Deep Learning for Bug-Localization in Student Programs

経営学部経営学科一年　村上広樹

DI-study_2019/12/07

論文情報

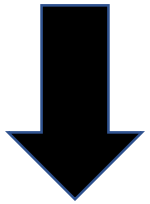
12月に行われる NeurIPS 2019 で有望な論文

10 EXCITING PAPERS TO LOOK OUT FOR AT THE　NEURIPS CONFERENCE　より

論文リンク： https://arxiv.org/abs/1905.12454

Abstract

・プログラムのバグの場所を見つける手法
・プログラムを動かさずに（使うのはコードの文字のみ）
・どこにバグがあるか見つける



プログラミング学習の助けになる

背景

現在でもエラーを返してくれるオンラインサイトありますね

これはコードを実行してみて成功するか失敗するか返してくれる

初心者にとっては不十分…

意味的なアプローチなので「エラーは起きてないけどタスクを満たせていない」といったときに役立つでしょう。

データセット

- C言語
- 簡単なプログラミングコースから29個の問題
- コードの長さは平均25行ぐらい

Table 1: Dataset statistics.

Avg. programs per task		Avg. tests per task	Avg. submissions per student	Avg. lines per submission
Correct	Buggy			
350	1007	8	18	25

（図1）

つまり、一つのタスクあたり大体8個のお手本コードと18個の生徒コードで学習されています

Technical Details

手法は2つのphaseに分かれる

① Success/Failure Prediction

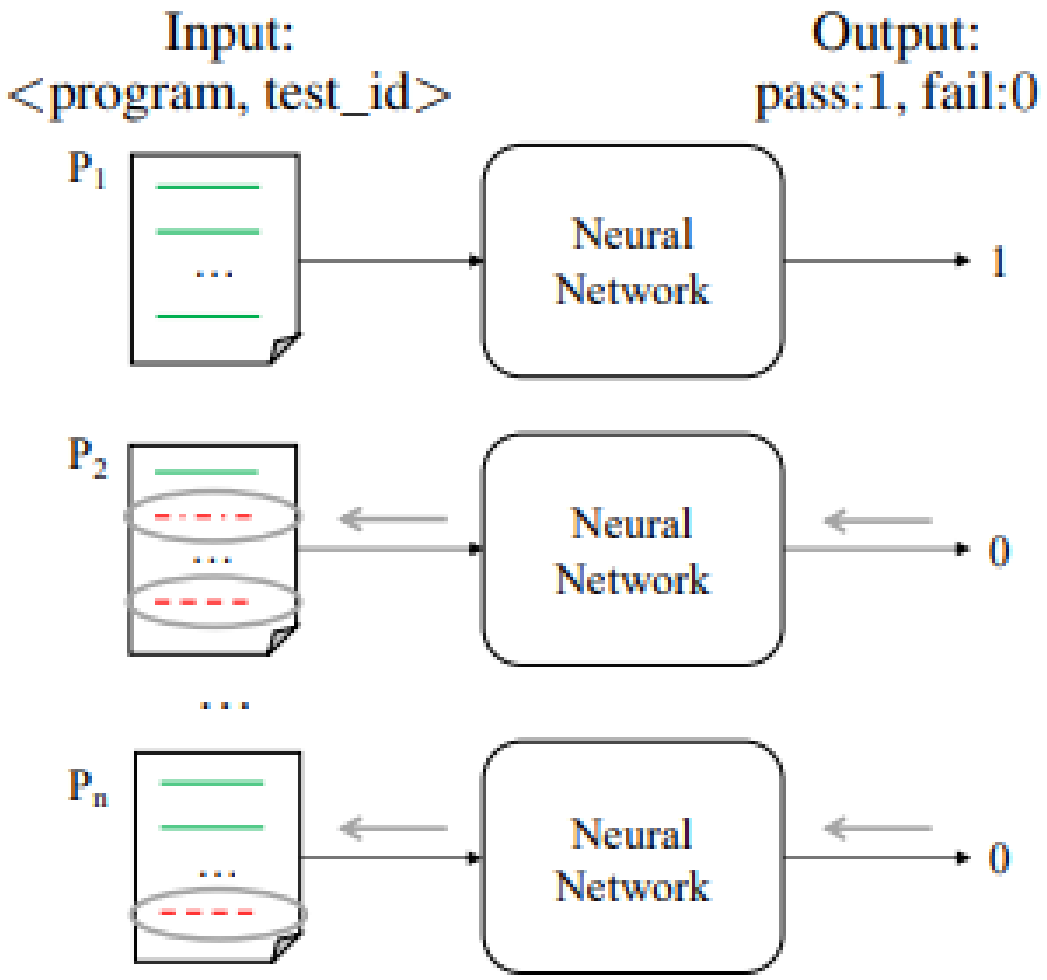
