Learning to Represent Student Knowledge on Programming Exercises Using Deep Learning

Lisa Wang, Angela Sy, Larry Liu, Chris Piech
Stanford University
{lisa1010@cs, angelasy, hrlarry, piech@cs}.stanford.edu

ABSTRACT

Modeling student knowledge while students are acquiring new concepts is a crucial stepping stone towards providing personalized automated feedback at scale. We believe that rich information about a student's learning is captured within her 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. 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 submission sequence into a recurrent neural network and train it on two tasks of predicting the student's future performance. By training on these tasks, the model learns nuanced representations of a student's knowledge, exposes patterns about a student's learning behavior, and reliably predicts future student performance. Even more importantly, the model differentiates within a pool of poorly performing students and picks out students who have true knowledge gaps, giving teachers early warnings to provide assistance.

Keywords

Educational data mining; Online education; Personalized learning; Knowledge tracing; Machine learning; Representation learning; Sequential modeling.

1. 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 struggle with an exercise, they see the student's strengths as well as their knowledge gaps. The process by which the student reaches the final solution is equally 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 MOOC on *Code.org*. The deep learning model trains on a student's history of past code submissions and predicts the student's future performance on the current or 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.

1.1 Motivation: Instructional Scaffolding

The widely used pedagogical concept of the zone of proximal development (ZPD) suggests that ideal learning objectives are in a sweet spot of difficulty called the ZPD: more difficult than what the student can accomplish on their own, but not so difficult that they cannot succeed even with guidance [3, 24]. The guidance for accomplishing such challenging-but-achievable objectives is called instructional scaffolding, and it is most effective when personalized to each student's mastery of the material [18].

Scaffolding is particularly difficult in MOOCs–it is hard to personalize instruction to thousands of students at once. While some research has explored the merits of academic habit scaffolding [6] or reciprocal scaffolding with peer collaboration [19] in MOOCs, the most promising work lies in expert scaffolding, which involves an expert, usually a teacher, in the relevant domain of knowledge providing guidance to help students acquire knowledge [**?**]. Effective teachers possess pedagogical content knowledge (PCK), or expertise about not only the domain of knowledge, but also how to best teach that material to learners [21]. Most importantly, PCK helps anticipate where students will struggle.

In existing MOOC research, the expert scaffolding usually takes the form of feedback to students' responses on assignments. Yet, many current systems for automating feedback in MOOCs relies on time-consuming and potentially arbitrary tasks of feature engineering [20] or defining rulesets [22] applicable only to single exercises. This manual encoding of PCK is task-specific and not a generalizable unsupervised process. A more generalizable signal of student failing learning objectives is student attrition from MOOCs. Limited work exists exploring correlations between attrition

and student engagement with MOOC materials [27] or other students (e.g. on discussion forums) [17, 26]. However, to the authors' knowledge, existing MOOC attrition research does not control for student achievement. Often, attrition is merely a downstream symptom of struggling with learning objectives. When students underachieve, their self-concept of themselves as learners may be threatened, which recursively reinforces lower achievement and disengagement [4]. In general, anticipating common domain-specific mistakes with PCK can help preempt them and mitigate subsequent disengagement, and thus the unsupervised *anticipation* of student mistakes is a worthwhile objective for automated systems that can ultimately improve learning.

2. RELATED WORK

Representation Learning with 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 combination of linear functions and nonlinearities stacked in layers allows deep neural networks (DNNs) to learn abstract representations in an efficient manner [1]. Empirically, DNNs do particularly well when the data has high semantic complexity and manually choosing features is not only tedious, but often insufficient. Once the representations are trained on one task, they can be used for other related tasks as well. E.g. In *word2vec* [12], word representations were trained on predicting context words but were then used for document classification and translation. 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 which take inputs over multiple timesteps and are therefore well-suited for learning representations on sequential data with temporal relationships.

2.1 Program Code Embeddings

In order to expand DKT to understand students as they produce rich responses over time within an exercise, a necessary step is to create meaningful embeddings of their program submissions. Piech et al. proposed to use recursive neural networks to create program embeddings for student code[15]. Recursive neural networks that learn embeddings on syntax trees were first developed by the NLP community to vectorize sentence parse trees [23], but are even more applicable to computer programs due to their inherent tree structure, since any program can be represented as an Abstract Syntax Tree (AST).

