Applying Sequential Modeling and Deep Learning Techniques to Large Scale Education Data to Infer Efficient and Personalized Learning Paths in Digital Learning Environments

By

Steven Tang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Education

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Zachary A. Pardos, Chair
Professor Mark R. Wilson
Professor Alan E. Hubbard

Fall 2017

Abstract

Applying Sequential Modeling and Deep Learning Techniques to Large Scale Education Data to Infer Efficient and Personalized Learning Paths in Digital Learning Environments

by

Steven Tang

Doctor of Philosophy in Education

University of California, Berkeley

Professor Zachary A. Pardos, Chair

In today's digital world, modern online services often make use of user data to create "personalized experiences" where the service infers what the user may want to see or do next based on what similar users have done in the past. In the education sphere, many students are turning to online and computer based learning environments as part of their ongoing education, thus generating vast logs of interesting student data, yet there does not exist much work in personalizing online learning experiences in a data driven fashion. This dissertation presents two methodologies that learn from large sets of student-created data to infer potentially efficient ways to utilize learning content on each respective digital learning platform.

The first methodology presented in this dissertation is somewhat tailored to the ASSISTments math-tutoring framework, but could be potentially applied to other data sets. This methodology seeks to uncover pedagogically advantageous orderings of items related to learning. The motivation for this methodology stems from the format of the ASSISTments platform: a random order of related questions is given to students until the students have "mastered" the knowledge component that underlies each question. It could be the case, however, that students might be able to master a knowledge component more efficiently if there exists an ordering of items that enables learning faster than an at-random order. To investigate this hypothesis, an item order extension is proposed to the Bayesian Knowledge Tracing (BKT) framework. The BKT framework is well suited to model student knowledge over a sequence of learning opportunities organized around a single knowledge component with the assumption that tutoring or learning occurs after each presented assessment opportunity. Using the publicly released ASSISTments 2012-2013 data set consisting of approximately 10 million student responses to math questions, the item order extension to the BKT framework is applied and compared to a baseline BKT result. Cross validation techniques are used to assess model fit, and a qualitative investigation is performed along with regression analyses to uncover in what circumstance a pedagogically advantageous item order might exist in the ASSISTments framework. An experiment is then conducted using the item order model to see if predicted learning outcomes are met in practice.

The second methodology presented enables learners in Massive Open Online Course (MOOC) contexts to be given data driven personalized recommendation related to what the learner might want to do next, based on what similar users have done in the past. In the world of MOOCs, billions of specific and granular actions taken by students have been recorded and logged. The types of actions that are logged vary in type: some actions are related to answering quiz questions, others are related to playing or pausing videos, while others are related to navigating to different course pages. This dissertation proposes a methodology whereby a model of student behavior is learned by training on how students have behaved in the past and proposes that insights gleaned from those behavior patterns can be utilized as part of a “personalized recommendation” framework that future MOOC learners can use, which will hopefully enhance the learning experience for those students. In this methodology, a general deep learning architecture utilizing recurrent neural networks is proposed, as such a methodology is well suited to model arbitrarily long sequences of data that may have complex relationships spanning multiple timepoints. Simpler N-gram (frequentist) baselines and course structure or “syllabus” baselines are used to assess RNN performance. A working demonstration of incorporating the personalized recommendation framework into a live MOOC using the edX framework is also conducted, showing that such a methodology can be reasonably implemented in practice. This behavior modeling methodology is applied to a wide variety of MOOC datasets in order to determine to what extent such a methodology works across a variety of different MOOCs. The size of the analyzed datasets spans tens of millions of navigational actions across tens of thousands of students.

Table of contents

List of figures

List of tables

Acknowledgments

My journey through graduate school would not have been possible without the help and support of my colleagues, friends, and family. First, I would like to thank my advisors Zachary Pardos, Mark Wilson, and Sophia Rabe-Hesketh. My journey through the QME program led me to a variety of different research paths, and all of you allowed me the right balance of freedom and guidance to pursue my own research path. All of my advisors have given me a substantial amount of perspective across each of their respective fields. In particular, I would like to thank Zachary for pushing me to complete my projects and helping me to understand how to frame research in the broader context of the education research community. I am also thankful for Zachary's ability to respond to e-mail at seemingly any time of day or night, sometimes at the last minute of deadlines. I am also thankful to Mark and Sophia for their concrete and actionable feedback throughout my graduate school career; their feedback has been clear and direct and has helped me move forward throughout the years. I am also thankful to Alan Hubbard for being on my committee and his helpful comments during the dissertation proposal process.

I also want to thank my colleagues and co-authors Hannah Gogel, Beth McBride, Joshua Peterson, Daniel Davis, and Christopher Le. I have grown intellectually by working with each of you, and have been shaped by everyone's unique perspectives. I also want to thank my fellow QME students who have been a part of my journey in some way. In particular, my immediate cohort of Rebecca Freund, SeungYeon Lee, Seth Corrigan, and JinHo Kim have all helped me in some way, perhaps through class project ideas or through general support. I am also grateful to the rest of the QME community; the QME community at UC Berkeley is truly friendly and supportive. David Torres Irribarra, Perman Gochyyev, and many of the other upper year QME students have been incredibly generous in spending time to explain the ins and outs of various measurement models or about navigating the QME program in general.

I also want to thank my close friends Charissa Tansomboon and Daniel Pineda. They have both provided a tremendous amount of support throughout my graduate career and have helped me to become a more well-rounded person.

Finally, I would not be where I am today without the support and encouragement from my parents Huixing and Jenny Tang and my brother Jason. I am incredibly lucky and grateful to have such a great support system from my family.

1 Introduction

1.1 Overview

In today’s digital age, the way people learn and gain new knowledge has shifted towards utilizing online and digital services, often in addition to or in place of traditional in-person teaching paradigms. Digital devices and media permeate learning processes in every learning environment, from K-12 to higher education to “life-long” learning processes. Massive Open Online Courses (MOOCs) are one form of online learning, where professors from renowned universities release university-level course content for anyone around the world to access, complete with lectures, problem sets, and discussion forums. These MOOC resources have impacted millions of learners around the world. MOOC-like resources have existed in their current form for about 6 years already, and while these have been shown to be useful and well utilized resources for many learners, research still suggests that there are many avenues for improvement and innovation in the MOOC space. Personalization techniques have impacted the millions of users across the web’s biggest websites for years already, from personalizing shopping experiences in Amazon [1] to optimizing advertising and social experiences in Facebook [2]. Personalization is often the path forward when a service has substantial user data, and personalization in the education sphere is a major theme underlying the methodologies discussed in this dissertation.

In this thesis, I perform an investigation into uncovering whether the order of test questions delivered by a digital learning environment can have an effect or impact on student learning in an effort to improve the efficiency of these types of learning environments at scale. This investigation presents a modification to the Bayesian Knowledge Tracing framework. Shifting to the MOOC sphere, I also investigate the granular click-stream logs generated by users and learners accessing MOOCs in recent years in an attempt to better understand the patterns of behaviors that MOOC learners partake in. I also investigate the potential for personalized navigational recommendation in MOOC contexts, while also showing that such work is possible now by implementing a recommender into a live MOOC administered by edX. This recommender uses a collaborative-filtering inspired deep learning model to infer which content the learner might be interested in next. I apply these methodologies to a broad survey of 99 edX MOOCs administered in 2015 and 2016 to see how well this methodology performs across many different kinds of MOOCs.

All of the methods presented and investigated in this thesis are unified by their goal of processing large amounts of student generated data in order to optimize the learning experience of the end user, the student. Thus, these methods are meant to work at scale, where large amounts of input data can be expected. These methods are particularly relevant in today’s educational environment, where more and more of student behaviors and student results are logged across all of the platforms that students use.

Some of the methods in this dissertation are focused on trying to provide recommendations for behavior in learning contexts. With any type of recommendation, there comes the risk that the recommendation is not actually good or beneficial for the learner. This caution is directly relevant to the collaborative-filtering based method of personalized recommendation I propose and investigate in chapters 4 and 5. It is certainly not the case that these recommendation systems are designed to produce the “absolute best” navigational path through the MOOC. Rather, these systems are intended to provide some support to the end user (the learner) based on what past learners have done. With this type of methodology comes the risk for perpetuating biases or behaviors that we may not necessarily want to perpetuate. With these risks in mind, this thesis presents an investigation into collaborative-filtering based recommendation methodologies, with the hope that these methodologies, while not perfect, can provide useful support and recommendation to learners at scale.

1.2 Contributions and outline of thesis

Because online learning *at scale* is a relatively recent phenomenon, there continues to be large opportunities for innovation in improving the delivery of learning content. In this thesis, I use deep learning approaches towards modeling the sequential nature of student actions in MOOC contexts. Essentially, I treat the sequence of clicks and actions learners take as an ordered sequence of events, and utilize a deep learning approach towards predicting what the student will attempt to do next based on their sequence of actions thus far. This method utilizes past behaviors of students in an effort to predict the behaviors of other similar students. This approach opens the door for a personalized recommendation framework that can enhance common use cases of MOOCs, which is a contribution towards personalized learning at scale. Understanding learner behavior in MOOC contexts is an ongoing research task, and this thesis contributes to that broader understanding. This thesis also presents an enhancement to the Bayesian Knowledge Tracing (BKT) framework, whereby responses to test questions are treated as an ordered sequence of events. The BKT framework models student knowledge at each timestep in the sequence, and I contribute to the BKT research framework by proposing that the order of items can have an impact on student learning. If such an enhancement is shown to be successful, then learning environments that use BKT as an underlying knowledge model can be made more efficient by automatically including information from item orders.

This dissertation contains 1 introductory chapter (this chapter), and then 4 chapters of methodological research and study. A final conclusion chapter summarizes some of the main findings from each chapter.

Summary Chapter 2: Modifying Bayesian Knowledge Tracing to Investigate the Effect of Item Order

This chapter focuses on an application of the Bayesian Knowledge Tracing framework applied to data collected from a digital assessment and tutoring platform known as ASSISTments. The data analyzed in this chapter still consists of sequential data generated by students at scale; however, the sequential data consists primarily of correct and incorrect responses to quiz questions. The BKT framework is used to determine when mastery of a knowledge component has occurred. The BKT framework assumes that each assessment opportunity also carries with it a tutoring component; thus, students are expected to gain knowledge after each quiz attempt. In this chapter, I investigated a modification to the BKT framework, whereby the modification attempts to determine the effect that the ordering of items in a test session has on the rate of learning; the rate of learning is one of the parameters of the BKT framework. In the ASSISTments framework, students are given randomized sets of quiz questions (that also contain tutoring components) that all measure a unified construct, and are simply assumed to all produce equal gains in learning. Because there exists a large set of existing responses to these quiz questions in a publicly released data dump from ASSISTments, it is hypothesized that efficient orderings of items can be discovered that may be more efficient at enabling students to master a construct than the at random order of item/tutoring delivery. This work [3] [4] was presented at Learning at Scale and at Educational Data Mining. Finally, an experimental study is also conducted, where students are split into two test groups and a control group to see if the item-order model can find effective orderings of content.

Summary Chapter 3: Predictive Modeling of Student Behavior using Granular Large Scale Action Data from a MOOC

In this chapter, I shift away from the cognitive domain from chapter 2 and move towards behavior modeling in the MOOC sphere, attempting to look at the potential for sequential modeling techniques to predict which granular action a student will take next in a MOOC context. In this chapter, I parse the click-stream log of an edX MOOC. Each student's sequence of actions is therefore converted into a parsed sequence of actions, and these parsed sequences can serve as input to a sequential model. The sequential model is then trained to predict which action the student is likely to take next. Two sequential models are investigated: an n-gram approach and a recurrent neural network based approach. Recurrent neural networks have shown great success in finding patterns in granular and complex data, such as in the case of language modeling, which tries to predict what kinds of language might follow a sequence of words or phrases. To my knowledge, such a sequential modeling approach has never been applied to behavioral data in MOOC contexts before. The chapter includes the results from training different models with differing sets of hyperparameters, and commentary about patterns and issues during training are discussed. The modeling techniques and results established in this chapter set the basis for modeling techniques explored in chapters 4 and 5. The work in this chapter has been published as part of a journal on learning analytics in educational contexts [5].

Summary Chapter 4: Enabling Real-Time Adaptivity in MOOCs with a Personalized Next-Step Recommendation Framework

In this chapter, a full-fledged recommendation framework is described and implemented into a live edX MOOC. The sequential modeling technique of student behavior in MOOC contexts first presented in chapter 3 is extended and modified to include a time-spent (dwell time) component. Thus, the sequential deep learning model is modified to be capable of predicting which resource a learner will visit next in addition to how much time that learner might spend on that resource. A working modification to the edX MOOC platform is presented in detail, and students taking the MOOC received live recommendations from the established recommendation framework. The details of coordinating the different components of the live recommendation framework are discussed as well. The implementation shown in this chapter shows that such a recommendation framework can be applied in practice, and that the underlying personalized recommendation model can be queried and utilized in a live website setting; students receive the actual recommendation in less than a second. A discussion on design considerations for real-time deployment is also included. The work from this chapter was presented as a full paper in a Learning at Scale conference [6].

Summary Chapter 5: Surveying Time Sensitive Behavior Modeling Across 99 MOOCs

In this chapter, the Time-augmented LSTM behavior model presented in Chapter 4 is applied across a wider survey of 99 edX MOOCs administered in 2015 and 2016. The investigation in this chapter seeks to discover how often the Time-augmented LSTM behavior modeling approach presented in previous chapters can be applied in practice to different kinds of MOOCs. The MOOCs investigated in the previous chapters were hand-picked because their click-stream data showed promise in the potential to generalize non-linear patterns of behavior. The MOOCs in this investigation, however, are not hand-picked in any way; the click-stream logs for these MOOCs were provided by edX in what was essentially large data dump. Through the investigation in this chapter, it is found that the majority of courses can benefit from the time-augmented LSTM behavior modeling approach, although a few of the MOOCs do not seem to benefit from this modeling approach. Discussion about the role of MOOCs for learners at large is also discussed, given that this investigation provides substantial evidence that learners skip large portions of the courses. Prior research from 2013 has established that students employ non-linear navigation patterns (such as deviating from the established syllabus order of the course), and the results from this chapter indicate that those behaviors still persist, and have even gotten worse. The results from this chapter suggest that there exists strong potential for personalized recommenders to persist in the MOOC world, given existing patterns of student behavior.

2 Modifying Bayesian Knowledge Tracing to Investigate the Effect of Item Order

2.1 Chapter Abstract

Online computer adaptive learning is increasingly being used in classrooms as a way to provide guided learning for students. Such tutors have the potential to provide tailored feedback based on specific student needs and misunderstandings. Bayesian knowledge tracing (BKT) is used to model student knowledge when knowledge is assumed to be changing throughout a single assessment period; in contrast, traditional Item Response Theory (IRT) models assume student knowledge to be constant within an assessment period. The basic BKT model assumes that the chance a student transitions from ``not knowing" to ``knowing" after each item is the same, and problems are considered learning opportunities. It could be the case, however, that learning is actually context sensitive, where students' learning might be improved when the items and their associated tutoring content are delivered to the student in a particular order. In this chapter, we use BKT models to find such context sensitive transition probabilities from real data delivered by an online tutoring system, ASSISTments. After empirically deriving orderings that lead to better learning, we qualitatively analyze the items and their tutoring content to uncover any mechanisms that might explain why such orderings are modeled to have higher learning potential. Additionally, we use BKT models to find such context sensitive transition probabilities in a mathematics tutoring system and offer a methodology to test the significance of our model based findings. We employ cross validation techniques to find models where including item ordering context improves predictive capability compared to the base BKT models. We then use regression testing to try to find features that may predict the effectiveness of an item ordering. Finally, an experiment is conducted, where students are given different tutoring conditions based on item orderings and then assessed on a single post test. The work presented in this chapter [3] [4] was developed with my co-authors Hannah Gogel and Elizabeth McBride.

2.2 Introduction

Online computer adaptive learning is increasingly being used in K-12 as a supplement to classroom instruction. Online systems, such as various MOOCs and the ASSISTments [7] platform, provide scaffolding and hints to students upon request or when the student answers a question incorrectly. This opportunity for learning after each question provides immediate formative feedback to the students, similar to how a human tutor might provide feedback during a lesson.

2.3 ASSISTments Data

The data set analyzed in this chapter comes from use of the ASSISTments platform in AY 2012-2013. The data set is publicly available and is rich with information that has been mined by other research projects [8] [9]. In this chapter, we focus on the *Skill Builder* sequences used in ASSISTments, where a problem set consists of items given in a random order, generated from a set of templates. Items generated from these templates are assumed to be answerable with knowledge of a single underlying knowledge component (KC). For example, one problem set might contain three item templates. Each template can be populated with a set of numbers to generate an item; thus, many different items can be derived from a single template. The number of templates per problem set varies; in this chapter, we look at problem sets with between 2 and 6 templates. The number of items delivered to the student depends on the student's performance; in the Skill Builder set, mastery is assumed to occur after three consecutive correct responses. After mastery of a KC is achieved, the student is finished with the specific Skill Builder and no more items are administered.

Each template in a Skill Builder sequence has an associated method of assistance; it is either a *hint* template or a *scaffolding* template. Scaffolding templates are bundled with a set of simpler questions to guide the student through the ideas in the item, while hint templates have guiding statements available to assist the students (usually the final hint provides the exact answer to the item). A student can directly ask for assistance, or the system will automatically provide assistance when the student answers a question incorrectly. For the sake of the BKT model, if a student asks for assistance, the response is modeled as incorrect. The BKT framework models this learning after each problem attempt through the transition parameter.

2.4 Bayesian Knowledge Tracing

Bayesian knowledge tracing [10] assumes a binary representation of student knowledge. There are noise parameters in the model that affect the relationship between knowledge and performance. For example, the model assumes that even when a student knows a KC, they might answer a question incorrectly (modeled as a *slip* parameter), and that when a student does not know a KC, they might guess and answer a question correctly (modeled as a *guess* parameter). Figure 1 depicts a BKT model representation as a hidden Markov model (HMM). In between each $\mathbf{K_i}$ and $\mathbf{K_{i+1}}$, there is an arrow representing a probability of *transition*, or learning. $\mathbf{O_1}$, $\mathbf{O_2}$, and $\mathbf{O_3}$ are binary indicators of correctness at opportunities 1, 2, and 3. $\mathbf{K_1}$, $\mathbf{K_2}$, and $\mathbf{K_3}$ represent the latent knowledge of the KC (assumed to be 0 or 1) at opportunities 1, 2, and 3.

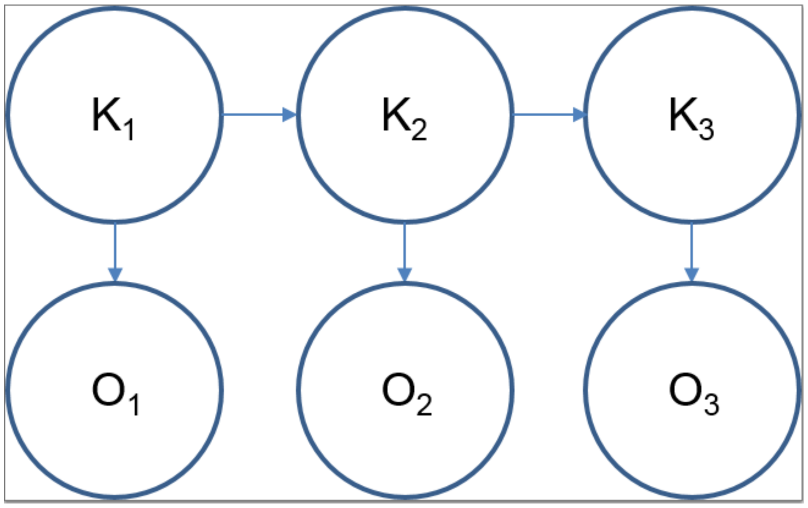

$$\begin{tikzcd}K_1 \arrow[r] \arrow[d] & K_2 \arrow[r] \arrow[d] & K_3 \arrow[d] \\ O_1 & O_2 & O_3\end{tikzcd}$$

Figure 1 Basic BKT Model

2.5 Item Ordering Effects

It is our hypothesis that there may exist pedagogically more advantageous orderings of problems than the default random orders. The BKT model can be extended to model a transition probability per particular item ordering. For example, the BKT model can estimate a transition probability after a student has specifically seen templates in the order (3, 1) and (1, 3). If one order has a higher transition probability than its reverse order, this could be seen as evidence that one ordering promotes more effective learning than another. Figure 2 depicts how this new model, drawn from work by Pardos and Heffernan [9], might be formulated as an HMM. Each item template is given their own set of guess and slip parameters. This student sees items from templates in the order of (3, 1, 2). **Q_3** and **Q_1** represent items from the 3rd and 1st template respectively, and are boxed together to show that this particular order affects the probability of knowledge at **K_3**. Then, **Q_1** and **Q_2** are boxed together to show that **K_4** would be affected by seeing items in the order (1, 2).

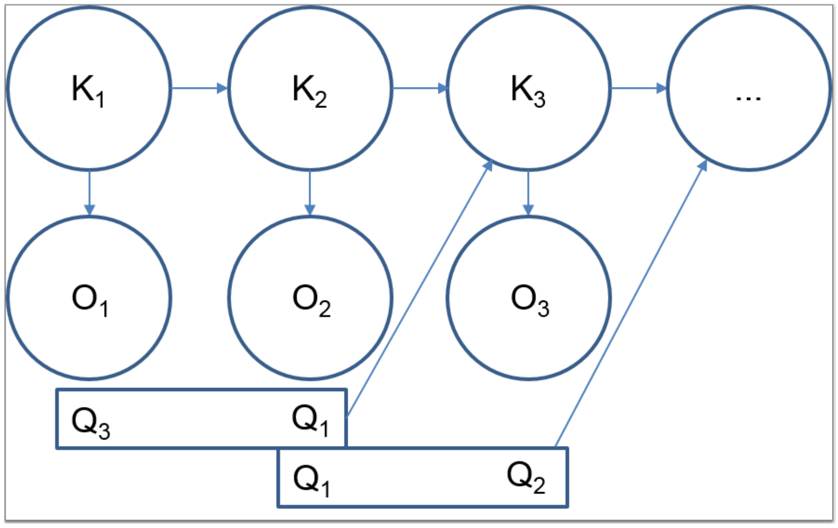


Figure 2 Item Order BKT Model

2.6 Procedure

Only student data within 24 hours of the student's first response in a SB was included. SBs with more than 2000 student responses, at least 16 unique students, and between 2 and 6 (inclusive) template types were used for this analysis resulting in 112 SBs.

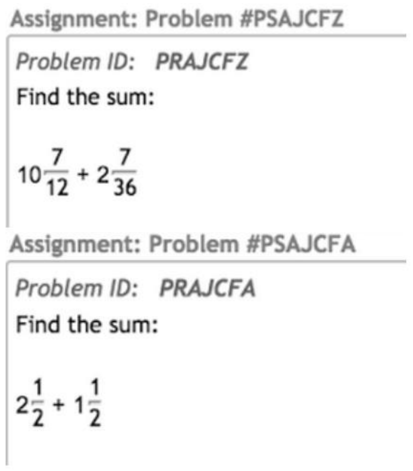
Two BKT models were run on each SB using the edX-hmm codebase. The first baseline model used standard BKT, where every item was assumed to have the same transition probability and the second model assumed a different transition probability based on the order of the previous two items administered.

2.7 Qualitative Analysis

We looked at the 48 problem pair orderings where the difference between one order and its reverse was between 0.3 and 0.7 (avoiding the uninteresting and degenerate cases). We examined each of the templates with the associated template hints and qualitatively coded for certain phenomena that included: the number of hints available, if there was context for the problem (e.g. a story problem), whether the answer to the problem was given in any of the hints, whether there were visualizations or images present in the problem or its hints, whether or not one template in the pair was more difficult than the other, and which item in the pair was more difficult (first or second). Due to the large range of grade levels and mathematical topics covered in the templates we analyzed, any qualitative coding by topic left us with 5 or fewer items per group. This was not enough data to make any conclusions based on the item order and the subject matter. Cohen's kappa between the two unique raters was found to be above .80 on all coded values, ensuring inter-rater agreement.

The sample size for this analysis was quite small, so significance testing on many of these criteria were inconclusive. However, 33 out of the 48 template orderings were found by both raters to have significant differences in difficulty between the two templates in the pair. Of these 33 more effective orderings, the first template was more difficult in 17 orderings and the second template was more difficult in 16 orderings. Theoretically, assigning a guess and slip for each problem template, as was done with this BKT item ordering extension, should control for problem difficulty. That is, if a problem is particularly easy, it should receive a high guess value from the fitting procedure. Thus, in order for the model to assume learning has occurred when a hard question is followed by an easy one, the expected performance on the easy question must outperform the expected independent outcome for the problem.

An example of an effective item ordering is shown in Figure 3. Although the exact numbers within a template can change, the templates can have certain consistencies. In this example, the second ordered template shown always has equal denominators across its addends making it a lower difficulty item than the first ordered template.


Assignment: Problem #PSAJCFZ

Problem ID: PRAJCFZ

Find the sum:

$10\frac{7}{12} + 2\frac{7}{36}$

Assignment: Problem #PSAJCFA

Problem ID: PRAJCFA

Find the sum:

$2\frac{1}{2} + 1\frac{1}{2}$

Figure 3 Example of Effective Item Ordering

One potential hypothesis that can help explain these findings is that of “desirable difficulties”. In a series of studies, Bjork and colleagues determined that some challenges to performance during learning activities may actually contribute to greater learning [11]

[12]. By introducing “desirable difficulties” that help learners engage in more active processing of information, learning tasks that may be perceived as challenging or inefficient may prove more beneficial than those completed with high fluency. In the case of item orderings where the first problem is more difficult than the second, the first (more difficult) problem may introduce a desirable difficulty, leading the student to learn more than they would with an easier problem. This learning then carries over into the second problem in the pair, thus leading to a higher overall rate of learning than found in the reverse order. When the first problem is easier than the second, this might be an instance where the material is better learned through a gentler or simpler introduction, as perhaps the second problem might be too difficult to be “desirable”.

Testing whether an item is of a ``desirable" difficulty would likely require a careful experimental design, rather than a post-hoc analysis of item properties when items are administered at random. Thus, numerous future research opportunities abound to further investigate how to best order items; in particular, it could be pertinent to explicitly engineer items with particular difficulty differences and test which orderings lead to the quickest learning of a KC.

2.8 Quantitative Approach

The previous section detailed a qualitative approach towards finding item ordering effects. This section introduces a quantitative approach instead. Among the Skill Builder response sets (SBs) from the 2012-2013 ASSISTments data set, we only looked at sets with more than 2000 student responses, more than 250 students, and between 2 and 6 (inclusive) templates. There were 112 Skill Builders that met these criteria. Some students had data that spanned several months, with long breaks in between responses. To remove confounding time effects, only student data produced within 24 hours of the student's first response in a Skill Builder was included.

