A Reinforcement Learning based Program Repair Framework

Homanga Bharadhwaj

Computer Science, IIT Kanpur

25 April, 2019

Motivation

- Repairing buggy programs manually is a very tedious task for programmers.
- The process of debugging typically involves going through each line of the program from the beginning and checking whether each character is correctly placed in each location.
- Editing syntax errors through this process involves checking whether a region of code conforms to the rules of the grammar of the particular programming language.

Motivation

- It seems imperative that a systematic algorithm for program correction be developed to root out syntax errors in the program and provide feedback to the programmer regarding the location and type of errors.
- In this respect a very good job is done by the compiler itself that points out syntax errors in the program which make it unfit for compilation.
- However, as pointed out in previous works, the compiler error messages do not pin-point the exact location of the errors and hence are not very helpful for non-experts in programming.

Previous Approaches

- Learn from data in an attempt to automate the correction process
- DeepFix used an end-to-end Recurrent Neural Network (RNN) model trained to predict the location of errors in the program along with the correct statement in an iterative manner
- For training, the algorithm requires both the incorrect program and multiple corrected versions, each corresponding to correction of one error

Previous Approaches

- A recent technique, RLAssist, achieves better performance than DeepFix by training entirely through self-exploration and with no direct supervision.
- It mimics a human agent repairing the program by modeling the program as the environment for reinforcement learning agent that proceeds sequentially through the program text and takes two types of actions at each character - edit action and move action
- Rewards are provided by the compiler - when the program executes successfully after resolution of all/some errors (sparse/dense reward space)

Setting - States

- Defined to be a pair $< \mathit{string}, \mathit{cursor} >$, where *string* denotes the text of the program and *cursor* denotes the position of the cursor in the text
- Specifically, *cursor* is an integer in the range $[1, \mathit{length}(\mathit{string})]$, where *length* denotes the number of characters
- We convert the program *string* into a sequence of lexemes

Setting - Actions

- Two types of actions, namely for navigation of the cursor and for modification of the program text
- The navigation actions modify only the *cursor* part of the state while modification actions modify only the *string* part
- Wrong edits i.e. those that do not reduce the number of compiler error messages are automatically rejected by the environment

