Why Deep Knowledge Tracing has less Depth than Anticipated

Xinyi Ding
Lyle School of Engineering
Southern Methodist University
xding@smu.edu

Eric C. Larson
Lyle School of Engineering
Southern Methodist University
eclarson@lyle.smu.edu

ABSTRACT

Knowledge tracing allows Intelligent Tutoring Systems to infer which topics or skills a student has mastered, thus adjusting curriculum accordingly. Deep Knowledge Tracing (DKT) uses recurrent neural networks (RNNs) for knowledge tracing and has achieved significant improvements compared with models like Bayesian Knowledge Tracing (BKT) and Performance Factor Analysis (PFA). However, DKT is not as interpretable as other models because the decision-making process learned by recurrent neural networks is not wholly understood by the research community. In this paper, we critically examine the DKT model, visualizing and analyzing the behaviors of DKT in high dimensional space. We modify and explore the DKT model and discover that Deep Knowledge Tracing has some critical pitfalls: 1). instead of tracking each skill through time, DKT is more likely to learn an 'ability' model; 2) the recurrent nature of DKT reinforces irrelevant information that it uses during the tracking task; 3) an untrained recurrent network can achieve similar results to a trained DKT model, supporting a conclusion that recurrence relations are not properly learned and, instead, improvements are simply a benefit of projection into a high dimensional, sparse vector space. Based on these observations, we propose improvements and future directions for conducting knowledge tracing research using deep models.

Keywords

knowledge tracing, recurrent neural network, visualization

1. INTRODUCTION

Knowledge tracing has been investigated for decades. It allows Intelligent Tutoring Systems to infer which topics or skills a student has mastered, thus adjusting curriculum accordingly. Two widely used models are Bayesian Knowledge Tracing (BKT) [2] and Performance Factor Analysis (PFA)[11]. These models are designed in a way that each parameter has a semantic meaning. For example, the guess and slip parameter in the BKT model reflect the probability that a student could guess the correct answer and make a mistake despite mastery of a skill, respectively. BKT attempts to explicitly model these parameters and use them to infer a binary set of skills as mastered or not mastered. In parallel with research in knowledge tracing models, deep neural networks have gained popularity in fields like Natural Language Processing and Computer Vision [3, 9]. Piech *et. al* proposed Deep Knowledge Tracing (DKT) [12], using recurrent neural networks for knowledge tracing. The DKT model achieves significantly improved results compared to BKT and PFA. However, its mechanisms are not well understood by the research community. That is, none of the parameters are mapped to a semantically meaningful measure which diminishes our ability to understand how DKT performs predictions. There have been some attempts to explain why DKT works well [8, 15], but these studies treat DKT model more like a black box, without studying the state space that underpins the recurrent neural network. In this work, we analyze and visualize the learned state space of the DKT model to better understand its mechanisms.

Recurrent neural networks can learn long range dependencies across many time steps. Long short term memory (LSTM) networks, gated-recurrent unit (GRU) networks, and numerous other variants enhance the vanilla RNNs in one way or another have achieved empirical success [6, 1, 5]. However, there are incredibly few works explaining what is happening under the hood. Karpathy *et al.* [7] provide a detailed analysis of the behaviors of recurrent neural network in language processing and find that some neurons are responsible for long range dependencies like quotes and brackets. We take a similar approach for analyzing the DKT model.

We aim to provide a better understanding of the DKT model and a more solid footing for using deep models for knowledge tracing. In this paper, we "open the box" of the DKT recurrent architecture, visualizing and analyzing the behaviors of the DKT model in a high dimensional space. We track activation changes through time and analyze the impact of each skill in relation to other skills. We modify and explore the DKT model, finding that some irrelevant information is reinforced in the recurrent architecture. Finally, we find that an untrained DKT model (with gradient descent applied only to layers outside the recurrent architecture) can be trained to achieve similar performance as a fully trained DKT architecture. Based on our analyses, we propose improvements and future directions for conducting knowledge

Xinyi Ding and Eric Larson "Why Deep Knowledge Tracing has less Depth than Anticipated" In: *Proceedings of The 12th International Conference on Educational Data Mining (EDM 2019)*, Collin F. Lynch, Agathe Merceron, Michel Desmarais, & Roger Nkambou (eds.) 2019, pp. 282 - 287

tracing with deep recurrent neural network models.