Two BKT models, estimated using the XBKT code base, were fit to each of the 112 SBs. The first model was standard BKT (baseline), where every item was assumed to have the same transition probability. In our standard BKT model, every template type was allowed to have its own guess and slip parameters. The second model allowed for both different guess and slips per template and different transition probabilities based on the previous two items administered. We enabled different guess and slips per template for our baseline model so that any difference between models would be attributed to the different item order learning transitions. Additionally, we modeled a transition probability for each template specifically when that template was the first item administered in the sequence.

For all 112 SBs, the log likelihood fit of the model improved when transition probabilities were estimated for each item ordering. This is expected since the item order model allows for a greater number of parameters than the base BKT model.

Table 1 shows the percentage of item order models that had a lower Akaike information criterion (AIC) than the corresponding base BKT model. The AIC statistic rewards goodness of fit but includes a penalty for the number of estimated parameters, to reduce the impact of over-fitting. A model that produces a lower AIC value is preferred.

Table 1 AIC Values

Templates	Proportion of SBs that had lower AIC with item order model
2	.91
3	.84
4	.5
5	.14
6	.1

The purpose of this chapter is not to contend that the item order model is *always* preferable to the base BKT model. Instead, this chapter investigates whether the item order model can be used to find a subset of item orders that have an impact on student learning. When viewed in this context, it can be expected that many item order models will not have the preferable AIC. The number of potential orderings (and therefore parameters) increases dramatically when adding additional templates, but without a proportionate increase in data points; AIC penalizes this dramatic increase in the number of parameters, despite the expectation that only a subset of the parameters will warrant further scrutiny. Thus, as expected, increasing the number of templates leads to a decreased proportion of SBs with a lower AIC from the item order model.

2.9 Cross-validation prediction to identify item orders of interest

To obtain statistical confidence in the generalization of a certain item ordering to unobserved students, we performed 5-fold cross validation (CV) on the data. This process starts by fitting both base and item order BKT models on a randomly selected 80% of student response data, and then using the trained models to predict student responses in the held out 20%, called the test set. Then, error rates can be computed by using the predicted student response and the actual student response in the test set for both models. This process is performed 5 times such that the responses from each student will be in a test set exactly once. Performing this cross-validation step allows us to obtain some statistical confidence that the item order model can successfully predict student responses on data that the model wasn't trained on.

By comparing the predicted responses to the actual responses, Mean Absolute Errors (MAE) were obtained for both the base and the item order models. The error rates were then compared using a paired t-test for each possible item order. Out of the 1789 possible item orders among all Skill Builder problem sets, 605 item orders were found to have statistically significant error differences between the two predictive models at the .05 level. Among the 605 item orders, 157 had their responses predicted better by the base BKT model, while the remaining 448 item orders had their responses predicted better when using the item order model. It is important to note that the item orders in this section include ordering situations where the same template is administered twice in a row.

The result that a large portion of the item orders had better response prediction when using the base BKT model is not surprising, considering that each addition of a single new template to an SB increases the number of potential item orders dramatically. Thus, as the number of templates increases, the number of responses per item order decreases, resulting in less data per parameter for the model to learn from.

The occurrence of 448 item orders whose responses were better predicted by the item order model suggests that the item order model could be able to uncover effective (or ineffective) item orderings.

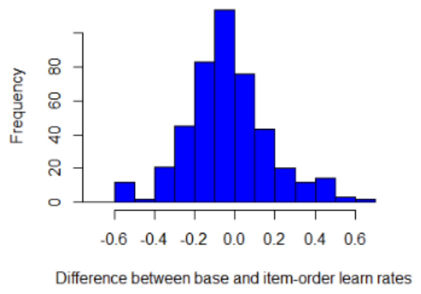


Figure 4 Difference between base and item-order learn rates

Figure 4 shows the distribution of differences between the 448 item order pair learn rates and their corresponding base model's learn rate. A difference above 0 means that the item order pair learn rate was higher than that of the standard BKT learn rate for that SB. This indicates it was an above average learning pair. The inverse is the case for differences below 0.

Figure 5 shows the distributions of base model learn rates among the SBs that were found to have significantly better predictions for at least one item order. Note that even if a SB has multiple significant item orders, the corresponding base learn rate for that SB only appears once in the histogram. Figure 6 shows the distribution of learn rates using the item order model. One key difference between the two distributions is that a higher proportion of learn rates are between 0.0 and 0.1 for the item order model.

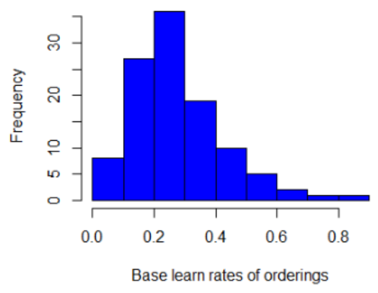


Figure 5 Distribution of base learn rates

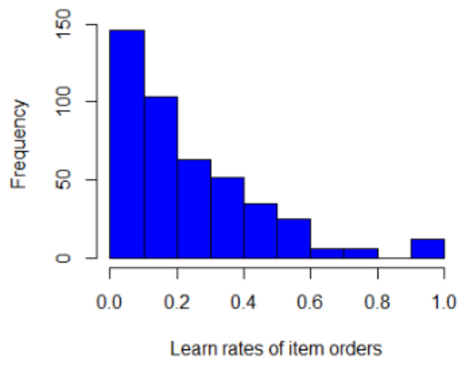


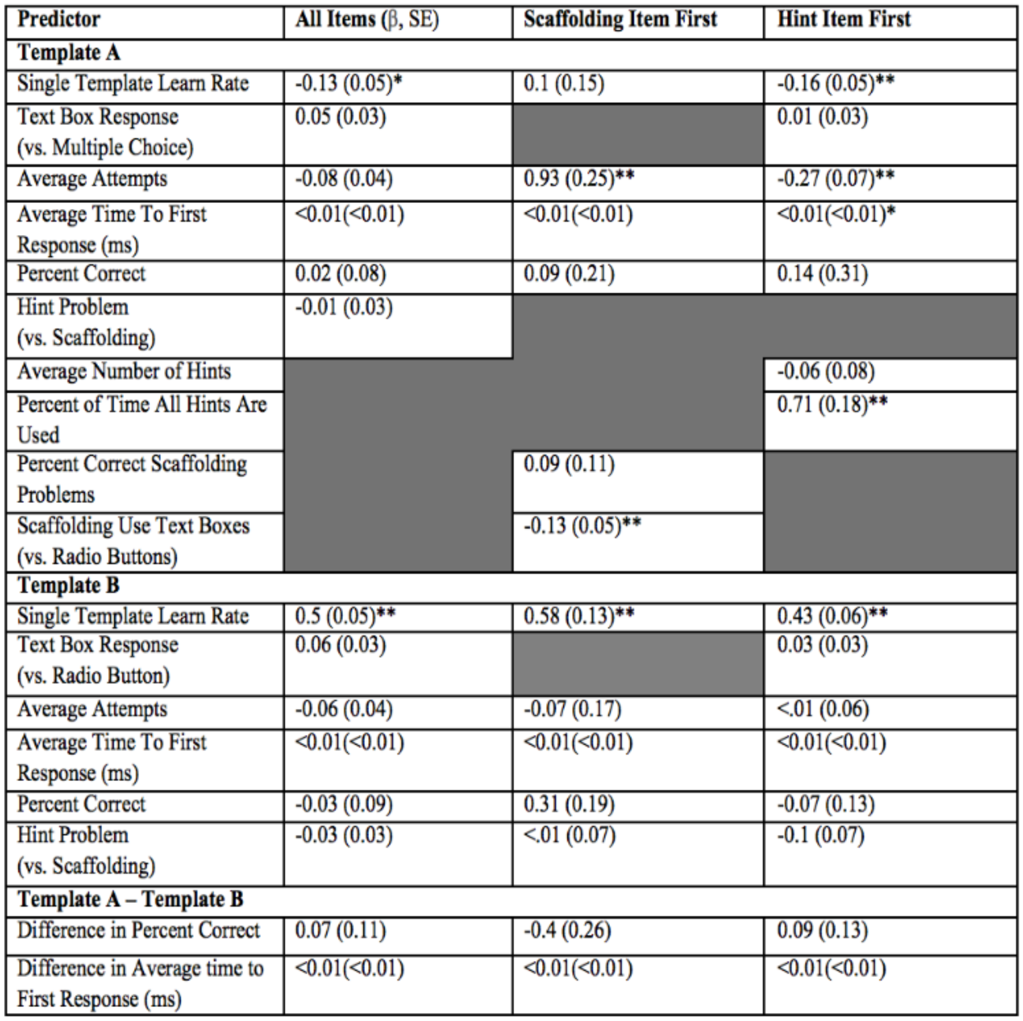
Figure 6 Distribution of item order learn rates

The combination of the item order model with the cross-validation approach provides a procedure that can determine when the item order model provides more accurate predictions compared to the base BKT model. Such a procedure can reveal when an item ordering might be considered effective or ineffective. These initial findings suggest promise in continued exploration of item ordering models and procedures.

2.10 Regression Analysis

The previous section detailed an approach for obtaining statistical confidence that the item order model can improve prediction of student responses over a base BKT model. This section details an attempt to find features of the templates and responses that might be able to predict the learn rate for a specific item order. The results of all regression analyses are shown in Table 2. Regression analyses were run on the 448 item orders found to be significantly better fitting from the cross-validation approach.

*Table 2 Regression predicting item order learn rate, * p < .05; ** p < .01*



Predictor	All Items (β, SE)	Scaffolding Item First	Hint Item First
Template A			
Single Template Learn Rate	-0.13 (0.05)*	0.1 (0.15)	-0.16 (0.05)**
Text Box Response (vs. Multiple Choice)	0.05 (0.03)	▨	0.01 (0.03)
Average Attempts	-0.08 (0.04)	0.93 (0.25)**	-0.27 (0.07)**
Average Time To First Response (ms)	<0.01(<0.01)	<0.01(<0.01)	<0.01(<0.01)*
Percent Correct	0.02 (0.08)	0.09 (0.21)	0.14 (0.31)
Hint Problem (vs. Scaffolding)	-0.01 (0.03)	▨	▨
Average Number of Hints	▨	▨	-0.06 (0.08)
Percent of Time All Hints Are Used	▨	▨	0.71 (0.18)**
Percent Correct Scaffolding Problems	▨	0.09 (0.11)	▨
Scaffolding Use Text Boxes (vs. Radio Buttons)	▨	-0.13 (0.05)**	▨
Template B			
Single Template Learn Rate	0.5 (0.05)**	0.58 (0.13)**	0.43 (0.06)**
Text Box Response (vs. Radio Button)	0.06 (0.03)	▨	0.03 (0.03)
Average Attempts	-0.06 (0.04)	-0.07 (0.17)	<.01 (0.06)
Average Time To First Response (ms)	<0.01(<0.01)	<0.01(<0.01)	<0.01(<0.01)
Percent Correct	-0.03 (0.09)	0.31 (0.19)	-0.07 (0.13)
Hint Problem (vs. Scaffolding)	-0.03 (0.03)	<.01 (0.07)	-0.1 (0.07)
Template A – Template B			
Difference in Percent Correct	0.07 (0.11)	-0.4 (0.26)	0.09 (0.13)
Difference in Average time to First Response (ms)	<0.01(<0.01)	<0.01(<0.01)	<0.01(<0.01)

For the regression analyses, we extracted template level features from both templates in an item order. For the rest of our analyses, Templates A and B will refer to the first and second template in an item order. Features included are: average time to first response (milliseconds), percent correct on first attempt, average number of attempts, and problem type.

Average number of attempts includes the number of times a student answers the original problem. A student can attempt a problem until they arrive at the correct response; however, as previously mentioned, only the first response will count for correctness for mastery sake. For problem type, items were either text response or used radio buttons. The use of radio buttons typically implies a multiple choice question. Other features included: difference in time to first response between Template A and Template B, the difference in percent correct between Template A and Template B, and the *individual* learn rates for templates A and B. Note that these learn rates for individual templates are calculated only from responses when that template is the first item in a response sequence. In our first model, denoted as the All Items model, stepwise regression was used with these features on all 448 item orders (R^2=.17, F = 46.19, p<.01).

For our first model, the only features that were found to be significant at the .05 level were the learn rates for each template. The learn rate of Template A had a negative effect on item order learn rate for the pair (β = -0.13, $p = .01$), and the learn rate of Template B had a positive effect (β = .502, $p < .01$) in the model. For every increase in .1 units in the learn rate for just Template A, the item order learn rate is predicted to decrease by .013. For every increase in .1 units in the learn rate for just Template B, the item order learn rate is predicted to increase by .05.

In the ASSISTments platform, templates either include *scaffolding* or *hints* (but not both) as the tutoring mechanism. When a student answers a question incorrectly or clicks the help button, the system will provide that template's form of assistance. Scaffolding refers to a sequence of sub-questions that the student has to answer, while the sequence of hints are simply pieces of text that the student reads to learn from. The last available hint in a hint sequence displays the exact answer to the item.

In our second model, denoted the Scaffolding Item First model (R^2=.37, F = 11.47, $p < .01$), we only examined item orderings where Template A was a problem that contained scaffolding as its assistance mechanism. In this model, all of the features from the All Items model were included, with the addition of: percentage of scaffolding problems answered correctly (which includes the sub-questions of the scaffolding), and the type of scaffolding problem, either text box or radio button response types. The scaffolding for a question consisted of problems that used either all text boxes or all radio buttons; a scaffolding session contained only one type of response. Thus, the original question could be a text box response item, but the scaffolding for that question used either all radio button items or all text box response items. All original templates that used scaffolding as the tutoring mechanism were text box response items (even though the scaffolding could use radio buttons or text box); thus, the problem type feature was omitted in this model. Template B was typically also always a text box response following a scaffolding template, so problem type was omitted for template B as well. Additionally, whether Template A was a scaffolding or a hint problem was no longer used as a predictor since only scaffolding problems were included.

Average attempts on Template A (β = .93, $p < .01$) and the learn rate for Template B (β = .58, $p < .01$) had a positive effect on the item order learn rate. When the scaffolding for Template A consisted of text responses, the learn rate of the ordering decreased (β = -.13, $p < .01$).

The third model, called the Hint Item First model, was fit using only orders where Template A was a hint item (R^2=.22, F = 20.94, $p < .01$). Hint features included the percentage of students who went through all the hints on Template A and the average amount of hints seen for the template. Whether the first template was a hint or scaffolding problem was not included as a potential predictor since only pairings where Template A was a hint problem were included in this analysis. Significant predictors of the learn rate for the item ordering in this model were: average number of attempts on Template A (β = .27, $p < .01$), average milliseconds to first response on Template A (β = < .01, $p = .03$), the percentage of students who accessed all of the hints on Template A (β = .71, $p < .01$), learn rate of Template A (β = -0.16, $p <.01$), and the learn rate for Template B (β = .43, $p < .01$).

Additional regression analyses were also conducted to look for feature predictors of individual (not item order) learn rates for the 321 individual templates included in these 448 orderings. As mentioned previously, these learn rates are only calculated from occurrences when the template appears first in a response sequence. The percent correct on the template ($\beta = .31$, $SE = .1$, $p < .01$) and the item requiring a text response ($\beta = .14$, $SE = .04$, $p < .01$) were both significant predictors ($R^2=.06$, $F = 10.84$, $p < .01$).

2.11 Discussion: Regression Findings

Regression analyses were run on all 448 significant item orderings as well as subsets of scaffold and hint templates. For all 448 items (the All Items model), significant predictors of learn rate for the ordered pair consisted of only the separate learn rates for the first item (Template A) and the second item (Template B). Template B's positive influence on the learn rate for the ordering is expected. Intuitively, it is understandable that a higher learn rate for one item would lead to a higher learn rate for the ordering.

However, a *lower* learn rate of Template A predicts a higher learn rate for the ordering, which is unexpected. It is important to note that this effect may be due to constraints in our current model. The learn rate of Template A alone is calculated when Template A occurs as the first item in a SB. Template A is also included as part of an item ordering pair made up of the first and second items in an SB. This item ordering, including Template A, also has an associated learn rate. If the learn parameter for Template A is high, the knowledge component is already known by the time we consider the learn rate for the ordering including Template A. Since the learning has already occurred for the KC, the transition parameter for the ordering is lower, due to the high probability of learning for Template A alone. This model behavior does not affect the relationship between the learn rate of Template B and the learn rate of the ordering because Template B would never be modeled as the first item of an SB while part of the associated pair as it occurs second by definition.

In future analyses, single template learn rates should be calculated from all template occurrences throughout the SB, not only when it is the first item. We would expect this change in future models to reduce the negative association between the learn rate for Template A and the learn rate for the ordering.

2.12 Desirable Difficulties

In previous proof-of-concept work [3], a qualitative analysis was performed to examine what might make certain item orderings more effective than other item orderings. One feature of item pairs that became obvious was that not all items had exactly the same level of difficulty, even though the ASSISTments system assumes that items within an SB are of equal difficulty. In addition, some effective orderings contain a harder item first whereas other effective orderings contain an easier item first.

One potential hypothesis that can help explain this difference in item ordering and difficulty is that of “desirable difficulties”. In a series of studies, Bjork and colleagues determined that some challenges to performance during learning activities may actually contribute to greater learning [13] [12]. By introducing “desirable difficulties” that help learners engage in the active processing of information, learning tasks that may be perceived as challenging or inefficient may prove more beneficial in the long run than those completed with high fluency.

In the case of item orderings where the first problem is more difficult than the second, the first (more difficult) problem may introduce a desirable difficulty, leading the student to learn more than they would with an easier problem. This learning then carries over into the second problem in the pair, thus leading to a higher overall rate of learning. This hypothesis works towards explaining our finding that a lower learn rate of the first template predicts a higher learn rate for an item ordering.

When the first problem is easier than the second, this might be an instance where the material is better learned through a gentler or simpler introduction, as perhaps the second problem might be more difficult than is ``desirable". In this case, a student would not properly learn from the more difficult problem unless it were preceded by an easier problem that would serve as a scaffold.

Using data from the BKT model to examine this hypothesis, we looked at how the difference between prior knowledge (at the start of an SB) and the percent correct on a template (as a proxy for template difficulty) compared to the probability of learning using regression. Finding *no difference* between a student's prior knowledge and the percent correct for a given template might show when an item has an “appropriate” difficulty. In this case, the difficulty of the item closely matches the prior knowledge of the student. Pedagogically, for an item to help the student learn, the difference between the student's prior knowledge and the item difficulty should be negative; in other words, the difficulty of the item should be above the level of the student's prior knowledge to promote learning.

Regressing the difference between prior knowledge and item difficulty (percent correct) on the probability of learning showed statistical significance at the 0.01 level. This statistical significance held when using the difference between prior knowledge at the beginning of an SB and the percent correct on the first item in a pair (Template A), as well as the difference between prior knowledge and the percent correct for the second item in the pair (Template B). Using the percent correct for Template A to find the difference between the student's prior knowledge and the item difficulty had a correlation of -0.3039 with the probability of learning, while using Template B had a -0.2146 correlation with the probability of learning.

These correlations are both relatively high, showing enough relationship between the variables to warrant further exploration in this area. Similar to the correlations, the regressions were also run using percent correct from Template A and from Template B in the difference between prior knowledge and item difficulty. For Template A the coefficient for regressing the difference between prior knowledge and item difficulty (percent correct) on the probability of learning was -0.187 (R^2=\$0.09, F=45.39); using template B, the coefficient was -0.120 (R^2=0.046, F=21.53). The negative correlations, as well as negative coefficients in each of the regressions, show that the more negative the difference between prior knowledge and item difficulty becomes (the larger the difference between these two variables in the right direction for a ``desirable difficulty"), the greater the probability of learning becomes. Scatterplots showing the relationship between these variables can be seen in Figure 7 and Figure 8.

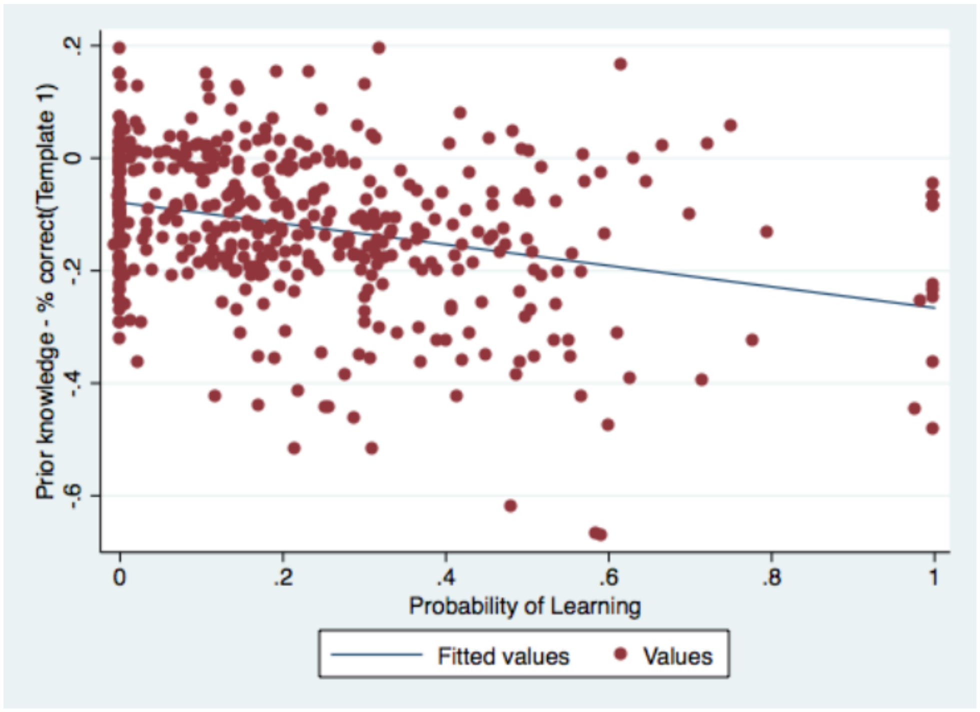


Figure 7 Desirable Difficulty Learning Template 1

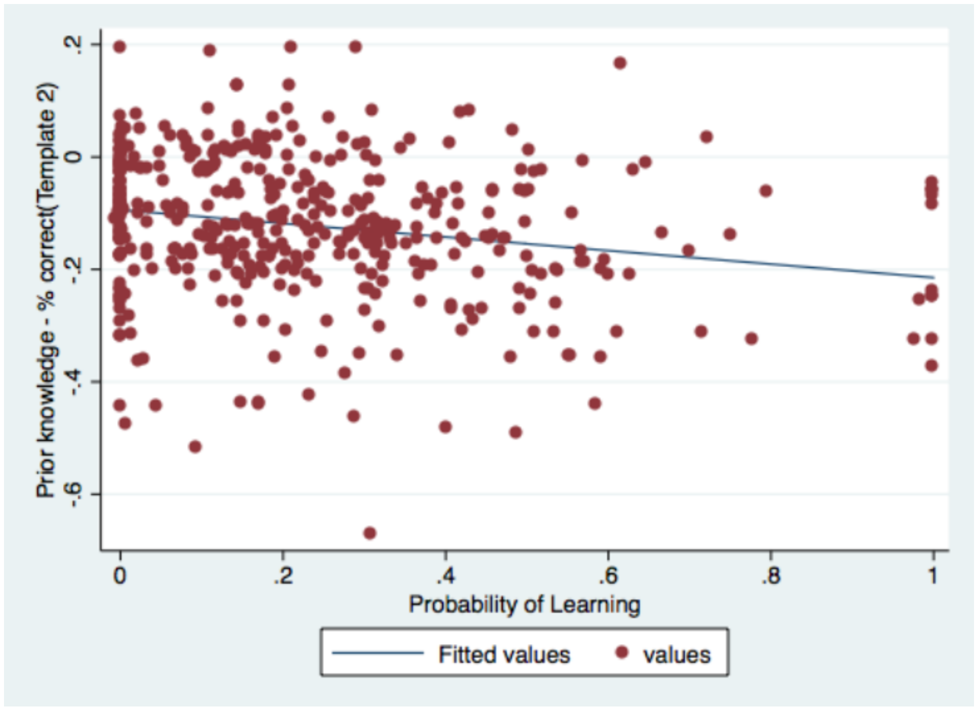


Figure 8 Desirable Difficulty Learning Template 2

Testing whether an item is of a "desirable" difficulty would likely require a careful experimental design, rather than a post-hoc analysis of item properties when items are administered at random. Thus, numerous future research opportunities abound to further investigate how to best order items; in particular, it could be pertinent to explicitly engineer items with particular difficulty differences and test which orderings lead to the quickest learning of a KC.

2.13 Experiment

The previous sections showed that 448 of the possible 1789 item orders showed a statistically significant better prediction when using the item order model. To test whether these item orders might actually yield a gain in learning performance in practice, an experiment is proposed. The experiment is set up so that there are three groups: a control group, a positive group, and a negative group. In all 3 groups, students are given a Skill Builder of 5 tutoring items, where tutoring is enabled. While all 3 groups received the same set of items, the order in which they received items differed by group. For the 5 item set of tutoring items, the control group received a randomly ordered set of items, while the positive group received a set of items that was constructed to have a higher learn rate overall, and the negative group received a set of items that was constructed to have a lower learn rate overall. After the 5 item tutoring set, each of the 3 groups was given the same post-test, consisting of 3 items. The items came from a Skill Builder that contained 5 statistically significant item orders, thus enabling a variety of possible sequences that might yield higher learn rates.

Although this test design is rather short, we were guided by the ASSISTments research team in that it is more feasible for teachers to try shorter sets of problems, thus maximizing the potential for us to gain a higher count of students actually joining the experiment. In a more ideal setting, a longer study with a pre-test and longer post-test would be desirable.

We released the constructed experimental problem sets to the ASSISTments research platform, allowing for teachers to administer this tutoring / test set to students. In total, we had 89 students complete the experiment, with 23 students completing the control group test, 33 students completing the positive group test, and 33 students completing the negative group test.

Table 3 Results from Item Ordering Experiment

	Students who completed the problem set	**PostTestScore**
Control	23	0.51 (0.15)
Experiment Positive	33	0.54 (0.1)
Experiment Negative	33	0.52 (0.13)

Table 3 shows the results from the ASSISTments experiment, with PostTestScore taking the form of Mean (Standard Deviation). The positive group had the highest post test score, but the difference was not large relative to the negative and control groups. The post test scores were not significantly different, indicating that the effect of ordering the same set of items in different manners could not produce noticeable gains in the 3-item post test score at least for this experimental set up.

3 Predictive Modeling of Student Behavior using Granular Large Scale Action Data from a MOOC

3.1 Transitional framing within dissertation

The previous chapter investigated a modification to the BKT framework by allowing for an effect of the ordering of tutoring questions. In this chapter, I shift away from the cognitive domain and into the behavioral domain. This chapter presents initial work in applying deep learning based methods towards sequential modeling of student actions and behaviors in MOOC contexts at scale.

