Locating Bugs in CS1 Code with Recurrent Neural Networks

Lucas Roy
University of Toronto
Mississauga
lucas.roy@mail.utoronto.ca

Haotian Yang
University of Toronto
Mississauga
haotian.yang@mail.utoronto.ca

Lisa Zhang
University of Toronto
Mississauga
lczhang@cs.toronto.edu

ABSTRACT
This paper presents the work in progress towards generating automatic feedback to student solutions to CS1 coding questions that highlights regions of the code that may be problematic. We use a Recurrent Neural Network to generate such feedback. We use a data-driven approach to train the model by re-purposing past student submissions and student corrections to their own code. We present preliminary results of a model that works on one problem. We hope to eventually integrate this kind of feedback in a CS1 setting.

Author Keywords
CS1, Intelligent tutoring systems, learning analytic, Machine learning, bug localization, recurrent neural network

CCS Concepts
•Social and professional topics → Computer science education;

INTRODUCTION
CS1 students typically receive feedback to programming questions in the form of unit test passes and failures. Such feedback is immediate, but does not help students understand *why* their solution may be incorrect. To that end, many Computer Science educators have developed ways to provide students automated and personalized feedback and hints [11, 9, 5, 12].

We would like to automatically generate feedback for submissions to programming problems by highlighting regions of the code that may be incorrect, where a student should pay particular attention. The feedback should identify both syntax and semantics issues. This approach is similar to the task of automatic bug detection and automatic bug correction [2, 14], though students should correct their own bugs.

We build a machine learning model that localizes errors in CS1 student code. We use a data-driven approach similar to that of [12]: we use past student submissions to programming problems, and the corrections that past students make to their own code. The model is summarized in Figure 2. We show preliminary results of the model trained on past student submissions to the programming problem in Figure 1.

Sixth SPLICE Workshop at L@S 2020 “Building an Infrastructure for Computer Science Education Research and Practice at Scale”, SPLICE’20, August 12, 2020, Virtual Event

```
def check_password(passwd: str) -> bool:
    """
    A strong password has a length greater than or equal
    to 6, contains at least one lowercase letter, at
    least one uppercase letter, and at least one digit.
    Return True iff passwd is considered strong.

    >>> check_password('I<3csc108')
    True
    """
```

Figure 1. The CS1 programming question that we study.

Term	Split	Submissions	Students	Pairs
2015	Train/Valid	7,131	768	419
2016	Train/Valid	8,869	816	515
2017	Train/Valid	11,456	1,066	755
2019	Train/Valid	6,310	880	559
2018	Test	3,676	375	247

Table 1. Summary of data used for training, validation and test.

RELATED WORK
Automatically locating and fixing bugs is an important problem with a rich literature [13]. Bhatia and Singh [2] corrects syntax errors by combining a generative recurrent neural network with a heuristic algorithm. Gupta et al. [6] uses a multilayer sequence-to-sequence network with attention to locate bugs, and correct them one-by-one. Pu et al. [14] correct both syntax and semantic errors by using a modified sequence-to-sequence network. We also draw inspiration from the work of Allamanis et al. [1] and Bhoopchand et al. [3], who uses sequence-of-tokens representation and neural representations.

STUDENT CODE SUBMISSIONS DATA
The CS1 course at our institution uses a flipped classroom, and delivers course content using an online educational computer programming platform called PCRS [10]. The system pairs video-based instruction with multiple choice, short-answer, and Python coding questions. Coding questions include those similar to Figure 1. Students receive immediate feedback when completing these questions, in the form of unit-test outputs.

PCRS collects interaction and performance data, and logs student attempts to each question. We repurpose the historical record from 2015-2019 to train our model. For our preliminary model, we use student submissions for the single coding question in Figure 1.