2. RELATED WORK

Bayesian Knowledge Tracing (BKT) [2] was proposed by Corbett *et al.* In their original work, each skill has its own model and parameters are updated by observing the responses (correct or incorrect) of applying a skill. Performance Factor analysis (PFA) [11] is an alternative method to BKT and it is believed to perform better when each response requires multiple skills. Both BKT and PFA are designed in a way that each parameter has its own semantic meaning. For example, the slip parameter of BKT represents the possibility of getting a question wrong even though the student has mastered the skill. These models are easy to interpret, but suffer from scalability issues and often fail to capture the dependencies between each skill because many elements are treated as independent to facilitate optimization.

Piech *et al.* recently proposed the Deep Knowledge Tracing model (DKT) [12], which exploits recurrent neural networks for knowledge tracing and achieves significantly improved results. Piesch *et al.* transformed the problem of knowledge tracing by assuming each question can be associated with a "skill ID", with a total of N skills in the question bank. The input to the recurrent neural network is a binary vector encoding of skill ID for a presented question and the correctness of the student's response. The output of the recurrent network is a length N vector of probabilities for answering each skill-type question correctly. The DKT model could achieve >80% AUC on the ASSISTmentsData dataset [4], compared with the BKT model that achieves 67% AUC. This is an exciting result because it demonstrates the possibility of using neural networks for knowledge tracing.

Despite the effectiveness of DKT model, its mechanism is not well understood by the research community. Khajah *et al.* investigate this problem by extending BKT [8]. They extend BKT by adding forgetting, student ability, and skill discovery components, comparing these extended models with DKT. Some of these extended models could achieve close results compared with DKT. Xiong *et al.* discover that there are duplicates in the original ASSISTmentsData dataset [15]. They re-evaluate the performance of DKT on different subsets of the original dataset. Both Khajah and Xiong's work are black box oriented—that is, it is unclear how predictions are performed within the DKT model. In our work, we try to bridge this gap and explain some behaviors of the DKT model.

Trying to understand how DKT works is difficult because the mechanisms of RNNs are not totally understood even in the machine learning community. Even though the recurrent architecture is well understood, it is difficult to understand how the model adapts weights for a given prediction task. One common method used is to visualize the neuron activations. Karpathy *et al.* [7] provide a detailed analysis of the behaviors of recurrent neural network using character level models and find some cells are responsible for long range dependencies like quotes and brackets. They break down the errors and partially explain the improvements of using LSTM. We use and extend their methods, providing a detail analysis of the behaviors of LSTM in the knowledge tracing setting.

3. EXPERIMENT

To investigate the DKT model, we perform a number of analyses based upon the activations within the recurrent neural network. We also explore different training protocols and clustering of the activations to help elucidate what is learned by the DKT model.

3.1 Experiment setup

In our analyses, we use the "ASSISTmentsData 2009-2010 (b) dataset" which is created by Xiong *et al.* after removing duplicates [15]. Like Xiong *et al.*, we also use LSTM units for analysis in this paper. Because we will be visualizing specific activations of the LSTM, it is useful to review the mathematical elements that comprise each unit. An LSTM unit consists of the following parts, where a sequence of inputs $\{x_1, x_2, ..., x_T\} \in \mathcal{X}$ are ideally mapped to a labeled output sequence $\{y_1, y_2, ..., y_T\} \in \mathcal{Y}$. The prediction goal is to learn weights and biases (W and b) such that the model output sequence ($\{h_1, h_2, ..., h_T\} \in \mathcal{H}$) is as close as possible to $\mathcal{Y}$:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{5}$$

$$h_t = o_t * \tanh(C_t) \tag{6}$$

Here, σ refers to a logistic (sigmoid) function, $\cdot$ refers to dot products, $*$ refers to element-wise vector multiplication, and $[,]$ refers to vector concatenation. For visualization purposes, we log the above 6 intermediate outputs for each input during testing and concatenate these outputs into a single "activation" vector, $a_t = [f_t, i_t, \tilde{C}_t, C_t, o_t, h_t]$. In the DKT model, the output of RNN, h_t is connected to an output layer y_t, which is a vector with the same number of elements as skills. We can interpret each element in y_t as an