Massive Open Online Courses generate a granular record of the actions learners choose to take as they interact with learning materials and complete exercises towards comprehension. There exist several methods for looking at longitudinal, sequential data like those recorded from learning environments. In the field of language modeling, traditional n-gram techniques and modern recurrent neural network (RNN) approaches have been applied to algorithmically find structure in language and predict the next word given the previous words in the sentence or paragraph as input. In this chapter, an analogy is drawn to this work by treating student sequences of resource views and interactions in a MOOC as the inputs and predicting students' next interaction as outputs. This approach learns the representation of resources in the MOOC without any explicit feature engineering required. This model could potentially be used to generate recommendations for which actions a student ought to take next to achieve success. Additionally, such a model automatically generates student behavioral state, allowing for inference on performance and affect. Given that the MOOC used in this chapter had over 3,500 unique resources, predicting the exact resource that a student will interact with next might appear to be a difficult classification problem. It is found that the syllabus (structure of the course) gives on average 23% accuracy in making this prediction, followed by the n-gram method with 70.4%, and RNN based methods with 72.2%. This research lays the ground work for behavior modeling of fine grained time series student data using feature engineering free techniques. The work presented in this chapter [5] was published with my co-authors Joshua Peterson in addition to my advisor Zachary Pardos.

3.2 Introduction

Today's digital world is marked with personalization based on massive logs of user actions. In the field of education, there continues to be research towards personalized and automated tutors that can tailor learning suggestions and outcomes to individual users based on the (often latent) traits of the user. In recent years, high volume sources of student-generated learning actions have been collected by higher education online learning environments such as Massive Open Online Courses (MOOCs). In this chapter, a goal is made to contribute to the growing body of research that aims to utilize large sources of student-created data towards the ability to personalize learning pathways to make learning as accessible, robust, and efficient as desired. To do so, a new strand of research is started, focused on modeling the behavioral state of the student, distinct from research objectives concerned primarily with performance assessment and prediction. In this research, an attempt is made to consider all actions of students in a MOOC such as viewing lecture videos or replying to forum posts. Such an approach makes use of the granular, non-assessment data collected in MOOCs and has potential to serve as a source of recommendations for students looking for navigational guidance.

Capturing the behavioral trends that successful students take through MOOCs can enable the development of automated recommendation systems so that struggling students can be given meaningful and effective recommendations to optimize their time spent trying to succeed. Generative sequential models are models that can take in a sequence of events as an input and generate a probability distribution over what event is likely to occur after that input sequence of events. Thus, the goal of this chapter is to evaluate the effectiveness of sequential models in predicting student behaviors captured during the administration of a MOOC. Two types of generative sequential models are utilized in this work, specifically the n-gram and the recurrent neural network model, which are models that have been traditionally successful when applied to other generative and sequential tasks.

This chapter specifically analyzes how well such models can predict the next action given a context of previous actions the student has taken in a MOOC. The purpose of such analysis would be to eventually create a system whereby an automated recommender could query the model to provide meaningful guidance on what action the student can take next. The next action in many cases may be the next resource prescribed by the course but in other cases it may be a recommendation to consult a resource that is back in a previous lesson or enrichment material that is buried in a corner of the courseware unknown to the student. The models being trained are known as generative, in that they can be used to generate what action could come next given a prior context of what actions the student has already taken. Actions can include things such as opening a lecture video, answering a quiz question, or navigating and replying to a forum post. This research serves as a foundation for applying sequential, generative models towards creating personalized recommenders in MOOCs with potential applications to other educational contexts with sequential data.

3.3 Related Work

In the case of the English language, generative models are used to generate sample text or to evaluate the plausibility of a sample of text based on the model's understanding of how that language is structured. A simple but powerful model used in natural language processing (NLP) is the n-gram model [14], where a probability distribution is learned over every possible sequence of n terms from the training set. In recent times, recurrent neural networks (RNNs) have been used to perform next-word prediction [15], where previously seen words are subsumed into a high dimensional continuous latent state. This latent state is a succinct numerical representation of all of the words previously seen in the context. The model can then utilize this representation to predict what words are likely to come next. Both of these generative models can be used to generate candidate sentences and words to complete sentences. In this work, rather than learning about the plausibility of sequences of words and sentences, the generative models will learn about the plausibility of sequences of actions undertaken by students in MOOC contexts. Then, such generative models can be used to generate recommendations for what the student ought to do next.

In the learning analytics community, there is related work where data generated by students, often in MOOC contexts, is analyzed. There are many different types of student-generated data whereby analytics are performed, and there are many different types of prediction tasks. [16] is an example of the paradigm where raw logs, in this case also from a MOOC, are summarized through a process of manual feature engineering. In this chapter's approach, feature representations are learned directly from the raw time series data. This approach does not require subject matter expertise to engineer features and is a potentially less lossy approach to utilizing the raw information in the MOOC click-stream. In [17], prior knowledge confounders were identified to help improve correlation of MOOC resource usage with knowledge acquisition. In that work, the presence of student self-selection is a source of noise and confounders. In contrast, student selection becomes the signal in behavioral modeling. In [18], multiple aspects of student learning in an online tutoring system are summarized together via an embedding. This embedding process maps assignments, student ability, and lesson effectiveness onto a low dimensional space. Such a process allows for lesson and assignment pathways to be suggested based on the model's current estimate of student ability. The work in this chapter also seeks to suggest learning pathways for students, but differs in that additional student behaviors, such as forum post accesses and lecture video viewings, are also included in the model. Additionally, different generative models are employed.

In this chapter, only event log data from MOOCs is used. While this user clickstream traverses many areas of interaction, there are examples of behavior research that have analyzed the content of the resources involved in these interaction sequences. Such examples include analyzing frames of MOOC videos to characterize the video's engagement level [19], analyzing the content of forum posts [20] [21], and analyzing the ad-hoc social networks that arise from interactions in the forums [22]. This chapter views all categories of possible student events at a more abstract level compared to these content-focused approaches.

In terms of cognition in Learning Analytics and Educational Data Mining (EDM), much work has been done to assess the latent knowledge of students through models such as Bayesian Knowledge Tracing (BKT) [10], including retrofitting the model to a MOOC [23] using superficial course structure as a source of knowledge components. This type of modeling views the actions of students' as learning opportunities to model student latent knowledge. The work in this chapter is related, but student knowledge is not explicitly modeled. Instead, the models in this chapter focus on predicting the complement of this performance data, which is the behavioral data of the student.

Deep Knowledge Tracing [24] uses recurrent neural networks to create a continuous latent representation of students based on previously seen assessment results as they navigate online learning environments. In that work, recurrent neural networks summarize all of the prior assessment results of a student by keeping track of a complex latent state. That work shows that a deep learning approach can be used to represent student knowledge, with favorable accuracy predictions relative to shallow BKT. Such results, however, are hypothesized to be explained by already existing extensions of BKT [25]. The use of deep learning to approach knowledge tracing still finds useful relationships in the data automatically, but potentially does not find additional representations relative to already proposed extensions to BKT. The work in this chapter is related to the use of deep networks to represent students, but differs in that all types of student actions are considered rather than only the use of assessment actions.

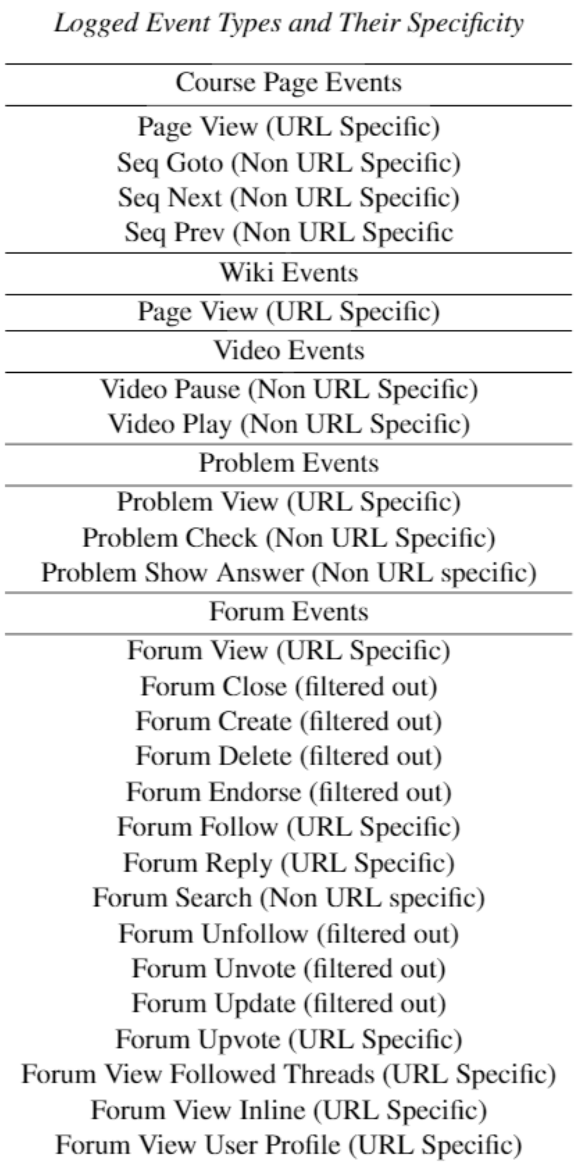
Specifically, in this chapter, both the n-gram approach as well as using a variant of the RNN known as the Long Short-Term Memory (LSTM) architecture [26] are considered. These two models are chosen as they are both used to model sequences of data and provide a probability distribution of what token should come next. The use of LSTM architectures and similar variants have recently achieved impressive results in a variety of fields that involve sequential data, including speech, image, and text analysis [27] [28] [29], in part due to its mutable memory that allows for the capture of long- and short-range dependencies in sequences. Since student learning behavior can be represented as a sequence of actions from a fixed action state space, LSTMs could potentially be used to capture complex patterns that characterize successful learning. In previous work, modeling of student click-steam data has shown promise with methods such as n-gram models [30].

3.4 Dataset

The dataset used in this chapter came from a Statistics BerkeleyX MOOC from Spring 2013. The MOOC ran for five weeks, with video lectures, homework assignments, discussion forums, and two exams. The original dataset contains 17 million events from around 31,000 students, where each event is a record of a user interacting with the MOOC in some way. These interactions include events such as navigating to a particular URL in the course, upvoting a forum thread, answering a quiz question, and playing a lecture video. The data is processed so that each unique user has all of their events collected together in sequential order. There are 3,687 types of events that are possible. Every row in the dataset is converted to a particular index that represents the action taken or the url accessed by the student.

Thus, every unique user's set of actions is represented by a sequence of indices, of which there are 3,687 unique kinds of indices. Each students history included navigating to different pages of the course which included forum threads, quizzes, video pages, and wiki pages. Within these pages, the actions that were taken within the page were also recorded, such as playing and pausing a video or checking a problem. Also recorded are javascript navigations called sequential events. In this rendition of the pre-processing, the sequence events are recorded by themselves, without explicit association with which url is navigated to by the sequential event. Table 4 catalogs the different types of events present in the dataset as well as whether we chose to associate the specific URL tied to the event or not. In this pre-processing, some of these events are recorded as URL specific, meaning that the model will be exposed to the exact URL the student is accessing for these events. Some events are recorded as Non URL specific, meaning that the model will only know that the action took place, but not which URL that action is tied to in the course. Note that any event that occurred fewer than 40 times in the original data set was filtered out. Thus, many of the Forum Events are filtered out, since they were URL specific, but did not occur very frequently. Seq goto, seq next, and seq prev refer to events triggered when students select navigation buttons visible on the browser page. Seq next and seq prev will move to either the next or previous content page in the course respectively, while a seq goto represents a jump within a section to any other section within a chapter.

Table 4 Types of Logged Events



Logged Event Types and Their Specificity

Course Page Events
Page View (URL Specific)
Seq Goto (Non URL Specific)
Seq Next (Non URL Specific)
Seq Prev (Non URL Specific
Wiki Events
Page View (URL Specific)
Video Events
Video Pause (Non URL Specific)
Video Play (Non URL Specific)
Problem Events
Problem View (URL Specific)
Problem Check (Non URL Specific)
Problem Show Answer (Non URL specific)
Forum Events
Forum View (URL Specific)
Forum Close (filtered out)
Forum Create (filtered out)
Forum Delete (filtered out)
Forum Endorse (filtered out)
Forum Follow (URL Specific)
Forum Reply (URL Specific)
Forum Search (Non URL specific)
Forum Unfollow (filtered out)
Forum Unvote (filtered out)
Forum Update (filtered out)
Forum Upvote (URL Specific)
Forum View Followed Threads (URL Specific)
Forum View Inline (URL Specific)
Forum View User Profile (URL Specific)

For example, if a student accesses the chapter 2 section 1 URL, plays a lecture video, clicks on the next arrow button which performs a javascript navigation to access the next section, answers a quiz question, then clicks on section 5 within the navigation bar which performs a javascript navigation to directly access the section 5 URL, that student's sequence would be represented by five different indices. The first index would correspond to the URL of chapter 2 section 1, the second index would correspond to a play video token, the third index would correspond to a *navigationnext* event, the fourth index would correspond to a unique identifier of which specific problem within the course the student accessed, and the fifth index would correspond to a *navigationgoto* event. The model would be given a list of these five indices, and trained to be able to predict what should come after seeing these five indices in order. The indices therefore represent the sequence of actions the student took. The length of five is not required; generative models can be given sequences of arbitrary length.

Of the 31,000 students, 8,094 completed enough assignments and scored high enough on the exams to be considered ``certified" by the instructors of the course. Note that in other MOOC contexts, certification sometimes means that the student paid for a special certification, but that is not the case for this MOOC. The certified students accounted for 11.2 million of the original 17 million events, with an average of 1,390 events per certified student. The distinction between certified and non-certified is important for this chapter, as we chose to train the generative models only on actions from students who were considered ``certified," under the hypothesis that the sequence of actions that certified students take might reasonably approximate a successful pattern of navigation for this MOOC.

Each row in the dataset contained relevant information about the action, such as the exact URL of what the user is accessing, a unique identifier for the user, the exact time the action occurs, and more. For this chapter, we do not consider time or other possibly relevant contextual information, but instead focus solely on the resource the student accesses or the action taken by the student. Events that occurred fewer than 40 times throughout the entire dataset were removed, as those tended to be discussion posts or user profile visits that were rarely accessed and are unlikely to be applicable to other students navigating through the MOOC.

3.5 Methodology

In this chapter, the use of two generative models is investigated: the recurrent neural network architecture and the n-gram. In this section, the details of each architecture are enumerated. In particular, the extension of the RNN known as the Long Short-Term Memory (LSTM) is discussed, which is the model that is hypothesized to perform best at next-action prediction. Other “shallow” models, such as the n-gram, are described afterwards.

3.6 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a family of neural network models that are designed to handle arbitrary length sequential data. Recurrent neural networks work by keeping around a continuous, latent state that persists throughout the processing of a particular sequence. This latent state captures relevant information about the sequence so far, so that prediction at later parts of the sequence can be influenced by this continuous latent state. As the name implies, recurrent neural networks utilize the computational approach utilized by feed forward neural networks while also imposing a recurring latent state that persists between time steps. Keeping the latent state around between elements in an input sequence is what gives recurrent neural networks their sequential modeling power. In this work, each input into the RNN will be a granular student event from the MOOC dataset. The RNN is trained to predict the student's next event based on the series of events seen so far.

RNNs maintain an ongoing latent hidden state that persists between each input to the model. This latent state can provide a representation of what has already been seen in the input sequence. Long Short-Term Memory (LSTM) is a modification of the RNN architecture, where the hidden latent state is replaced with a more powerful memory component. LSTMs are chosen due to their stronger performance in modeling longer range dependencies [31] [32].

RNNs maintain a latent, continuous state, represented by $\boldsymbol{h_t}$ in the equations below. This latent state persists in the model between inputs, such that the prediction at $\boldsymbol{x_{t+1}}$ is influenced by the latent state $\boldsymbol{h_t}$. The RNN model is parameterized by the input weight matrix $\boldsymbol{W_x}$, recurrent weight matrix $\boldsymbol{W_h}$, initial state $\boldsymbol{h_0}$, and output matrix $\boldsymbol{W_y}$. $\boldsymbol{b_h}$ and $\boldsymbol{b_y}$ are biases for the latent and output units.

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b_h) \qquad (1)$$
$$y_t = \sigma(W_y h_t + b_y) \qquad (2)$$

LSTMs, a popular variant of the RNN, augment the latent, continuous state with additional gating logic that helps the model learn longer range dependencies. The gating logic learns when to retain and when to forget information in the latent state. Each hidden state $\boldsymbol{h_t}$ is instead replaced by an LSTM cell unit with the additional gating parameters. The update equations for an LSTM are:

$$f_t = \sigma(Wf_x x_t + W_{fh} h_{t-1} + b_f) \qquad (3)$$
$$i_t = \sigma(W_{ix} x_t + W_{ih} h_{t-1} + b_i) \qquad (4)$$
$$C'_t = \tanh(W_{Cx} x_t + W_{Ch} h_{t-1} + b_C) \qquad (5)$$
$$C_t = f_t \times C_{t-1} + i_t \times C'_t \qquad (6)$$
$$o_t = \sigma(W_{ox} x_t + W_{oh} h_{t-1} + b_o) \qquad (7)$$
$$h_t = o_t \times \tanh(C_t) \qquad (8)$$

$\boldsymbol{f_i}$, $\boldsymbol{i_t}$, and $\boldsymbol{o_t}$ represent the gating mechanisms used by the LSTM to determine when to forget, input, and output data from the cell state, $\boldsymbol{C_t}$. $\boldsymbol{C'_t}$ represents an intermediary candidate cell state that is gated to update the next cell state. Figure 9 illustrates the anatomy of a cell, where the numbers in the figure correspond to the previously mentioned update equations for the LSTM

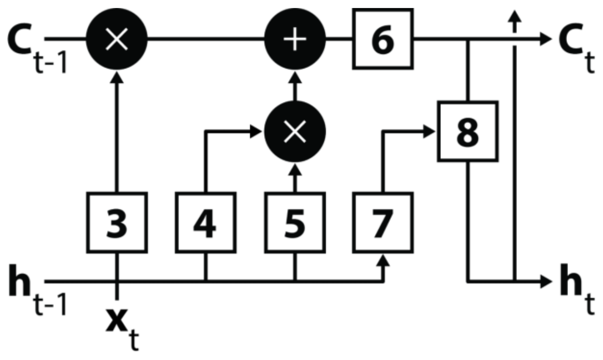


Figure 9 Anatomy of LSTM Cell

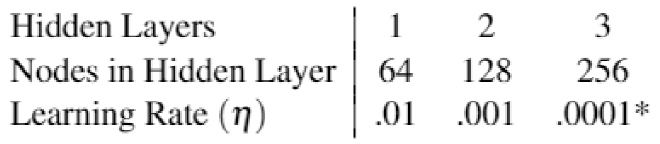
3.7 LSTM Implementation

The generative LSTM models used in this chapter were implemented using Keras [33], a Python library built originally on top of Theano [34] [35], but has since been updated to run on top of the Tensorflow package as well. The model takes each student action represented by an index number. These indices correspond to the index in a 1-hot encoding of vectors, also known as dummy variabilization. The model converts each index to an embedding vector, and then consumes the embedded vector one at a time. The use of an embedding layer is common in natural language processing and language modeling [36]as a way to map words to a multi dimensional semantic space. An embedding layer is used here with the hypothesis that a similar mapping may occur for actions in the MOOC action space. The model is trained to predict the next student action, given actions previously taken by the student. Back propagation through time [37] is used to train the LSTM parameters, using a softmax layer with the index of the next action as the ground truth. Categorical cross entropy is used calculating loss, and RMSprop is used as the optimizer. Drop out layers were added between LSTM layers as a method to curb overfitting [38]. Drop out randomly zeros out a set percentage of network edge weights for each batch of training data.

3.8 LSTM Hyperparameter Search

To find the best performing set of LSTM hyperparameters, 24 LSTM models each with a different set of hyperparameters was trained for 10 epochs each. An epoch is the parameter fitting algorithm making a full pass through the data. The searched space of hyperparameters for these LSTM models are shown in Table 5. These hyperparameters were chosen for grid search based on previous work which prioritized different hyperparameters based on effect size [39]. For the sake of time, learning rates of .0001 were not trained for the 3-layer LSTM models with learning rates of .0001.

Table 5 LSTM Hyperparameter Grid



Hidden Layers	1	2	3
Nodes in Hidden Layer	64	128	256
Learning Rate (η)	.01	.001	.0001*

This initial hyperparameter search is used to inform an extended investigation, where the most successful initial results are tested for more epochs. Because training RNNs is relatively time consuming, the extended investigation consisted of a subset of promising hyperparameter combinations. Results of the extended investigation are included in the Results section.

3.9 Cross Validation

To evaluate the predictive power of each model, 5-fold cross validation was used. Each model was trained on 80% of the data and then validated on a remaining 20%; this was done five times so that each set of student actions was in a validation set once. For the LSTMs, the model held out 10% of its training data to serve as the hill climbing set to provide information about validation accuracy during the training process. Each row in the held out set consists of the entire sequence of actions a student took. The proportion of correct next action predictions produced by the model is computed for each sequence of student actions. The proportions for an entire fold are averaged together to generate the model's performance for that particular fold, and then the performances across all 5 folds are averaged together to generate the *CV-accuracy* for a particular LSTM model hyperparameter set.

3.10 Shallow Models

N-gram model

N-gram models are simple, yet powerful probabilistic models that aim to capture the structure of sequences through the statistics of n-sized sub-sequences called *grams* and are equivalent to n-order Markov Chains. Specifically, the model predicts each sequence state x_i using the estimated conditional probability $P(x_i \mid x_{i-(n-1)}, \ldots, x_{i-1})$, which is the probability that x_i follows the previous n-1 states in the training set. N-gram models are both fast and simple to compute, and have a straightforward interpretation. It is expected that n-grams will be a competitive standard, as they are relatively high parameter models that essentially assign a parameter per possible action in the action space.

N-gram Model Structure

For the n-gram models, models were evaluated where n ranged from 2 to 10, the largest of which corresponds to the size of the LSTM context window during training. To handle predictions in which the training set contained no observations, backoff was employed, a method that recursively falls back on the prediction of the largest n-gram that contains at least one observation. The validation strategy was identical to the LSTM models, wherein the average cross-validation score of the same five folds was computed for each model.

Course Structure Models

A number of alternative models aimed at exploiting hypothesized structural characteristics of the sequence data are also proposed. In the student sequences, there are certain actions that are repeated several times in a row. For this reason, it is important to know how well this assumption alone predicts the next action in the dataset. Next, since course content is most often organized in a fixed sequence, it is important to evaluate the ability of the course syllabus to predict the next page or action. This was accomplished by mapping course content pages to student page transitions in the action set, which yielded an overlap of 174 matches out of the total 300 items in the syllabus. Since this process relied on matching content ID strings that were not always present in the action space, a small subset of possible overlapping actions were not mapped. Finally, both models were combined, wherein the current state was predicted as the next state if the current state was not in the syllabus.

3.11 Results

In this section, results are discussed from the previously mentioned LSTM models trained with different learning rates, number of hidden nodes per layer, and number of LSTM layers. Model success is determined through 5-fold cross validation and is related to how well the model predicts the next action. N-gram models, as well as other course structure models, are validated through 5-fold cross validation.

3.12 LSTM Model Results

Table 6 shows the CV-accuracy for all 24 LSTM models computed after 10 iterations of training. For the models with learning rate of .01, accuracy on the hill climbing sets generally peaked at iteration 10. For the models with the lower learning rates, it would be reasonable to expect that peak CV-accuracies would improve through more training.

Results after 10 iterations provide a snapshot of how well these models are performing during the training process. It can be hypothesized that model performance is unlikely to improve drastically over the .01 learning rate model performances in the long run, and there is also a need to maximize the most promising explorations to run on limited GPU computation resources. The best CV-accuracy for each learning rate is bolded for emphasis.

One downside of using LSTMs is that they require the use of a GPU and are relatively slow to train. Thus, when investigating the best hyperparameters to use, additional models were trained based only on a subset of the initial explorations. In this additional exploration, the amount of context exposed to the model was also extended, extending past context from 10 elements to 100 elements. Table 7 shows these extended results. Each LSTM layer has 256 nodes and is trained for either 20 or 60 epochs, as opposed to just 10 epochs in the previous hyperparameter search results. The extended results show a large improvement over the previous results, where the new accuracy peaked at .7223 compared to .7093.

Table 6 LSTM Performance (10 Epochs)

Learn Rate	Nodes	Layers	Accuracy
0.01	64	1	0.7014
0.01	64	2	0.7009
0.01	64	3	0.6997
0.01	128	1	0.7046
0.01	128	2	0.7064
0.01	128	3	0.7056
0.01	256	1	0.7073
0.01	256	2	**0.7093**
0.01	256	3	0.7092
0.001	64	1	0.6941
0.001	64	2	0.6968
0.001	64	3	0.6971
0.001	128	1	0.6994
0.001	128	2	0.7022
0.001	128	3	0.7026
0.001	256	1	0.7004
0.001	256	2	**0.7050**
0.001	256	3	**0.7050**
0.0001	64	1	0.6401
0.0001	64	2	0.4719
0.0001	128	1	0.6539
0.0001	128	2	0.6648
0.0001	256	1	0.6677
0.0001	256	2	**0.6894**

Table 7 Extended LSTM Performance (256 Nodes, 100 Window Size)

Learn Rate	Epochs	Layers	Accuracy
0.01	20	2	0.7190
0.01	60	2	0.7220
0.01	20	3	0.7174
0.01	60	3	**0.7223**
0.001	20	2	0.7044
0.001	60	2	0.7145
0.001	20	3	0.7039
0.001	60	3	**0.7147**

Figure 10 shows validation accuracy on the 10% hill-climbing hold out set during training by epoch for the 1 and 2 layer models from the initial exploration. Each data point represents the average hill-climbing accuracy among all three learning rates for a particular layer and node count combination. Empirically, having a higher number of nodes is associated with a higher accuracy in the first 10 epochs, while 2 layer models start with lower validation accuracies for a few epochs before approaching or surpassing the corresponding 1 layer model. This figure provides a snapshot for the first 10 epochs; it is clearly the case that for some parameter combinations, more epochs would result in a higher hill-climbing accuracy, as shown by the additional extended LSTM search. Extrapolating, 3-layer models may also follow the trend that the 2-layer models exhibited where accuracies may start lower initially before improving over their lower layer counterparts.