2.2 Knowledge Tracing (KT) and Deep KT

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} [5]. Piech et al. applied RNNs to data from Khan Academy's online courses to perform knowledge tracing by predicting student performance [14]. The authors found that RNNs can robustly predict whether or not a student will solve a particular problem correctly given their performance on prior problems. Other models that are designed to take low dimensional inputs, such as IRT and modifications of Bayesian Knowledge Tracing [28] [13], sometimes outperform the initial version of Deep Knowledge Tracing (DKT) [25] [10]. 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 [7]. These properties make the model an appropriate fit to understand trajectories of open-ended student responses, which have unbounded input spaces.

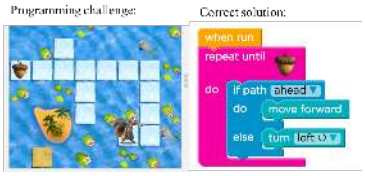

Figure 1: Exercise 18 in the Code.org Hour of Code. Left, the programming challenge. Right, the solution. The challenge is to program the squirrel to reach the acorn, while using as few coding blocks as possible. `https://studio.code.org/hoc/18`.

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 they used only contained binary information about a student's final answer (i.e. correct or incorrect). In contrast, the *Hour of Code* dataset comprises program submissions that each have a boundless solution space. These infinite variations represent richly structured data which we can encode as program embeddings. The ideas presented in this paper work towards a model with the representative capacity to tackle **open-ended** knowledge tracing [9]. In addition, 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 knowledge as they work through the exercise and incrementally submit programs. When focusing exclusively on the final submission, these incremental steps are ignored.

3. EXPERIMENTS: TASK DEFINITIONS

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

- **Task A**:
  Based on a student's sequence of k code submission attempts ***over time*** *(hereby, their "trajectory")* $T = [AST_1, AST_2, ..., AST_k]$ *on a programming exercise, predict at the end of the sequence whether the student will successfully complete or fail the* ***next*** *programming exercise within the same course.*

- **Task B**:
  At each $t \leq k$*, given a student's sequence thus far of t code submission attempts* $T = [AST_1, AST_2, ..., AST_t]$ *on a programming exercise, predict whether the student*

*will successfully complete or fail the **current** programming exercise.*

Task A is pedagogically comparable to predicting whether or not a student will be able to learn a new concept given the way they did or did not learn previous concepts. Phrased differently, a student who quickly (e.g. in few time steps) demonstrates some level of mastery of material (i.e. the goodness of their final submission) should be considered more likely to outperform a student who took a long time and may have struggled before eventually demonstrating the same level of mastery. Meanwhile, **Task B** is pedagogically comparable to detecting whether or not a student is struggling to acquire the present concept as they incrementally engage with the learning objective. In other words, teachers can get real-time information about the learning of the students. We expect **Task B** to be more difficult but also more pedagogically powerful. **Task B** is unlike **Task A** in that **Task B** does not use the full trajectory of a problem, *which would contain post-hoc knowledge of whether or not a student gave up in an earlier learning interaction*, for prediction. All of the students used as inputs in **Task B** can be considered not attrited at least at *some* point in the prediction task. Critically, success on **Task B** would enable teachers to predict at-risk students who may *eventually* give up and not complete the exercise but have not yet given up, where in **Task A**, by the time a teacher knows a student has given up one exercise (the inputs of **A**) as you are trying to guess their success on the next exercise, it may be too late to get the attrited student to rejoin in the learning environment (e.g. re-enroll after dropping out).

4. DATASET: HOUR OF CODE EXERCISE

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. We chose Exercise 18 (over Exercise 4) because it covers multiple concepts such as loops, if-else statements, and nested statements, resulting in more complex and varied code submissions. This Exercise 18 data set contains 1,263,360 code submissions, and, in turn, more varied trajectories of student learning, 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. In comparison, there were 1,138,506 code submissions, of which only 10,293 were unique. The 509,405 students who attempted Exercise 4 succeeded at a 97.8% rate.

Since the *Hour of Code* exercises do not have a bounded solution space, students could produce arbitrarily long trajectories. 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, for both tasks **A** and **B**, we chose to only include trajectories of length 3 or above. Pedagogically, we are also more interested in students who don't get the answer right away, and we speculate that longer trajectories should roughly correlate with greater struggling with the learning objective.

