コード内フォールト自動修正のための機械学習に関する考察

2016SE010 羽賀 文哉
指導教員：野呂 昌満

1 はじめに

深層学習を用いたプログラムの修正に関する研究が多く行なわれている．深層学習を用いた修正では，その修正に適した内部表現を学習する必要があり，どのような特徴を用いるかが，処理の性能を左右する重要な要因になっている [4].

制御フロー解析では，制御フローを静的に解析し，実行パスに依存したフォールトを検出する．プログラム修正に関する既存の研究の多くは, 字句列や実行トレースを解析対象としているので, 制御フロー解析で検出されるフォールトは対象としていない．このことから，深層学習を用いてプログラムを分析を分析する方法の 1 つとして制御フローに関する特徴を用いたニューラルネットワークを設計する方法が考えられる．

本研究の目的は，制御フローに関する分析を行うニューラルネットワークモデルを実現することである．このニューラルネットワークモデルは，変数の def-use に関するフォールトを検出する．各変数の定義・参照に関する情報を入力とし，フォールトを含むプログラムか含まないプログラムかを分類する．このニューラルネットワークモデルが実現できれば，制御フロー解析が扱う他の特徴を抽出できたと考えられる．このニューラルネットワークモデルを応用してプログラムの修正が実現可能になると考えた．

2 ニューラルネットワークモデルの設計

変数の def-use について分析を行うニューラルネットワークモデルを設計する．def-use を対象にしているので, プログラム中の変数の定義・参照状態 (以下, 環境と呼ぶ) をベクトルにエンコードし, ニューラルネットワークモデルの入力とする．変数の数はプログラムごとに異なるので, 環境のエンコードでは可変長のデータを扱えるようにする必要がある．

本研究で設計するニューラルネットワークモデルは，Wang らの State Trace Model[2] の拡張し，カテゴリカル属性を入力として扱うニューラルネットワークモデルを設計する．State Trace Model では，入力に数値属性を扱っていたが，本研究で入力として扱うのはカテゴリカル属性である．入力で扱うデータの属性の違いについて，変数の定義・参照・定義及び参照なしを One-hot ベクトルで表せると仮定して，入力層を設計する．

図 1 に本研究で設計したニューラルネットワークモデルを示す．特定の環境におけるすべての変数を前述の One-hot ベクトルにエンコードする．このベクトルを入力とした RNN によってフォールトの分類を行なう．環境に変化が起きるすべての時点について入力する．図 1 中の GRU は長期記憶可能な RNN のユニットである．Embedding は入力データをエンコードし, RNN に与えるものである．$u_{1...n}$ はある時点での環境を表す．$u_{n,1}...u_{n,m}$ はある時点の環境における特定の変数の One-hot ベクトルが順に入力されることを表す．すべての時点における計算結果を入力とした RNN により, フォールトの有無を分析する．

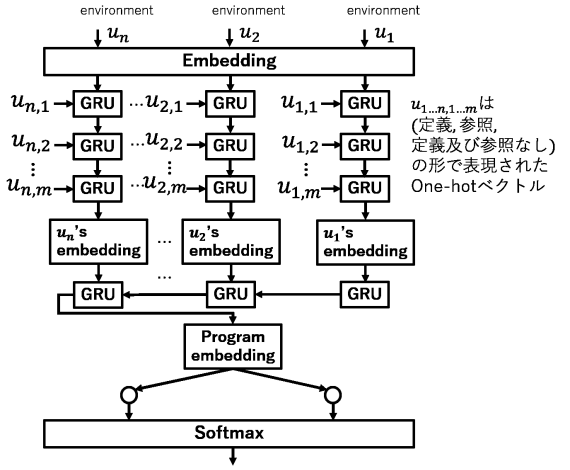

図 1 設計したニューラルネットワークモデル

```
int j,k,i,x;  (1)
i=1;          (2)
if(i==1){     (3)
 j=1;         (4)
}else{        (5)
 k=1;         (6)
}             (7)
x=k+3;        (8)
```

図 2 プログラム例

3 考察

3.1 設計したニューラルネットワークモデルを用いたフォールトの検出

図 2 に示すプログラムを例として設計したニューラルネットワークモデルを用いたフォールトの検出について説明する．フォールト検出ツールにおける学習は図 3 に示す手順に従って行われる．

図 3 の 1. に示すように入力プログラムから抽象構文木を作成する．図 2 のプログラムから作成される抽象構文木として図 4 のような木が作成される．

2. の変数一覧の作成では, 抽象構文木を走査し, プログラム中に出現する変数の一覧を定義する．この例では,

```
1. 抽象構文木の作成
2. 変数一覧の作成
While(パスの数){
  3. 訓練データの作成
  4. 学習器へ入力
}
```