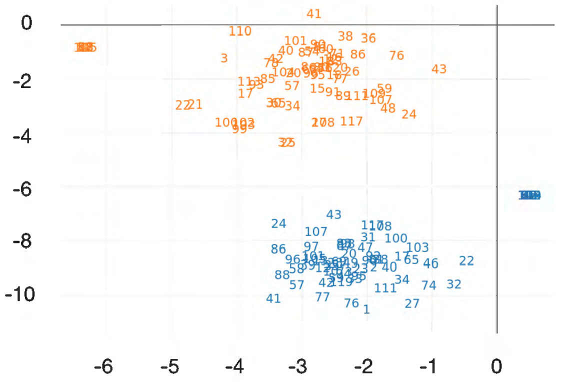

Figure 1: First two components of T-SNE of the activation vector for first time step inputs. Numbers are skill identifiers, blue for correct input, orange for incorrect input

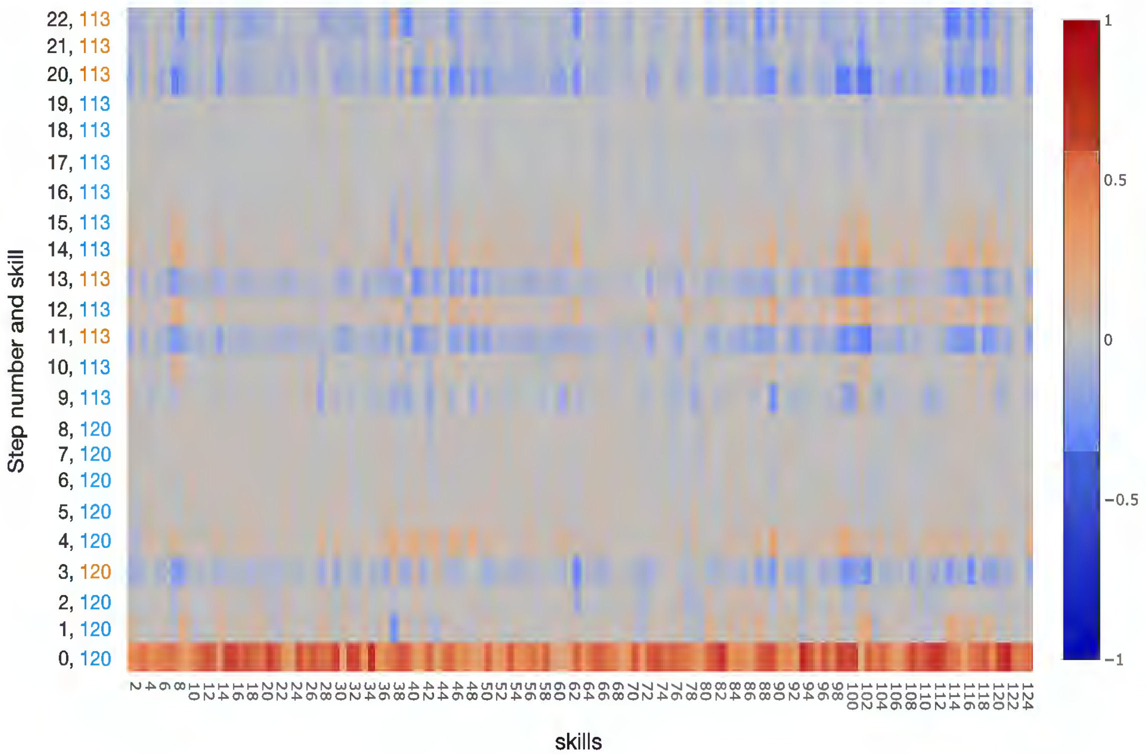

Figure 2: The prediction changes for one student, 23 steps, correct input is marked blue, incorrect input is marked orange

estimate that the student would answer a question from each skill correctly, with larger positive number denoting that the student is more likely to answer correctly and more negative numbers denoting that the student is unlikely to respond correctly. Thus, a student who had mastered all skills would ideally obtain an y_t of all ones. A student who had mastered none of the skills would ideally obtain an y_t of all negative ones.

