Analyzing CS1 Student Code Using Code Embeddings

Robert Bazzocchi[1], Micah Flemming[2], Lisa Zhang[2] [1]University of Toronto, [2]University of Toronto Mississauga

Research Questions:

Can we learn good vector representations (vector *embeddings)* of student code?

Can we use these representations to reason about student code?

Background:

Many automated tools have been developed to understand and provide feedback to computer programming students [4].

Neural networks have been used to learn embeddings of words (e.g. word2vec [1] and GloVe [2]) and images (e.g. image auto encoders [3]).

Can we do the same with student code?

Data:

CS1 student code submission to a Python programming problem about modifying lists. A total of 8,476 syntactically correct submissions, split into 5,086 for training, 1,695 validation and 1,695 test.

Automatically generated 100 unit tests.

Future Work:

Can we use code embeddings to predict student understanding of concepts?

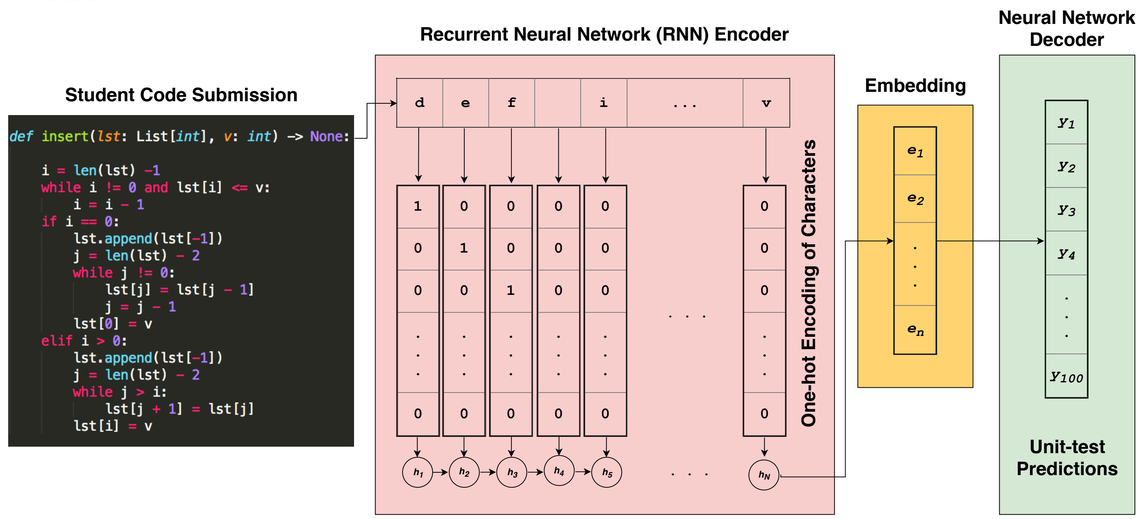
Can we use code embeddings to help give students feedback?

Key References:

[1] Mikolov et al. (2013) *Distributed Representations of Words and Phrases and their Compositionally*
[2] Pennington et al. (2014) *GloVe: Global Vectors for Word Representation*
[3] Cheng et al. (2018) *Deep Convolutional AutoEncoder-based Lossy Image Compression*
[4] Keuning et al. (2019) *A Systematic Literature Review of Automated Feedback Generation for Programming Exercises*

Acknowledgements: We thank Prof. Andrew Petersen for the data and the helpful feedback. We thank Haotian Yang for some auxiliary model results.

MODEL:



We treat each syntactically-correct student code submission as a sequence of characters.

A character-level RNN that uses bi-directional Gated Recurrent Units (GRU). It takes the sequence of characters in a code submission as input, and outputs its vector embedding.

A multi-layer perceptron (MLP) with two fully connected layers. It takes the vector embedding as input, and predicts the pass/fail of each of the 100 unit tests.

Training:

The entire model (encoder + decoder) is trained together to predict the unit test output (pass/fail) of the code submissions in our training set.

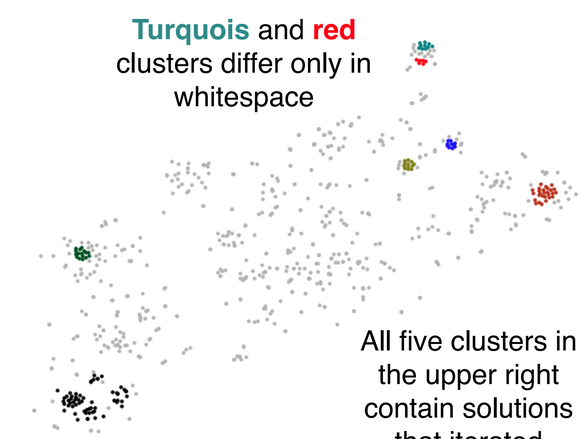
Although the encoder is the only part of the model that we use in our analysis, the decoder helps the encoder learn good embeddings that captures the unit test pass/fail information.

Final Training Accuracy	90%
Final Validation Accuracy	84%
Final Test Accuracy	85%

ANALYSIS:

Code Clustering:

Distances in the embedding space appears to be meaningful. We find seven tight clusters of *correct* solutions (dimensionality reduction via tSNE):



Auxillary (New) Tasks:

Can we use the embeddings to make predictions about other features of the code?

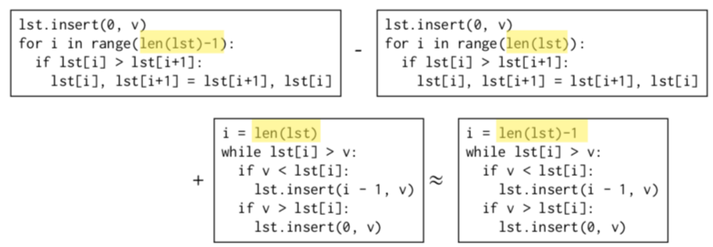
Task 1 Error Identification: Identify whether a code submission throws a Python exception exception.

Task 2 Error Classification: Identify what type of error a code submission throws. We grouped the errors in to four classes: No error, IndexError, RecursionError or Timeout, and Other.

	Test Accuracy	
Input	**Task1**	**Task2**
Unit test	79%	62%
Code	81%	75%
Embedding	**91%**	**87%**

Analogies:

Embeddings of *words* [1] often have structures with analogies like "king - man + woman = queen". Is there evidence of similar analogies in our code embedding?



In this example, we:

1. Embed the first three pieces of code
2. Compute the addition and subtraction of the vectors
3. Find the code submission closests to resulting vector

This result suggests that direction vectors in the embedding space are meaningful, and can represent complex programming concepts.