5. MODELS

5.1 Recurrent Neural Network Model for Student Trajectories

Since we would like to capture aspects of a student's learning behavior over time, RNN's are a suitable neural network architecture for our experiments, as RNN's have empirically performed well on sequence modeling tasks in other domains. For both tasks **A** and **B**, 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 [8]. A student's trajectory consists of k program submissions, which are represented as ASTs. Note that an AST contains all the information about a program and can be mapped back into a program. These ASTs are converted into program embeddings using a recursive neural network similar to the one described in [15]. The program embedding is a more compact representation of the original AST, which captures aspects of the program; in particular its functionality. This sequence of program embeddings gets fed into an RNN, as illustrated in Figure 2.

For **task A**, we used a three layer deep LSTM. To make the prediction at the end of the sequence, we pass the hidden state at the last timestep through a fully connected layer and a subsequent softmax layer. The output $\hat{y}$ of the softmax layer is an estimated probability distribution over two binary classes, indicating whether the student successfully solved the next exercise. For task **B**, we built a dynamic three layer LSTM, which makes a prediction at every timestep t based on the hidden state at t. Hence, if a student submits three times, we will use the sequence thus far to make three predictions.

5.2 Baselines

Task A: The goal here is to show that our model can learn from the program embeddings alone whether a student is likely to succeed on the subsequent exercise and contrast its performance against the state of the art baseline using handpicked features. For the baseline, we chose the following two features for a student's trajectory T, which have been shown to be highly correlated with learning outcome and performance on the next exercise and trained a logistic regression model.

1. The Poisson path score of the trajectory T as defined in [16]. 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 [16]. $pathScore(T) = \sum_{x \in T} \frac{1}{\lambda_x}$ where λ_x is the number of times AST x appears in student submissions.

2. Indicator feature of student success on current exercise 18. A student succeeded if they ended the trajectory with the solution AST.

Task B: Here, we would like to demonstrate that an LSTM is able to capture more information about a student's trajectory and capture the temporal relationships within the sequence. Hence, we picked logistic regression on program

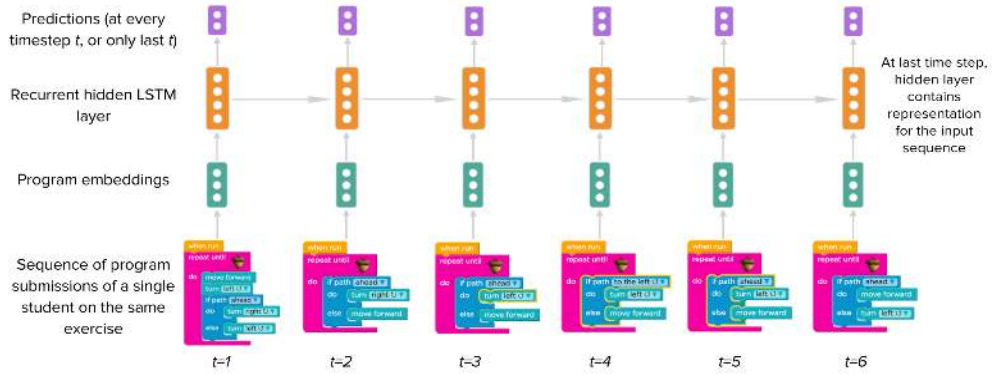

*Figure 2: Simplified Sequential RNN Model. For Experiment **A**, the model only predicts at the last timestep. For Experiment **B**, the model predicts at every timestep. Note that the RNN can be unrolled any number of times, since the parameter weights are shared across timesteps. Note that our models for both tasks stack multiple LSTM layers to increase expressivity.*

embeddings as our baseline. Since logistic regression cannot take in a sequence of embeddings, we consider every embedding within a trajectory sequence as a separate sample that we pass into the logistic regression model. Hence, this model ignores any previous temporal information; e.g. at timestep t, it ignores all embeddings from timestep 1 to $t-1$. Note that this is fairly high baseline, since we feed in program embeddings which are learned using neural networks. We also included a random baseline as a sanity check.

6. RESULTS

6.1 Quantitative Results

Task A: For both the pathscore baseline model and the LSTM model, we used 90% of the data set to perform training and validation and the remaining 10% for testing. The LSTM model consistently outperforms the path score baseline by around 5% on test accuracy at every trajectory length. This result is significant since the input we feed into the LSTM model consists of program embeddings, and not hand-picked features like success on current problem. Our model identified trajectories that show more promise. 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.