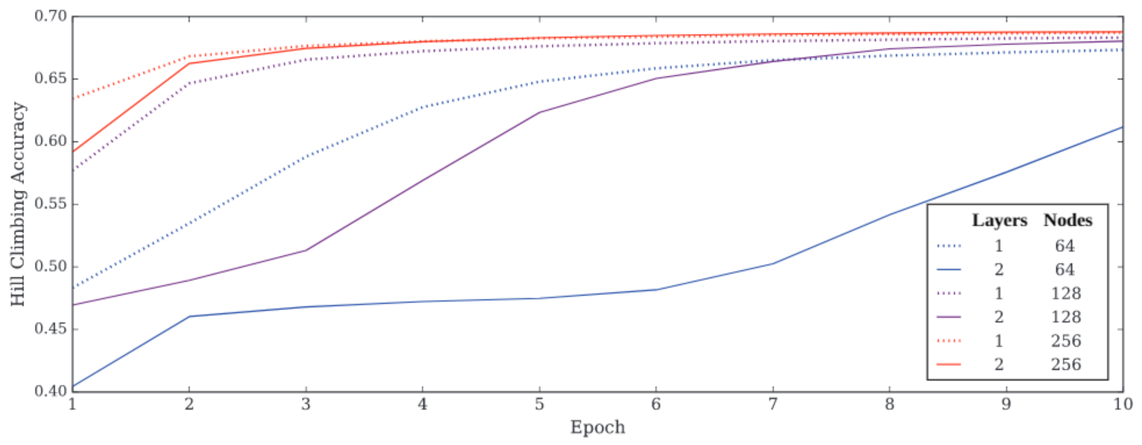


Figure 10 Average accuracy by epoch on hill climbing data

3.13 Course Structure Model Results

Model performance for the different course structure models is shown in Table 8. Results suggest that many actions can be predicted from simple heuristics such as stationarity (same as last), or course content structure. Combining both of these heuristics ("syllabus + repeat") yields the best results, although none of the alternative models obtained performance within the range of the LSTM or n-gram results.

Table 8 Structural Model Results

Structural Model	Accuracy
repeat	0.2908
syllabus	0.2339
syllabus + repeat	**0.4533**

3.14 N-gram Model Results

N-gram model performance is shown in Table 9. The best performing models made predictions using either the previous 7 or 8 actions (8-gram and 9-gram respectively). Larger histories did not improve performance, indicating that the range of n was sufficiently large. Performance in general suggests that n-gram models were competitive with the LSTM models, although the best n-gram model performed worse than the best LSTM models. Table 10 shows the proportion of n-gram models used for the most complex model (10-gram). More than 62% of the predictions were made using 10-gram observations. Further, less than 1% of cases fell back on unigrams or bigrams to make predictions, suggesting that there was not a significant lack of observations for larger gram patterns. Still, about 6% fewer data points looks to be predicted by successively larger n-grams.

Table 9 N-gram Performance by N

N-gram	Accuracy
2-gram	0.6304
3-gram	0.6658
4-gram	0.6893
5-gram	0.6969
6-gram	0.7012
7-gram	0.7030
8-gram	**0.7035**
9-gram	**0.7035**
10-gram	0.7033

Table 10 Proportion of 10-gram Prediction by N

n	% Predicted by
1	0.0003
2	0.0084
3	0.0210
4	0.0423
5	0.0524
6	0.0605
7	0.0624
8	0.0615
9	0.0594
10	**0.6229**

3.15 Validating on Uncertified Students

The previous models were all trained on data only from learners who achieved certification from the MOOC. It could be interesting to see how well this model performs on the data produced by learners who did not achieve certification. Therefore, the best performing initial LSTM model after 10 epochs of training (.01 learn rate, 256 nodes, 2 layers) is used to predict actions on streams of data from students who did not ultimately end up certified. Many uncertified students only had a few logged actions, so this analysis is restricted to students who had at least 30 logged actions. There were 10,761 learners who met these criteria, with a total of 2,151,662 actions. The LSTM model was able to correctly predict actions from the uncertified student space with .6709 accuracy, compared to .7093 cross validated accuracy for certified students. This difference shows that actions from certified students tend to be different than actions from uncertified students, perhaps showing potential application in providing an automated suggestion framework to help guide students.

3.16 Summarizing Results

In this chapter, the problem of modeling granular student action data by modeling all types of interactions within a MOOC was approached. This differs in approach from other work which primarily focuses on modeling latent student knowledge using assessment results. In predicting a student's next action, the best performing LSTM model produced a cross-validation accuracy of .7223, which was an improvement over the best n-gram model accuracy of .7035. This amounts to 210,000 more correct predictions of the total 11-million possible. Table 11 shows the number of times the two models agreed or disagreed on a correct or an incorrect prediction during cross validation. Both LSTM and n-gram models provide significant improvement over the structural model of predicting the next action by syllabus course structure and through repeats, which shows that there are patterns of student engagement that clearly deviate from a completely linear navigation through the course material.

Table 11 Cross Validated Model Comparison

	N-gram Correct	N-gram Incorrect
LSTM Correct	7,565,862	577,683
LSTM Incorrect	367,960	2,735,702

The work thus far in this chapter marks the first time that behavioral data has been predicted at this level of granularity in a MOOC. It also represents the first time Recurrent Neural Networks have been applied to behavioral MOOC data. This technique for representing student's behavioral states from raw time series data, without feature engineering, has broad applicability in any learning analytics context with high volume time series data. While the current framing suggests how behavioral data models could be used to suggest future behaviors for students, the representation of their behavioral states could prove valuable for making a variety of other inferences on constructs ranging from performance to affect.

4 Enabling Real-Time Adaptivity in MOOCs with a Personalized Next-Step Recommendation Framework

4.1 Transitional Framing Within Dissertation

The previous chapter established that RNN based behavior models in MOOC contexts can be effective, finding improvements over N-gram and course structure models in predicting what learners will do next. This next chapter shows that such an RNN model can be deployed as part of a content recommendation framework in a live MOOC setting. Developing and deploying a recommendation framework in a live setting also demands careful design considerations, which are also discussed within this chapter. This work was developed with my co-authors Daniel Davis and Christopher Le, over-arching guidance from my chair Zachary Pardos, and was published [6] as part of the Learning at Scale conference proceedings.

In this chapter, a first-of-its-kind adaptive intervention in a MOOC utilizing real-time clickstream data and a novel machine learned model of behavior is presented. Included are the implementation details of how the edX platform can be augmented with the capabilities necessary to support this type of intervention which requires both tracking learners' behaviors in real-time and dynamically adapting content based on each learner's individual clickstream history. This chosen pilot intervention was in the category of adaptive pathways and courseware and took the form of a navigational suggestion appearing at the bottom of every non-forum content page in the course. This pilot intervention is designed to help students more efficiently navigate their way through a MOOC by predicting the next page they were likely to spend significant time on and allowing them to jump directly to that page. While interventions which attempt to optimize for learner achievement are candidates for this adaptive framework, behavior prediction has the benefit of not requiring causal assumptions to be made in its suggestions. A novel extension is presented, where the behavioral model takes into account students' time spent on pages and forecasts the same. Several approaches to representing time using Recurrent Neural Networks are evaluated and compared to baselines without time, including a basic n-gram model. Finally, design considerations and handling of edge cases for real-time deployment are discussed, including considerations for training a machine learned model on a previous offering of a course for use in a subsequent offering where courseware may have changed. The work from this chapter opens the door to broad experimentation with adaptivity and serves as a first example of delivering a data-driven personalized learning experience in a MOOC.

4.2 Introduction

The path towards a more democratized learner success model for MOOCs has been hampered by a lack of capabilities to provide a personalized experienced to the varied demographics MOOCs aim to serve. Primary obstacles to this end have been insufficient support of real-time learner data across platforms and a lack of maturity of recommendation models that accommodate the learning context and breadth and complexity of subject matter material in MOOCs. In this chapter, we address both shortfalls with a framework for augmenting a MOOC platform with real-time logging and dynamic content presentation capabilities as well as a novel course-general recommendation model geared towards increasing learner navigational efficiency. We piloted this intervention in a portion of a live course as a proof-of-concept of the framework. The necessary augmentation of platform functionality was all made without changes to the open-edX codebase, our target platform, and instead only requires access to modify course content via an instructor role account.

The organization of the chapter begins with related work, followed by technical details on augmentation of the platform's functionality, a description of the recommendation model and its back-tested prediction results, and finally an articulation of the design decisions that went into deploying the recommendation framework in a live course.

4.3 Related Work

In searching for answers to the problem of dismal completion rates in MOOCs, previous research has shown that MOOC learners often feel lost or isolated in their learning experience [40]. So far, the attempts to address this problem have largely come in the form of self-regulated learning (SRL) support interventions. For example, [41] tested the effectiveness of recommending self-regulating learning strategies to MOOC learners in the pre-course survey, but did not observe any significant changes in behavior as a result. As an example of a MOOC experiment integrated in the course content, [42] ran experiments in two MOOCs evaluating the effectiveness of providing learners with retrieval cues (to facilitate the active retrieval of information from memory) and study planning support (planning and reflecting on one's learning activities each week)—both foundational techniques in self-regulation. However, in both studies the authors report null results, with no evidence that providing this support to learners was beneficial. Another approach to instructional interventions in MOOCs is found in [43] where the authors manipulated the course discussion forum. In one condition, the course instructor was active in the discussion forum and provided support to the learners in answering their questions; in the other, the instructor was absent and the learners were on their own to discuss amongst themselves. Just as in the previous two studies, this yielded no significant change in behavior between the conditions.

To address the challenge of implementing a real-time, adaptive intervention in a MOOC, we act on the need to find a way to effectively support learners in improving their navigational efficiency with the course materials. We here present a new form of support for MOOC learners in our next step recommendation system, as prior work has shown a strong relationship between the success of a MOOC learner (measured by course completion) and the characteristics of their learning path through the course [44]. While novel to the MOOC context specifically, such recommender systems have been applied to educational settings in the past, namely in intelligent tutoring systems (ITS). Both [45] and [41] provide an overview of the various approaches used to recommend and adapt course content and resources to learners in the context of ITS.

To highlight some example use cases of learning path adaptivity in prior research, we begin with an early example of real-time “task-loop adaptivity” (defined in [45] as the guiding of learners from task to task) offered in [13]. The authors here present a tutoring system which models a student’s learning path in terms of correct and incorrect actions, and would adaptively intervene to guide students back to the correct path of action with immediate feedback.

The authors in [46] [47] provide real-time adaptive hints to coding assignments in the context of computer programming MOOCs. Both approaches are “step-loop” [45] in that they provide adaptive hints regarding the learners’ problem solving process. However, they take different approaches in doing so; [46] models the ideal process of solving the problem in a “Problem Solving Policy,” as defined by an expert, and guides learners towards this behavior. [47], on the other hand, leverages the scale of MOOCs and proposes algorithms which use the surrounding context of a code snippet to identify the problem and recommend a solution to the learner. The authors in [48] present a personalized navigation support system in the context of a JavaScript programming course. By monitoring the learner’s performance on previous problems, the system presented learners with a next-step suggestion to try problems of the appropriate, or “optimal,” difficulty level. By addressing the issue of learners navigating themselves to tasks that are too easy or too difficult, this system increased learner achievement and engagement.

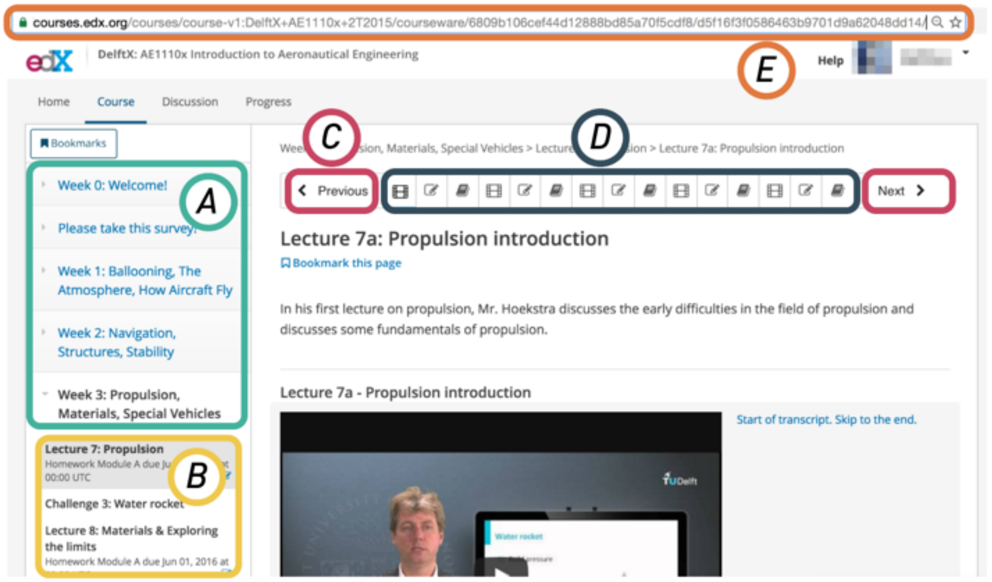
[49], [50], and [51] describe the design and deployment of an adaptive hint generator in an ITS on the topic of logic. This system uses past learner activity data as input for a Markov decision process which, when prompted by the learner requesting a hint, provides personalized support based on the current progress through the problem. This step-loop adaptivity was empirically tested in [51] where, compared to a tutor system without adaptive hints, learners receiving the adaptive hint system earned higher grades, tried more problems, and persisted deeper into the course. While the next-step recommender system we present here does not provide hints about how to solve a given quiz or assessment problem, the suggestions we provide can be thought of as hints on how to most efficiently navigate the course.

The next-step recommendation system proposed here is course content-general and concerned solely with modeling learner behavior from the navigational patterns of peers from previous offerings of the course. This is in contrast to studies described above which are based on modeling a learner’s mastery of the course topic/domain or helping them through a given task. It also differs in that the system does not acknowledge any “correct” or “incorrect” learning path as described in [13]. The system could be trained to bias towards the behaviors of certificate earners but this would miss out on serving those who do not intend to complete but nevertheless wish to make use of portions of the courseware. While the objective of the recommender is not explicitly focused on improving cognitive aspects, as was attempted to be modelled in [52], it will facilitate this in so far as past behavior has been a means to these ends, for example by recommending resources for review before a quiz. These considerations are key when it comes to the eventual evaluation of recommendation quality. A review of the work in the area of recommender systems suggests that every context in which a system operates has its own special aspects against which both the system and its success metrics must be evaluated [53]. Although outside of the scope of this chapter, future evaluation of this intervention might include: increasing navigational efficiency (clicks per performance), affective experience (feeling supported), as well as common outcomes such as grade and completion rate.

Thinking back to the challenge of addressing MOOC learners feeling lost in the course, we propose next-step recommendations as a service that could reach learners most in need of engagement. Pointing to recent findings from HCI research, [54] found that people are stimulated and respond positively to recommendations when they are bored. The potentially-overwhelming selection of possible next steps in a MOOC compounded with the complexity of course content can, understandably, leave a learner frustrated. A friendly next-step recommendation can be the support they need to move forward and persist.

4.4 Platform Augmentation

Several technical hurdles had to be overcome in order to add base functionality that would enable at-scale deployment of a real-time recommendation system within the edX platform. All solutions can be achieved without modification to open-edx and only require standard instructional design team / instructor access to edit course material. Figure 11 shows an annotated screen shot of the edX interface. Label (A) shows what is henceforth referred to as "Chapters," (B) refers to "Sequentials," (C) refers to navigation/goto buttons, (D) refers to "Verticals," and (E) is the page URL


courses.edx.org/courses/course-v1:DelftX+AE1110x+2T2015/courseware/6809b106cef44d12888bd85a70f5cdf8/d5f16f3f0586463b9701d9a62048dd14/
edX
DelftX: AE1110x Introduction to Aeronautical Engineering
E
Help
Home
Course
Discussion
Progress
Bookmarks
C
D
Week 3: Propulsion, Materials, Special Vehicles > Lecture 7: Propulsion > Lecture 7a: Propulsion introduction
A
Week 0: Welcome!
Please take this survey!
Week 1: Ballooning, The Atmosphere, How Aircraft Fly
Week 2: Navigation, Structures, Stability
Week 3: Propulsion, Materials, Special Vehicles
Previous
Next
Lecture 7a: Propulsion introduction
Bookmark this page
B
Lecture 7: Propulsion
Homework Module A due Jun 01, 2016 at 00:00 UTC
Challenge 3: Water rocket
Lecture 8: Materials & Exploring the limits
Homework Module A due Jun 01, 2016 at
Lecture 7a - Propulsion introduction
Start of transcript. Skip to the end.
TU Delft
Water rocket

In his first lecture on propulsion, Mr. Hoekstra discusses the early difficulties in the field of propulsion and discusses some fundamentals of propulsion.

Figure 11 Annotated breakdown of edX interface components

4.5 Enabling real-time logging

Our real-time recommendation requires knowledge of the student's most recent navigational events, some of which may have occurred only seconds earlier. The edX platform provides a daily event log delivery to its X consortium members but does not have a real-time data API. In order to enable access to real-time learner event logs, we set up a JavaScript logger within the xml of every page in the course which communicated to the recommendation server which events to store in the logging database. This process is illustrated in Figure 12.

The client-side logging, which we describe as the sensor code, was written in JavaScript. The sensor code was responsible for gathering four items of information from the client at every page: (1) the learner's userID (2) the page's chapter (3) the page's sequential (4) the page's vertical.

The learner's anonymous ID can be queried simply enough from Segment's analytics library used by edX:

```
userid = analytics.user().anonymousId();
```

The anonymousId call has the shortcoming that it will change if the user switches devices or browsers. A non-anonymized userID call is also available, which will remain stationary throughout.

Next is the retrieval of chapter and sequential ID, both of which can be parsed from the browser URL:

```
var url = window.location.href;

var split = url.split("/");

chap = split[6];

seq = split[7];
```

The vertical ID, also known as the position ID within a sequential, is non-trivial to retrieve. While verticals can be accessed by adding the vertical number to the sequential URL, this is rarely how verticals are accessed in the course. They are most commonly accessed via the "next" and "previous" arrow buttons which are graphical navigational elements on either side of the sequential accordion view. When these "seq" events are triggered, the desired page's content dynamically replaces the current page. This dynamic loading keeps the browser URL the same which means that the vertical position must be queried from a different source. We find this vertical position information in the edX document object model (DOM[1]).

```
var block = $('#sequence-list .nav-item.active').data('id');

vert = block.split("@").pop();
```

Arbitrarily clicking on a vertical in the accordion triggers a "seq_goto" event which is much the same as the next and previous events in how they load the page.

[1]All DOM related function calls used in this work are undocumented by edX and subject to change. After conducting this pilot study of the framework, we contacted edX in regards to the supportability of our approach, including providing persistent anon IDs. This support is currently under review.

With all of these elements now stored, the full description of the page a learner is on can be described:

```
origin = chap+"/"+seq+"/"+vert;
```

The userID and origin are sent to a local server for logging via a cross domain aJax POST method.

Table 12 Example of entries in local mongo database

Row ID	Anon Stu. ID	Origin	Rec	Followed	Previous ID	Timestamp	Time Category
100	C103	5	6	0	99	1477142712	2
101	C103	35	45	1	100	1477142732	1
102	C548	89	101	0	82	1477142736	2

Table 12 shows the columns stored in the logging database and a few example entries. At the time of the event, only the following columns are populated: row id (transaction id), stu_id, origin, timestamp, and previous ID (the previous transaction id of the user). The remainder of the columns are populated on the subsequent event. Full client side javascript can be found here[2].

4.6 Enabling real-time recommendation

An html <div> container is inserted at the bottom of every page which contains a template of the recommendation text. The container is marked as hidden using “display: none” until a recommendation is received successfully, upon which time the template is populated with the actual page being recommended and its title. By hiding the template until a recommendation is received, we are able to fail gracefully and shield learners from any error that may occur along the recommendation pipeline; in the case of an unsuccessful recommendation request, the page would appear to the user the same way as it would as if no intervention was added.

[2] https://github.com/CAHLR/adaptive_mooc_LAS/

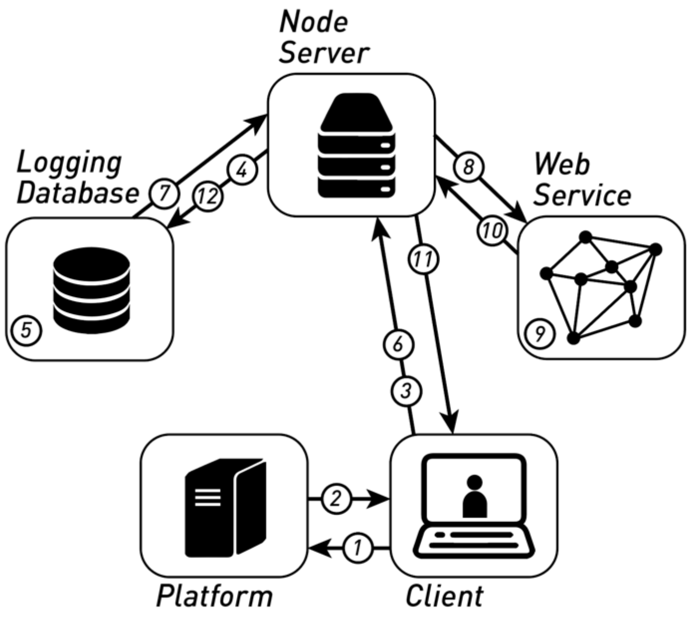


Figure 12 Diagram visualizing the entire process of delivering a recommendation to the learner

The recommendation URL and title is populated by (i) sending an aJax POST to the recommendation server, which in turn (ii) looks up the learner's event history from the logging server and then (iii) passes that information to a web service which interfaces with the machine learned model. The model returns a recommendation which is passed back through to the web service. This is then sent to the recommendation server and then to the requesting client. At this point, the "Rec" column of Table 12 is filled in representing the internal index of the recommended URL, and "Followed" is set to 0. If the learner clicks on the recommended URL, a request is sent to the recommendation server, the "Followed" is set to 1, and the learner is redirected to the recommended URL. Upon loading a subsequent page, either by following the recommendation or clicking on a different navigational component, the sensor code will look up the previous event of the learner and update the time category of the past event. This is necessary since it is unknown how long the learner will spend on the page when it is first logged.

1. The learner requests a page in the course
2. The platform sends the page to the client. In the case of a "seq" event, the page is loaded in dynamically.
3. Client sensor code sends a logging event to the server
4. The server writes the event to a Mongo database
5. If a previous event exists for this student, the time category of that event is calculated and updated.
6. Client sends a request to the server for recommendation
7. The database is queried for all of the learner's past events and respective time categories.
8. The server relays this information to a Flask web service that parses the information and passes it along to the machine learned model written in python.

9. The machine learned model predicts forward until it finds a page that the user is predicted to spend more than 10 seconds on.
10. The recommended page is returned to the server.
11. The server sends this page to the client which parses a valid “200” response into a proper hyperlink and populates the <div> to display the recommendation.
12. The server will update the logging database for this learner with the recommendation simultaneously
13. If user clicks on the recommendation, the server is contacted and the database is updated to indicate that user followed the recommendation.

The term "learner" is used when an event is triggered due to a deliberate action on the part of a human, such as clicking on a link. The term “client” is used when actions are initiated, invisible to the learner (e.g. sending a logging request), by code processed by their web browser.

4.7 Choice of Technology

In order to create this live intervention, we used a range of different technologies. NodeJS and Express were used to create the server API; Python Flask served as a light-weight web service; Python Keras was used to create the machine learned model; and Mongo was used for persistent database storage.

Server - NodeJS with Express

We decided to create our server using Node primarily because it is fast and performs well under stress. It handles operations asynchronously and facilitates a large number of simultaneous connections very well. It integrates nicely with MongoDB and can easily create routes with the Express framework.

Our API has several local lookup tables including a mapping of url to index (used for the machine learned model), index to url, url to edX path, and edX path to display name.

When the server receives a post request from the client it creates a new event with a unique user_id, origin, and timestamp. It will then check if the student has had a previous entry. If yes, the server will update the previous the timeSeq column of the previous entry and update the Recents database with this current entry. If no previous entry exists, it will skip the update in the Events database and go straight to updating the Recents database for this student. It will create an entry in the Recents database if this is the student’s very first event.

After successful logging and updating, the client will ask for a recommendation for this particular student. The server will then take the student’s unique user_id and query the Events database for the sequence of events and timeSeqs connected with this student. The output will then be sent to the Python web service for a recommendation.

When the web service responds, the response is checked. A lookup is then done to go from index to url as well as Edx path to name and then sent back to the client. The final JSON response will have the url, Sequential display name, and Vertical display name of the recommendation.

Web Service - Python & Flask

We decided to create web service using Python and Flask because our machine learning model was written using Python. It made it easiest to get the input into the correct format and parse the output into a simple response. Flask also allowed us to create multiple processes for parallelizability.

The web service is called after the server requests for a recommendation for a particular student. It takes in a list of the student's events and associated time categories, and then queries the machine learned model. It will receive either a -1 or an index from the machine learned model. If the response is a -1, then there is no valid recommendation (i.e., no recommendations meet the minimum time anticipated for the learner to spend on the page).

Machine Learned Model - Python Keras

Keras is a neural network machine learning framework providing functions for fast model prototyping. It has the option of utilizing Theano or tensorflow for the backend computations, both of which can utilize GPUs for accelerated training.

Database - MongoDB

We decided to use Mongo as our database of persistent storage because it is scalable and quickly handles simultaneous queries. It also has fast in-place updates and has documents stored in JSON, which makes it efficient to work with our client and server code.

4.8 Choice of Course

This framework is generally applicable to different backend recommendation algorithms with different objectives. For our purposes of navigational behavior recommendation, there were several criteria that we anticipated as important in selecting a reasonable pilot course.

Given our objective of increasing the navigational efficacy of learners, courses with more numerous pages to navigate are better candidates for demonstrating the utility of navigational recommendation. In order to learn non-trivial navigational patterns from past course events, we also wanted a course with a high amount of variation in navigational pathways exhibited by its learners. To measure this variation, we chose to treat student paths through a particular course as a Markov chain and then computed the entropy of the transition probability matrix for each course [18]. There were 13 courses evaluated offered by our deployment University partner, DelftX. Table 13 shows the entropy calculated for a variety of courses where entropy was 20 or greater. A higher amount of entropy indicates larger amounts of non-linear navigation. Since the Intro to Aeronautical Engineering course had both a high entropy and candidate assets to recommend, we selected that course for deployment.

Table 13 Course suitability evaluation based on navigational entropy and asset quantity

Course	Entropy	Assets	Normalized Entropy+Assets
Intro to Aeronautical Engineering (2014)	343	1175	1.782
Intro to Water & Climate (2013)	149	1503	1.434
Intro to Drinking Water treatment (2015)	86	745	0.806
Economics of Cybersecurity (2015)	78	323	0.746

4.9 Modeling Navigation Behavior

The literature on cognition and learning has several theories for describing how knowledge acquisition develops over time. Far fewer theories exist for behavior, however, as it is an amalgam of many cognitive and affective factors. As such, the lack of existing theory to adequately predict navigational behavior to a high degree of accuracy means that there is also a lack of knowledge of which manually engineered features may capture student behavior. As such, we use a model that makes no assumption about behavior and instead learns these features from the raw time series data itself.

To model student navigation behavior, we chose to use the Recurrent Neural Network (RNN) architecture. RNNs are able to model time sensitive dependencies between events in arbitrarily long sequences without the need for manual feature engineering.

