A Self-Attentive model for Knowledge Tracing

Shalini Pandey
University of Minnesota
Twin Cities, MN 55455, USA
pande103@umn.edu

George Karypis
University of Minnesota
Twin Cities, MN 55455, USA
karypis@umn.edu

ABSTRACT

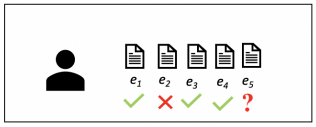
Knowledge tracing is the task of modeling each student's mastery of knowledge concepts (KCs) as (s)he engages with a sequence of learning activities. Each student's knowledge is modeled by estimating the performance of the student on the learning activities. It is an important research area for providing a personalized learning platform to students. In recent years, methods based on Recurrent Neural Networks (RNN) such as Deep Knowledge Tracing (DKT) and Dynamic Key-Value Memory Network (DKVMN) outperformed all the traditional methods because of their ability to capture complex representation of human learning. However, these methods face the issue of not generalizing well while dealing with sparse data which is the case with real-world data as students interact with few KCs. In order to address this issue, we develop an approach that identifies the KCs from the student's past activities that are *relevant* to the given KC and predicts his/her mastery based on the relatively few KCs that it picked. Since predictions are made based on relatively few past activities, it handles the data sparsity problem better than the methods based on RNN. For identifying the relevance between the KCs, we propose a self-attention based approach, Self Attentive Knowledge Tracing (SAKT). Extensive experimentation on a variety of real-world dataset shows that our model outperforms the state-of-the-art models for knowledge tracing, improving AUC by 4.43% on average.

Keywords

Knowledge Tracing, Massive Open Online Courses, Self-attention, sequential recommendation

1. INTRODUCTION

The availability of massive dataset of students' learning trajectories about their *knowledge concepts* (KCs), where a KC can be an exercise, a skill or a concept, has attracted data miners to develop tools for predicting students' performance and giving proper feedback [8]. For developing such personalized learning platforms, knowledge tracing (KT) is considered to be an important task and is defined as the task of tracing a student's *knowledge state*, which represents his/her mastery level of KCs, based on his/her past learning activities. The KT task can be formalized as a supervised sequence learning task - given student's past exercise interactions $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t)$, predict some aspect of his/her next interaction $\mathbf{x}_{t+1}$. On the question-answering platform, the interactions are represented as $\mathbf{x}_t = (e_t, r_t)$, where e_t is the exercise that the student attempts at timestamp t and r_t is the correctness of the student's answer. KT aims to predict whether the student will be able to answer the next exercise correctly, i.e., predict $p(r_{t+1} = 1 | e_{t+1}, \mathbf{X})$.



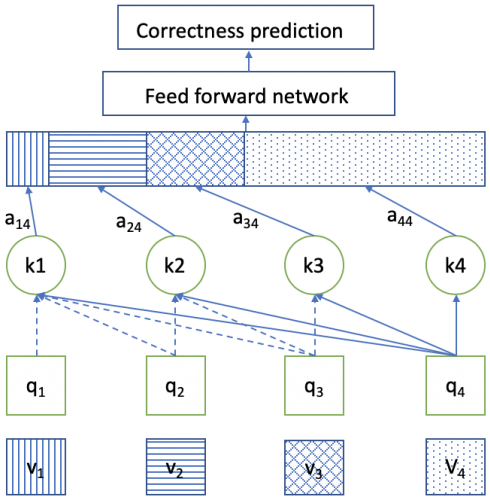
Exercise tag	Knowledge Concept
e_1	Multiplication
e_2	Square root
e_3	Division
e_4	Linear Equations
e_5	Quadratic equations

Figure 1: Left subfigure shows the sequence of exercises that the student attempts and the right subfigure shows the knowledge concepts to which each of the exercises belong.

