Deep Knowledge Tracing On Programming Exercises

Lisa Wang
Stanford University
Stanford, USA
lisa1010@cs.stanford.edu

Angela Sy
Stanford University
Stanford, USA
angelasy@stanford.edu

Larry Liu
Stanford University
Stanford, USA
hrlarry@stanford.edu

Chris Piech
Stanford University
Stanford, USA
piech@cs.stanford.edu

ABSTRACT
Modeling a student's knowledge state while she is solving exercises is a crucial stepping stone towards providing better personalized learning experiences at scale. This task, also referred to as "knowledge tracing", has been explored extensively on exercises where student submissions fall into a finite discrete solution space, e.g. a multiple-choice answer. However, we believe that rich information about a student's learning is captured within their responses to open-ended problems with unbounded solution spaces, such as programming exercises. In addition, sequential snapshots of a student's progress while she is solving a single exercise can provide valuable insights into her learning behavior. In this setting, creating representations for a student's knowledge state is a challenging task, but with recent advances in machine learning, there are more promising techniques to learn representations for complex entities. In our work, we feed the embedded program submissions into a recurrent neural network and train it on the task of predicting the student's success on the subsequent programming exercise. By training on this task, the model learns nuanced representations of a student's knowledge, and reliably predicts future student performance.

Author Keywords
Educational data mining; Online education; Personalized Learning; Knowledge tracing; Machine learning; Deep learning, Representation learning; Sequential modeling.

INTRODUCTION
With the inception of online learning platforms, educators around the world can reach millions of students by disseminating course content through virtual classrooms.

However, in these online environments, teachers' ability to observe students is lost. Understanding a student's incremental progress is invaluable. For instance, if a teacher watches a student work through an exercise, she can observe the student's strengths, knowledge gaps as well as motivation. Hence, the process by which the student reaches the final solution is as important as the solution itself. We attempt to encode these markers of progress.

We performed representation learning with recurrent neural networks to understand a student's learning trajectory as they solve open-ended programming exercises from the *Hour of Code* course, a Massive Open Online Course (MOOC) on *Code.org*. The deep learning model trains on a student's history of past code submissions and predicts the student's performance on the next exercise. The model is able to learn meaningful feature representations for a student's series of submissions and hence does not require manual feature selection, which would be very difficult for open-ended exercises. Furthermore, the learned representations can be used for other related tasks, such as predicting an intervention.

RELATED WORK

Representation Learning with Recurrent Neural Networks
In the field of machine learning, representation learning is the task of learning a model to create meaningful representations from low-level raw data inputs. The goal of representation learning is to reduce the amount of human input and expert knowledge needed to preprocess data before feeding it into machine learning algorithms [1].

In contrast to manually selecting high-level features, representation learning algorithms are trained to extract features directly from raw input, e.g. from words in a document.

The learned representations can be used for other related tasks as well. E.g. In *word2vec* [6], word representations were trained on predicting context words but were then used for document classification and translation.

In recent years, representations learned by Deep Neural Networks (DNNs) have outperformed other methods in many tasks including image classification [1]. Empirically, DNNs do particularly well when the raw data has high semantic complexity and manually choosing features is not only tedious, but often insufficient.

Recurrent neural networks (RNNs) are a subtype of neural networks that takes inputs over multiple timesteps, which makes them particularly suited for learning representations on sequential data. RNNs have been successfully applied to modeling and translating natural language sentences, performing speech recognition, and completing other tasks on data with temporal relationships.

Knowledge Tracing

The task of knowledge tracing can be formalized as: given observations of interactions $x_0 \dots x_t$ taken by a student on a particular learning task, predict aspects of their next interaction x_{t+1} [2].

RNNs have been applied to the knowledge tracing task in the past to understand how students progress through different problems. Piech et al. applied RNNs to data from Khan Academy's online courses to predict student performance [8]. The authors found that RNNs can robustly predict whether or not a student will solve a particular problem correctly given the accuracy of historic solutions. More recent answers are a more accurate representation of students' current state, hence our DKT models use RNNs with long short-term memory (LSTM) to weight recent inputs more heavily while still taking into account historic inputs, a method developed in Hochreiter and Schmidhuber's paper [4]. Other models that are designed to take low dimensional inputs, such as IRT and modifications of Bayesian Knowledge Tracing [13] [7], sometimes outperform the initial version of Deep Knowledge Tracing (DKT) [12] [5]. However, DKT does not require student interactions to be manually labeled with relevant concepts and the RNN paradigm was designed to take vectorized inputs, hence it can utilize inputs that extend beyond the discrete inputs of traditional models [3]. These properties make the model an appropriate fit to understand trajectories of open-ended student responses.

A limitation with the work of Piech et al. is that it does not fully leverage the promise of using neural networks to trace knowledge. The dataset used only contained binary correct/incorrect information about a student's final answer. In contrast, the *Hour of Code* dataset offers richly structured data in the form of program submissions.

