演算子の推定を行ったときの所要時間を表 5 に示す[1]．M_{FB} の場合，推定時間

[1]Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz を 2 つと 64 GB の RAM を使用した．GPU は使用していない．

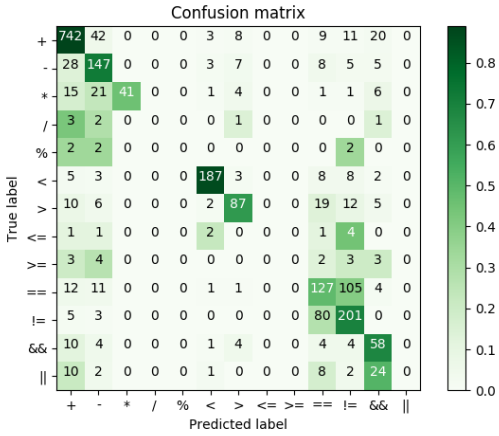

図 7. M_{FB} を用いた実験の結果の混同行列

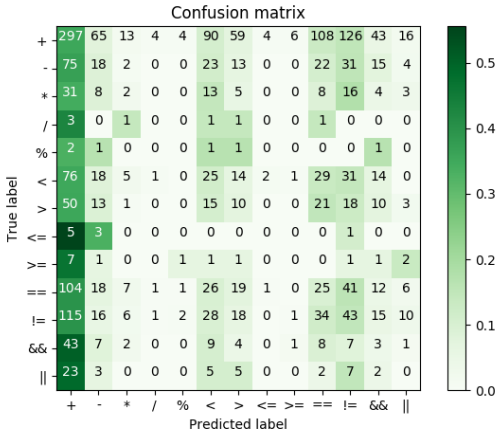

図 8. M_R を用いた実験の結果の混同行列

表 4. 各モデルの実験結果の F 値

	F 値
M_F	0.653
M_B	0.609
M_{FB}	0.721
M_R	0.192

表 5. 各モデルの学習時間と推定時間

	学習時間（秒）	推定時間（秒）
M_F	19,524	174
M_B	9,311	84
M_{FB}	40,468	181
M_R	-	-

は 181 秒でありテストデータのファイル数は 2,204 個であるので，1 つの演算子を推定するのに約 0.08 秒の時間を要することがわかる．また，M_R は学習モデルではないため計測していない．

7. 考察

7.1. 研究設問への回答

実験結果をもとに，研究設問に回答する．

RQ1: 空欄より前の情報から空欄に入る演算子をどの程度の精度で推定できるか．
約 65%の正解率で推定できた．

RQ2: 空欄より後の情報から空欄に入る演算子をどの程度の精度で推定できるか．
約 61%の正解率で推定できた．

RQ3: 空欄以外の情報から空欄に入る演算子をどの程度の精度で推定できるか．
約 72%の正解率で推定できた．

RQ4: 以上の 3 つの方法で推定した結果に差異はあるか．
4〜11%程度の正解率の差があった．

7.2. 結果全体に関して

表 3 より，'+' はソースコードに出現する演算子の約 4 割を占めるが，他の演算子の出現率は 1 割程度以下となっている．'+' を除くと演算子の出現率の偏りが小さいことから，重み付けされた確率からランダムに推定する手法の正解率は低くなることが予想され，その結果は表 4 より約 19%となった．一方，提案手法では最大 72%の正解率を得ることができた．これらの結果から，空欄の前後のソースコードと欠落した演算子の間には何らかの関係性があり，深層学習を用いることでその関係性を学習できると考えられる．

空欄より後の情報から推定した M_B の実験結果の正解率が 61%であった．M_B の推定手法では，例えば "`if(a < b)`" というソースコードがあったとき，トークナイズして演算子を '??' に置き換えると，"`if a ?? b`" というトークン列になり，"`if`" を読まずに "`??`" を推定することになる．しかし，図 6 より，'<' は約 82%，'>' は約 46%の正解率が得られた．また，'<' と '>' の出現回数に大きな差はない．これらのことから，'if' などのキーワードが無くともその後の文からある程度の正解率で空欄に当てはまる演算子を推定できることを示している．空欄より前の部分を使用した場合には劣るが，空欄より後の部分にも演算子の正しい推定に重要な特徴が多く含まれていると考えられる．しかし，'<' と '>' の正解率の差については構文解析を利用するなど，別途考察する必要があると考えられる．

