Personalized Feedback for Open-Response Mathematical Questions using Long Short-Term Memory Networks

Joshua J. Michalenko
Rice University
jjm7@rice.edu

Andrew S. Lan
Princeton University
andrew.lan@princeton.edu

Richard G. Baraniuk
Rice University
richb@rice.edu

ABSTRACT

In this paper, we explore the problem of automatic grading and feedback generation for open-response mathematical questions. We resort to the long short-term memory (LSTM) network to learn the simple task of polynomial factorization and use the trained network for grading and feedback. We use Wolfram Alpha to synthetically generate a training dataset that consists of step-by-step responses to polynomial factorization questions to train the LSTM network. Preliminary results validate the efficacy of LSTMs in learning to factor low-order polynomials; we also demonstrate how to leverage the trained network for automatic grading and personalized feedback generation.

Keywords

Automatic grading, Feedback generation, Long short-term memory networks, Mathematical expressions

1. INTRODUCTION

In spite of tremendous advances in technology for education, learning today largely remains a "one-size-fits-all" approach. Personalized learning is the manifestation of *differentiation*, the idea that all students access content and develop mastery differently. The personalized learning experience necessitates a scalable approach since the number of students is much larger than the number of teachers. Many recent advances focus on using machine learning algorithms to analyze student data, but mostly resort to limited utility multiple-choice questions for grading a feedback [5].

The mathematical language processing (MLP) framework proposed in [4] is the first automatic grading and feedback generation tool for open-response mathematical questions. MLP is capable of automatically grading a large number of student responses requiring minimal human effort, but lacks an effective feedback mechanism because it not capable of truly understanding mathematics, and is therefore unable to provide informative feedback. A series of recent tools based on recurrent neural networks (RNNs) [3] have found great success in various NLP tasks (e.g., machine translation, image captioning, etc.) and predicting the output of simple computer code [7]. Natural language processing for the purposes of grading and feedback has also made substantial progress in several restricted domains including essay evaluation and mathematical proof verification [2, 6]. These successes inspires us to use RNNs to analyze responses to mathematical questions due to their sequential, step-by-step format and their algorithmic nature. They support our belief that LSTMs have the ability to learn simple mathematical operations such as factoring polynomials from data and providing relevant feedback.

1.1 Contributions

In this paper, we apply the LSTM network [3], a type of RNN, to try to understand simple mathematics for automatic grading and feedback generation for open-response mathematical questions. In particular, we study the simple problem of *polynomial factorization* due to the fact that responses to polynomial factorization questions are typically short and require only simple mathematical operations. We first generate a synthetic dataset using the Wolfram Alpha API consisting of responses (step-by-step solutions with mathematical expressions and text explaining the mathematical operations performed) to polynomial factorization questions. We then train multiple LSTM networks on the dataset and evaluate their performance on factoring previously unseen polynomials. Preliminary results show that the trained character level networks can factor previously unseen polynomials up to the second order with sufficient accuracy, after training on enough examples. More importantly, we showcase how the trained networks have the potential for automatic grading and feedback generation for open-response mathematical questions.

We emphasize that our proposed method has the capability to go beyond Wolfram Alpha. First, the ability of the trained LSTM networks to generalize to previously unseen examples enables *transfer* between domains, i.e., these networks have the capability of learning a rule in a certain context and apply it in another context. This property enables a LSTM network to build on its own knowledge as more and more training data becomes available, which is a much more scalable approach than the rules-based Wolfram Alpha system, which requires new rules to be manually coded for every new domain.

2. EXPERIMENTS

Experimental setup. We generate factorable polynomials that are subsequently used by the Wolfram Alpha API to produce responses on how to fully factor these polynomials. The responses include step-by-step solutions that consist of a series of mathematical expressions that end up in a fully-factored final form, together with concise text describing the mathematical operations involved. The data generation process is limited to polynomials with a single variable, co-

	Character Level % Error			Expression Level % Error		
# units	1 Layer	2 Layer	3 Layer	1 Layer	2 Layer	3 Layer
50	31.11	20.98	20.40	87.93	80.76	78.28
200	11.79	10.68	10.12	68.55	59.39	56.80
512	12.94	**8.21**	10.32	42.38	39.94	**38.95**

Table 1: Character and expression level misclassification errors on the test set. Performance of the best models are highlighted in bold.

efficients that are less than 10 and up to the third order. We construct a training dataset including 200,000 responses to various factoring questions this way. A test dataset is constructed with 20 first, 20 second, and 20 third order polynomials to be factored. We emphasize that, while for the simple task of polynomial factorization, Wolfram Alpha is able to generate the correct response, our aim is to develop a method that can generalize to more complicated mathematical operations that are too complicated for a rules-based system like Wolfram Alpha to cover. We train our LSTM networks to operate on a character-by-character level, i.e., use each character in a response as input and output data at each time instant. We train 9 different LSTM networks with varying number of hidden units ($N \in \{50, 200, 512\}$) and layers (1, 2, and 3). We use 95% of the generated training dataset for training and 5% as the validation dataset; We train the LSTM networks for a total of 50-150 epochs or terminate the training process early if the validation error shows minimal change across 10 epochs. In order to achieve faster training, we apply the curriculum learning approach [1], i.e., we start by training the LSTM networks on factorizations of first order polynomials until the validation error cannot be further reduced, and then proceed to train on responses factoring second order polynomials and beyond.