Task B: We trained on trajectories of variable lengths 5 to 15, using 90% for training/validation and 10% for testing. At every timestep, we perform a binary prediction. Let's call these two classes "success" and"failure". Since the "failure" class is pedagogically more important, we reported recall, precision and F1 score for the "failure" class at each timestep for our LSTM model as well as for the logistic regression baseline and the random baseline (see Figure 3). We can observe that logistic regression on program embeddings appears to be a very strong baseline. This is potentially due to a high correlation between certain ASTs and the "success" or "failure" classes. Our model does particularly well on recall on the "failure" class, which is pedagogically more important than precision. In education settings, it is much worse to miss students who will fail then giving superfluous support to students who would be successful anyway. It is also worth noting that with increasing number of timesteps, the gaps between the LSTM model and the logistic regression baseline on recall and F1 are increasing. In particular, while recall and precision roughly remain constant after timestep 5 for logistic regression, recall is improving significantly for the LSTM while precision stays roughly constant.

6.2 Analysis of Trajectory Representations

The hidden layer outputs of the trained neural net can be interpreted as the learned feature representations. Input samples that share patterns in the context of the learned task should ideally be mapped close to each other in the representation space.

Visualizing the learned representations of a neural net is an empiric method to explore what the neural net has learned. t-Stochastic Neighbor Embedding (t-SNE) [11] is particularly suited for visualization of high-dimensional data, as it can uncover structures at different scales. Figure 4 shows a t-SNE visualization of student trajectory embeddings for trajectories of length 6. We can observe five distinct clusters, labeled **A** through **E**, which we were also able to identify using the K-Means clustering algorithm with number of centroids set to 5. Each cluster contains trajectories sharing some high-level properties. Some statistics for the clusters are summarized in table 1.

Within these clusters of student trajectories, qualitative analysis found 3 distinct learning groups. **Cluster A** contains the best students who make consistent progress, showing logical debugging steps to apply programming concepts. Each step fixes an existing error and moves towards the correct final solution. A notable differentiator for **Cluster A** students is that they did not return to sections of their solution that they had already corrected. This demonstrates comprehension of the error and that they have digested the concept.

Students in **Clusters C** and **E** make inconsistent progress and show signs of random guessing. Some students methodically test combinations of elements to engineer a passing solution. This behavior likely represents uncertain or distrusted knowledge. This kind of behavior is overlooked by the current grading system as *Code.org* only considers cor-

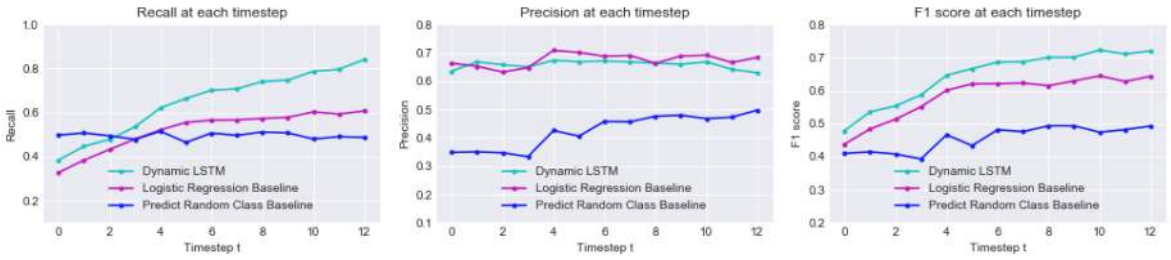

*Figure 3: Recall, Precision and F1 Score at each timestep on task **B**, for the "failure" class.*