To provide an example, an RNN can be given a sequence of URLs a learner has already visited. The RNN maintains a hidden, continuous state that represents the past behavior exhibited by the learner. The RNN model can then output a probability distribution over the next URL the student is likely to visit. Thus, we can then take the output of the RNN as a potential recommendation to serve to the learner. The output can be augmented to also be able to predict the amount of time that a learner will spend on the resource. With this augmentation, we can then choose to only provide recommendations where there is expected to be a significant amount of time spent on the URL. This helps expedite the learner's navigation through the course by skipping less useful content.

To use an RNN model, the logs of student actions must be parsed so that each student can be represented by a single list which contains each unique course URL the student has visited. Additionally, the timestamp associated with each course URL visit is also tracked. These timestamps are used to create a proxy for the amount of time spent on a resource. We investigate whether adding time spent as an input to the RNN model improves its predictive accuracy, and investigate two model modifications to incorporate time spent as an input.

Understanding edX logging of navigational events

Parsing a data log of student actions is not trivial. In this work, the ultimate goal of parsing through the data log is to obtain the sequence of course URLs that each student has visited, as well as the timestamp associated with each visit. The data log contains other student events, such as pausing videos and answering quiz questions. For this work, such rows were dropped. Thus, only navigation events were kept, where navigation is defined as visiting a specific course URL. These navigation events were then parsed to resolve to a specific course URL. Each URL contains a chapter hash, a sequential hash (which refers to sections within a chapter), and a vertical hash (which refers to a specific course page within a section). For example, a URL represented by ‘abc123/zzz444/2’ would have a chapter hash of ‘abc123’, a sequential hash of ‘zzz444’, and a vertical value of ‘2’. Thus, each navigation event in the edX data log can be resolved to a specific URL. However, each event in the raw log unfortunately does not directly map to a URL without an extra step of processing. Navigation events can be found in rows where either:

1. The row is a ***seq event***. Seq events include *seq_next*, *seq_prev*, or *seq_goto*. Next and prev refer to moving directly forward or backwards one vertical. Goto is a jump to any vertical within a single sequential.

   OR

2. The row contains a direct course page URL. In the URL, the vertical may be given directly, or the vertical may be missing.

Both types of navigation events mentioned above have data processing quirks. *Seq_next* and *seq_prev* events contain the sequential hash and the vertical that is navigated to. Using the sequential hash, the chapter hash can be inferred, since there is only one sequential hash per section in the course, and each section only belongs to one chapter. The vertical displayed by the row, however, may need to be additionally processed when *seq_prev* is invoked on the first vertical in a section or *seq_next* is invoked on the last vertical in a section. For example, the row in the data log may contain a *seq_next* to vertical 7 in a particular section. However, that section might only contain 6 verticals. This event should actually point to vertical 1 of the next section. Thus, the processing code must be able to handle when navigating to the previous and next sections when the current vertical is at the beginning or the end of the section. Once the corresponding sequential, chapter, and vertical hashes are resolved, a URL can be constructed to represent the URL that the student is now at in this row.

For the second type of navigation event, where the row contains a direct course URL, when the vertical is included in the row, the URL can be directly taken from the row itself. When the vertical is not included in the row, which means that the row contains a chapter hash and a sequential hash, but no vertical value, then the vertical must be inferred from the student’s past actions. The server stores the most recent vertical a student was at for each section in the course. Thus, the processing code must keep track of the most recent vertical accessed for each section in the course, and when a row contains a direct course URL without a vertical, the vertical must be inferred from the previously stored most recent vertical for that section.

One other important note is that the rows of the original data file may not actually be in sorted, ascending order by time. In our processing, we found that while some rows seemed to be in ascending order, some rows were actually sorted in descending order.

Thus, each student is associated with a list of URLs they visited, processed from the original data log. There are a fixed number of possible course page URLs, which can be represented by the possible combinations of chapter, sequential, and vertical hashes. If there are 200 unique URLs in a course, then the indices from 1 to 200 can each correspond to one of the URLs. Once this mapping between index and URL is established, each student's set of actions can be represented as a list of indices.

Recommendation model design

This sub-section provides context to how the RNN and LSTM architectures function. RNNs maintain an ongoing latent hidden state that persists between each input to the model. This latent state can provide a representation of what has already been seen in the input sequence. Long Short-Term Memory (LSTM) is a modification of the RNN architecture, where the hidden latent state is replaced with a more powerful memory component. We chose to use LSTMs due to their stronger performance in modeling longer range dependencies [31] [32].

RNNs maintain a latent, continuous state, represented by $\boldsymbol{h_t}$ in the equations below. This latent state persists in the model between inputs, such that the prediction at $\boldsymbol{x_{t+1}}$ is influenced by the latent state $\boldsymbol{h_t}$. The RNN model is parameterized by the input weight matrix $\boldsymbol{W_x}$, recurrent weight matrix $\boldsymbol{W_h}$, initial state $\boldsymbol{h_0}$, and output matrix $\boldsymbol{W_y}$. $\boldsymbol{b_h}$ and $\boldsymbol{b_y}$ are biases for the latent and output units.

$$\boldsymbol{h_t = tanh(W_x x_t + W_h h_{t-1} + b_h)}$$
$$\boldsymbol{y_t = \sigma(W_y h_t + b_y)}$$

LSTMs, a popular variant of the RNN, augment the latent, continuous state with additional gating logic that helps the model learn longer range dependencies. The gating logic learns when to retain and when to forget information in the latent state. Each hidden state $\boldsymbol{h_t}$ is instead replaced by an LSTM cell unit with the additional gating parameters. The update equations for an LSTM are:

$$\boldsymbol{f_t = \sigma(Wf_x x_t + W_{fh} h_{t-1} + b_f)}$$
$$\boldsymbol{i_t = \sigma(W_{ix} x_t + W_{ih} h_{t-1} + b_i)}$$
$$\boldsymbol{C'_t = tanh(W_{Cx} x_t + W_{Ch} h_{t-1} + b_C)}$$
$$\boldsymbol{C_t = f_t \times C_{t-1} + i_t \times C'_t}$$
$$\boldsymbol{o_t = \sigma(W_{ox} x_t + W_{oh} h_{t-1} + b_o)}$$
$$\boldsymbol{h_t = o_t \times tanh(C_t)}$$

$\boldsymbol{f_i}$, $\boldsymbol{i_t}$, and $\boldsymbol{o_t}$ represent the gating mechanisms used by the LSTM to determine when to forget, input, and output data from the cell state, $\boldsymbol{C_t}$. $\boldsymbol{C'_t}$ represents an intermediary candidate cell state that is gated to update the next cell state.

LSTM Model Description and Training

LSTM models have several hyperparameters, which refer to values that affect how the model performs on a given set of data. Evaluating which hyperparameters work best for a given model and dataset can be done in one of several ways, and is usually resolved with some empirical experimentation. For this analysis, we varied the following hyperparameters: number of LSTM layers and number of hidden nodes per LSTM layer.

Each model was trained using either 1, 2, or 3 LSTM layers, as well as 64, 128, and 256 nodes per LSTM layer. Thus, each LSTM model is trained with 9 different hyperparameter sets.

To create a behavior prediction LSTM model, the model needs to be trained to predict the next URL given a prior sequence of URLs visited. This is our baseline LSTM model, where the inputs and outputs are simply indices corresponding to unique URL accesses. The model is trained in batches of 64 student sequences at a time using back propagation through time [37]. Categorical cross entropy is used to calculate loss and RMSprop is used as the optimizer. Drop out layers were added between LSTM layers as a method to curb overfitting [38]. An embedding layer with 160 dimensions is added to convert input indices to a continuous multi dimensional space, a technique commonly used in language modeling [15]. LSTM models were created using Keras [33], a Python library originally built on top of Theano [34].

Figure 13 details an example pipeline where the first two timesteps of a student sequence of URL accesses is shown. The two URLs in the student's sequence are converted to an index representation of that URL, which is then fed to the LSTM model. The index is implicitly converted to a one-hot vector representation by the embedding layer used by the Keras LSTM model. The output of the model uses the softmax function to normalize the outputs to sum to 1, so that the values within the output vector could be thought of as probabilities of that index being the predicted next URL. If there are 300 unique course URLs, for example, then the output vector would be of length 300, where each value of the vector corresponds to the probability that the next URL in the sequence will be that index value. Thus, to find the most likely next URL, one needs to find the index of the vector that has the maximum probability, and then consult the one to one mapping between indices and URLs to find which URL that index corresponds to. Note that in the example figure, index 32 of the softmax output in timestep 1 has the highest probability. Thus, according to the model, the most likely next URL would be the URL corresponding to index 32. In the example, this prediction turns out to be correct, as it is shown that the actual input in the next timestep is associated with that URL.

Incorporating time into the model

The previous subsection described a baseline LSTM model, where only the sequence of URL visits was modeled. We hypothesize that prediction accuracy of the next URL can go up if the model were to incorporate the amount of time spent on each resource. Unfortunately, there is no way to know exactly how much time the student is truly paying attention to a particular URL. We can approximate time spent, however, by calculating the time difference between each URL visit. Thus, we approximate the time spent on a URL by taking the time difference before accessing the next URL.

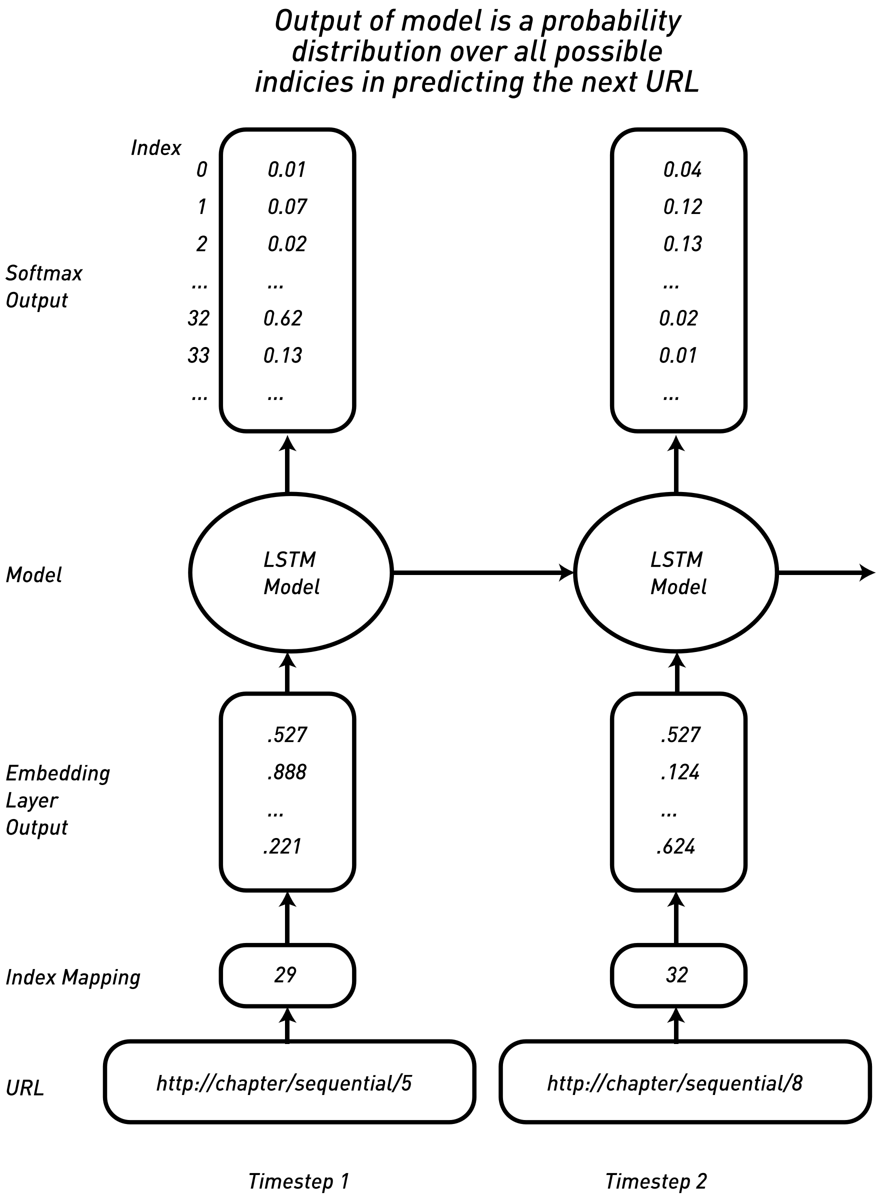


Figure 13 Depiction of baseline LSTM architecture

The baseline LSTM model [32] can be augmented to be able to incorporate time spent in addition to the standard input and output of the current and next URL index.

We propose two methods for incorporating time spent into the input of the model. These two methods are referred to as bucketed-time-input and normalized-time-input. These two methods of input are explained next.

Bucketed-time-input refers to an augmented input, where an additional one-hot vector is concatenated with the original baseline input. Figure 14 depicts this additional time input processing step in a graphical format. This additional one-hot vector indicates the amount of time spent on the resource relative to four pre-determined buckets: between

0-10 seconds, 11-60 seconds, 61-1799 seconds, and finally 1800 and beyond seconds. These buckets were chosen qualitatively, rather than with a data driven approach, to be able to prescribe real world interpretation to the time buckets.

Normalized-time-input refers to an augmented input where an additional two-dimensional vector is concatenated with the original baseline input. Figure 14 depicts how the normalized-time-input is incorporated into the architecture. The first dimension of this vector takes a value between 0 and 1, which is calculated by dividing the time spent on the resource by 1800, or if the time spent is greater than 1800, then the value is taken to simply be 1. Thus, a time spent of 900 seconds would be converted to 0.5. This is considered normalizing the time by 1800 seconds. The second dimension of the vector is simply 1 if the time spent is over 1800 seconds, and 0 otherwise.

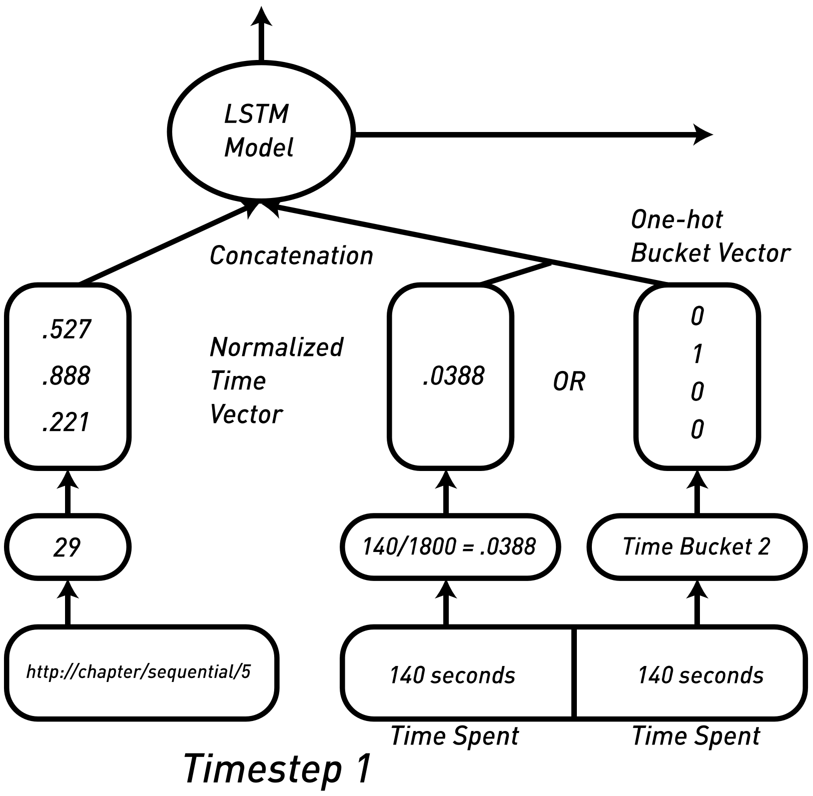


Figure 14 Depiction of two methods of adding dwell time to the model: Normalized continuous (0-1) and time bucketed (1, 2, 3, 4)

It is also possible to incorporate time into the output of the model. The non-time version is referred to as non-concatenated-output, while the time incorporated version is referred to as concatenated-output.

Non-concatenated-output refers to the standard output, where the object of prediction is simply an index, where each index has a one-to-one mapping with a course URL. Concatenated-output refers to an output space where the number of indices possible is multiplied by four, so that one could think of a time bucket being concatenated with each index. Each possible course URL now has four associated indices with it, where each index represents a course URL and the amount of time spent on that URL, where time spent is bucketed in the same fashion as the bucketed-time-input. We can compute the overall likelihood for a particular URL by adding the probabilities among all four indices associated with a particular URLs. Since each output is also associated with a time, one can look at only indices associated with a particular time bucket. Thus, the output can now be queried to find the most likely URL to visit among each possible time category.

With these methods of input and output defined, we propose the following models:

Attributes:

(a) Input time treated as continuous
(b) Input time treated as categorical
(c) Input time concatenation with vertical after embedding
(d) Time category concatenated with vertical in the output

1. Baseline LSTM model: Inputs and outputs are indices, where each index has a one to one mapping to a unique course URL.

2. Bucketed-time-input, non-concatenated-output (b,c)

3. Bucketed-time-input, concatenated-output (b,c,d)

4. Normalized-time-input, non-concatenated-output (a,c)

5. Normalized-time-input, concatenated-output (a,c,d)

Deployment Course Dataset and Prediction Results

The pilot course, DelftX Intro to Aeronautical Engineering 2015, contained log data from 27024 unique learner ids. However, for the purposes of behavior recommendation, we chose to filter the data to only include learners who attempted at least one problem check, resulting in data logs from 9,172 learners. From the data logs, we again filter the data to only include data regarding course page navigations, thus excluding events related to lecture video pausing, problem viewing, and so on. We chose to also filter out contiguous repeats of URL accesses. This means that if there are multiple visits to the same URL in a row, we removed duplicates such that there only remained one access to that URL for a student sequence representation. For the time spent associated with sole URL used in place of the duplicate contiguous URLs, we took the maximum time spent among the duplicated URL accesses. Time spent is calculated, in general, by taking the timestamp of a URL access and calculating the future difference to the timestamp of the next URL access in the sequence. There were 336127 navigation events in the 0-10 second bucket, 248918 in the 11-60 second bucket, 338144 events in the 61-1799 bucket, and 123287 events in the 1800 seconds and beyond bucket.

There was a total of 286 possible course URLs, which means there were 286 possible unique verticals to model, spread over 38 sequentials. The median number of verticals in a sequential was 6, with a maximum of 19. The course was self-paced, which means that assignment due dates were not fixed, and all of the course content was released at the beginning of the course. Log data was filtered to only include data from roughly the time period that the course officially ran, from May 31, 2015 to June 3, 2016.

Hill-climbing Validation Early Stopping

The 5 LSTM models described in the previous section were each trained under the 9 different hyperparameter settings described in section 3.1.4. The data was split into two sets, a training set and a held-out test set. The training set comprised sequences from a randomly selected 70% of the users, while the test set contained the remaining 30%. Within the training set, 10% of the sequences were held out as a hill-climbing validation set. During training of a particular model, if the loss calculated on the hill-climbing set did not obtain a best result for 3 consecutive epochs, then training was halted for that model and the best result was recorded. This was our early stopping criterion.

Baselines

An n-gram model is included as another sequential model for comparison. N-gram models capture the structure of sequences through the statistics of *n*-sized sub-sequences. The model predicts each sequence state x_i using the estimated conditional probability that x_i follows the previous *n-1* states in the training set. We trained n-grams with values of n between 2 and 10, while also instituting a "back-off" policy when there are too few subsequences. For each n-gram, we instituted back-off policies of between 0 and 10 occurrences, so that a particular sub-sequence of size n must occur at least the number of times as the back-off policy, or else that sub-sequence is not used. The back-off policy prevents the n-gram model from using very sparse data, requiring a minimum number of occurrences for that sub-sequence to be used. If there are too few occurrences of a particular n-sized sub-sequence, the model "backs off" and uses the values for the (n-1)-gram model, and so on. The best performing n-gram model on the validation set had an n-value of 7 and a back-off value of 8. A 2-gram model is also included, representing a model predicting the most common URL following a particular URL. We call this model the "Next most common" model. The last baseline is dubbed the "Next syllabus URL" model, which predicts the next URL in the course structure; this is equivalent to the page learners are taken to when they click on the "Next" button in the native edX interface.

4.10 Back-tested Prediction Results

Validation accuracy and test set accuracy is shown in Table 14. For each model, the hyperparameter set that reached the highest validation accuracy was used. Thus, for each LSTM model listed in the table, only the highest achieving hyperparameter set results are shown, where training stopped according to the early stopping rules described previously. Accuracy refers to average accuracy per student sequence; thus a next URL prediction accuracy is established per student sequence, and then the averages from all students are averaged together. For baseline outputs, the models produce an index which has a one to one mapping with a URL. Thus, if the most likely index produced by the model matches the actual next URL in the sequence, that is counted as a correct prediction within a student sequence. For concatenated outputs, the models produce an index which has a four to one mapping with a URL, meaning there are four possible indices that all correspond to the same URL, just with a different time spent predicted. For the purpose of accuracy, as long as the URL mapping of the index is correct, then the prediction is counted as correct. Thus, accuracy for concatenated outputs drops the time component from the output in calculating correctness.

Table 14 Prediction Accuracy Results

Model Input / Output	**Validation Acc.**	**Test Set Acc.**
Bucket /Non-Concat.	63.5	64.0
Norm / Concat.	62.6	63.5
Bucket / Concat.	63.0	63.3
Norm / Non-Concat.	62.9	63.3
Baseline LSTM	62.0	62.5
Best *n*-gram (7)	61.6	61.7
Next most common	55.1	55.6
Next syllabus URL	51.5	52.0

4.11 Real-Time Deployment

Recommendation Interface

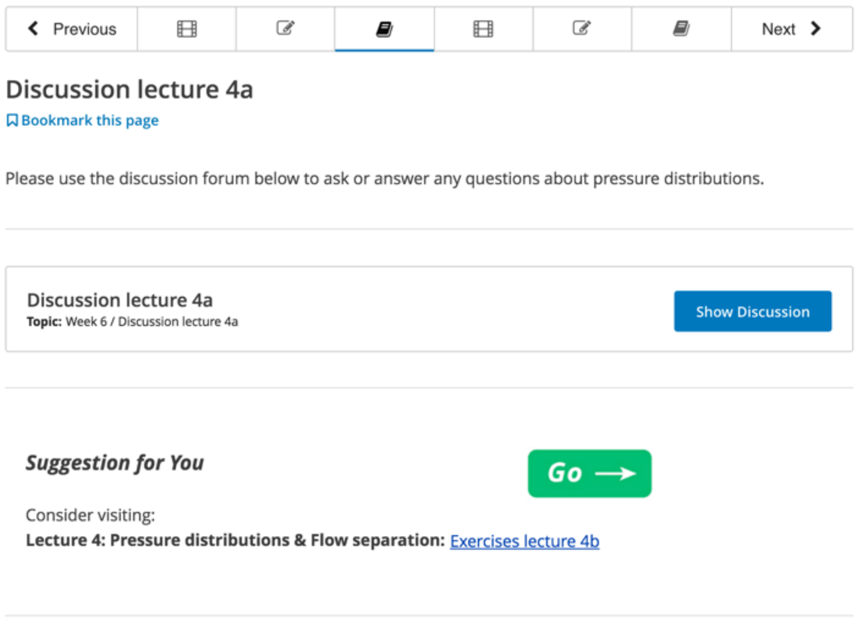
This section describes our rationale for how to best integrate the recommendations into the learner's course experience. We primarily consider two key aspects of the interface: (i) the visual appearance of the recommendations and (ii) the linguistic framing of the accompanying text.

Visual Appearance

As the interface is housed within the edX platform and course materials, it is important that the appearance of the recommendations is seamless. This ensures both a sense of trust from the user---in that it looks like it's a natural part of the edX course---and assuages the risk that the recommendations act as distractions to the learners. Given the simplicity of the edX user interface design, this was not hard to achieve. And to make following the recommendations more intuitive, we also add a "Go" button that learners can use as an alternative to clicking on the plain text link. These appear at the bottom of every page in the course---made directly available to the learner at all times.

Linguistic Framing

Just as we did not want the visual appearance of the recommendation to be too overwhelming in the existing course interface, we likewise aimed to present the accompanying text in a way that clearly communicates the benefit of this resource while not sounding overly authoritative. While definitely an avenue for future experimentation (what is the most effective way to frame such recommendation text to learners?), we eventually decided on "Suggestion for you… Consider visiting: [Recommended next-step]." This text accomplishes the task of communicating to the learner that this recommendation is indeed personalized and unique to him or her (without explaining how) and also making it clear that following this recommendation is optional. Figure 15 shows the final design of the recommendation interface.



Discussion lecture 4a

Please use the discussion forum below to ask or answer any questions about pressure distributions.

Discussion lecture 4a
Topic: Week 6 / Discussion lecture 4a

Suggestion for You

Consider visiting:
Lecture 4: Pressure distributions & Flow separation: Exercises lecture 4b

Figure 15 Final design of the recommendation interface

We are able to show the text of sequential and vertical being suggested through a lookup table we created from the course xml.

4.12 Model Usage Considerations

Training a model based on a previous offering of the course

Since the navigation behavior model proposed in this chapter is behavior and data driven, a requirement to deploy such a model in a live course is that behavior from the course must already exist. To perform our live case study, we selected a MOOC that had multiple offerings over time so that we could use behavior from a completed iteration of the course to train our behavior models. Since the model is trained on a specific structure of course URLs, the current iteration of the course should not deviate too much, preferably at all, from the iteration that the model was trained on.

To deploy our behavior model in the 2016 offering of the Aeronautics Engineering course, we trained on the behavior data from the 2015 offering of the course.

Taking into Account Changes in The Courseware

In our case study, the majority of the course structure was held exactly the same. However, the first chapter to the course was re-ordered. Our behavior model implicitly incorporates the ordering of the course in its predictions, so that any re-ordering of course content would adversely affect its prediction. Therefore, we chose to drop all events related to the first chapter of the course from both the training and the live recommendation data sets. This was deemed acceptable since the first chapter for this course was an introduction to the course staff and logistics.

Additionally, one unique URL was added to the current version of the course. Thus, the trained model has no knowledge or ability to recommend that URL. However, the actual recommendation code can be altered so that when it is detected that a student is near the new URL, the recommendation code can choose to temporarily suspend usage of the behavior model and either suggest going to the new URL directly or simply not suggesting a URL temporarily.

Any deviation in course structure from the training environment needs consideration in handling. Special recommendation logic must be put in place when the live version of the course differs in ways that the original model cannot account for.