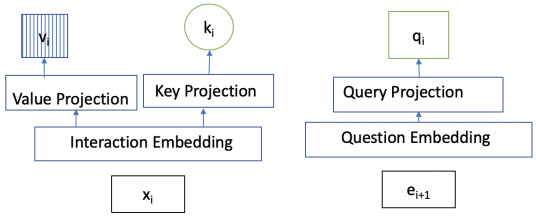
Recently deep learning models such as Deep Knowledge Tracing (DKT) [6] and its variant [10] used Recurrent Neural Network (RNN) to model a student's knowledge state in one summarized hidden vector. Dynamic Key-value memory network (DKVMN) [11] exploited Memory Augmented Neural Network [7] for KT. Using two matrices, *key* and *value*, it learns the correlation between the exercises and the underlying KC and student's knowledge state, respectively. The DKT model faces the issue of its parameters being non-interpretable [4]. DKVMN is more interpretable than DKT as it explicitly maintains a KC representation matrix (*key*) and a knowledge state representation matrix (*value*). However, since all these deep learning models are based on RNNs, they face the issue of not generalizing while dealing with sparse data [3].

In this paper, we propose to use a purely attention mechanism based method, *transformer* [9]. In the KT task, the skills that a student builds while going through the sequence of learning activities, are related to each other and the performance on a particular exercise is dependent on his performance on the past exercises related to that exercise. For example, in figure 1, for a student to solve an exercise on "Quadratic equation" (exercise 5) which belongs to the knowl-

edge concept "Equations", he needs to know how to find "square roots" (exercise 3) and "linear equations" (exercise 4). SAKT, proposed in this paper first identifies *relevant* KCs from the past interactions and then predicts student's performance based on his/her performance on those KCs. For predicting student's performance on an exercise, we used exercises as KCs. As we show later, SAKT assigns weights to the previously answered exercises, while predicting the performance of the student on a particular exercise. The proposed SAKT method significantly outperforms the state-of-the-art KT methods gaining a performance improvement of 4.43% on the AUC, on an average across all datasets. Furthermore, the main component (self-attention) of SAKT is suitable for parallelism; thus, making our model order of magnitude faster than RNN based models.



(a) Network of SAKT. At each timestamp the attention weights are estimated for each of the previous element only. Keys, Values and Queries are extracted from the embedding layer shown below. When jth element is query and ith element is key, attention weight is $a_{i,j}$.



(b) Embedding layer embeds the current exercise that the student is attempting and his past interactions. At every time stamp $t+1$, the current question e_{t+1} is embedded in the query space using Exercise embedding and elements of past interactions $\mathbf{x}_t$ is embedded in the key and value space using the Interaction embedding.

Figure 2: Diagram showing the architecture of SAKT.

2. PROPOSED METHOD

Our model predicts whether a student will be able to answer the next exercise e_{t+1} based on his previous interaction sequence $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$. As shown in figure 2, we can transform the problem into a sequential modeling problem. It is convenient to consider the model with inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}$ and the exercise sequence with one position ahead, $e_2, e_3, \ldots, e_t$ and the output being the correctness of the response to exercises $r_2, r_3, \ldots, r_t$. The interaction tuple $\mathbf{x}_t = (e_t, r_t)$ is presented to the model as a number $y_t = e_t + r_t \times E$, where E is the total number of exercises. Thus, the total values that an element in the interaction sequence can take is $2E$, while elements in the exercise sequence can take E possible values.

Table 1: Notations

Notations	Description
N	total number of students
E	total number of exercises
$\mathbf{X}$	Interaction sequence of a student: $(x_1, x_2, \ldots, x_t)$
x_i	ith exercise-answer pair of a student
n	maximum length of sequence
d	latent vector dimensionality
$\mathbf{e}$	Sequence of exercises solved by the student
$\mathbf{M}$	Interaction embedding matrix
$\mathbf{P}$	Positional embedding matrix
$\mathbf{E}$	Exercise lookup matrix
$\hat{\mathbf{M}}$	Past interactions embedding
$\hat{\mathbf{E}}$	Exercise embedding

We now describe the different layers of our architecture.