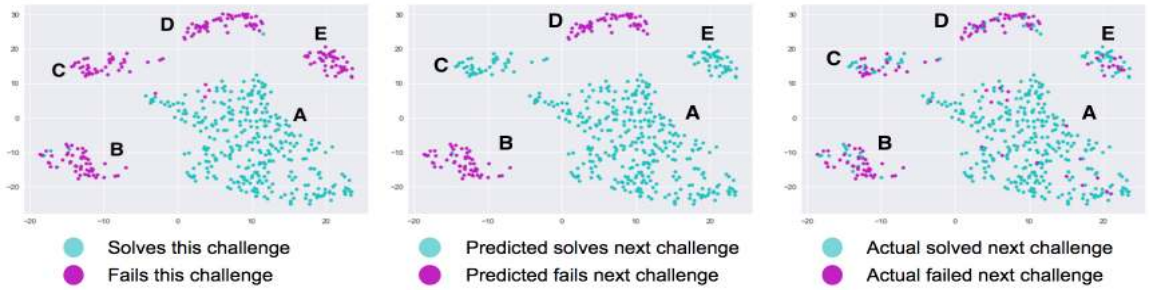

Figure 4: t-SNE visualization of trajectory representations. Left: Ground truth student success for current challenge. Center: Predictions for next challenge given student trajectories on this challenge. Right: Ground Truth student success for next challenge.

rectness of the final submission when scoring, a "number-right scoring" policy. The alternative is a "negative marking" policy, which would penalize students for wrong submissions along with the final answer. Educators have found that number-right scoring is a less reliable grading policy that overestimates student achievement particularly for students with more distrusted knowledge because it obscures whether responses represent true understanding or a lucky guess [2]. Anecdotally, we speculate that the students' success in the next problem may come from being able to reverse-engineer conceptual knowledge through repeated guessing.

Students in **Clusters B** and **D** appear to miss important concepts tested in this exercise. Students in **Cluster D** used an average of 9.21 blocks for every solution (see Table 1), almost twice as many total blocks as other clusters. Rather than solving the challenge with one generalized program, they break the challenge down into segments, hardcoding steps to pass each segment. Students in **Cluster D** have the highest usage of *move forward* blocks and *turn* blocks since students rely on these simple elements rather than the more complex *if-else* and *while* statements, both crucial learning components of this challenge. An ideal solution would include one *if-else* statement and one *while* loop. Students in **Cluster D** used the *if* statement only an average of 0.87 times and the *while* statement 0.60 times. Inspection of their programs show that students in **Cluster B** and **Cluster D** often disregard the *while* statement completely, unlike other clusters where students' solutions consistently contain the *while* loop) even if used incorrectly or inefficiently.

In summary, this analysis shows that our model can create more nuanced representations that lead to better predictions than a model that only looks at binary success indicators. Given that all students in **Clusters B, C, D,** and **E** performed poorly on the current exercise, a binary input model analyzing student success on Exercise 18 could not have distinguished between these poorly performing students. However, our model predicted that students in **Clusters C** and **E**, despite getting an incorrect answer for Exercise 18, would be successful on the next exercise. See Figure 4 *Left* and Figure 4 *Center*. Students in **Cluster C** went from a success rate of 0% in the current problem to a success rate of 48% in the next problem. Students in **Cluster E** went from 15% to 71%. This high success rate for **Clusters C** and **E** is visually noticeable in Figure 4 *Right*. The students' learning trajectories provided our model information to understand the students' learning at a deeper level and make these nuanced predictions, validating the claim that analyzing student trajectories provides richer data for the model.

Table 1: Statistics on student clusters (K-means)

Cluster	**A**	**B**	**C**	**D**	**E**
# students	316	51	44	58	62
Avg # total blocks	4.42	5.10	5.45	9.21	5.41
Avg # *if* statements	0.95	0.95	1.03	0.87	0.97
Avg # *while* blocks	0.89	0.83	0.97	0.60	0.92
Avg # *move forward* blocks	1.38	1.20	2.00	5.81	2.12
Avg # *turn* blocks	1.20	1.32	1.44	1.94	1.40
Success rate on current problem	99.7%	1.7%	0.0%	1.7%	14.5%
Success rate on next problem	95.3%	25.5%	47.7%	17.2%	71.0%

7. CONCLUSION

Our work focuses on multi-step exercises with unbounded solution spaces. While open-ended exercises encourage more flexible problem solving (e.g. in comparison to multiple-choice questions), understanding a student's progress is more challenging due to unbounded variations in student submissions. Given that digital learning platforms can easily archive the temporal dimension of student submissions, we proposed a new approach for learning representations of student knowledge by using program embeddings of student code submissions *over time* instead of hand-picked features. We showed that the trajectories of these representations produce distinct clusters of different student learning behaviors not picked up by a model that only observes binary success outcomes. We also showed that these representations can predict future student performance. We envision creating automated hint systems, where deep knowledge tracing has the potential to identify weaknesses and provide personalized feedback. By being able to anticipate student struggles in particular, we are in essence *capturing pedagogical content knowledge in an unsupervised fashion*. These applications could help improve and personalize the learning experience of students both in the classroom and on online education platforms.