Deep neural networks usually work in high dimensional space and are difficult to visualize. Even so, dimensionality reduction techniques can help to identify clusters. For example, Figure 1 plots the first two reduced components (using t-SNE [10]) of the activation vector, a_t, at the first time step ($t = 0$) for a number of different students in the ASSISTmentsData. The numbers in the plot are skill identifiers. We use color blue to denote a correct response and the color orange to denote an incorrect response. From reducing the dimensionality of the a_t vector for each student, we can see that the activations show a distinct clustering between whether the questions were answered correctly or incorrectly. We might expect to observe sub-clusters of the skill identifiers within each of the two clusters but we do not. This observation supports the hypothesis that correct and incorrect responses are more important for the DKT model than skill identifiers. However, perhaps this lack of sub-clusters is inevitable because we are only visualizing the activations after one time step—this motivates the analysis in the next section.

3.2 Skill relations

In this section, we try to understand how the prediction vector of one student changes as a student answers more questions from the question bank. Figure 2 plots the prediction difference (current prediction vector - previous prediction vector) for each question response from one particular student (steps are displayed vertically and can be read sequentially from bottom to top). The horizontal axis denotes the skill identifier and the color of the boxes in the heatmap denote the change in the output vector y_t. The initial row in the heatmap (bottom) is the starting values for y_t for the first input. As we can see, if the student answers correctly, most of the y_t values increase (warm color). When an incorrect response occurs, most of the predictions decreases (cold color). This makes intuitive sense. We expect a number of skills to be related so correct responses should add value and incorrect responses should subtract value. We can further observe that changes in the y_t vector diminish if the student correctly or incorrectly answers a question from the same skill several times repeatedly. For example, observe from step 14 to step 19, where the student correctly answers questions from skill #113—eventually the changes in y_t come to a steady state. However, occasionally, we can also notice, a correct response will result in decreases in the prediction vector (observe step 9). This behavior is difficult to justify from our experience, as correctly answering a question should not decrease the mastery level of other skills. Yeung *et al.* have similar findings when investigating single skills [16]. Observe also that step 9 coincides with a transition in skills being answered (from skill #120 to #113). Even so, it is curious that switching from one skill to an-

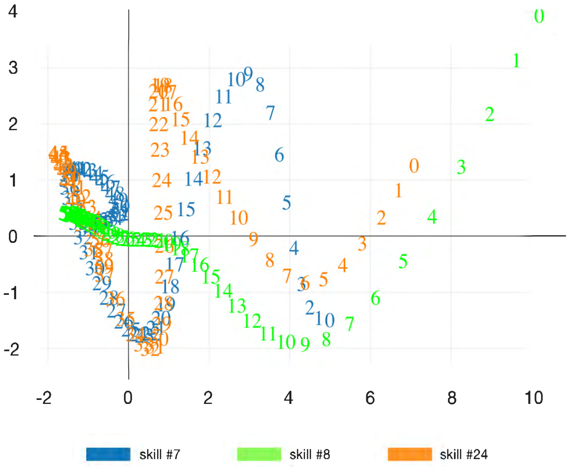

Figure 3: Activation vector changes for 100 continuous correctness of randomly picked 3 skills

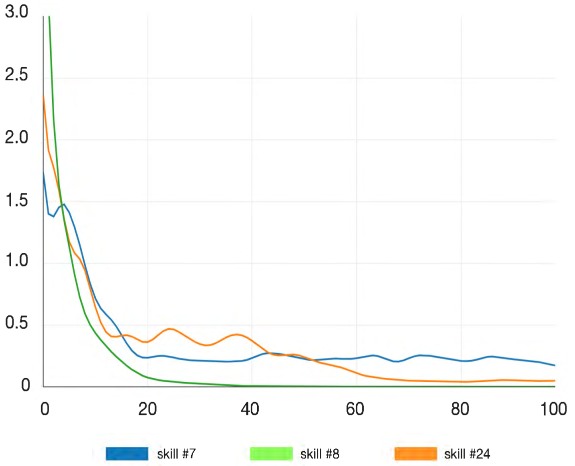

Figure 4: Activation vector difference of randomly picked 3 skills through time

other would decrease values in y_t even when the response is correct. From this observation, one potential way to improve the DKT model could be adding punishment for such unexpected behaviors (for example, in the loss function of the recurrent network).