Embedding layer: We transform the obtained input sequence $\mathbf{y} = (y_1, y_2, \ldots, y_t)$ into $s = (s_1, s_2, \ldots, s_n)$, where n is the maximum length that the model can handle. Since the model can work with inputs of fixed length sequence, if the sequence length, t is less than n, we repetitively add a *padding* of question-answer pair to the left of the sequence. However, if t is greater than n, we partition the sequence into subsequences of length n. Specifically, when t is greater than n, y_t is partitioned into t/n subsequences each of length n. All these subsequences serve as input to the model.
We train an *Interaction embedding matrix*, $\mathbf{M} \in \mathbb{R}^{2E \times d}$, where d is the latent dimension. This matrix is used to obtain an embedding, $\mathbf{M}_{s_i}$ for each element, s_i in the sequence. Similarly, we train exercise embedding matrix, $\mathbf{E} \in \mathbb{R}^{E \times d}$ such that each exercise in the set e_i is embedded in the e_ith row.
Position Encoding: Position Encoding is the layer in the self-attention neural network which is used for encoding the position so that like convolution network and recurrent neural network, we can encode the order of the sequence. This layer is particularly important in knowledge tracing problem because a student's knowledge state evolves gradually and steadily with time. The knowledge state at a particular time instance should not show wavy transitions [10]. In order to incorporate this we use a parameter, position embedding, $\mathbf{P} \in \mathbb{R}^{n \times d}$ which is learned while training. The ith row of position embedding matrix, $\mathbf{P}_i$ is then added to the interaction embedding vector of the ith element of the interaction sequence.

The output from the embedding layer is embedded interac-

tion input matrix, $\hat{\mathbf{M}}$ and embedded exercise matrix, $\hat{\mathbf{E}}$:

$$\hat{\mathbf{M}} = \begin{bmatrix} \mathbf{M}_{s_1} + \mathbf{P}_1 \\ \mathbf{M}_{s_2} + \mathbf{P}_2 \\ \cdots \\ \mathbf{M}_{s_n} + \mathbf{P}_n \end{bmatrix}, \ \hat{\mathbf{E}} = \begin{bmatrix} \mathbf{E}_{s_1} \\ \mathbf{E}_{s_2} \\ \cdots \\ \mathbf{E}_{s_n} \end{bmatrix}. \quad (1)$$

Self-attention layer: In our model, we use the scaled dot-product attention mechanism [9]. This layer finds the relative weight corresponding to each of the previously solved exercise for predicting the correctness of the current exercise.

We obtain query and key-value pairs using the following equations:

$$\mathbf{Q} = \hat{\mathbf{E}}\mathbf{W}^Q, \mathbf{K} = \hat{\mathbf{M}}\mathbf{W}^K, \mathbf{V} = \hat{\mathbf{M}}\mathbf{W}^V, \quad (2)$$

where $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V \in \mathbb{R}^{d \times d}$ are the query, key and value projection matrices, respectively, which linearly project the respective vectors to different space [9]. The relevance of each of the previous interactions with the current exercise is determined using the attention weights. For finding the attention weights we use the scaled dot product [9], defined as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}. \quad (3)$$

Mutiple heads: In order to jointly attend to information from different representative subspaces, we linearly project the queries, keys and values h times using different projection matrices.

$$\text{Multihead}(\hat{\mathbf{M}}, \hat{\mathbf{E}}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O, \quad (4)$$

where $\text{head}_i = \text{Attention}(\hat{\mathbf{E}}\mathbf{W}_i^Q, \hat{\mathbf{M}}\mathbf{W}_i^K, \hat{\mathbf{M}}\mathbf{W}_i^V)$ and $\mathbf{W}^O \in \mathbb{R}^{hd \times d}$.

Causality:
In our model, we should consider only first t interactions when predicting the result of the $(t+1)$st exercise. Therefore, for a query $\mathbf{Q}_i$, the keys $\mathbf{K}_j$ such that $j > i$ should not be considered. We use, causality layer to mask the weights learned from a future interaction key,

Feed Forward layer:
The self-attention layer described above results in weighted sum of values, $\mathbf{V}_i$ of the previous interactions. However the rows of the matrix obtained from the multihead layer, $\mathbf{S} = \text{Multihead}(\hat{\mathbf{M}}, \hat{\mathbf{E}})$ is still a linear combination of the values, $\mathbf{V}_i$ of the previous interactions. To incorporate non-linearity in the model and consider the interactions between different latent dimensions, we use a feed forward network.

$$\mathbf{F} = \text{FFN}(\mathbf{S}) = \text{ReLU}(\mathbf{S}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \quad (5)$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times d}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times d}$, $\mathbf{b}^{(1)} \in \mathbb{R}^d$, $\mathbf{b}^{(2)} \in \mathbb{R}^d$ are parameters learned during training.