8. ACKNOWLEDGMENTS

The authors would like to thank Code.org for providing the *Hour of Code* data set and Mehran Sahami, Nishith Khandwala and Daniel Guo for constructive feedback and help.

9. REFERENCES

[1] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[2] R. F. Burton. Misinformation, partial knowledge and guessing in true/false tests. *Medical Education*, 36(9):805–811, 2002.

[3] S. Chaiklin. The zone of proximal development in vygotsky's analysis of learning and instruction. *Vygotsky's educational theory in cultural context*, 1:39–64, 2003.

[4] G. L. Cohen, J. Garcia, V. Purdie-Vaughns, N. Apfel, and P. Brzustoski. Recursive processes in self-affirmation: Intervening to close the minority achievement gap. *science*, 324(5925):400–403, 2009.

[5] A. T. Corbett and J. R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

[6] I. Gutiérrez-Rojas, C. Alario-Hoyos, M. Pérez-Sanagustín, D. Leony, and C. Delgado-Kloos. Scaffolding self-learning in moocs. *Proceedings of the Second MOOC European Stakeholders Summit, EMOOCs*, pages 43–49, 2014.

[7] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

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

[9] T. Jenkins. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, volume 4, pages 53–58. Citeseer, 2002.

[10] M. Khajah, R. V. Lindsey, and M. C. Mozer. How deep is knowledge tracing? *arXiv preprint arXiv:1604.02416*, 2016.

[11] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[13] Z. A. Pardos and N. T. Heffernan. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 255–266. Springer, 2010.

[14] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*, pages 505–513, 2015.

[15] C. Piech, J. Huang, A. Nguyen, M. Phulsuksombati, M. Sahami, and L. J. Guibas. Learning program embeddings to propagate feedback on student code. *CoRR abs/1505.05969*, 2015.

[16] C. Piech, M. Sahami, J. Huang, and L. Guibas. Autonomously generating hints by inferring problem solving policies. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*, pages 195–204. ACM, 2015.

[17] C. P. Rosé, R. Carlson, D. Yang, M. Wen, L. Resnick, P. Goldman, and J. Sherer. Social factors that contribute to attrition in moocs. In *Proceedings of the first ACM conference on Learning@ scale conference*, pages 197–198. ACM, 2014.

[18] R. K. Sawyer. *The Cambridge handbook of the learning sciences*. Cambridge University Press, 2005.

[19] A. Sharif and B. Magrill. Discussion forums in moocs. *International Journal of Learning, Teaching and Educational Research*, 12(1), 2015.

[20] S. Shatnawi, M. M. Gaber, and M. Cocea. Automatic content related feedback for moocs based on course domain ontology. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 27–35. Springer, 2014.

[21] L. S. Shulman. Those who understand: Knowledge growth in teaching. *Educational researcher*, 15(2):4–14, 1986.

[22] R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. *ACM SIGPLAN Notices*, 48(6):15–26, 2013.

[23] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

[24] L. Vygotsky. Interaction between learning and development. *Readings on the development of children*, 23(3):34–41, 1978.

[25] K. H. Wilson, Y. Karklin, B. Han, and C. Ekanadham. Back to the basics: Bayesian extensions of irt outperform neural networks for proficiency estimation. *arXiv preprint arXiv:1604.02336*, 2016.

[26] D. Yang, T. Sinha, D. Adamson, and C. P. Rosé. Turn on, tune in, drop out: Anticipating student dropouts in massive open online courses. In *Proceedings of the 2013 NIPS Data-driven education workshop*, volume 11, page 14, 2013.

[27] C. Ye and G. Biswas. Early prediction of student dropout and performance in moocs using higher granularity temporal information. *Journal of Learning Analytics*, 1(3):169–172, 2014.

[28] M. V. Yudelson, K. R. Koedinger, and G. J. Gordon. Individualized bayesian knowledge tracing models. In *International Conference on Artificial Intelligence in Education*, pages 171–180. Springer, 2013.