3.3 Simulated data

From the above analysis, we see from step 14 to step 19, the student correctly answers question from skill #113 and the changes in y_t diminish—perhaps an indication that the vector is converging. Also, from Figure 2, we see that for each correct input, most of the elements of y_t increase by some margin, regardless of the input skill. To have a better understanding of this convergence behavior, we simulate how the DKT model would respond to an *Oracle Student*, which will always answer each skill correctly. We simulate how the model responds to the *Oracle Student* correctly answering 100 questions from one skill. We repeat this for three randomly selected skills.

We plot the convergence of each skill using the activation vector a_t reduced to a two-dimensional plot using t-SNE (Figure 3). The randomly chosen skills were #7. #8, and #24. As we can see, each of the three skills starts from a different location in the 2-D space. However, they each converges to near the same location in space. In other words, it seems DKT is learning one “oracle state” and this state can be reached by practicing any skill repeatedly, regardless of the skill chosen. We verified this observation with a number of other skills (not shown) and find this behavior is consistent. Therefore, we hypothesize that DKT is learning a ‘student ability’ model, rather than a ‘per skill’ model like BKT. To make this observation more concrete, in Figure 4 we plot the euclidean distance between the current time step activation vector, a_t, and the previous activations, a_{t-1}, we can see the difference becomes increasingly small after 20 steps. Moreover, the euclidean distance between each activation vector learned from each skill becomes extremely small, supporting the observation that not only is the y_t output vector converging, but all the activations inside the LSTM network are converging. We find this behavior curious because it means that the DKT model is not remembering what skill was used to converge the network to an ‘oracle state.’ Remembering the starting skill would be crucial for predicting future performance of the student, yet the DKT model would treat every skill identically. We also analyzed a process where a student always answers responses incorrectly and found there is a similar phenomenon with convergence in an anti-oracle state.

Figure 5 shows the skills prediction vector after answering correctly 20 times in a row. We can see the predictions of most skills are above 0.5, regardless of the specific practice skill used by the *Oracle Student*. Now, we can safely say that the DKT model is not really tracking the mastery level of *each skill*, it is more likely learning an ‘ability model’ from the responses. Once a student is in this oracle state, DKT will assume that he/she will answer most of the questions correctly from any skill. We hypothesize that this behavior could be mitigated by using an “attention” vector during the decoding of the LSTM network [13]. Self attention in recurrent networks decodes the state vectors by taking a weighted sum of the state vectors over a range in the sequence (weights are dynamic based on the state vectors). For DKT, this attention vector could also be dynamically allocated based upon the skills answered in the sequence, which might help facilitate remembering long-term skill dependencies.

3.4 Temporal impact

RNNs are typically well suited for tracking relations of inputs in a sequence, especially when the inputs occur near one another in the sequence. However, long range dependencies are more difficult for the network to track [13]. In other words, the predictions of RNN models will be more impacted by recent inputs. For knowledge tracing, this is not a desired characteristic. Consider two scenarios as shown below: For each scenario, the first line is the skill numbers and the second line are responses (1 for correctness and 0 for incorrectness). Both two scenarios have the same number of attempts for each skill (4 attempts for skill #9, 3 attempts for skill #6 and 2 attempts for skill #24). Also, the ordering of correctness within each skill is the same (*e.g.*, 1, 0, 0, 0