We extract pairs of consecutive submissions submitted by the same student, where the earlier submission does not pass all unit tests, but the later submission *does* pass all unit tests. We

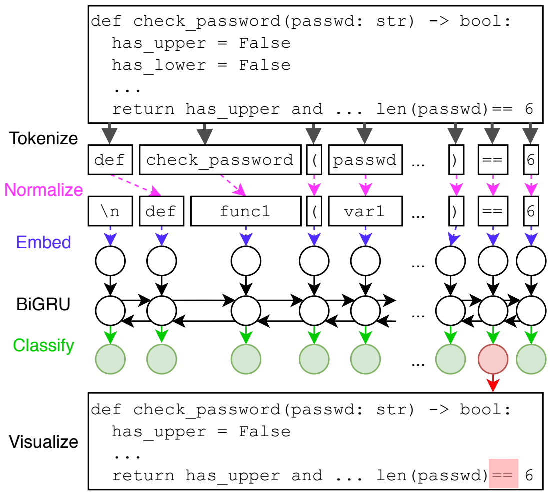

Figure 2. RNN model for generating visual feedback.

keep only consecutive submissions where less than 30% has changed. The difference between the two submissions is key, as we use it to generate ground-truth labels for identifying which part of the incorrect code should be changed. We hold out the 247 consecutive submission pairs from the 2018 term for testing, and split the remaining data from the other terms into 1,987 for training and 221 pairs for validation.

To generate the ground-truth classification labels for the earlier, incorrect submission, we identify tokens that changed in the later, correct submission. We use the Python libraries `tokenize` and `difflib` to tokenize both submissions into sequences of tokens, and compare these sequences. We annotate each token in the earlier code submission with one of 6 labels, shown in Table 2.

1. Token unchanged	2. Token removed	3. Token replaced
4. Token unchanged + new token appended	5. Token removed + new token appended	6. Token replaced + new token appended

Table 2. The six label classes that we use to annotate each token in the incorrect submission code.

CODE HIGHLIGHT RNN MODEL

The Recurrent Neural Network Model for localizing bugs is shown in Figure 2. We represent each code submission as a sequence of tokens, extracted using the Python library `tokenize`. We normalize the token sequences by prepending a newline token (to capture addition of code at the very beginning), and replacing function and variable names with canonical names (like `func1`, `func2`, `var1`, `var2`) for consistency across similar programs with different variable names.

The neural model has three trainable layers: an embedding layer that maps each unique token to a vector, a bi-directional GRU [4] layer, and a classification layer that predicts the label of each token. We also use Batch Normalization [7] before the final classification layer, which prevents overfitting.

We build and test two variations of the model in Figure 2. In the "6-Class" model, we use the labels in Table 2, and perform 6-way classification on each token. In the "Binary" model, we combine classes 2-6 and perform binary classification on

```
def check_password (passwd : str) -> bool :
    if len (passwd) < 7 :
        return False
    alpha = "abcdefghijklmnopqrstuvwxyz"
    l_strong = False
    u_strong = False
    d_strong = False
    for a in passwd :
        if a in alpha :
            l_strong = True
        elif a in alpha.upper () :
            u_strong = True
        elif a.isdigit () :
            d_strong = True
    return l_strong & u_strong & d_strong
```

Figure 3. Example feedback generated using our model on an element of the test set. The model identifies the actual issue in line 2, but also highlights the unusual bitwise & operations in the last line.

whether the token needs to be modified. We choose an embedding size of 128, a GRU hidden size of 128, cross-entropy loss, and the Adam optimizer [8] with a learning rate of 0.001. We used the validation sets to select these hyperparameters.

RESULTS

Table 3 shows the model accuracies. Since the labels are heavily imbalanced, we report three other accuracy measures (1) "Unchanged" accuracy over all unchanged tokens, (2) "Binary" accuracy in identifying tokens that requires modification, without regard for the actual label, and (3) "Modified" accuracy in correctly classifying a token in classes 2-6. Note that for the "Binary" model, we cannot compute the latter figure because we combined these class labels.