Previous work in deep knowledge tracing has looked at student responses over multiple exercises, but not within an exercise. Our method focuses on a student's sequence of submissions within a *single* programming exercise to predict future achievement. We model student learning and progress by capturing representations of the current state of a student's program as they work through the exercise. When focusing exclusively on the final submission, these steps are ignored.

TASK DEFINITION

In order to create representations of a student's current state of knowledge, we chose the following training task:

Based on a student's sequence of code submission attempts ***over time*** *(hereby, their "trajectory") on a programming exercise, predict whether the student will successfully complete the next programming exercise within the same course.*

DATASET: HOUR OF CODE EXERCISE 18

The *Hour of Code* course consists of twenty introductory programming exercises aimed at teaching beginners fundamental concepts in programming. Students build their programs in a drag-and-drop interface that pieces together blocks of code. The number of possible programs a student can write is infinite since submissions can include any number of block types in any combination. A student can run their code multiple times for any exercise. These submissions provide temporal snapshots to track the student's learning progress. The student submission data for Exercises 4 and 18 from this course are publicly available on `code.org/research`.
For our experiments, we focus on the sequences of intermediate submissions on Exercise 18. It covers multiple concepts such as loops, if-else statements, and nested statements. This Exercise 18 data set contains 1,263,360 code submissions, of which 79,553 are unique, made by 263,569 students. 81.0% of these students arrived at the correct solution in their last submission.

Student Trajectory Within A Single Exercise

For each student, we focus on her sequence of code submissions for Exercise 18 and her success on Exercise 19. Each code submission is represented as an abstract syntax tree (AST). Given a student's trajectory on Exercise 18, our task is to predict their success on the next coding challenge. We believe success on the next exercise is a good indicator of the student's learning progress since succeeding challenges add new concepts incrementally.

Since the *Hour of Code* exercises do not have a bounded solution space (open-ended solutions), students could produce an infinitely long trajectory of submissions. We noted that the accuracy of student submissions have a high correlation with trajectory lengths. For instance, the vast majority of students with trajectory length 1 solved the problem with their very first submission. Hence, we chose to control for trajectory length and train our models independently for each trajectory length. We ran experiments on 9 data subsets with trajectory lengths ranging from 2 to 10, including submissions from 81,880 students.

MODEL

Recurrent Neural Network Model for Student Trajectories

For our model, we used a Long Short Term Memory (LSTM) RNN architecture, which is a popular extension to plain RNNs since it reduces the effect of vanishing gradients [4].

For our task, x_t is the program embedding vector of a student's trajectory at time step t. E.g., assume that a student's trajectory consists of k submissions. These are converted into program embeddings which form a sequence of k embeddings. This sequence is fed into an RNN, whose final hidden state is passed through a fully connected layer and a subsequent softmax layer. The output $\hat{y}$ of the softmax layer is a probability distribution over two binary classes, indicating whether the student successfully solved the next exercise.

Recursive Neural Network for Program Embeddings

In order to expand DKT to understand students as they produce rich responses over time within an exercise, a necessary task is to create meaningful embeddings of their responses. Based on Piech et al.'s previous work on creating program embeddings for student code[9], we trained a *recursive* neural network which allowed us to vectorize the AST representation of student programs. Recursive neural networks that learn embeddings on trees were developed by the NLP community to vectorize sentence parse trees [11]. We extended this idea to coding programs.

In our program embedding based model, a subtree of the AST rooted at a node j is represented by a vector which is computed by a linear combination of subtree representations rooted at the children of j. Representation of leaf nodes in the AST are parameters learned by the model.The hidden activations at the root of the AST are used as embeddings.

Baseline Model

For the baseline, we chose two features for each student's trajectory $T = trajectory(s)$, which is the published state of the art at predicting completion of the next exercise on this dataset, shown to be highly correlated with learning outcome and performance on the next exercise.

- The Poisson path score of the trajectory T as defined in [10]. Intuitively, the path score is an estimate of the time it will take a student to complete the trajectory series. The path score of a student trajectory has previously been related to student retention in sequential challenges [10].

$$pathScore(T) = \sum_{x \in T} \frac{1}{\lambda_x}$$

  where λ_x is the number of times AST x appears in student submissions.

- Indicator feature of student success on current exercise 18. A student succeeded if they ended the trajectory with the solution AST. (In *Hour of Code*, there is a unique solution to the exercises, since the solution has to satisfy both functionality and count requirements.)

To summarize, the two-dimensional feature vector $\phi(s)$ for a student s is built as follows:

$$\phi(s)[0] = pathScore(trajectory(s))$$

$$\phi(s)[1] = \begin{cases} 1 \text{ if } s \text{ solved current exercise,} \\ 0 \text{ otherwise .} \end{cases}$$

Using the features $\phi(s)$ as input and $successNextChallenge(s)$ as a binary label, we trained a simple logistic regression model. A separate model for each data subset of trajectory length.