図 3 学習手順

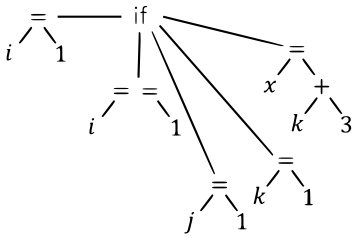

図 4 抽象構文木

```
int j,k,i,x;  (1)
i=1;          (2)   →   def i
if(i==1){     (3)   →   use i
 j=1;         (4)   →   def j
}else{        (5)
 k=1;         (6)
}             (7)
x=k+3;        (8)   →   def x
                        use k
```

図 5 変数の環境

	i	j	k	x
(2)	(1,0,0)	(0,0,1)	(0,0,1)	(0,0,1)
(3)	(0,1,0)	(0,0,1)	(0,0,1)	(0,0,1)
(4)	(0,0,1)	(1,0,0)	(0,0,1)	(0,0,1)
(8)	(0,0,1)	(0,0,1)	(0,1,0)	(1,0,0)
正解ラベル:フォールト有				

図 6 訓練データ

i, m, k, x の一覧が定義される．

3. の訓練データの作成では，特定の実行パス毎に入力と正解ラベルの組を作成する．プログラム中の変数に対する操作行毎に全ての変数の状態を入力とする．正解ラベルは，フォールトの有無となる．例えば，図 5 に示すように，if 文の then ブロックを通るパスにおいては (2),(3),(4),(8) で変数に対する操作が行われる．この各行毎の全ての変数の状態がニューラルネットワークモデルの入力となる (図 6)．if 文の then ブロックを通るパスでは変数 k に値が定義されないので，フォールト有が正解ラベルとなる．

4. では 3. で作成した訓練データを用いて学習を行なう．

3. から 4. は，全てのパス毎に行なう．予測器による予測では,3. で入力データを作成し，4. で予測器に入力する．

本研究では,sapid を用いて 1. 2. の手順を行なった．sapid を用いることで抽象構文木やデータフローグラフを作成することができる．

3.2 関連研究との比較

本研究は, 制御フローに関する特徴についてニューラルネットワークモデルを用いて分析することを目的とした．解析対象のプログラムを抽象構文木で表現し, それを基に各パスごとの解析対象変数の状態に着目したベクトル化を行なう．ニューラルネットワークモデルに与えることで, Def-Ref 問題を検出する．

Gupta らの研究 [3] は, 変数の状態を考慮していないので, 制御フローに関する特徴について分析を行なうことはできないと考えられる．本研究では, 変数の def-use を解析対象とし, 変数の定義及び参照に関する特徴を取り扱うことができるので, def-use に関するフォールトを扱うことができる．

Bader らの研究 [1] は, 構文上の形式的な修正パターンを学習しているだけであるので, 変数の状態を考慮した分析はできないと考えられる．本研究では, 変数の def-use を解析対象とし, 変数の定義及び参照に関する特徴を取り扱うことができるので, def-use に関するフォールトを扱うことができる．

4 おわりに

近年, 深層学習を用いたプログラムの自動修正などに関する研究が注目されているが, そこで扱われるプログラムの特徴は構文的なものが多いので, データと制御の依存関係や実行時に定まる意味を考慮することが難しい．

本研究では, 分析可能である制御フローに関する特徴を分析するニューラルネットワークモデルの実現を目的とした．目的を達成するために, 変数の def-use に関するフォールトを検出するニューラルネットワークモデルを設計した．

今後の課題として，実装による妥当性の検証を行なう必要がある．本研究では問題を簡潔にするために対象プログラムに制約をつけたが, 制約のないプログラムでも目的を達するか確認を行なう必要がある．他の研究への応用が可能であるかを考察し, 確認する必要がある．

参考文献

[1] A. Scott, J. Bader, M. Pradel and S. Chandra, “ Getafix: Learning to Fix Bugs Automatically, ” *arXiv preprint arXiv:1902.06111*, 2019.

[2] K. Wang, R. Singh, Z. Su, ”Dynamic Neural Program Embeddings For Program Repair,” *arXiv preprint arXiv:1711.07163*, 2018.

[3] R. Gupta, S. Pal, A. Kanade, and S. Shevade, “ DeepFix: Fixing Common C Language Errors by Deep Learning, ” In *Thirty-First AAAI Conference on Artificial Intelligence*, pp. 1341-1351, 2017.

[4] 手塚太郎，しくみがわかる深層学習，朝倉書店，2018.