　　-プログラムを行列に変換

　　-SuccessかFailureか二値分類

② Prediction Attribution

　　-何が分類を左右しているか？というパターンを見つけ、

　　バグがある場所を特定



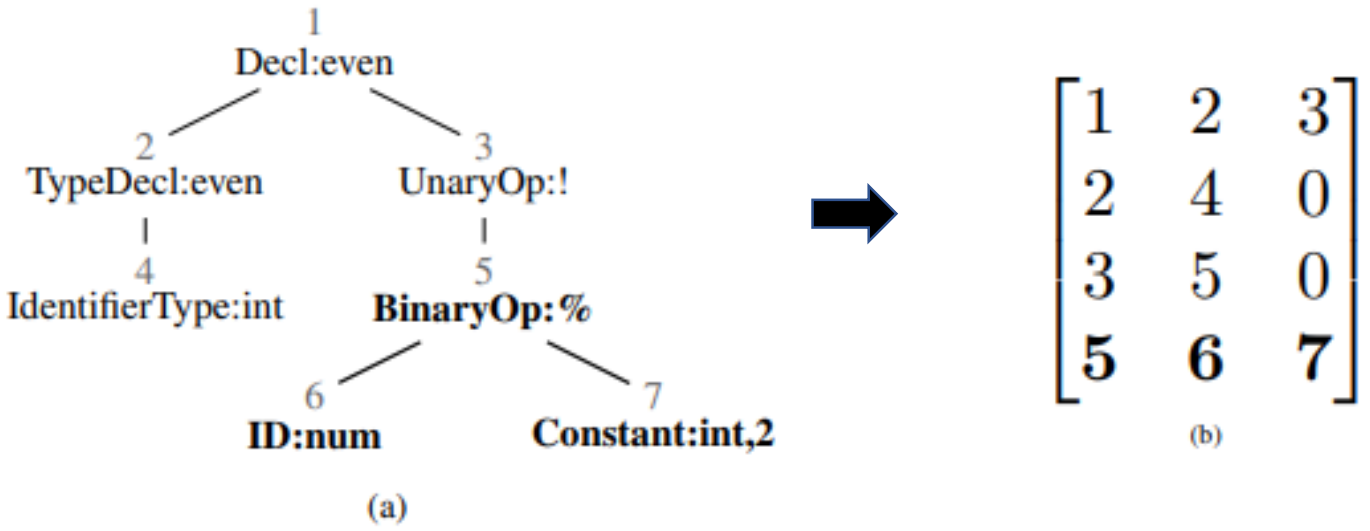
（図2）

①TCNN(Tree CNN)を使い、与えられたプログラムがテストを通るかどうか推測します

②outputが0(fail)だった場合、どの行が問題を起こしていそうか推測します

① Success/Failure Prediction -Technical Details

ASTs(abstract syntax trees)とは



（図3）

・ASTsはプログラムの構造的な特徴を得ることを可能にする。

・この性質を使ってプログラムをエンコードします。

・shared vocabulary(全データセット内の語彙リスト)も作る

① Success/Failure Prediction -Technical Details

Encode

プログラムをASTsに従って変換＆パディング

$max\_subtrees \times max\_nodes$

↓

それぞれの要素をembeddingされたshared vocabularyに置き換える

$max\_subtrees \times max\_nodes$ x 24

↓

TCNN(Tree CNN)の入力とする

max_nodes

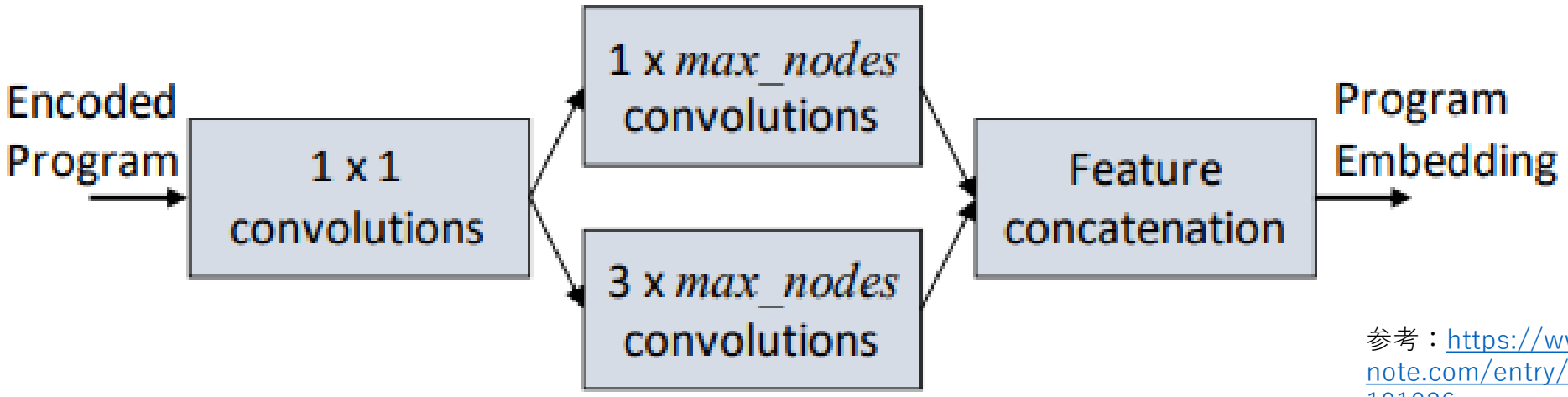
$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 0 \\ 3 & 5 & 0 \\ \mathbf{5} & \mathbf{6} & \mathbf{7} \end{bmatrix}$$