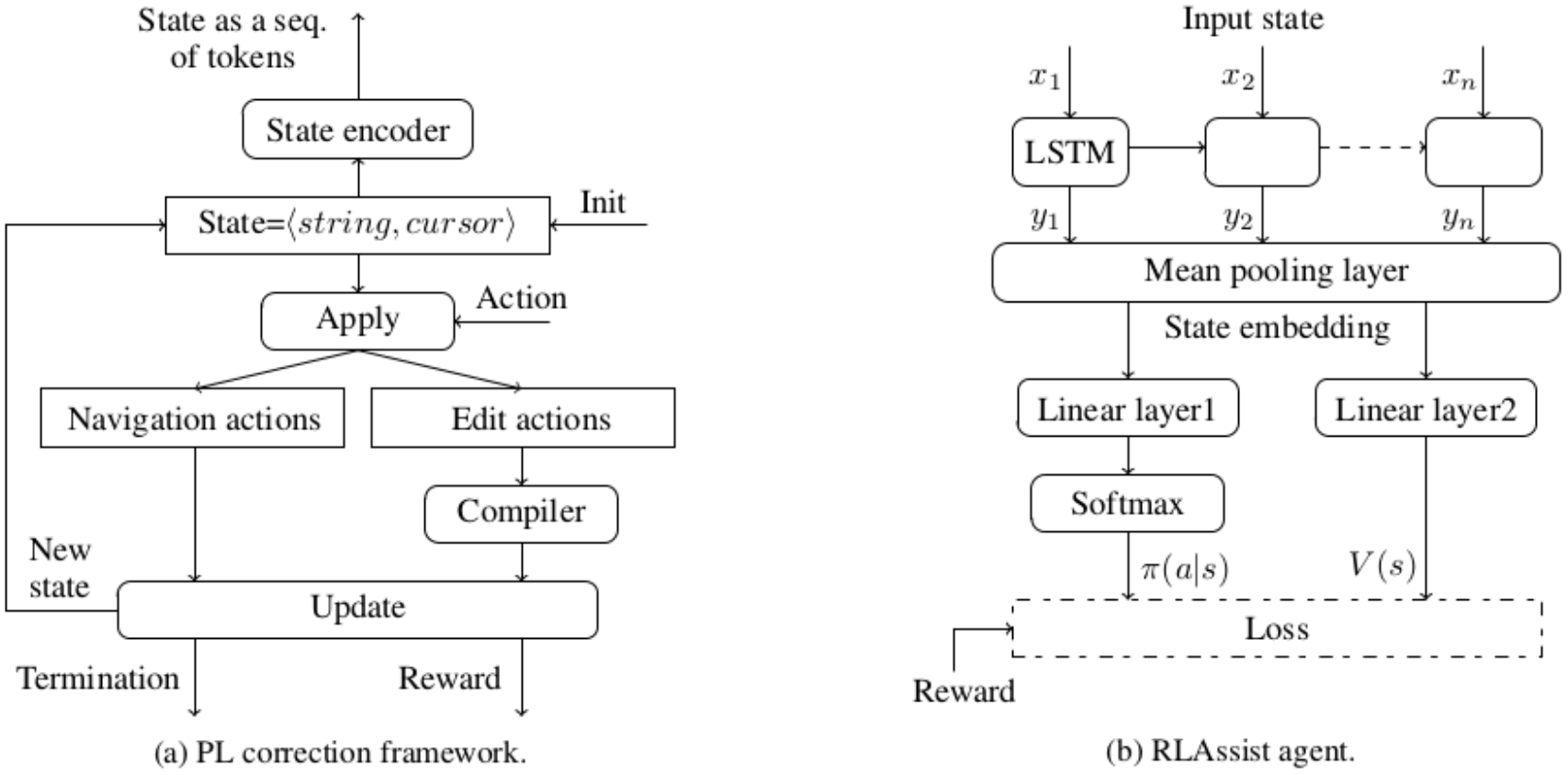
Setting - Episodes

- We define each episode to start with a program ridden with errors, denoted by the program *string* and set the *cursor* to its first token
- The goal state is said to have been reached when the edited program compiles successfully
- Define a maximum length of each episode to denote the maximum number of discrete time-steps that the agent is allowed to reach the goal from the start state

Setting - Rewards

- We evaluate using two reward settings - sparse reward and dense reward
- In the sparse reward setting, upon successful removal of all errors, a positive reward (+1) is given to the agent and there is 0 reward at all intermediate time-steps
- In the dense reward setting, a large positive reward is given after the program compiles successfully and small intermediate rewards are given for edit actions that reduce at least one error in the program

The Basic Framework



(a) PL correction framework.

(b) RLAssist agent.

Figure 1: The Design of RLAssist

Embedding - Mean Pooling

- LSTM for mapping each token x_i of an input sequence $(x_1, x_2, .., x_n)$ to a real vector y_i
- The final embedding is obtained by taking an element-wise mean over all the output vectors $y_1, y_2, ...y_n)$
- Mean-pooling startegy works well when the program length is *less*
- In lengthy programs, this simplistic pooling method does not capture the semantic constructs which appear in a syntactically identical form at different locations of the code

Detour $\rightarrow$ P-SIF embeddings

- Partition Word-Vectors Averaging (P-SIF) - first partition, then average only within each partition; concatenate all partition embeddings
- Consider a corpus C with N documents and the vocabulary of frequent words denoted by V. Let $\overrightarrow{w_{it}}$ represent a word vector i from topic t. By SIF, a document d can be represented by $\overrightarrow{v_d}$, such that:

$$\overrightarrow{v_d} = \sum_t \sum_i \overrightarrow{w_{it}} \tag{1}$$

- Mean-pooling strategy works well when the program length is *less*
- Here, it could be the case that same words belong to different topics (different semantic meanings for the same word in different contexts).