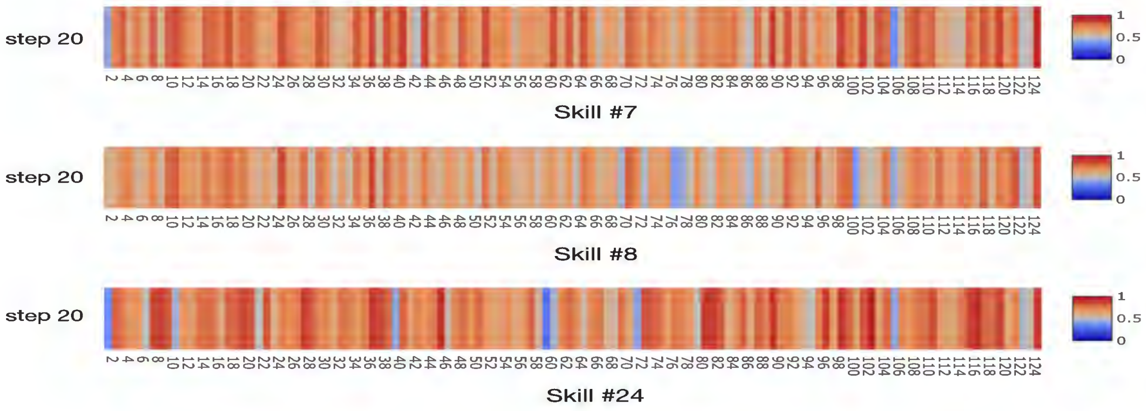

Figure 5: Prediction vector after 20 steps for skill #7, #8, #24

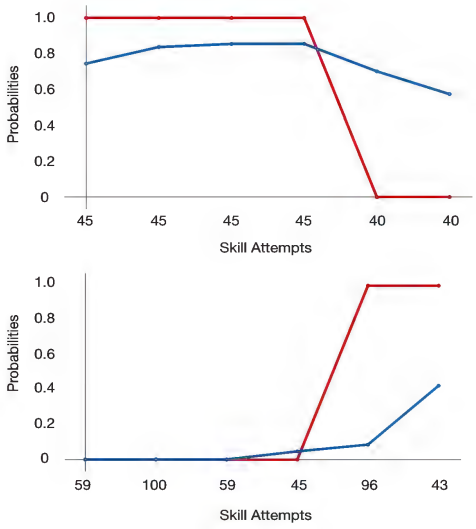

Figure 6: DKT predictions from two different students. The blue line is the prediction of correctness from DKT. The red line is the actual response correctness(1 or 0).

for skill #9).

Scenario #1									
Skill ID	6	6	9	9	9	9	**24**	24	6
Correct	1	1	1	0	0	0	0	0	1

Scenario #2									
Skill ID	9	9	9	9	6	6	6	**24**	24
Correct	1	0	0	0	1	1	1	0	0

For models like BKT, there is a separate model for each skill. Thus, the relative order of different skills presented has no influence, as long as the ordering within each skill remains the same. In other words, for each skill the ordering of correct and incorrect attempts remains the same, but different skills can be shuffled into the sequence. For BKT, it will learn the same model from these two scenarios, but it may not be the case for DKT. The DKT model is more likely to predict incorrect response after seeing three incorrect inputs in a row because it is more sensitive to recent inputs in the sequence. This means, for the first scenario, first attempt of skill #24 (in bold) will be more likely predicted incorrect because it follows three incorrect responses. For the second scenario, first attempt of skill #24 (in bold) is more likely to be predicted correct. Thus the DKT model might perform differently on the given scenarios.

Figure 6 gives two typical excerpts from the real dataset for two students. In the top example, after several correct inputs, the DKT model has a high probability of predicting the next item correct, regardless of the skill (70%). Similarly, in the bottom example, after several incorrect inputs, the DKT model has a low probability of predicting the next item correct (8%), regardless of the skill. That means, if a student has mastered an easy skill previously but then fails three attempts of more difficult exercises, the DKT would predict that the student would also fail the already mastered skill. We are only giving two samples here due to limited space, but this kind of behavior is universal across students, which we will talk more next. Again, we hypothesize that this behavior could be mitigated by using an "attention" vector that allows the DKT to use the whole weighted history as additional inputs.

Table 1: Area under the ROC curve

	PFA	BKT	DKT	DKT (spread)	DKT (untrained)
09-10 (a)	0.70	0.60	0.81	0.72	0.79
09-10 (b)	0.73	0.63	0.82	0.72	0.79
09-10 (c)	0.73	0.63	0.75	0.71	0.73
14-15	0.69	0.64	0.70	0.67	0.68
KDD	0.71	0.62	0.79	0.76	0.76

Table 2: Square of linear correlation (r^2) results

	PFA	BKT	DKT	DKT (spread)	DKT (untrained)
09-10 (a)	0.11	0.04	0.29	0.15	0.25
09-10 (b)	0.14	0.07	0.31	0.14	0.26
09-10 (c)	0.14	0.07	0.18	0.14	0.15
14-15	0.09	0.06	0.10	0.08	0.09
KDD	0.10	0.05	0.21	0.17	0.17