Description of the Recommendation Engine

The machine learned model is the contact point between the underlying LSTM behavioral model and the code that serves a clickable link on the learner's browser. The LSTM model has been trained to produce an output that contains a probability distribution over all possible course pages. Our time-concatenated output LSTM models additionally contain time information in each of the output indices. With this time-concatenated probability distribution, it is reasonable to simply take the most likely page and serve that as the model's recommendation. With the time-augmented output, however, the recommendation engine can instead be configured to recommend a URL that the learner is likely to spend a significant amount of time on, for example between 10 seconds and 30 minutes. The hypothesis behind this logic is that if the model only expects the learner to spend fewer than 10 seconds on a resource, then it may be the case that the learner is trying to skip over it on her way to the eventual resource of interest. The recommender gives the learner the skip directly to that eventual resource of interest. It could be reasonable to recommend pages where the learner is expected to spend more than 30 minutes on; However, we chose not to include these as part of our recommendation engine configuration, since it could be possible that such a lengthy time spent on a page could really be indicative of a time-out event, where the learner has actually just left the page, potentially after consulting an ineffective page.

Another method for producing a recommendation could instead be to repeatedly query the behavioral model until the most likely page corresponds to a desirable time bucket, where each repeated query has a "hypothetical action" appended as the most recent event. For example, if a student is currently at a quiz page, then the behavioral model would be queried using that student's past behavior as well as the current quiz page. The time spent on the current quiz page is not known yet since the learner has not navigated away from that page at the time of the query. Thus, time spent on the current page must be approximated in some way; we use the modal time bucket as a place holder (11-60 seconds) and the real time spent is filled in after the next navigation event. The model then produces a probability distribution over time-concatenated indices, as usual. If the most probable page is in a desirable time bucket range, then the engine recommends that URL. However, if the most probable page is not in a desirable time bucket range, then instead of recommending this page, the engine temporarily appends it to the student's "hypothetical" path until a desirable time bucket recommendation is reached. Thus, through repeated querying of the model, eventually a page in the desirable time bucket range would be reached, and the engine would use this as the recommendation. The reasoning behind such a model would be, for example, to skip through many URL accesses that are under ten seconds (undesirably short time spent) and instead recommend a URL that the student would likely have eventually dwelled on. We refer to this as a forward-stepping process, where we create hypothetical forward steps to the model. The page used in the case shown in Figure 16 is *Video 1*, a recommendation which is inserted into the page after the query completes and which the student followed in this example. After the learner in the example visits *Video* 1, she is suggested to return to the quiz but instead navigates to *Text* 3.

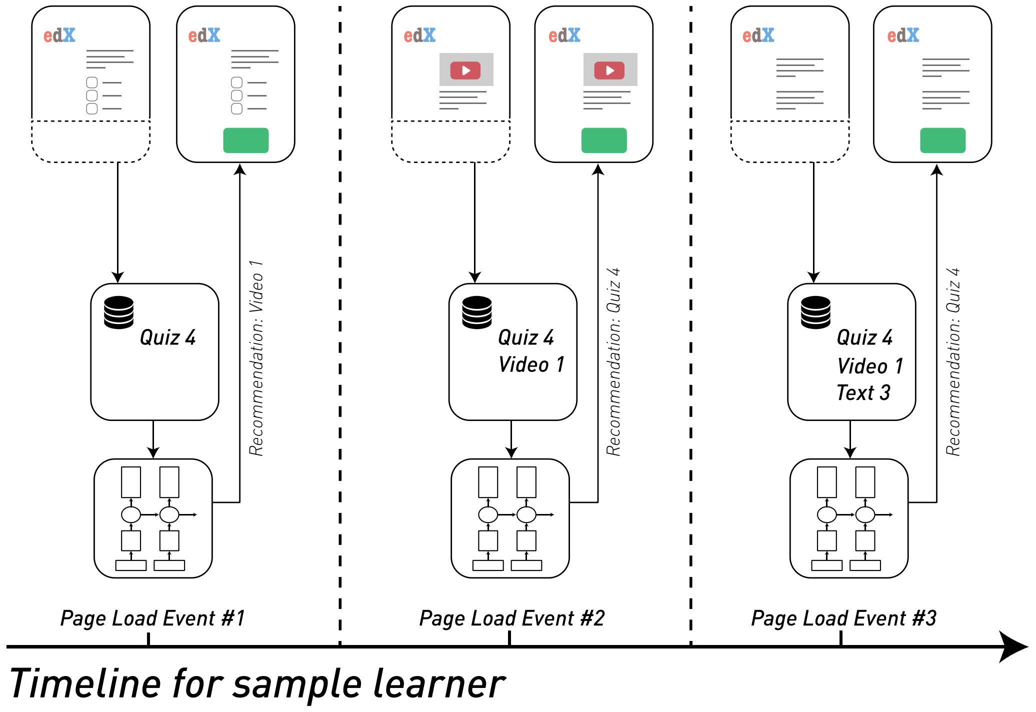


Figure 16 Example of framework delivering three recommendations at three consecutive page visits for a learner

We chose to use the time-bucketed-input with time-concatenated output model discussed previously. For the live recommendation pilot, we chose to retrain the LSTM model on the entire set of the previous course offering's data, as opposed to the original training only using 70% of the data. We only used the validation set's best hyperparameters.

Our next step recommendation can be seen as predicting what page a learner wants or will eventually want, and directly linking them to that page in advance. When to consider if what learners want is different from what they need to achieve their goals or the goals of the course is a matter for consideration by future work as well as the appropriate role of a platform, courseware, and personalization in facilitating these goals.

4.13 Contributions

In this chapter, we made three contributions to adaptive personalization in a MOOC. The first was to solve the issue of real-time learner event logging required for data-driven intervention with a client-side JavaScript solution that records learner navigational events. The second was to introduce a novel behavioral model which predicted the next page a learner was likely to spend significant time on which outperformed existing prediction baselines. Lastly, we combined the first two contributions to provide the first proof-of-concept realization of a real-time data-driven recommendation framework in a live MOOC along with the edge cases and design considerations that needed to be handled in order to deploy.

4.14 Acknowledgements

We would like to acknowledge our use of the edX partner’s Research Data Exchange (RDX) program and the support contributed by the edX data team. We also acknowledge the support from TU Delft’s Office of Online Learning, its director, and the DelftX Aeronautics Engineering course team for their cooperation.

5 Surveying Time Sensitive Behavior Modeling Across 99 MOOCs

5.1 Transitional framing within dissertation

The previous two chapters gave in depth case studies towards applying sequential modeling of student behavior in MOOC contexts. Those courses were handpicked for research since they seemed to have promising amounts of non-linear navigational behavior which could theoretically be modelled through a sequential model such as a time-augmented LSTM. In this chapter, I investigate to what extent the time-augmented LSTM (TLSTM) behavior model can model student behaviors across a much wider survey of edX MOOCs. edX provided the click-stream logs to 99 edX MOOCs administered in 2015 and 2016. This work is the first attempt to apply the TLSTM approach more broadly. This chapter investigates to what extent the TLSTM approach can generalize across MOOCs more broadly and seeks to find behavioral patterns and tendencies across many different kinds of MOOCs. This work was done in collaboration with my advisor, Zachary Pardos.

MOOCs have substantial opportunity for personalization. In this chapter, we investigate using a collaborative filtering methodology to find “non-linear” navigations in MOOC settings. In general, MOOCs follow a linear course syllabus, where students are expected to follow a straight path through the course. However, there is the case where students deviate from this established course structure. This chapter investigates a “Personalized Wayfinding Methodology” that learns from past behaviours of students navigating through a MOOC and seeks to provide possible recommendations for future students, in the hope that such a recommendation can help students navigate through the course more quickly as well as receive personalized support, which can help students stay motivated to stay in the course. We investigate the use of the wayfinding methodology on a survey of 99 edX courses administered in 2015 and 2016; these edX courses span a variety of subjects such as literature, computer science, physics, and math. These 99 courses are also administered from a variety of host countries and host universities, and span a wide range of enrolled learners. This chapter provides a more comprehensive survey of when such a methodology might or might not be successful and looks at potentially determining factors at the course and student population levels.

5.2 Introduction

Massive Open Online Courses (MOOCs), much like their residentially offered ancestors, were not constructed with a shared critical pedagogy. Instead, they are a digitization of their residential offered, mostly lecture based counterparts, some with elements of their in-class labs ported online. Also like their residential offerings, MOOCs were not designed with adaptivity or personalization in mind. Types of adaptivity brought to bear in most Intelligent Tutoring Systems have been prescribing the amount of practice based on performance, and model tracing tutoring, which can provide the student with adaptive hints based on their current state in the solving of a problem. Systems employing these approaches have been fruitful in seeing increases in student outcomes, moving median scores up 8 percentage points in high school algebra [55]. While the topics of MOOCs are diverse, among the courses offered on one of the major MOOC platforms, their data are stored in a uniform schema and with instructional material also sharing a single format between courses. This universality presents an opportunity to develop personalized and adaptive augmentations of the platform that could gracefully scale to all courses across the platform. This chapter continues work on using collaborative sources of personalization to overlay ITS like properties of adaptive hinting onto MOOCs. This hinting approach, based on hot linking to resources that learners with similar event histories spent time on, is ultimately a collaborative filtering approach, based on the hypothesis that learners have something to gain from the navigational patterns of previous cohorts.

Our "personalized wayfinding" methodology consists of comparing a student's actions and navigations to what similar students from the past performed to predict what the current student will navigate to next. The methodology consists of a "time spent" component, where the model can accept as input how much time the student is spending on each page and output a probability distribution over which course page a student will visit next as well as how much time the student will spend on that page. Such a methodology can underlie a personalized recommender system that seeks to optimize a student's navigational pathway through a MOOC. The basic idea is that not all students will navigate through a MOOC in a strictly forward path; some students may deviate more than others, and perform more backwards or forwards navigations. The personalized recommender can personalize an efficient path through the course while simultaneously providing a level of personalized support to the student, which can help replace the lack of personal instructor interaction that pervades online courses compared to in-person course interactions. Research has shown that personalized support, at the human tutor level, can provide substantial gains in testing outcomes [56], and this research is on the path towards higher levels of personalized tutoring.

5.3 Related Work

In this chapter, a collaborative filtering recommendation approach is applied to a survey of 99 edX MOOCs, spanning a variety of subjects and slight variations in presentations of learning content. This modeling approach is content agnostic and relatively easy to deploy. Prior research has clearly laid out that the population of learners that take MOOCs is diverse, and that different learners behave quite differently when interacting with MOOCs [57] [58]. This chapter extends research in the fields of both education recommenders, such as Intelligent Tutoring Systems, and in survey analysis of behavior trends in MOOCs.

Intelligent Tutoring Systems (ITS) [59] prescribe practice according to performance. In this chapter, a recommender is proposed where content is recommended based on navigational behaviors, including dwell time, which could be seen as overlaying ITS-like hinting mechanisms over a MOOC framework. Collaborative filtering [1] in education contexts refers to a methodology whose chief goal is to find a few items that are relevant to a user's interests in a large pool of documents. Collaborative filtering ignores content completely, instead attempting to match users who are interested in the same documents. An alternative approach would be content-based filtering, which inspects the content of resources to make recommendations. In this chapter, we focus on a collaborative filtering approach as a means towards a light weight and quick to deploy recommendation framework. However, content-based filtering approaches are another viable path forward towards personalized tutoring in MOOC contexts.

Adaptive and Intelligent Web-based Educational Systems (AIWBES) [60] refer to educational systems that seek to be adaptive and/or intelligent by building a model of the "goals, preferences and knowledge of each individual student and using this model throughout the interaction with the student in order to adapt to the needs of that student." In this chapter, a recommendation framework based solely on the navigations of students throughout their progress through a MOOC is proposed, whereby the recommender would ideally provide support for adaptive presentation, adaptive navigation support, and in some ways curriculum sequencing, in the sense that it is possible for different students to be potentially recommended different paths through the same set of linear content.

Personalized learning has been pursued from a variety of angles. There exist many knowledge-based personalization frameworks, including various ITS systems [61] [62] [63] as well as knowledge tracking systems based on item response theory [64]. Multiple general frameworks for deploying personalized recommendation in education or social contexts have been suggested [65] [66]. The breadth of research in personalized recommendation in education contexts and beyond is quite wide. In the realm of recommendation in MOOCs, research has focused on understanding how to better navigate forum contexts [67] and also on investigating the prediction or prevention of dropout [68] [69] [70].

Prior research has established that navigation patterns of students often vary quite significantly from the prescribed, linear, straightforward path prescribed by the instructor via the inherent course structure. In a study of four edX MOOCs from 2014, it was found that certificate earners skipped 22% of course content, certificate earners engaged in non-linear navigation behaviors such as jumping backward to revisit earlier lectures, and that age and country were significant predictors in finding non-linear navigation patterns [57]. This chapter analyzes a much larger sample of 99 edX courses from 2015-2016, and finds that some of these patterns still holds. These findings of non-linear navigations and different behaviors from different populations highlights the possibility of adding a touch of personalization and tutoring to online learning resources. Stereotype modeling [71] has been used to predict student problem-solving learning in programming courses and in MOOCs in order to address whether students in different groups learn differently. Survey data [58] has been used as a means to understand the different kinds of motivations students have for taking MOOCs in the first place; fewer than half of students in this survey indicated an intention to earn a certificate, showing that certificate earning is not necessarily a motivator for a majority of students. The survey data also indicated that students had a wide variety of motivations for taking MOOCs, once again highlighting the potential benefit for a personalized and adaptive recommendation framework, since students take MOOCs for a variety of different goals.

5.4 Dataset

Our data consisted of course content data, anonymized user info, and event tracking logs from 99 edX MOOCs administered from Fall 2015 through Fall 2016. Course content data included xml of the courses' content element; videos meta info, html, problem text and answer types, and course structure files (chapters, sequences, and verticals). The anonymized user info contained summative stats of the user in the course, such as their certification status and limited demographic info (age, country, sex). The event tracking logs, the primary focus of our studies, contained a variety types of interactions[3] between the learner and the courseware. These logs included information such as the correctness of problem submissions, answer text given, page navigations of learners, and time stamps. The MOOCs spanned both STEM and Non-STEM disciplines (Table 15) and were administered by 16 universities participating in the edX Research Data Exchange (RDX[4]), a program allowing for the mutual exchange of anonymized learner logs among participating edX partner institutions.

[3] http://edx.readthedocs.io/projects/devdata/en/latest/internal_data_formats/event_list.html#event-list

[4] http://edx.readthedocs.io/projects/devdata/en/latest/rdx/

Table 15 Titles of the 99 edX Courses in the Dataset

NONSTEM	**BerkeleyX**	Jane Eyre by Bronte: BerkeleyX Book Club	A Christmas Carol by Dickens: BerkeleyX Book Club
		A Room with a View by Forster: BerkeleyX Book Club	Adventures of Huckleberry Finn by Twain: BerkeleyX Book Club
		Call of the Wild by London: BerkeleyX Book Club	Frankenstein; Or, The Modern Prometheus by Shelley: BerkeleyX Book Club
		English Grammar and Essay Writing	The Picture of Dorian Gray by Wilde: BerkeleyX Book Club
		Academic and Business Writing	Solving Public Policy Problems: UC Berkeley's Eightfold Path
		How to Save Money: Making Smart Financial Decisions	
	BUx	The Art of Poetry	AP® Spanish Language and Culture
	DARTx	The American Renaissance: Classic Literature of the 19th Century	
	DelftX	Building With Nature	Responsible Innovation: Ethics, Safety and Technology
		Leadership for Engineers	Creative Problem Solving and Decision Making
		The Next Generation of Infrastructure	IMAGE | ABILITY - Visualizing the Unimaginable
		Open Government	
	GeorgetownX	The Divine Comedy: Dante's Journey to Freedom, Part 1 (Inferno)	
	HKUSTx	English for Doing Business in Asia - Speaking	Making Sense of News
	Pennx	Intellectual Property Law and Policy: Part 1	Analyzing Global Trends for Business and Society
		Intellectual Property Law and Policy: Part 2	
	TsinghuaX	China's Perspective on Addressing Climate Change	Introduction to Mao Zedong Thought
		Chinese History From Warring States to the Tang Dynasty	Introduction to Psychology
		Chinese History From Warring States to the Tang Dynasty	Geology and Engineering Geology (offered twice)
		Relics in Chinese History - Part 1: Agriculture and Manufacturing	User Experience (UX) Design: Human Factors and Culture in Design
		Relics of Chinese History - Part 2: Astronomy and Medicine	Will China Rise as a Disruptive Force? The Insiders' Perspective
		Relics of Chinese History - Part 3: Writing System, Rites and Music	
	TorontoX	Death 101: Shaping the future of global health	
STEM	**BerkeleyX**	The Beauty and Joy of Computing (CS Principles), Part 3	Engineering Software as a Service, Part 2
		The Beauty and Joy of Computing (CS Principles), Part 4	Electronic Interfaces
	BUx	Force and Motion: Pedagogical Content Knowledge for Teaching Physics	AP® Physics 1
		Linear Differential Equations	
	DelftX	Advanced Credit Risk Management	Introduction to Drinking Water Treatment
		Introduction to Aeronautical Engineering	Data Analysis: Visualization and Dashboard Design
		Pre-University Calculus (Self-Paced)	Topology in Condensed Matter: Tying Quantum Knots
	EPFLx	Electrotechnique II	Introduction à l'astrophysique
		Space Mission Design and Operations	Plasma Physics and Applications
		MATLAB et Octave pour débutants	
	GeorgetownX	Preparing for the AP Physics C: Electricity & Magnetism Exam	
	HKUSTx	Introduction to Java Programming - Part 1	Introduction to Mobile Application Development using Android
		Introduction to Java Programming - Part 2	A System View of Communications: From Signals to Packets (Part 3)
	IITBombayX	Programming Basics	Introduction to Computer Programming, Part 1
	KIx	Behavioral Medicine: A Key to Better Health	Pragmatic Randomized Controlled Trials in Health Care
		eHealth – Opportunities and Challenges	
	RiceX	AP® Biology - Part 3: Evolution & Diversity	AP® Physics 1 - Part 3: Electricity and Waves
		AP® Biology - Part 4: Ecology	AP® Physics 1 - Part 4: Exam Prep
		AP® Biology - Part 5: AP Exam Review	AP® Physics 2 - Part 3: Optics and Modern Physics
		Preparing for the AP Calculus AB Exam - Part 2: Integrals	AP® Physics 2 - Part 4: Exam Review
		AP® Environmental Science - Part 3: Pollution	Proteins: Biology's Workforce
		AP® Environmental Science - Part 4: Exam Review	DNA: Biology's Genetic Code
		AP® Physics 1 - Part 2: Rotational Motion	Fundamentals of Immunology, Part 2
	SNUx	Introduction to Economics: Microeconomics	
	TsinghuaX	Principles of Electric Circuits (offered twice)	Combinatorial Mathematics
		Finite Element Analysis and Application (offered twice)	Water and Wastewater Treatment Engineering:Physicochemical Technology
		Data Structures and Algorithm Design Part I	Electromagnetism
		Data Structures and Algorithm Design	Financial Analysis and Decision Making (offered twice)
		Water and Wastewater Treatment Engineering: Biochemical Technology	Hydraulics
	UCSDX	Computer Graphics	
	TorontoX	Behavioural Economics in Action	

5.5 Processing edX click-stream logs

Many learners merely sign-up for a MOOC but do not enter and interact with its content. In our analyses, we focus on a subset of enrolled learners exhibiting a minimum engagement with the courseware. We defined this minimum engagement to be an attempt to answer at least one assessment problem. Only this subset of learners were included in the studies presented, unless otherwise stated. There was a total of 427,969 learner-course pairs in the dataset, 171,791 of which satisfied the problem attempt criterion. The 171,791 logged a total of 12,044,919 navigational events. A navigational event was defined as viewing a page of the courseware. This could be initiated by clicking the “Next” or “Previous” buttons related to the current sequence, directly clicking on a page in the sequence navigation bar, clicking on a different sequence within the same chapter, clicking on a different chapter, or simply typing a direct url address to access a different page. Each of these navigational actions was mapped to a page in the courseware, called a “vertical.” Using the nomenclature of edX courseware, the course is organized into a hierarchy of chapters, sequentials, and verticals. Thus, there might be five chapters in a course, each containing its own set of “sequential” sections. Each of these sequential sections is comprised of “verticals,” with verticals containing assessment, video, or text content. When we refer to the course syllabus, it is the collection of vertical pages indexed in the order prescribed by this hierarchical structure of the course. On average, there were 1,735 learners and 158 verticals per course, 65 of which were videos, 52 problems, and 41 text (book) type vertical content.

```
{
    username: "username_123123",
    context: {
        course_id: "BerkeleyX/BJC.3x/1T2016",
        org_id: "BerkeleyX",
        user_id: 123123
    },
    event_source: "browser",
    name: "seq_next",
    agent: "Mozilla/5.0 (X11; CrOS x86_64 7520.62.0) Apple
WebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.80 Safari/537.36",
    event: {
        new: 5,
        old: 4,
        id:
"i4x://BerkeleyX/BJC.3x/sequential/ed28b5e6ccba45dd8dfeac59303d13f0"
    },
    session: "abcabc",
    accept_language: "en-US,en;q=0.8",
    time: "2016-12-01T01:01:01.100100+00:00",
    augmented: {
        country_code: "US"
    },
    event_type: "seq_next"
}
```

An example event of type “seq_next”

```
{
    username: "username_123123",
    context: {
        course_id: "BerkeleyX/BJC.3x/1T2016",
        course_user_tags: { },
        user_id: 123123,
        org_id: "BerkeleyX"
    },
    event_source: "server",
    event_type:
"/courses/BerkeleyX/BJC.3x/1T2016/courseware/275dd5728c1f46ecbd748f8a8
eabc3f5/",
    agent: "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3)
AppleWebKit/601.4.2 (KHTML, like Gecko) Version/9.0.3 Safari/601.4.2",
    accept_language: "en-us",
    time: "2015-12-01T01:01:01.100100+00:00",
    event: "{}",
    augmented: {
        country_code: "US"
    }
}
```

An example “slash” event type

Extracting all navigation events required additional parsing from what was found in the edX event tracking logs. Learners clicking on next, previous, or an arbitrary vertical within a sequential resulted in log entries with clearly defined event types of, “seq_next, seq_prev, and seq_goto.” We mapped these events to the verticals they were referring to, including the edge cases where a seq_next was intended to lead to the first vertical in the next sequence. All other types of navigation events manifested themselves as resource paths, similar to URLs, in the “event_type” field. These events were colloquially referred to as “slash events.” Both seq and slash type events were resolved to the course page vertical as part of our pre-processing with examples of both json events depicted above.

5.6 Anatomy of edX MOOC

The general interface components of an edX MOOC are shown in a screenshot of a DelftX Aeronautics course in Figure 17, representing how the interface would have been presented to learners during the period in which our data were collected, 2015-2016. The left side of the interface displayed a list of chapter headings (A), each of which could be expanded with a click to show the names of the sequences (B) they contained. Within a sequence were verticals (D). Learners could navigate to the “Previous” or “Next” vertical in the sequence by clicking the navigational arrow elements (C), directly clicking on a vertical in the sequence (D), or by arriving at the vertical through other direct means, such as entering the URL of the vertical into the browser or via a save bookmark (E).

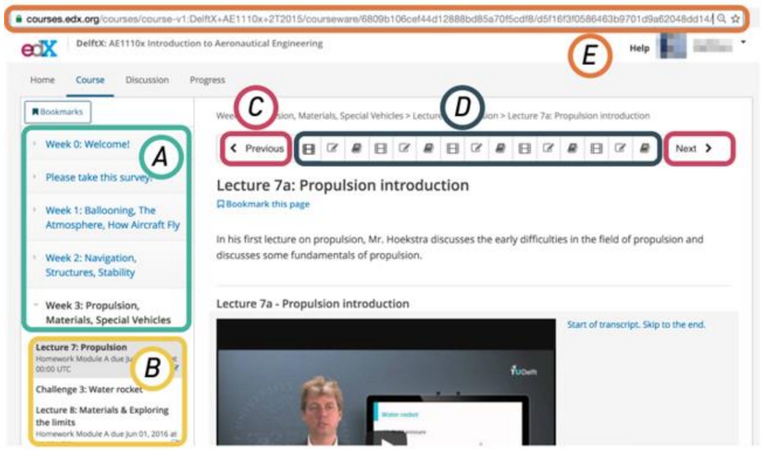

courses.edx.org/courses/course-v1:DelftX+AE1110x+2T2015/courseware/6809b106cef44d12888bd85a70f5cdf8/d5f16f3f0586463b9701d9a62048dd14
DelftX: AE1110x Introduction to Aeronautical Engineering
Help
E
Home
Course
Discussion
Progress
Bookmarks
C
D
A
Week 0: Welcome!
Please take this survey
Week 1: Ballooning, The Atmosphere, How Aircraft Fly
Week 2: Navigation, Structures, Stability
Week 3: Propulsion, Materials, Special Vehicles
Previous
Next
Lecture 7a: Propulsion introduction
Bookmark this page
In his first lecture on propulsion, Mr. Hoekstra discusses the early difficulties in the field of propulsion and discusses some fundamentals of propulsion.
Lecture 7a - Propulsion introduction
Start of transcript. Skip to the end.
B
Lecture 7: Propulsion
Challenge 3: Water rocket
Lecture 8: Materials & Exploring the limits
Homework Module A due Jun 01, 2016 at

Figure 17 Screenshot of 2016 edX MOOC

edX verticals are comprised of video lectures, problem sets, readings, and links to the discussion forum. These components are embedded in the vertical by way of the same is-a-part-of hierarchical structure that defines the chapter, sequential, vertical relationships. We model navigations at the vertical level, instead of the more granular component level, because verticals are the level at which learners navigate the course and thus the level at which navigational recommendations can be made.

5.7 Additional Dataset Descriptives

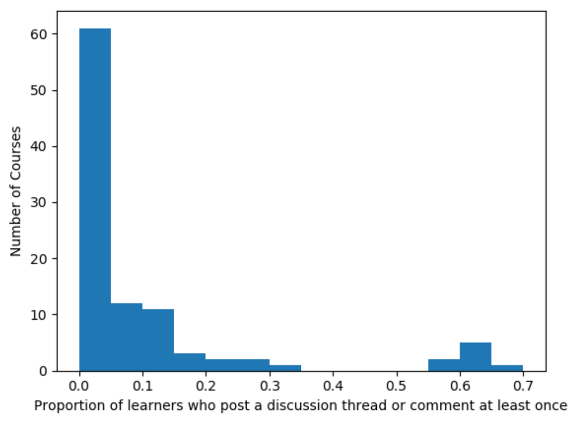
Our modeling of learner pathways utilizes only the navigation events from the event logs, however, we were interested in investigating the course attributes and contexts most amenable to this paradigm of models. Table 16 summarizes host country of MOOC and the number of learners by course host country. In this dataset, there are 41 MOOCs that come from institutions based in the United states. However, the 32 MOOCs that originated from institutions in China had the highest number of learners who attempted at least one problem in the MOOC. The remaining MOOCs come from institutions based in the Netherlands, Switzerland, Sweden, Canada, India, and South Korea. Across the 99 courses, there are 171,791 learners who attempted at least one problem check, and a total of 427,969 learners who logged at least one action in the MOOC during the timeframe in which the MOOC was scheduled to originally run. For many of the MOOCs, users can still audit and view course content, even outside of the time window in which the course was intended to be run. In this chapter, we analyze the behaviors of students who completed at least 1 problem, and only their actions that were performed during the timeframe of the course (generally around 5-10 weeks long) are included in analysis.