Results and discussion. We evaluate the performance of our trained LSTM networks on factoring previously unseen polynomials using two metrics. The first metric computes the character-level misclassification error rate by comparing every character in the correct factorization to the maximum-likelihood predicted character by the trained LSTM network. The second metric computes the expression-level misclassification error rate by comparing every full mathematical expression in the correct factorization to the full predicted expression by the trained LSTM network; a successful classification means that the entire expression is correctly predicted.

Experimental results for all 9 LSTM networks on both metrics are shown in Table 1. In general, LSTM networks with more hidden units and layers achieves lower misclassification error rates. We note that the expression-level misclassification rate is much higher (the best model achieves an error rate of 38.95%) than the character-level misclassification rate (the best model achieves an error rate of 8.21%). This observation is not surprising since correctly predicting the entire expression is much more difficult than successfully predicting a character. Moreover, we observe that the best model achieves error rates of 0% and 15%, respectively, on factoring first and second order polynomials but a 100% error rate on third order polynomials. This result is due to the fact that factoring third order polynomials is hard since it requires first factoring out a second order polynomial as an intermediate step.

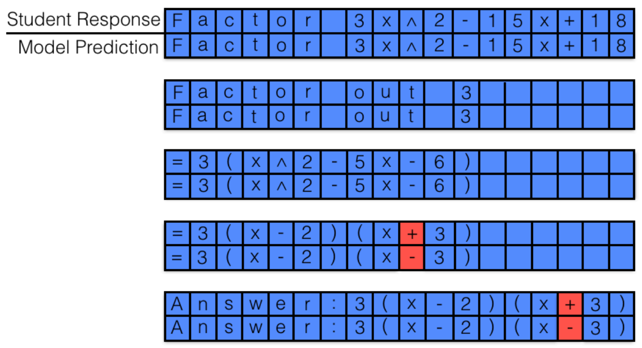

Figure 1: Illustration of how to use of a trained LSTM network to detect when a student's response deviates from the correct response.

Using trained LSTM networks for grading and feedback. We now illustrate how the trained LSTM networks can be used for automatic grading and feedback generation. Figure 1 shows a typical use case with an actual student response and a direct comparison to the maximum-likelihood character the trained LSTM network predicts given the previous characters as input. For automatic grading, we can calculate the predictive likelihood of every character in a student's response using a trained LSTM network. We can then assign a grade to a response by its total predictive likelihood; since our LSTM networks are trained on correct responses, a correct response will have a higher predictive likelihood than an incorrect one. For personalized feedback generation, we can automatically alert a student that they might have made an error if the predictive likelihood of the next input character is lower than a certain threshold. In Figure 1, such an error is shown in red where the student response contains a character that the trained LSTM network predicts as highly unlikely. Using these predictive probabilities, we can also automatically provide hints to a student about the most likely next expression in case they get stuck.

3. REFERENCES

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proc. 26th Intl. Conf. Mach. Learn.*, pages 41–48, June 2009.

[2] M. Cramer, B. Fisseni, P. Koepke, and D. Kühlwein. The naproche project controlled natural language proof checking of mathematical texts. In *Cont. Nat. Lang.*, pages 170–186, 2009.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[4] A. S. Lan, D. Vats, A. E. Waters, and R. G. Baraniuk. Mathematical language processing: Automatic grading and feedback for open response mathematical questions. In *Proc. 2nd ACM Conf. Learn. at Scale*, pages 167–176, Mar. 2015.

[5] A. S. Lan, A. E. Waters, C. Studer, and R. G. Baraniuk. Sparse factor analysis for learning and content analytics. *J. Mach. Learn. Res.*, 15:1959–2008, June 2014.

[6] A. Naumowicz and A. Kornilowicz. A brief overview of mizar. In *In Proc. 22nd Intl. Conf Theorem Proving in Higher Order Logics*, pages 67–72, 2009.

[7] W. Zaremba and I. Sutskever. Learning To Execute. *arXiv preprint arXiv:1410.4615*, pages 1–25, Feb. 2015.