Residual Connections: The residual connection [2] are used to propagate the lower layer features to the higher layers. Hence, if low layer features are important for prediction, the residual connection will help in propagating them to the final layers where the predictions are performed. In the context of KT, students attempt exercises belonging to a specific concept to strengthen that concept. Hence, residual connection can help propagating the embeddings of the recently solved exercises to the final layer making it easier for model to leverage the low layer information. A residual connection is applied after both self-attention and feed forward layer.

Layer normalization: In [1], it was shown that normalizing inputs across features can help in stabilizing and accelerating neural networks. We used layer normalization in our architecture for the same purpose.Layer normalization is also applied at both the self-attention and feed forward layer.

Prediction layer:
Finally, each row of the matrix $\mathbf{F}_i$ obtained above is passed through the fully connected network with Sigmoid activation to predict the performance of the student.

$$p_i = \text{Sigmoid}(\mathbf{F}_i\mathbf{w} + \mathbf{b}), \quad (6)$$

where p_i is a scalar and represents the probability of student providing correct response to exercise e_i, $\mathbf{F}_i$ is the ith row of $\mathbf{F}$ and $\text{Sigmoid(z)} = 1/(1 + e^{-z})$

Network Training: The objective of training is to minimize the negative log likelihood of the observed sequence of student responses under the model. The parameters are learned by minimizing the cross entropy loss between p_t and r_t.

$$\mathcal{L} = -\Sigma_t(r_t \log(p_t) + (1 - r_t) \log(1 - p_t)) \quad (7)$$

3. EXPERIMENTAL SETTINGS

3.1 Datasets

To evaluate our model, we used four real-world datasets and one synthetic dataset.

- *Synthetic*[1]*:* This dataset is obtained by simulating 4000 virtual students' answering trajectories. Each student answers the same sequence of 50 exercises, which are drawn from 5 virtual concepts with varying difficulty level.
- *ASSISTment 2009*[2] *(ASSIST2009):* This dataset is provided by ASSISTment online tutoring platform and is widely used for KT tasks. We conducted our experiments on the updated "skill-builder" dataset. The dataset is sparse as the density of this dataset is 0.06, shown in Table 2.
- *ASSISTment 2015*[3] *(ASSIST2015):*ASSISTment 2015 contains students' responses on 100 skills. There are 19,917 students and 708,631 interactions. Although the number of records in this dataset is more than ASSISTment 2009, the average number of records per student is smaller because the number of students is larger. This dataset is the most sparse of all the available datasets, with a density of 0.05.

[1]https://github.com/chrispiech/DeepKnowledgeTracing/tree/master/data/synthetic
[2]https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data/skill-builder-data-2009-2010
[3]https://sites.google.com/site/assistmentsdata/home/2015-assistments-skill-builder-data

Table 2: Dataset Statistics

Datasets	#Users	#Skill tags	#Interactions	#Unique Interactions	Density
Synthetic-5	4000	50	200K	200K	1
ASSIST2009	4417	124	328K	35K	0.06
ASSIST2015	19917	100	709K	102K	0.05
ASSIST-Chall	686	102	943K	57K	0.81
STATICS	333	1223	190K	129K	0.31

The columns corresponding to #Users, #Skill tags and #Interactions represent the number of students, total number of exercise tags and the number of records, respectively. The column Density represents the density of each dataset (i.e., Density = #Unique Interactions/(#Users × #Skill tags)).

- *ASSISTment Challenge (ASSISTChall):* This data is obtained from ASSISTment 2017 competition[4]. It is the richest dataset in terms of the number of interactions with 942,816 interactions, 686 students and 102 skills. This dataset is the most dense dataset of all the available datasets because its density is 0.81.
- *STATICS2011 (STATICS):* This dataset contains the interaction from an engineering statics course with 189,927 interactions, 333 students and 1223 skill tags. We adopted the processed data from [11]. It is also a dense dataset with a density of 0.31.