Table 16 MOOC Host Country Statistics

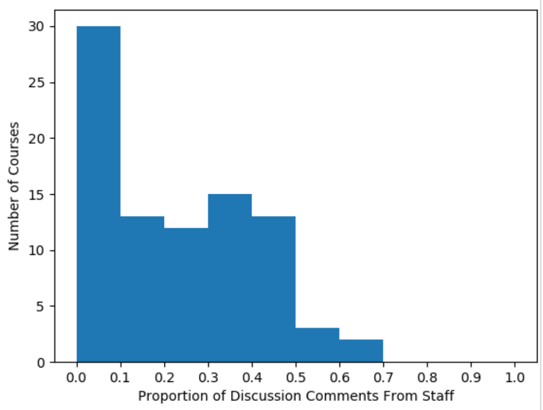
COUNTRY	NUMBER OF MOOCS	LEARNERS WHO ATTEMPTED AT LEAST 1 PROBLEM	ALL LEARNERS WHO SIGNED UP FOR COURSE
UNITED STATES	41	57,421	190,003
CHINA	32	60,428	137,399
NETHERLANDS	13	38,395	67,651
SWITZERLAND	5	3,563	9,467
SWEDEN	3	2,327	5,580
CANADA	2	3,583	9,623
INDIA	2	5,300	6,517
SOUTH KOREA	1	774	1,729
TOTAL	99	171,791	427,969

Figure 18 shows two trends of discussion usage. Figure 18A shows the proportion of learners (who attempt at least one problem) who also use the discussion forums at least once. Around 60 of the MOOCs had fewer than 5% of learners make a post at least once. 9 of the MOOCs had at least half of all users posting a discussion post at least once. All 9 of these MOOCs were from the Berkeley College Writing group of courses. This indicates that it was likely a requirement for learners to post on the discussion board as part of the learning experience. Figure 18B shows what proportion of all comments made in discussion environments were from staff members. This gives a degree of how much staff involvement there is in the discussion forums. Around 30 MOOCs had fewer than 10% of discussion comments originate from staff origin. The median proportion of staff discussion comments was .215.

Figuring out which of the users from the provided click-stream logs were actually staff users required parsing the *user_course_roles* field from each row's *event* attribute. The vast majority of rows are either *Student* or simply do not contain a *user_course_roles* attribute; the user_ids associated with these rows can therefore be considered student roles. The remaining roles were *beta_tester, instructor,* and *staff*. Thus, user_ids associated with rows that contained the role *instructor* or *staff* were considered staff ids.



(A) Proportion of users who make a discussion post at least once

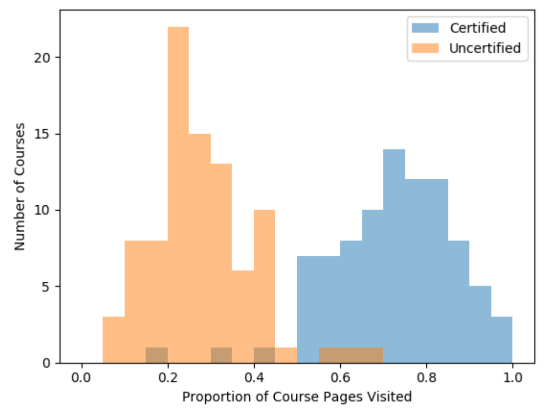


(B) Proportion of Discussion Comments made by Staff over all MOOCs (median of .215)

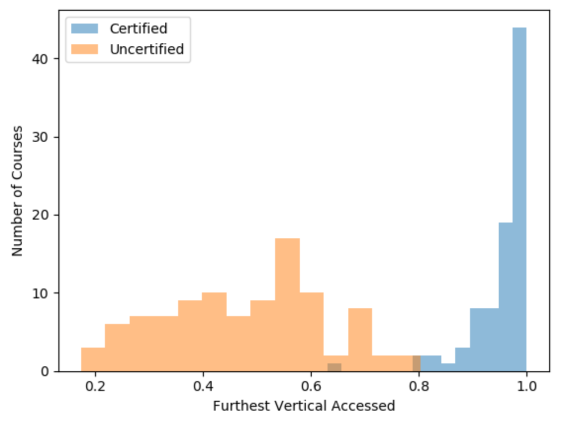
Figure 18 Discussion Comment Usage From Both Learners and Staff

Figure 19 shows two histograms. In Figure 19A, the average proportion of course pages visited by both certified and uncertified learners is shown. The certificate earning learners clearly visit a higher proportion of course pages on average. The distribution mostly ranges from 0.5 to 1.0. Conversely, the distribution for uncertified learners ranges mostly between 0.1 to 0.45 proportion of course pages visited. In Figure 19B, the furthest course vertical accessed (normalized to between 0 and 1) by both certified and uncertified learners is shown. For example, if there are 200 possible course page verticals in a course, then a student who accesses the 180th course page (ordered by the given course syllabus) at least once but does not visit any pages after that page would have his furthest vertical accessed proportion be 0.9. Thus, this is a measure of how far into a course learners get. On average, certified learners access very close to the end of the course, with over 40 courses in the highest possible furthest vertical accessed bucket. The furthest vertical accessed for uncertified learners varies widely by course, where the distribution stretches from 0.2 on average to all the way up to 0.8. Learners who end up certified generally visit the latter parts of courses, whereas uncertified learners might stop only halfway on average.

Taking both Figure 19A and Figure 19B together, certified learners are generally reaching the end pages of a course, but do not necessarily visit every page in the course, indicating a degree of skipping course content. Uncertified learners might be both skipping and stopping early.



(A) How Many Course Pages Certified and Non-certified Learners Visit



(B) Furthest Course Vertical Accessed by Certificate Earners

Figure 19 Course Page Access Separated by Certified and Non-Certified

Figure 20 shows a stacked bar chart for all 99 MOOCs in our dataset. These bar charts show the distribution of vertical types. There are "Problem" verticals, "Video" verticals, and "Text" verticals. These titles refer to the type of content displayed on a particular course page vertical. In the event that a vertical has more than one content type, such as having both a video and a problem set, we follow the standard chosen by the edX web staff. If a vertical any problem questions on the page, the vertical is considered a Problem vertical. Then, if there is no problem but there is a video anywhere on the vertical, then that vertical is considered a Video vertical. Finally, if there is neither a quiz item nor a video, then the vertical is considered Text. Along the x-axis is a label for whether the particular course is Non-STEM or STEM. The chart is ordered by increasing quantities of Problem vertical percentage. MOOCs vary in the percentage of verticals that are Problem verticals, all the way from as few as 3% up to around 60%. Interestingly, it seems to be the case that STEM courses generally have a higher percentage of problem verticals compared to Non-STEM courses.

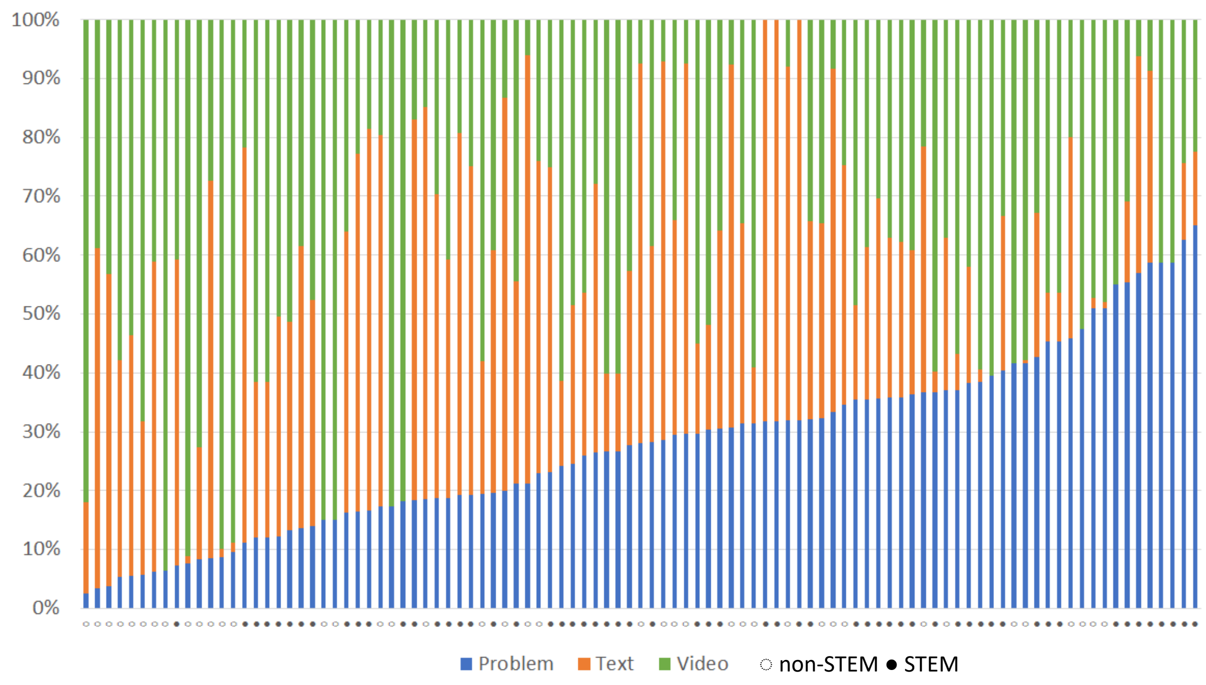


Figure 20 Proportion of Verticals by Content Type

5.8 Methods for personalized wayfinding

The proposed collaborative-filtering based wayfinding model needs to be a sequential model that is capable of modelling sequences of navigational behaviors of arbitrary length. A navigational behavior is defined as accessing a particular course page. Thus, the model needs to be able to accept a sequence of course pages that a student has already visited, and produce a probability distribution over all possible course pages that the student will visit to next. A recurrent neural network (RNN) architecture is suitable for this approach [72]. RNNs have been successfully applied to other sequential modelling tasks, such as language modeling [73], which seeks to predict the next word given a sequence of previous words. The sequential behavior model can be augmented to also take as input "dwell time" [74] or the amount of time that the student has spent on the page. Thus, RNNs are well suited for this task, as they can find general patterns of behavior in each student's sequence as well as seamlessly handle a dwell time input. In particular, the Long Short-Term Memory (LSTM) variant of RNN is used, as LSTMs have shown better success in retaining longer range dependencies in modelling sequences [75].

The input to the LSTM model consists of the history of courseware URLs that the student has visited. The output of the model consists of a probability distribution over all possible courseware URLs. Additionally, dwell time can be augmented to both the input and the output of the model. Thus, the model outputs a probability for every possible next courseware URL navigation and its associated dwell time, and these probabilities sum to 1. In this chapter, accuracy is measured as "recall @ 1," meaning that a prediction is considered correct if the highest probability course page predicted in the output (regardless of time category) is the actual next course page visited by the student. This style of sequential modeling, whereby the input is a long sequence of previous elements and the output is a probability distribution over all possible elements, is analogous to the approach used in language modeling [75].

An RNN uses all the same components of a simple feed-forward neural network, consisting of an input layer, an arbitrary number of hidden layers, and an output layer. The difference is that this structure is repeated for every time slice of the model, dictated by the maximum length sequence among the learners. The nodes in the hidden layer(s) of the network are all-connected to the hidden nodes in the hidden layer of the next time slice of the model. The weights of the network, including the weights associated with the between time slice connections, are shared in every time slice of the model.

Formally, RNNs maintain a latent, continuous state $\boldsymbol{h_t}$. The RNN model is parameterized by the input weight matrix $\boldsymbol{W_x}$, recurrent weight matrix $\boldsymbol{W_h}$, initial state $\boldsymbol{h_0}$, and output matrix $\boldsymbol{W_y}$. $\boldsymbol{b_h}$ and $\boldsymbol{b_y}$ are biases for the latent and output units. $\boldsymbol{\sigma}$ and ***tanh*** are sigmoidal squashing functions that compress input, representing the sigmoid and hyperbolic tangent functions respectively.

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b_h)$$
$$y_t = \sigma(W_y h_t + b_y)$$

Long Short-Term Memory (LSTM) [75] is a popular variant of the RNN, whereby the hidden memory unit is augmented with additional gating logic that helps the model learn about long range dependencies. This gating logic learns when to retain and when to forget information in the latent state. Each hidden state $\boldsymbol{h_t}$ is replaced by an LSTM cell unit that maintains the additional gating parameters.

$$f_t = \sigma(Wf_x x_t + W_{fh} h_{t-1} + b_f)$$
$$i_t = \sigma(W_{ix} x_t + W_{ih} h_{t-1} + b_i)$$
$$C'_t = \tanh(W_{Cx} x_t + W_{Ch} h_{t-1} + b_C)$$
$$C_t = f_t \times C_{t-1} + i_t \times C'_t$$
$$o_t = \sigma(W_{ox} x_t + W_{oh} h_{t-1} + b_o)$$
$$h_t = o_t \times \tanh(C_t)$$

$\boldsymbol{f_i}$, $\boldsymbol{i_t}$, and $\boldsymbol{o_t}$ represent the gating mechanisms used by the LSTM to determine when to forget, input, and output data from the cell state, $\boldsymbol{C_t}$. $\boldsymbol{C'_t}$ represents an intermediary candidate cell state that is gated to update the next cell state. $\boldsymbol{h_t}$ and $\boldsymbol{C_t}$ are the memory components that are used to propagate information between timesteps and inputs, thus passing information to future predictions in the sequence.

LSTM models contain several hyperparameters that affect how the model performs on training data [76]. For these analyses, the number of LSTM layers and the number of nodes in each layer were varied. LSTM layers varied between 1 and 2, the number of nodes per layer were 64 or 256, and Adam was used as the optimization method [77]. This indicates that a total of 4 different hyperparameter sets were fit per dataset. These hyperparameter sets were informed by previous iterations of grid searching on similar datasets in prior work [74] [72]. For each MOOC dataset, 30% of students' actions were held out as a test set. Of the 70% of the student data used as the training set, 10% was held out as a hill-climbing set. When the calculated loss on the hill-climbing set did not improve for 3 consecutive epochs, training was stopped and the learned weights were saved; this process is known as "early-stopping," as a method to efficiently account for over-fitting.

The set of hyperparameters that had the highest accuracy on the hill-climbing set was then saved, and the model was then tested on the 30% held-out test set. Thus, of the 4 hyperparameter sets, only the best performing set according to hill-climbing accuracy was tested on the 30% held out set. This process was repeated for both the vanilla LSTM approach and the time-augmented LSTM approach [74].

5.9 Results – Evidence of Skipping and Non-Linear Navigation

Across the 88 courses that had at least 5 learners receiving certificates, certified learners accessed 72.11% of course content on average, indicating that certified students completely skipped approximately 28% of course content. This finding is in line with the numbers from the Guo research [57], indicating that the patterns of skipping course content behavior found in 2012 can still be found in 2015/2016 MOOC data. In fact, it seems that students are skipping even more content than before. Learners, as real people with time constraints, are concerned with making most of their limited time and will selectively target MOOCs and parts of MOOCs that specifically address their current needs. 100% course completion may not necessarily be a goal of students [58] or even course designers; students may decide to skip content that is outside of their interest, either because they already know the content, feel that the content is way over their heads, or they are just not interested in those details of the course. Students may also decide to stop accessing materials as soon as they achieve certification. There are a variety of plausible explanations into why course completion, even among students who receive certification, is only 72% on average. Given that almost 30% of course content is skipped, a personalized recommender may make navigating course content more efficient for students who want to receive certification, skipping content they are uninterested in.

On average, certificate earners are skipping 28% of course content. This begs the question of which verticals students are skipping, and what do access patterns look like for courses that have varying amounts of skipped content? Some courses have certificate earners skipping much more content (as much as 85% of the course on average), while other courses have content-skipping rates of under 10% of content. In this section, we take a look at 1 course that had significant content-skipping, 1 course that had an average amount of content-skipping, and 1 course that had a very low content-skipping rate.

Table 17 Snapshot of 3 Courses

	Intro to Psych	Intro to Aero. Engineering	English for Doing Business in Asia
Host University	Tsinghua	Delft	HKUST
Host Country	China	Netherlands	China
Average Percentage of Verticals Accessed by Cert. Earners	33%	71%	90%
Average Percentage of Content Skipped by Cert. Earners	67%	29%	10%
Certificate Earners	128	502	124
STEM Course?	No	Yes	No
Instructor Paced?	No	No	Yes
Learners with 1+ Problem Checks (PC)	447	9196	1,874
Proportion of 1+ PC Learners That End Up with Certificate	28%	5%	7%
Navigations Logged	13,002	886,533	157,694
Syllabus Model Accuracy	.412	.545	.549
Next Most Common Model Accuracy	.589	.561	.551
TLSTM Model Accuracy	.631	.644	.576
%TLSTM Gain Over Syllabus Model Accuracy	52.94%	17.98%	5.06%
Chapter Count	14	13	11
Sequential Count	69	41	47
Vertical Count	69	291	154
Problem Verticals	12	89	57
Video Verticals	57	104	57
Book/Discussion Verticals	0	98	40
Total Individual Problems	13	332	205
Total Forum Comments	99	2941	1,746
Total Forum Comment Threads	26	3,528	162
Forum Comments Made by Staff	0	112	100
Forum Threads Made by Staff	1	1	12

Figure 21 – Certificate Earners Accessing Course Pages At Least Once - 3 Separate Cases

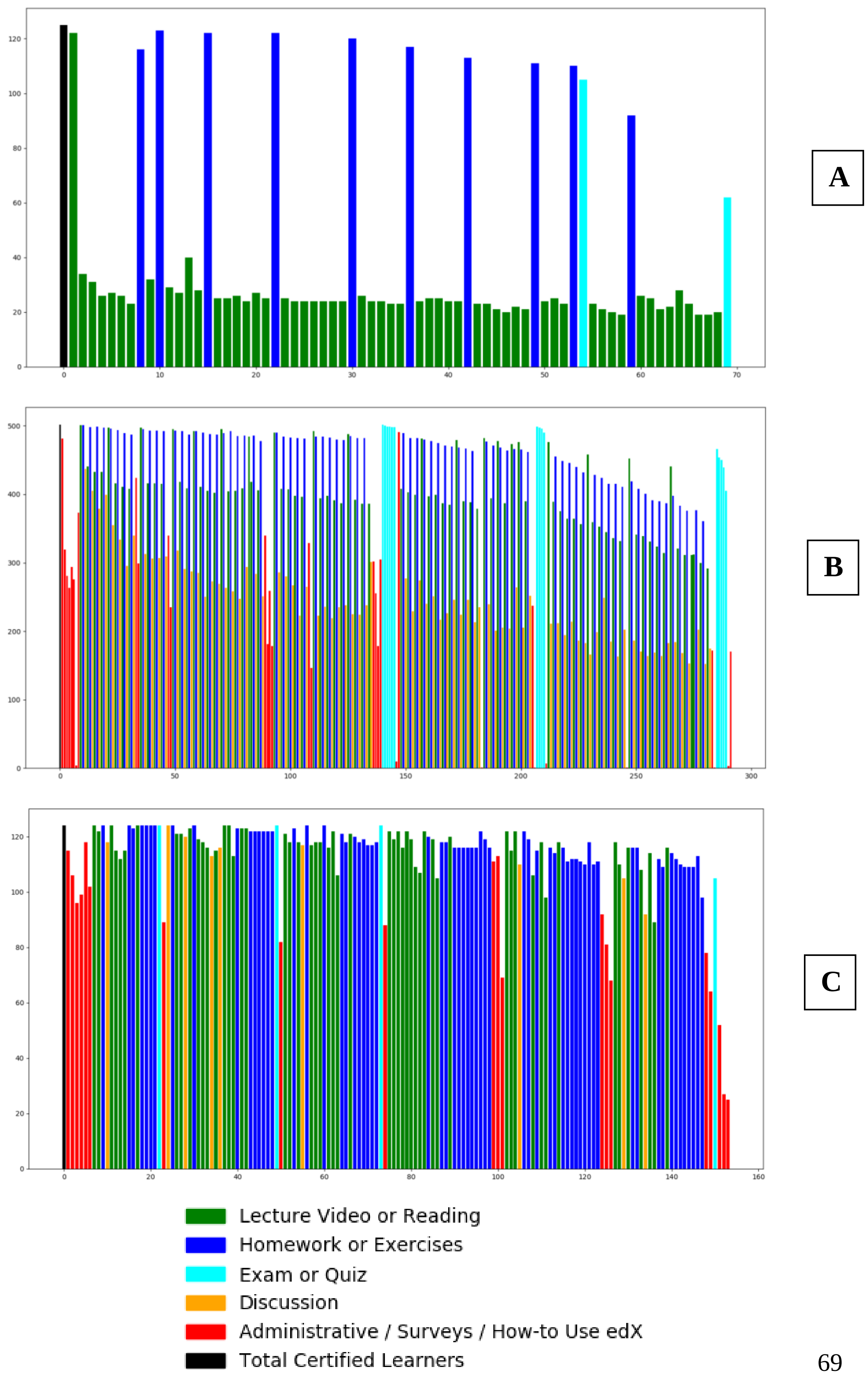


Figure 21 contains 3 graphs that show how often each course page was accessed at least once for each of the 3 courses selected. Graph A refers to Tsinghua 30700313x Introduction to Psychology, Graph B refers to Delft Aeronautics Engineering, and Graph C refers to HKUSTx EBA101x, English for Doing Business in Asia – Speaking. The color legend is located at the bottom of the figure.

For Graph A, it is clear that some course pages were accessed much more frequently than other pages. With the color coding, it is clear that the blue and cyan verticals were accessed much more often than the green verticals. From the legend, it can be seen that the blue and cyan verticals are for homeworks and exams, while the green verticals are lecture content. In this course, certificate earners only accessed 32.57% of course content on average. This course showcases a behavior pattern of only accessing homework assignments. Thus, certificate earners are skipping large portions of the course. Interestingly, the final exam seems to only be accessed by roughly half of certificate earners, indicating that it is possible to achieve a certificate without accessing the final exam.

Graph B showcases more typical access patterns. Certified learners accessed 71.85% of all possible course verticals on average, which is close to the mean among all MOOCs. Once again, the blue and cyan colored bars are the most commonly accessed verticals. Next, the green verticals are accessed less than the blue verticals, but are generally accessed more often than the orange and red verticals. This shows that nearly all certified learners are accessing the homework/exercise verticals, and that there still exists a portion of learners that seem to generally skip verticals that are not exercise verticals. Thus, the skipping behaviors that are seen in graph A can still be seen in graph B, but to a lesser degree. Finally, the discussion verticals (in orange) are often skipped over, with many certificate earners choosing to completely skip these pages.

Graph C shows the behavior patterns from a course that had a relatively high average proportion of course content visited at least once, at 90.13% of accessed content on average by certified learners. Most verticals are accessed by the vast majority of certificate earners. The verticals that were sometimes skipped by a portion of the class were generally related to administrative course pages, such as “Weekly Outline” verticals and “Grading Rubric / Submission Walkthrough” verticals. These access patterns generally show a large amount of course coverage by most learners, with dips on the administrative verticals.

5.10 Results – Model Performance Compared

There are 4 models of behavior discussed in this section. Table 18 describes how each model makes predictions.

Table 18 Sequential Model Descriptions

Model Name	**How the model makes predictions**
(S) Syllabus	Next page will be the next page in the syllabus
(NMC) Next Most Common	Next page will be the most commonly visited next page according to the behaviors in the training set of data
(LSTM) LSTM	Next page will be predicted by a LSTM network trained on behaviors from the training set of data
(TLSTM) Time-augmented LSTM	Next page will be predicted by a LSTM network augmented with dwell time buckets in both the input and output from behaviors from the training set of data

The ultimate goal of such sequential modeling is to develop a personalized recommender system in which learners can be recommended resources that they can be expected to spend a large amount of time on. Thus, ideally, the TLSTM model performs the best in predicting the next action, as that is the only model that accounts for dwell time as well. The NMC and LSTM models are used to provide comparable baselines to compare TLSTM model results. It could very well be the case that the TLSTM requires more data to train well.

Each dataset is split into a 70/30 split, where 70% of student click-streams are lumped into the training set and the remaining 30% form the test set. These click-streams are randomly split into such sets, and the same test set is applied for each of the 4 models. Table 19 displays the results of each of the 4 models averaged across all 99 courses. When interpreting these results, it is important to remember that the syllabus model is already the "pedagogically ideal" pathway through the course, and that the other models capture **common deviations** from the syllabus model. Thus, any signal above the syllabus model shows significant and consistent deviation from the syllabus model.

Table 19 Aggregate Model Results

Model Name	**Average Prediction Accuracy (Test set)**	**Average % Improvement Over Syllabus**
S	.5561	0.00
NMC	.5789	4.79
LSTM	.6033	9.28
TLSTM	.6024	9.09

It appears that the LSTM model had the best overall average accuracy. However, the TLSTM actually had a higher accuracy than the LSTM model in 52 out of the 99 courses. Thus, in the remaining 47 courses, the LSTM performed higher than the TLSTM. The two models are generally comparable when predicting next course page accuracy. However, the TLSTM also simultaneously models time in the output as well, so it is capable of also predicting how much time a student will spend on a resource.

It is not expected that every course will exhibit consistent non-linear navigation. The degree to which learners follow the syllabus model is influenced by factors such as learner motivations (are they certification seeking?), course design (how much content is on each page?), necessity of revisiting difficult topics more than once, and more. It is expected there will be a degree of variation in the extent to which behavior modelling actually exceeds a syllabus model. In particular, the NMC, LSTM, and TLSTM models are all trained on a training set, so any improvement over the syllabus model needs to have learned patterns of behaviors that generalized well to the test set.

The average % improvement over syllabus accuracy for TLSTM models was 9.09%. However, the distribution for this model is quite wide. Figure 22 shows the distribution of TSLSTM improvement over the syllabus model. 7 out of the 99 courses showed negative improvement; this means that the trained model did not generalize well on the test set, and that perhaps there are not many consistent non-linear navigations that can be learned by the model. The remaining 92 of the 99 courses showed that the TLSTM learned to predict behaviors that generalized to the test set, showing an improvement over the syllabus model in predicting what students will navigate next to. The amount of % improvement ranges from 0.27% to 52.94%. In total, 33 of the 99 courses displayed TLSTM improvements of over 10% compared to the syllabus model, perhaps indicating that there is substantial room for personalization in these courses as the TLSTM model was able to find general patterns of non-linear navigations. This can be interpreted that, in general, students in these courses do exhibit consistent non-linear behaviors, and that some courses contain substantially more patterns of patterns that can be generally learned by sequential models.