Detour $\rightarrow$ P-SIF embeddings

- However, if all the words are averaged together, it reduces the representation of documents to a single point in space and loses information of the different semantic topics.
- P-SIF proposes topic-wise averaging of word vectors followed by concatenation of each topic vector (the averaged word vectors) to represent the document
- So, by P-SIF, a document d can be represented by $\overrightarrow{v_d}$, such that:

$$\overrightarrow{v_d} = \Theta_t \left(\sum_i \overrightarrow{w_{it}} \right) \tag{2}$$

- Here, Θ denotes concatenation

Embedding - P-SIF

- Given an input sequence of tokens $(x_1, x_2, .., x_n)$, we use word2vec for generating word vectors y_i for each token
- The corpus for us is the entire set of program texts.
- The next step is generating word-cluster vectors $(y_1, y_2, ...y_n)$
- The final step is concatenation of all word-cluster vectors.

Agent - Model Free RL

- RL Assist used the A3C (Asynchronous Actor Critic algorithm) for modeling the RL agent
- The state embeddings pass through two different channels - one for inferring the value of state s $V(s)$ and the other for inferring the probability distribution of actions given the current state $\pi(a|s;\theta)$
- In the former channel, the state embeddings pass through a Linear layer, while in the latter channel, the state embedding are passsed through a linear layer followed by a softmax layer

Agent - Model Based RL

- Model Free RL is sample inefficient and takes a lot of time to train
- The basic idea is to learn a good dynamics model $f(s, a)$ that is capable of predicting the next state given the current state and the current action
- If this model is learned well, we can take the actions according to this dynamics model

Agent - Model Based RL

- We first run a base random policy $\pi_0(a_t|s_t)$ that takes random action (from the set of all allowed actions) at each time-step in order to collect the first batch of data $D = \{(s, a, \hat{s})_j\}$
- The dynamics model is a two layer fully connected NN that takes the concatenation of state s_t and action a_t as input at time-step t and outputs an inferred next state $\hat{s}_t$
- The loss used for training the dynamics model $f(s, a)$ is $\sum_j ||f(s_j, a_j) - \hat{s}_i||^2$. We iteratively execute some gradient steps to minimize this loss and take actions in the environment to observe new data in order to replenish the buffer of data

Experiments - Dataset

- The programs in the dataset are students' solutions to 93 programming tasks in an introductory programming course
- For each correct program x, five training examples x'_i were generated (total 165,000 training examples - without labels)

Dataset statistics		
Erroneous programs	Error messages	Average tokens
6975	16766	203

Table 1: Summary of the test dataset for programming assignments

Experiments - Baselines

We evaluate the following variants of our model:

- **MFreeTopic** - Model free RL with a topic based partition embedding of states
- **MBasedMean** - Model based RL with a mean pooling based embedding of states
- **MBasedTopic** - Model based RL with a topic based partition embedding of states

It is important to note that the fourth variant, Model based RL with a mean pooling based embedding of states is precisely the RLAssist model.

Experiments - Results

Method	Evaluation				
	Completely fixed programs	**Partially fixed programs**	**Error messages resolved**	**Average reward received**	**Average length of episodes**
DeepFix	1625	1129	5156	–	–
RLAssist	1699	1310	5884	0.26	52
MFreeTopic	1716	1321	5954	0.30	53
MBasedMean	1708	1319	5927	0.27	45
MBasedTopic	1720	1330	6109	0.32	46

Table 2: Summary of results of all the models evaluated on the test data, whose statistics are mentioned in Table 1

Experiments - Results

Method	% Error messages resolved				
	75 - 150	**150 - 225**	**225 - 300**	**300 - 375**	**375 - 450**
DeepFix	30.6	26.7	30.2	35.1	27.4
RLAssist	46.6	42.5	41.2	34.3	32.9
MFreeTopic	45.3	41.5	43.6	38.7	38.6
MBasedMean	47.2	44.6	37.6	35.7	34.1
MBasedTopic	44.9	43.2	41.7	39.3	38.7

Table 3: Variation of % error messages resolved with length of the test program for all baseline models

Conclusion + Future Works

- We observe that the partition topic based averaging method outperforms mean-pooling based embeddings for *longer* programs
- The Model-based approach slighly outperforms the model-free approach. More detailed experimentation is required to infer the precise reasons for this improvement
- Currently experimenting to compare the sample efficiencies of the model based and model free approaches
- One important future work is to incorporate the correction of semantic errors (in addition to syntactic errors)