The complete statistical information for all the datasets can be found in Table 2.

3.2 Evaluation Methodology

Metrics: The prediction task is considered in a binary classification setting i.e., answering an exercise correctly or not. Hence, we compare the performance using the Area Under Curve (AUC) metric.

Approaches: We compare our model against the state-of-the-art KT methods, DKT [6], DKT+ [10], and DKVMN [11]. These methods are described in the introduction.

Model Training and parameter selection: We trained the model with 80% of the dataset and test it on the remaining. For all the methods, we tried the hidden state dimension $d = \{50, 100, 150, 200\}$. For the competing approaches, we used the same hyperparameters as reported in their respective papers. For initialization of weights and optimization, we used a similar procedure as [10]. We implemented SAKT with *Tensorflow* and used ADAM [5] optimizer with learning rate of 0.001. We used a batch size of 256 for the ASSISTChall dataset and 128 for the others. For datasets with a larger number of records, e.g., ASSISTChall and ASSIST2015, we used a dropout rate of 0.2, while for the remaining datasets, we used a dropout rate of 0.2. We set the maximum length of the sequence, n as roughly proportional to the average exercise tags per student. For ASSISTChall and STATICS dataset we use $n = 500$, for the ASSIST2009 $n = 100$ and 50 , for the synthetic and ASSIST2015 datasets n is set to 50.

[4]https://sites.google.com/view/assistmentsdatamining

Table 3: Student Performance prediction comparison.

Datasets	AUC				
	DKT	**DKT+**	**DKVMN**	**SAKT**	**Gain%**
Synthetic	0.823	0.824	0.822	**0.832**	0.97
ASSIST2009	0.820	0.822	0.816	**0.848**	3.16
ASSIST2015	0.736	0.737	0.727	**0.854**	15.87
ASSISTChall	**0.734**	0.728	0.689	**0.734**	0.00
STATICS	0.815	0.835	0.814	**0.853**	2.16
Average	0.786	0.789	0.773	**0.824**	4.43

[1] **Bold** numbers are the best performance.

[2] The reported results are obtained by the best hyperparameter selection for each dataset individually.

4. RESULTS AND DISCUSSION

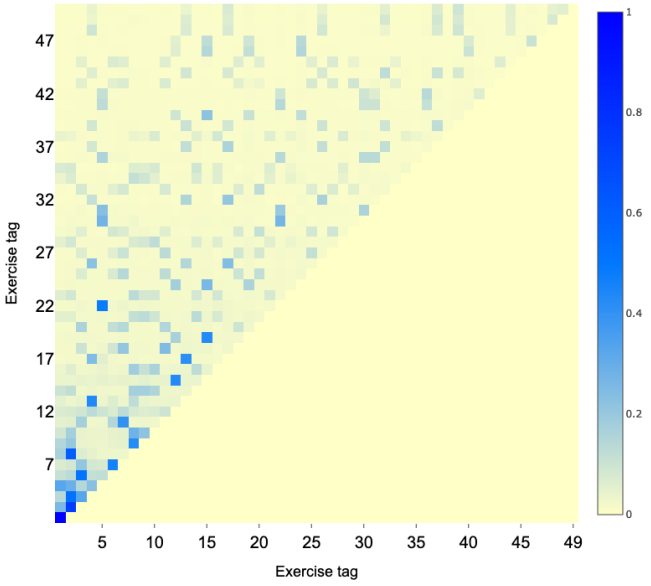
Student Performance Prediction: Table 3 shows the performance comparison of SAKT with the current state-of-the-art methods. On the Synthetic dataset, SAKT performs better than the competing approaches, achieving an AUC of 0.832 compared to 0.824 by DKT+. Even though Synthetic is the most dense dataset, SAKT outperforms RNN based methods because of the methodology used for generating Synthetic. For this dataset, each individual exercise is derived from only one concept. The probability of a student answering an exercise from this dataset correctly is determined using Item Response Theory [8] as, $p(\text{correct}|\alpha, \beta) = c + \frac{1-c}{1+\exp(\beta-\alpha)}$, where c denotes the probability of guessing it correctly, α and β are randomly chosen numbers to indicate the concept ability and exercise difficulty, respectively. Thus, in this dataset, the exercises belonging to the same concept are strongly correlated. SAKT, unlike other benchmarks, directly attempts to identify exercises belonging to the same concept and hence performs better than other methods. On ASSIST2009, SAKT performs better than competing approaches, gaining a performance improvement of 3.16% over the second best performing method. For ASSIST2015 dataset, SAKT shows an impressive improvement of 15.87%. We attribute this gain to the fact that attention mechanism leveraged by SAKT can learn and generalize well even when the dataset is sparse, which is the case with ASSIST2015 as its density is the least among the other datasets. For STATICS2011, our method achieves a performance improvement of 2.16% compared to DKT+. For ASSISTChall, our method performs at par with DKT. This can be attributed to the fact that ASSISTChall is the most dense dataset of all the real-world datasets.