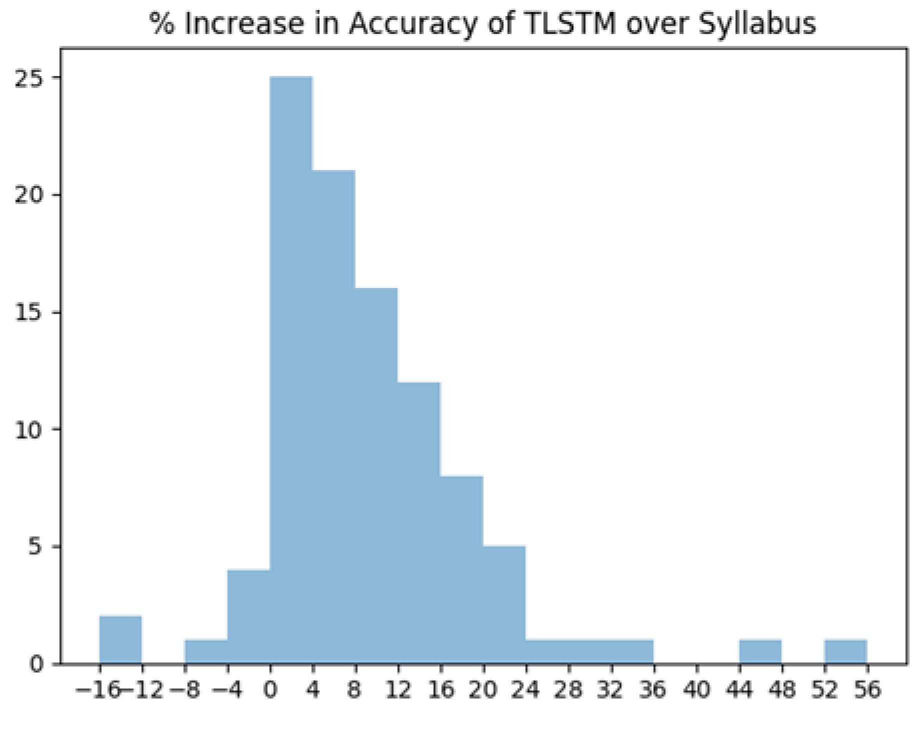


Figure 22 Distribution of TLSTM Improvement

It is often useful to compare how often competing models match and do not match in their predictions. Figure 23 shows how often the TLSTM and Syllabus models had ***different*** predictions for the same set of data, but one model had a correct prediction and the other did not, measured as a proportion of the total number of navigations that were correctly predicted by either model. This figure contains a clustered bar graph, where each blue line has a corresponding orange line; these lines refer to predictions within one course. When the blue and orange lines have small magnitudes, that means that both models generally made similar predictions. For example, if the blue (TLSTM) line had a value of .09, and the orange (Syllabus) line had a value of .06, that means the TLSTM and Syllabus differed on .15 of all correct predictions (.09 and .06 summed together yields .15). That means for the remaining 85% of correct predictions, both models made the same correct prediction. Thus, when the magnitudes are bigger in this figure, there is more disagreement in correct predictions. When one line is large, that means the models had larger disagreement, as a proportion of all correctly predicted navigations for that course.

In 92 of the 99 courses, the TLSTM model outperformed the Syllabus model. This shows that the TLSTM model had a higher overall accuracy of predicting the next prediction. In the 7 courses where the Syllabus outperformed TLSTM, it could be the case that the TLSTM was unable to find patterns of navigations in the training set that would generalize well in the test set. The courses where the blue line is much larger than the orange line indicate significant improvement in predicting navigational behaviors over the Syllabus model.

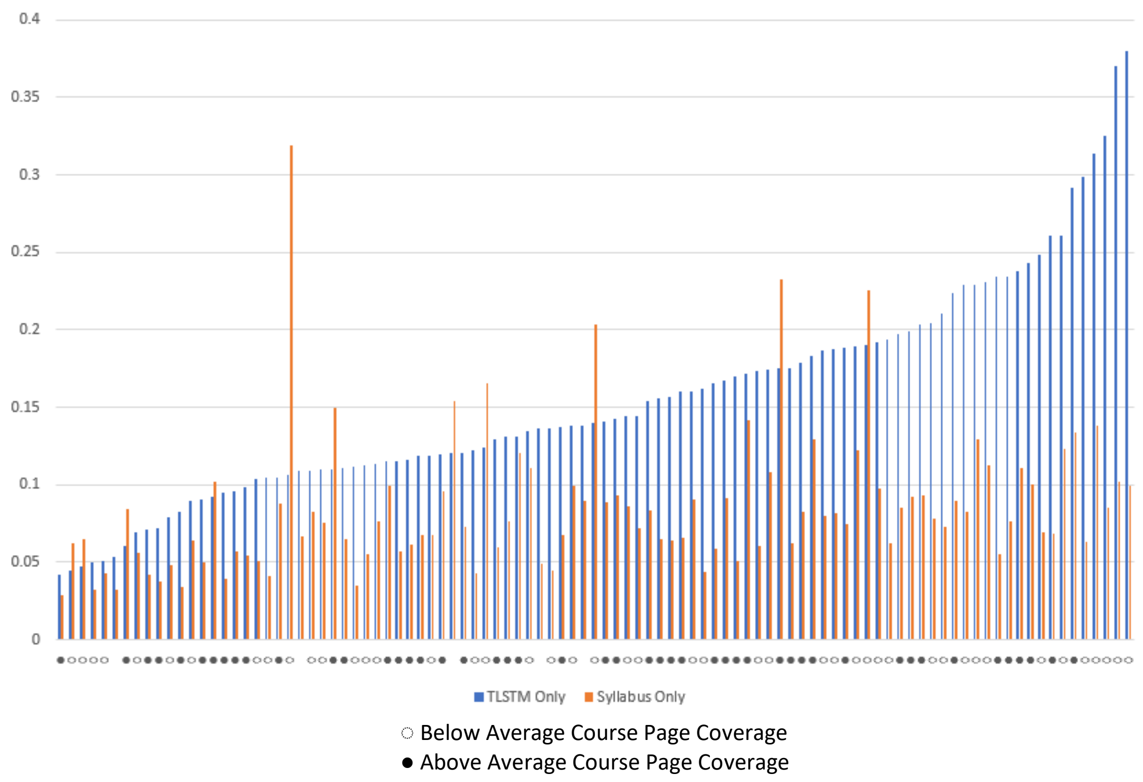


Figure 23 Differing Correct Predictions between TLSTM and Syllabus

Table 20 Description of Courses That Were Best Uniquely Correctly Predicted by TLSTM and Syllabus Respectively

	Introduction to Computer Programming, Part 1	**Relics in Chinese History - Part 1: Agriculture and Manufacturing**
Host University	IITBombay	Tsinghua
Host Country	India	China
TLSTM Unique Correctness Proportion	.380 (Best Among All Courses)	.107
Syllabus Unique Correctness Proportion	.099	.319 (Best Among All Courses)
Average Percentage of Verticals Accessed by Cert. Earners	16%	70%
Average Percentage of Verticals Skipped by Cert. Earners	84%	30%
Certificate Earners	592	37
STEM Course?	Yes	No
Instructor Paced?	No	No
Learners with 1+ Problem Checks (PC)	3885	92
Proportion of 1+ PC Learners That End Up with Certificate	.15	.40
Navigations Logged	179,134	7619
Syllabus Model Accuracy	.3638	.4645
Next Most Common Model Accuracy	.4811	.5638
TLSTM Model Accuracy	.5272	.4028
%TLSTM Gain Over Syllabus Model Accuracy	44.91%	-13.28
Chapter Count	34	6
Sequential Count	116	38
Vertical Count	203	79
Problem Verticals	33	6
Video Verticals	73	72
Book/Discussion Verticals	97	1
Total Individual Problems	139	14
Total Forum Comments	449	23
Total Forum Comment Threads	207	15
Forum Comments Made by Staff	127	10
Forum Threads Made by Staff	5	3

Table 20 provides the description of the two courses that obtained the highest relative unique correctness for both the TLSTM and Syllabus models respectively.

Introduction to Computer Programming, Part 1 from IITBombayX had the highest TLSTM unique correctness proportion at .380, while Relics in Chinese History – Part 1: Agriculture and Manufacturing from Tsinghua had the highest Syllabus unique correctness proportion. One feature that stands out is that the Tsinghua course only had 92 learners with at least 1 problem check. It is certainly possible that, given the test set is only comprised of a couple dozen learner sequences, the TLSTM was unable to generalize well with small training and test sets. This is one potential explanation as to why the TLSTM did worse than the Syllabus model in this one course. On the other side, in the Bombay course, the TLSTM was significantly better in making correct predictions than the Syllabus model. In this course, a huge proportion of the course was skipped by certificate earners. This is one area where the TLSTM can improve over the Syllabus model, by learning when learners are likely to skip content (and therefore not follow the syllabus).

5.11 Discussion

It is clear that many students are not following the prescribed order of the Syllabus. It is somewhat interesting that in 2013, Guo [57] found that certificate earners were completely skipping 22% of all course content. In this chapter, the findings indicate that 3 years later across 99 MOOCs, certificate earners are skipping even more content, at 28%. Thus, the trend of skipping content has actually increased over time. The approach presented in this chapter suggests that MOOCs can instead invest in personalized recommendation to at least enhance the existing experience of learners taking the MOOC. Utilizing a personalized recommendation framework using a TLSTM model can help guide learners to where they want to go. In 92 of the 99 MOOCs analyzed in this study, the TLSTM approach was able to better predict the behaviors of learners in a held out test set of learner sequences of behaviors compared to the actual Syllabus of the course. It has been well established that learners take MOOCs for a wide variety of motivations and reasons [58], and it is likely that these differing motivations as well as general learner backgrounds manifest in a variety of non-linear behavior patterns that can be observed in the click-stream logs of MOOCs.

The TLSTM approach is suitable for the majority of MOOCs. When it comes to actually implementing this type of personalized recommendation in practice, the implementer may choose to focus on courses that show the greatest promise in capturing non-linear behavior, perhaps through a metric such as the one presented in this chapter, the %TLSTM gain over Syllabus metric.

Personalized recommendation, in general, seems to be a promising path forward when it comes to education at scale. The methods investigated in this chapter are just one of many types of possible recommendations. This type of recommendation can be seen as trying to get students to where they want to go more quickly. A significant and large proportion of learners taking MOOCs are skipping large portions of the course; a recommender utilizing the methods from this chapter would then simply encourage this type of behavior to get learners to where they are probably going to end up anyways. This has both positives and negatives. The instructional design team might think that skipping content is inherently wrong, and that the course was organized to be consumed in its entirety and in order. However, the reality is that learners have many different motivations and needs when taking the course. The question becomes whether the course should adapt to the needs of its users or whether users should instead just be discouraged from taking this MOOC, since it is not suitable for them according to the instructional design. This is certainly an open question with many different viewpoints.

The role of MOOCs in a learners educational journey is still not entirely clear. On the one hand, it makes sense to try to bring university-level courses to everyone around the world. However, these courses are generally designed with a homogenous population of learners in mind. Additionally, the stakes for taking these in-person university courses are usually quite high, where the students have committed both their time and their money to taking the course. On the other hand, MOOCs are generally low stakes and low cost. Thus, the intrinsic population attributes and the intrinsic motivation for taking such courses is quite different, which can easily manifest in very different behaviors when interacting with course content. Thus, while MOOCs in their current form closely mimic in-person course structures, it may be the case that MOOCs might be best envisioned as a personalized online learning system, enhanced in different ways than in-person courses.

6 Conclusion

In this dissertation, I investigated an item order modification to the BKT framework and introduced a new methodology for understanding student behaviors in MOOC contexts. These methodologies enable personalization and seek to improve efficiency in their respective contexts.

Chapter 2 investigated the potential for including an item-order learning component into the base Bayesian Knowledge Tracing (BKT) framework. In that chapter, a qualitative analysis was performed, in addition to utilizing a quantitative cross-validation approach towards model fitting. The cross-validation framework for establishing significantly different item order effects was presented, and a regression test was performed to try and better predict when an item order might be more effective. Finally, a small experiment was conducted, although it did not yield immediate results.

Chapter 3 showed the first steps towards building a behavior model in MOOC contexts, where we drew an analogy between recurrent neural networks used in language modeling and their ability to potentially find general patterns of MOOC behaviors. Using this structure of a sequential model, we were able show that deep learning based sequential models were able to capture more information than a baseline of an n-gram structures with backoff.

Chapter 4 expanded on this model by adding a time-dwell component to the model, enhancing the model to be able to predict the amount of time spent on each resource. We found that the time-augmented version of the recommendation model improved prediction accuracy. Along with my co-authors, we built this recommendation model into a full-fledged recommendation framework, allowing students to actively receive recommendations based on their own personal history of actions within the MOOC.

Chapter 5 showed that the time-augmented recommendation framework generalized across most courses in a survey of 99 edX MOOCs. This type of research shows that the patterns of non-linear navigations and behaviors did seem to be common across different MOOCs, and that avenues for personalization in MOOC context seem promising.

As the education world continues to veer more and more towards digital solutions, smart methodologies that can handle large sets of data will become more and more important. Some of the methodologies discussed in this thesis show strong promise in finding solutions to personalization in learning at scale, and it is my hope that the models introduced and investigated in this thesis can be extended and eventually impact personalized learning at scale.

References

[1] G. Linden, B. Smith and J. York, "Item-to-Item Collaborative Filtering," IEEE, 2003.

[2] J. Xu, A. Lada and V. Kant, "News Feed FYI: Showing You More Personally Informative Stories," 11 August 2016. [Online]. Available: https://newsroom.fb.com/news/2016/08/news-feed-fyi-showing-you-more-personally-informative-stories/.

[3] S. Tang, E. McBride, H. Gogel and Z. A. Pardos, "Item Ordering Effects with Qualitative Explanations using Online Adaptive Tutoring Data," *Learning at Scale,* 2015.

[4] S. Tang, H. Gogel, E. McBride and Z. A. Pardos, "Desirable Difficulty and Other Predictors of Effective Item Orderings," *International Conference on Educational Data Mining,* 2015.

[5] S. Tang, J. Peterson and Z. A. Pardos, "Predictive Modeling of Student Behavior using Granular Large Scale Action Data from a MOOC," *Handbook of Learning Analytics and Educational Data Mining,* 2017.

[6] Z. A. Pardos, S. Tang, D. Davis and C. V. Le, "Enabling Real-Time Adaptivity in MOOCs with a Personalized Next-Step Recommendation Framework," *Learning at Scale,* 2017.

[7] M. Feng, N. Heffernan and K. Koedinger, "Addressing the assessment challenge in an Intelligent Tutoring System that tutors as it assesses," *The Journal of U"ser Modeling and User-Adapted Interaction,* vol. 19, pp. 243-266.

[8] J. Ocumpaugh, R. Baker, S. Gowda, N. Heffernan and C. Heffernan, "Population validity for Educational Data Mining models: A case study in affect detection," *British Journal of Educational Technology,* vol. 45, no. 3, pp. 487-501, 2014.

[9] Z. A. Pardos and N. T. Heffernan, "Determining the Significance of Item Order in Randomized Problem Sets," *International Conference on Educational Data Mining,* 2009.

[10] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User modeling and user-adapted interaction,* vol. 4, no. 4, pp. 253-278, 1994.

[11] M. C. Anderson, J. H. Neely, E. L. Bjork and R. Bjork, "Memory, Chapter 6: Interference and inhibition in memory retrieval," 1996.

[12] R. W. Christina and R. A. Bjork, "In the mind's eye: Enhancing human performance: Optimizing long-term retention and transfer," 1991.

[13] J. R. Anderson, A. T. Corbett, K. R. Koedinger and R. Pelletier, "Cognitive tutors: Lessons learned," *The Journal of the Learning Sciences,* vol. 4, no. 2, pp. 167-207, 1995.

[14] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. Pietra and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics,* pp. 467-479, 1992.

[15] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky and S. Khudanpur, "Recurrent neural network based language model," in *Interspeech*, 2010.

[16] S. Crossley, L. Paquette, M. Dascalu, D. S. McNamara and R. S. Baker, "Combining click-stream data with NLP tools to better understand MOOC completion," in *Learning Analytics and Knowledge*, 2016.

[17] Z. A. Pardos and X. Yanbo, "Improving efficacy attribution in a self-direct learning environment using prior knowledge individualization," in *Learning Analytics and Knowledge*, 2016.

[18] S. Reddy, I. Labutov and T. Joachims, "Latent Skill Embedding for Personalized Lesson Sequence Recommendation," [Online]. Available: http://arxiv.org/abs/1602.07029.

[19] A. Sharma, A. Biswas, A. Ghandi, S. Patil and O. Deshmukh, "LIVELINET: A Multimodal Deep Recurrent Neural Network to Predict Liveliness in Educational Videos," *International Educational Data Mining Society,* 2016.

[20] M. Wen, D. Yang and C. P. Rose, "Sentiment Analysis in MOOC Discussion Forums: What does it tell us?," in *Educational Data Mining*, 2014.

[21] J. Reich, B. Stewart, K. Mavon and D. Tingley, "The Civic Mission of MOOCs: Measuring Engagement across Political Differences in Forums," in *Learning @ Scale*, 2016.

[22] P. Oleksandra and D. Shane, "Untangling MOOC learning networks," in *Learning Analytics & Knowledge*, 2016.

[23] Z. A. Pardos, Y. Bergner, D. T. Seaton and D. E. Pritchard, "Adapting Bayesian Knowledge Tracing to a Massive Open Online Course in edX.," *Educational Data Mining,* pp. 137-144, 2013.

[24] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas and J. Sohl-Dickstein, "Deep Knowledge Tracing," *Advances in Neural Information Processing Systems,* pp. 505-513, 2015.

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

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[27] A. Graves, A.-r. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," *Acoustics, Speech and Signal Processing (ICASSP),* pp. 6645-6649, 2013.

[28] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever and G. Hinton, "Grammar as a Foreign Language," *Advances in Neural Information Processing Systems 28,* pp. 2755-2763, 2015.

[29] O. Vinyals, A. Toshev, S. Bengio and D. Erhan, "Show and Tell: A Neural Image Caption Generator," *IEEE Conference on Computer Vision and Pattern Recognition,* 2015.

[30] W. Miaomiao and C. P. Rose, "Identifying latent study habits by mining learner behavior patterns in massive open online courses," *Information and Knowledge Management,* pp. 1983-1986, 2014.

[31] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks,* vol. 5, no. 2, pp. 157-166, 1994.

[32] F. A. Gers, J. Schmidhuber and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation,* vol. 12, no. 10, pp. 2451-2471, 1999.

[33] F. Chollet, *Keras,* Github repository, 2015.

[34] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio, "Theano: a {CPU} and {GPU} Math Expression Compiler," in *Proceedings of the Python for Scientific Computing Conference ({SciPy})*, Austin, 2010.

[35] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard and Y. Bengio, *Theano: new features and speed improvements,* Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[36] Y. Goldberg and O. Levy, *word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding,* http://arxiv.org/abs/1402.3722, 2014.

[37] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks,* vol. 1, no. 4, pp. 339-356, 1988.

[38] V. Pham, T. Bluche, C. Kermorvant and J. Louradour, "Dropout improves recurrent enural networks for handwriting recognition," in *Frontiers in Handwriting Recognition*, 2014.

[39] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink and J. Schmidhuber, "LSTM: A search space odyssey," *arXiv preprint arXiv:1503.04069,* 2015.

[40] H. Khalil and M. Ebner, "Moocs completion rates and possible methods to improve retention-a literature review," *Word Conference on Educational Multimedia, Hypermedia and Telecommunications,* vol. 1, pp. 1305-1313, 2014.

[41] R. F. Kizilcec, M. Perez-Sanagustin and J. J. Maldonado, "Recommending self-regulated learning strategies does not improve performance in a MOOC," *Learning at Scale,* pp. 101-104, 2016.

[42] D. Davis, G. Chen, T. van der Zee, C. Hauff and G. J. Houben, "Retrieval practice and study planning in moocs: Exploring classroom-based self-regulated learning strategies at scale," *European Conference on Technology Enhanced Learning,* pp. 57-71, 2016.

[43] J. H. Tomkin and D. Charlevoix, "Do professors matter?: Using an a/b test to evaluate the impact of instructor involvement on MOOC student outcomes," *Learning @ Scale,* pp. 71-78, 2014.

[44] D. Davis, G. Chen, C. Hauff and G. J. Houben, "Gauging mooc learners adherence to the designed learning path," *Educational Data Mining,* pp. 54-61, 2016.

[45] V. Aleven, E. McLaughlin, R. A. Glenn and K. Koedinger, "Instruction based on adaptive learning technologies," *Handbook of research on learning and instruction,* 2016.

[46] C. Piech, M. Sahami, J. Huang and L. Guibas, "Autonomously generating hints by inferring problem solving policies," *Learning @ Scale,* pp. 195-204, 2015.

[47] A. Nguyen, C. Piech, J. Huang and L. Guibas, "Codewebs: scalable homework search for massive open online programming courses," *Conference on World wide web,* pp. 491-502, 2014.

[48] I. H. Hsiao, S. Sosnovsky and P. Brusilovsky, "Guiding students to the right questions: adaptive navigation support in an e-learning system for java programming," *Journal of Computer Assisted Learning,* vol. 26, no. 4, pp. 270-283, 2010.

[49] J. Stamper, T. Barnes, L. Lehmann and M. Croy, "The hint factory: Automatic generation of contextualized help for existing computer aided instruction," *Intelligent Tutoring Systems Young Researchers Track,* pp. 71-78, 2008.

[50] T. Barnes and J. Stamper, "Toward automatic hint generation for logic proof tutoring using historical student data," *International Conference on Intelligent Tutoring Systems,* pp. 373-382, 2008.

[51] J. Stamper, M. Eagle, T. Barnes and M. Croy, "Experimental evaluation of automatic hint generation for a logic tutor," *International Journal of Artificial Intelligence in Education,* vol. 22, no. 1-2, pp. 3-17, 2013.

[52] Z. A. Pardos, Y. Bergner, D. Seaton and D. E. Pritchard, "Adapting Bayesian Knowledge Tracing to a Massive Open Online College Course in edX," *Educational Data Mining,* pp. 137-144, 2013.

[53] J. L. Herlocker, J. A. Konstan, L. G. Terveen and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS),* vol. 22, no. 1, pp. 5-53, 2004.

[54] M. Pielot, T. Dingler, J. S. Pedro and N. Oliver, "When attention is not scarce-detecting boredom from mobile phone usage," *ACM International Join Conference on Pervasive and Ubiquitous Computing,* pp. 825-836, 2015.

[55] J. F. Pane, B. A. Griffin, D. F. McCaffrey and R. Karam, "Effectiveness of Cognitive Tutor Algebra I at Scale," *Educational Evaluation and Policy Analysis,* vol. 36, no. 2, 2014.

[56] B. S. Bloom, "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring," *Educational researcher,* vol. 13, no. 6, pp. 4-16, 1984.

[57] P. J. Guo and K. Reinecke, "Demographic Differences in How Students Navigate Through MOOCs," in *Learning at Scale*, 2014.

[58] R. F. Kizilcec and E. Schneider, "Motivation as a Lens to Understand Online Learners: Toward Data-Driven Design with the OLEI Scale," *ACM Transactions on Computer-Human Interaction,* vol. 22, no. 2, 2015.

[59] A. T. Corbett and K. R. Koedinger, "Intelligent Tutoring Systems," in *Handbook of Human-Computer Interaction*, 1997.

[60] P. Brusilovsky and C. Peylo, "Adaptive and Intelligent Web-based Educational Systems," *International Journal of Artificial Intelligence in Education,* no. 13, pp. 156-159, 2003.

[61] P. Brusilovsky, E. Schwarz and G. Weber, "ELM-ART: An intelligent tutoring system on world wide web," *International Conference on Intelligent Tutoring Systems,* pp. 261-269, 1996.

[62] A. C. Graesser, P. Chipman, B. C. Haynes and A. Olney, "AutoTutor: an intelligent tutoring system with mixed-initiative dialogue," *IEEE Transaction on education,* vol. 48, no. 4, pp. 612-618, 2005.

[63] R. Baker, A. T. Corbett, K. R. Koedinger, S. Evenson, I. Roll, A. Z. Wagner, M. Naim, J. Raspat, D. J. Baker and J. E. Beck, "Adapting to When Students Game an Intelligent Tutoring System," *International Conference on Intelligent Tutoring Systems,* pp. 392-401, 2006.

[64] C.-M. Chen, H.-M. Lee and Y.-H. Chen, "Personalized e-learning system using Item Response Theory," *Computers & Education,* vol. 44, no. 3, pp. 237-255, 2005.

[65] J. Lu, "A Personalized e-Learning Material Recommender System," in *International Conference on Information Technology for Application*, 2004.

[66] O. R. Zaiane, "Building a recommender agent for e-learning systems," in *Computers in Education*, 2002.

[67] D. Yang, M. Piergallini, I. Howley and C. Rosé, "Forum thread recommendation for massive open online courses," in *Educational Data Mining*, 2014.

[68] K. Hanan and M. Ebner, "MOOCs completion rates and possible methods to improve retention-A literature review," *Word Conference on Educational Multimedia, Hypermedia and Telecommunications,* vol. 1, 2014.

[69] J. Whitehill, J. J. Williams, G. Lopez, C. A. Coleman and J. Reich, "Beyond prediction: First steps toward automatic intervention in MOOC student stopout," 2015.

[70] D. S. Chaplot, E. Rhim and J. Kim, "Predicting Student Attrition in MOOCs using Sentiment Analysis and Neural Networks," *AIED Workshops,* 2015.

[71] R. Hosseini, P. Brusilovsky, M. Yudelson and A. Hellas, "Stereotype Modeling for Problem-Solving Performance Predictions in MOOCs and Traditional Courses," in *UMAP*, Bratislava, Slovakia, 2017.

[72] S. Tang, J. C. Peterson and Z. A. Pardos, "Predictive Modeling of Student Behavior Using Granular Large Scale Action Data from a MOOC," *Handbook of Learning Analytics and Educational Data Mining,* 2017.

[73] T. Mikolov, M. Karafiát, L. Burget and S. Khudanpur, "Recurrent neural network based language model," *Interspeech,* 2010.

[74] Z. A. Pardos, S. Tang, D. Davis and C. V. Le, "Enabling Real-Time Adaptivity in MOOCs with a Personalized Next-Step Recommendation Framework," *Learning at Scale,* 2017.

[75] M. Sundermeyer, R. Schlüter and H. Ney, "LSTM Neural Networks for Language Modeling," *Interspeech,* 2012.

[76] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE transactions on neural networks and learning systems,* 2016.

[77] K. Diederik and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[78] 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 *NIPS Data-driven education workshop*, 2013.

[79] L. Breslow, D. Pritchard, J. DoBoer, G. Stump, A. D. Ho and D. T. Seaton, "Studying Learning in the Worldwide Classroom Research into edX's First MOOC," *Research & Practice in Assessment,* vol. 8, pp. 13-25, 2013.