	6-Class Model			Binary Model		
Accuracy	Train	Valid	Test	Train	Valid	Test
Overall	99.3%	97.2%	96.0%	99.6%	97.2%	97.0%
Unchanged	99.8%	99.0%	99.1%	99.8%	99.1%	99.5%
Binary	84.7%	39.2%	31.8%	93.7%	34.3%	26.4%
Modified	83.9%	35.5%	8.6%	N/A	N/A	N/A

Table 3. Model accuracy over tokens with various class labels.

The binary classification accuracy for the 6-class model is about 32%. We did not yet fine-tune the model, and always predict the *most likely* token. The 6-class and binary models has an Area Under the ROC Curve (AUC) of 0.829 and 0.832, respectively.

Figure 3 shows one particularly interesting example of feedback generated on a test set code. Our model highlights the actual semantic issue on line 2, but also highlights where the return statement uses bitwise & rather than logical and, even though using the bitwise operation still passes the unit tests.

CONCLUSION AND FUTURE WORK

We build a preliminary RNN model for locating bugs in student code to be used for providing feedback on which part of their code a student should pay particular attention to. This model shows promise in being able to identify both syntax and semantic errors. We hope to leverage more recent approaches in program representation to develop tools to help CS1 students receive better automated feedback.

Acknowledgements
We would like to thank Andrew Petersen for providing the PCRS data.

REFERENCES

[1] Miltiadis Allamanis, Hao Peng, and Charles A. Sutton. 2016. A Convolutional Attention Network for Extreme Summarization of Source Code. *CoRR* abs/1602.03001 (2016). http://arxiv.org/abs/1602.03001

[2] Sahil Bhatia and Rishabh Singh. 2016. Automated correction for syntax errors in programming assignments using recurrent neural networks. *arXiv preprint arXiv:1603.06129* (2016).

[3] Avishkar Bhoopchand, Tim Rocktäschel, Earl T. Barr, and Sebastian Riedel. 2016. Learning Python Code Suggestion with a Sparse Pointer Network. *CoRR* abs/1611.08307 (2016). http://arxiv.org/abs/1611.08307

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[5] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing Syntax Error Messages Appears Ineffectual. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)*. ACM, New York, NY, USA, 273–278. DOI: http://dx.doi.org/10.1145/2591708.2591748

[6] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. Deepfix: Fixing common c language errors by deep learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[7] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*. 448–456.

[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] V.C.S. Lee, Y.T. Yu, C.M. Tang, T.L. Wong, and C.K. Poon. 2018. ViDA: A virtual debugging advisor for supporting learning in computer programming courses. *Journal of Computer Assisted Learning* 34 (02 2018). DOI:http://dx.doi.org/10.1111/jcal.12238

[10] Andrew Petersen. PCRS. https://mcs.utm.utoronto.ca/~pcrs/pcrs/. (????). Accessed: 2019-01-20.

[11] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. 2015. Learning Program Embeddings to Propagate Feedback on Student Code. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 1093–1102. http://proceedings.mlr.press/v37/piech15.html

[12] Thomas Price, Rui Zhi, and Tiffany Barnes. 2017. Evaluation of a Data-Driven Feedback Algorithm for Open-Ended Programming. *International Educational Data Mining Society* (2017).

[13] Thomas W Price, Yihuan Dong, Rui Zhi, Benjamin Paaßen, Nicholas Lytle, Veronica Cateté, and Tiffany Barnes. 2019. A comparison of the quality of data-driven programming hint generation algorithms. *International Journal of Artificial Intelligence in Education* 29, 3 (2019), 368–395.

[14] Yewen Pu, Karthik Narasimhan, Armando Solar-Lezama, and Regina Barzilay. 2016. sk_p: a neural program corrector for MOOCs. *CoRR* abs/1607.02902 (2016). http://arxiv.org/abs/1607.02902