PRELIMINARY RESULTS

To compare our proposed model to the baseline model, we trained and evaluated each model on each data subset separately. Each data subset contains student trajectories of the same length. For both the baseline model and the LSTM model, we used 90% of the data set to perform training and the remaining 10% for testing.

Figure 1 illustrates test accuracies of both the baseline model and the LSTM model on the 9 data subsets. We can observe that the LSTM model consistently outperforms the state of the art baseline by around 5%.

This result is significant since the input we feed into the LSTM model consists of program embeddings, and not handpicked features like success on current problem. Our model identified trajectories that show more promise and improved student learning.

The ability to understand trajectories suggests that the representations used for the programs within the trajectories were also meaningful. The program embeddings were trained to predict the output of any given student program. Our program embedding model was able to correctly predict the output for 96% of the programs in a hold out set, compared to a 54% accuracy from always predicting the most common output.

FUTURE WORK

Further Analysis of Representations

Since our model is able to predict future student success reliably purely based on a sequence of program submissions, it learned representations for the trajectories that are meaningful at least for this particular task. We have noticed a distinct clustering of trajectory representations, and will further investigate whether these clusters are indicative of certain types of learning behaviors or knowledge states. These trajectory representations of students could potentially be used for subsequent decision making, e.g. choosing the next exercise or intervention.

Extending to Multiple Exercises

Subject to data availability, we would like to extend our analysis to cover a series of exercises in the *Hour of*

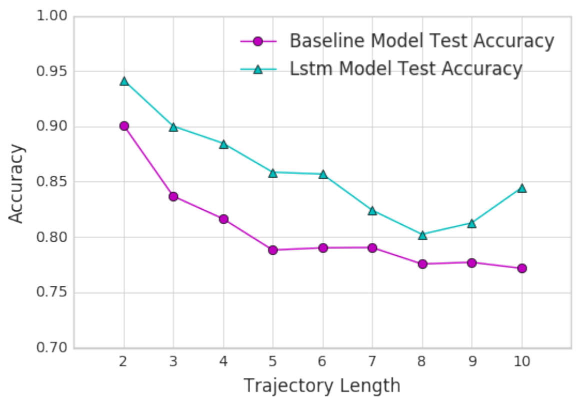

Figure 1. Test accuracies of baseline model and LSTM model.

Code course. By tracing a students' progress not just over submissions to multiple related exercises, we hope to learn a more accurate picture of the student's knowledge state. In addition, by jointly training our model over multiple exercises, the model could potentially learn more general representations of student knowledge and behavior that are not exercise specific.

CONCLUSION

We have shared a preliminary investigation into knowledge tracing over richly structured data with the goal of gaining a deeper understanding of a student's current knowledge and the progression of their learning while she is progressing through a complex exercise. The only input we use to trace a student's knowledge are the raw code submissions. The model we proposed is generalized, reducing the need for handpicked features by leveraging the power of internal feature learning in deep neural networks.

Our results suggest that our model can reliably predict a student's success on the next problem purely based on their sequence of code submissions. Thus, the learned trajectory representations contain meaningful information about a student's learning and could be potentially used for making decisions about interventions, e.g. giving a hint, providing feedback or suggesting a new exercise. Not just if, but how a student arrives at a solution, is crucial to understanding a student's knowledge acquisition and could improve the effectiveness of intelligent tutoring systems.

ACKNOWLEDGEMENTS

We thank *Code.org* for providing the *Hour of Code* data set to the research community.

REFERENCES

1. Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.
2. Albert T Corbett and John R Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction* 4, 4 (1994), 253–278.
3. Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507.
4. Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
5. Mohammad Khajah, Robert V Lindsey, and Michael C Mozer. 2016. How deep is knowledge tracing? *arXiv preprint arXiv:1604.02416* (2016).
6. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
7. Zachary A Pardos and Neil T Heffernan. 2010. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *International Conference on User Modeling, Adaptation, and Personalization*. Springer, 255–266.
8. Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. 2015a. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*. 505–513.
9. Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas J Guibas. 2015b. Learning program embeddings to propagate feedback on student code. *CoRR abs/1505.05969* (2015).
10. Chris Piech, Mehran Sahami, Jonathan Huang, and Leonidas Guibas. 2015c. Autonomously generating hints by inferring problem solving policies. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*. ACM, 195–204.
11. Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, Vol. 1631. Citeseer, 1642.
12. Kevin H Wilson, Yan Karklin, Bojian Han, and Chaitanya Ekanadham. 2016. Back to the Basics: Bayesian extensions of IRT outperform neural networks for proficiency estimation. *arXiv preprint arXiv:1604.02336* (2016).
13. Michael V Yudelson, Kenneth R Koedinger, and Geoffrey J Gordon. 2013. Individualized bayesian knowledge tracing models. In *International Conference on Artificial Intelligence in Education*. Springer, 171–180.