$max\_subtrees$

（図4）

① Success/Failure Prediction -Technical Details

Encodeしたプログラムを特徴量をとらえた行列に変換　→
TCNN(Tree CNN)を用いる



参考：https://www.robotech-note.com/entry/2017/12/24/191936
（1x1 convolutionについて）

（図5）

~正方形ではないConvolutionを用いて構造的特徴を維持します~

② Prediction Attribution -Technical Details

同じタスクのCorrect Programのうち最も似てるものを使ってバグの原因推定を行う -Buggy Programとのコサイン類似度で判定

Integrated Gradient Techniqueを使ってノードごとのsuspiciousness scoresを求める

Integrated Gradient

baseline（お手本コード）と原因推定したいプログラムを使う。調べたい行の(baseline,プログラム)の特徴量を混ぜ合わせてバグがある可能性が上がる/下がるを見ることによってどのくらいその行がバグがある可能性を導き出すことに寄与しているかを調べる手法

```
1.  #include <stdio.h>
2.  int main(){
3.    char c;
4.    scanf("%c", &c);
5.    if('A'<=c && c<='Z'){
6.      printf("%c", c+('a'-'A'));}
7.    else if ('a'<=c && c<='z'){
8.      printf("%c", c-('a'-'A'));}
9.    else if(0<=c && c<=9){
10.     printf("%d", 9-'c');}
11.   else{
12.     printf("%c", c);}
13.   return 0;}
```

（図6）

suspiciousness scoresの高い行からランク付けされたリストを推測結果として返す。

ここではヒートマップの形式でそれを表しています

評価

Technique & Configuration	Evaluation metric	Localization queries	Bug-localization result Top-10	Top-5	Top-1
Proposed technique	$\langle P, t \rangle$ pairs	4117	3134 (76.12%)	2032 (49.36%)	561 (13.63%)
	Lines	2071	1518 (73.30%)	1020 (49.25%)	301 (14.53%)
	Programs	1449	**1164** (80.33%)	833 (57.49%)	294 (20.29%)
Tarantula-1	Programs	1449	964 (66.53%)	456 (31.47%)	6 (0.41%)
Ochiai-1			1130 (77.98%)	796 (54.93%)	227 (15.67%)
Tarantula-*	Programs	1449	1141 (78.74%)	791 (54.59%)	311 (21.46%)
Ochiai-*			1151 (79.43%)	**835** (57.63%)	**385** (26.57%)
Diff-based	Programs	1449	623 (43.00%)	122 (8.42%)	0 (0.00%)
NBL rank			(1/6)	(2/6)	(3/6)

（図7）

最もsuspiciousness scoreが高いと予測した列がバグがある列であった割合は20.29%
suspiciousness score が高い順の10番目までの中にバグがある列が含まれていた割合は80.33%

今後の課題

- ほかのプログラミング言語でどのような結果になるのか
- 生徒のコードだけでなくいろんなコードで試す
- この技術を使ってバグを修理する

個人的には意味的なアプローチができるなら、プログラムを別のプログラミング言語で訳したりできたらいいなと思いました。

こんな感じです

Wrong `for` Loop

```
1  #include <stdio.h>
2  int rot(int [],int,int);
3  int main() {
4      int n,d,i;
5      scanf("%d\n",&n);
6      int arr[n];
7      for(i=0;i<n;i++) {
8          scanf("%d ",&arr[i]); }
9      scanf("\n%d",&d);
10     rot(arr,n,d);
11     return 0; }
12
13 int rot(int arr[],int n,int d) {
14     int j,k;
15     for(j=d+1;j<n;j++) { \\ suspiciousness score: 0.0006181474
16         printf("%d ",arr[j]); }
17     for(k=0;k<=d;k++) { \\ suspiciousness score: 0.0006690205
18         printf("%d ",arr[k]); }
19     return 0; }
```

This program is supposed to right shift a given array of ‘n’ numbers by a given number ‘d’. To correctly implement this, the programmer needs to change the two `for` loops at lines 15 and 17 to `for(j=n-d;j<n;j++)` and `for(k=0;k<n-d;k++)`, respectively. Our technique ranks these two lines as its second and third most suspicious buggy lines, respectively.

（図8）