Khajah *et al.* also alluded to this recency effect in [8]. In this paper, we examine this phenomenon in a more quantitative way. We shuffle the dataset in a way that keeps the ordering within each skill the same, but spreads out the responses in the sequence. This change should not change the prediction ability of models like BKT. The results are shown in Table 1 and Table 2 using standard evaluation criteria for this dataset. All results are based on a five-fold cross validation of the dataset. When comparing DKT on the original dataset to the "spread out" dataset ordering, we see that the relative ordering of skills has significant negative impact on the performance of the model. From these observations, we see the behaviors of DKT is more like PFA which counts prior frequencies of correct and incorrect attempts other than BKT and the design of the exercises could have a huge impact on the model (For example, the arrangements of easy and hard exercises).

3.5 Is the RNN representation meaningful?

Recurrent models have been successfully used in practical tasks like natural language processing [3]. These models can take days or even weeks to train. In a recently published paper, Wieting *et al.* [14] argue that RNNs might not be learning a meaningful state vector from the data. They show that a randomly initialized RNN model (with only W_o and b_o trained) can achieve similar results to models where all parameters are trained. This result is worrying because it may indicate that the RNN performance is due mostly to simply mapping input data to random high dimensional space. Once projected into the random vector space linear classification can perform well because points are more likely to be separated in a sparse vector space. The actual vector space may not be meaningful. We perform a similar experiment in training the DKT model. We randomly initialize the DKT model and only train the last linear layer (W_o and b_o) that maps the output of LSTM h_t to the skill vector, y_t. As shown in Table 1 and Table 2, the untrained recurrent network performs similarly to the trained network.

4. CONCLUSION AND FUTURE WORK

In this paper, we dive deep into the Deep Knowledge Tracing model. We have visualized and analyzed the behaviors of DKT through time using dimensionality reduction of the activations vector, a_t. We have also analyzed the temporal sequence behavior of DKT using qualitative and quantitative analyses. We find that the DKT model is most likely learning an 'ability' model, rather than tracking each individual skill. Moreover DKT is significantly impacted by the relative ordering of skills presented. We also discover that a randomly initialized DKT with only the final linear layer trained achieves similar results to the fully trained DKT model. In other words, the DKT model performance gains may stem from mapping input sequences into a random high dimensional vector space where linear classification is easier because the space is sparse. This is a worrying conclusion because it means the underlying recurrent representation may not be reliable nor semantically meaningful. Several mitigating measures are suggested in this paper, including the use of a loss function that mitigates unwanted behaviors and the use of an attention model to better capture long term skill dependencies. We leave evaluation of these suggestions to future work in the educational data mining community.

5. REFERENCES

[1] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

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

[3] Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*

[4] Feng, M., Heffernan, N. T., and Koedinger, K. R. 2006. Addressing the testing challenge with a web-based e-assessment system that tutors as it assesses. In *Proceedings of the 15th international conference on World Wide Web*, 307–316.

[5] Graves, A., Wayne, G., and Danihelka, I. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

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

[7] Karpathy, A., Johnson, J., and Fei-Fei, L. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*

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

[9] Krizhevsky, A., Sutskever, I., and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

[10] Maaten, L. V. D., and Hinton, G. 2008. Visualizing data using t-sne. *Journal of machine learning research*,9, (Nov, 2008), 2579–2605.

[11] Pavlik Jr, P. I., Cen, H., and Koedinger, K. R. 2009. Performance factors analysis–a new alternative to knowledge tracing. *Online Submission*.

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

[13] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, 5998–6008.

[14] Wieting, J., and Kiela, D. 2019. No training required: Exploring random encoders for sentence classification. *arXiv preprint arXiv:1901.10444*.

[15] Xiong, X., Zhao, S., Van Inwegen, E. G., and Beck, J. E. 2016. Going deeper with deep knowledge tracing. In *International Educational Data Mining Society*.

[16] Yeung, C. K., and Yeung, D. Y. 2018. Addressing two problems in deep knowledge tracing via prediction-consistent regularization. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*.