空欄より前の情報から推定する M_F と空欄の前後の情報から推定する M_{FB} の間には約 7%の正解率の差があった．空欄より前の情報だけでは演算子の候補を絞りきれない場合に，空欄より後の情報も用いることによって正しく推定できるようになったと考えられる．演算子の種類や空欄の場所により，空欄の前後の情報のどちらが演算子を正しく推定するための情報として有用であるかが変化することが考えられる．

表 5 より，M_F と M_B の学習時間と推定時間に差があるのは空欄前後のトークンの数が異なるためであると考えられる．空欄より前の部分のトークン t_F には import 文やクラス名などがトークンとなって含まれており，空欄より後の部分のトークン t_B に比べてトークンの量が多くなる傾向があると考えられる．それぞれの合計ファイルサイズを測定した結果，t_F は 26 MB，t_B は 16 MB であった．

7.3. 妥当性への脅威

本研究ではソースコードをトークンに変換するため自作したトークナイザを使用している．しかし，一部のソースコードに対してこのトークナイザを適用する場合に不具合があることが確認されている．例えば，'<' や '>' が演算子として使われていない場合でも演算子として扱ってしまう．そのため，構文解析を利用するなどして，より精度の高いトークナイザを準備する必要があると考え

られる．

空欄より前の情報から推定する M_F は後の情報から推定する M_B に比べて正解率が 4%程度優れていた．しかし，7.2 節にて言及した，トークンファイル t_F と t_B の間にサイズ差があることが M_F と M_B の推定精度の差に影響することが考えられる．そのため，空欄より前の部分と後の部分のどちらがより多くの特徴量を含んでいるか議論するには，トークンファイルに手を加えるなどして検証する必要がある．

7.4. 今後の課題

LSTM 層の多層化

本研究で作成したモデルでは LSTM を 1 層だけ使用している．LSTM を複数使用して層を深くすることによりモデルの表現力が増し性能が向上することが一般的に示されている．ただし，層を深くしすぎると反対に性能が下がる場合もあるので，適切な深さを調整する必要がある．本研究のモデルはそのような調整を行っていないので，LSTM 層の深さを調整する必要がある．

演算子の種類の追加

本研究では推定する演算子の種類を限定した．しかし，実用することを視野に入れるには推定できる演算子の種類を増やす必要がある．そのため，実験対象の演算子の種類を増やし，モデルやパラメータの調整をする必要がある．

他の言語での実験

本研究では Java のみを実験対象とした．他の言語に対応したトークナイザを用意することで，様々な言語のソースコードを対象に実験を行うことができる．それぞれの言語に対応させるため，実験を行いモデルの改良をする必要がある．

8. まとめ

本研究では，ソースコードをトークンに変換して機械学習モデルの入力とすることで空欄に当てはまる演算子を推定する手法を提案した．実験の結果，61〜72%の正解率で推定できた．このことから，ソースコードには空欄に入る演算子を推定するための特徴が含まれており，深層学習を用いることでその特徴を学習できると考えられる．

今後の課題としては，トークナイザやモデルの改良，実験対象となる演算子や言語を追加して汎用性を向上させることが挙げられる．

参考文献

[1] Y. Yang, K. Sakamoto, H. Washizaki, and Y. Fukazawa, “A tool for suggesting similar program element modifications,” 研究報告ソフトウェア工学（SE）, vol.2014, no.20, pp.1–6, jul 2014.

[2] 山本哲男，吉田則裕，肥後芳樹，“ソースコードコーパスを利用したシームレスなソースコード再利用手法,” 情報処理学会論文誌，vol.53，no.2，pp.644–652，feb 2012.

[3] Keras, Home - Keras Documentation, （オンライン）, 入手先〈https://keras.io/ja/〉（参照 2019-2-13）.

[4] D.P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv e-prints, p.arXiv:1412.6980, Dec. 2014.

[5] The Apache Software Foundation, Welcome to The Apache Software Foundation!, （オンライン）, 入手先〈https://www.apache.org/〉（参照 2019-2-1）.