Attention weights visualization: Visualizing the attention weights between the elements of past interactions (which serve as keys) and the exercise that the student is going to solve next (which serves as query) can help in understanding which exercises in the past interactions are relevant to the query exercise. With this motivation, we compute the sum of attention weights of each exercise pair $(e1, e2)$ across all the sequences where $e1$ serves as query and interaction with exercise $e2$ serves as key. We then normalize the attention weights so that the sum of the weights for each query is one. This results in a *relevance* matrix in which each element, $(e1, e2)$ represents the influence of $e2$ on $e1$. We perform our analysis on Synthetic because this dataset was generated with known hidden concepts and hence the ground

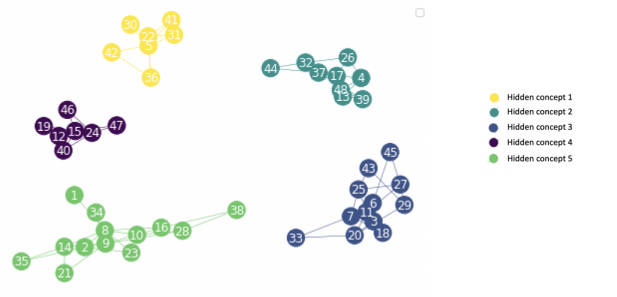
Table 4: Example of attention weights for some sequences in ASSIST2009 dataset.

Exercise tag	Past Interactions
Scale Factor	(Probability of Two Distinct Events,1): 0.000001, (Circle Graph, 1): 0.0001, (Circle Graph,1):0.001, (Division Fractions, 0): 0.99
Ordering integers	(Intercepts,0): 0.21, (Ordering positive decimals,1): 0.611, (Multiplication whole numbers,1): 0.09, (Proportion,1):0.033
Rate	(Interior Angles Figures, 0):0.005, (Algebraic Simplification,0) : 0.009, (Rate,0):0.5, (Interior Angles Figures, 0):0.1, (Algebraic Simplification,0) : 0.12

- The columns corresponding to Exercise tag refers to the query (i.e., the exercise for which we have to predict the student's performance) and Past Interactions refers to the sequence of interactions that has been observed for that student, respectively.
- red colored elements in the right column represent the most important element among the past interaction elements.



(a) Heatmap depicting the attention weights between each pair of exercises. Note that, the weight assigned for pair (i, j), where $j > i$ is always zero because all the sequences consists of exercises in the same order from



(b) Graph depicting the relevance between exercises. The relevance is determined by the attention weights learned between the exercises using SAKT. We observe a perfect clustering of latent concepts.

Figure 3: Visualizing attention weight of Synthetic dataset.

truth regarding the relevance of different exercises are known to us. Figure 3a shows the heatmap corresponding to the *relevance* matrix of exercises in Synthetic. For Synthetic, all the sequences consist of all exercise tags in the same sequence starting from 1 to 50.

In order to build the influence graph between the exercise tags, as shown in Figure 3b, we use the *relevance* matrix. Firstly, we draw out the first exercise in the sequence that belongs to each hidden concept, and visit each row of the *relevance* matrix, and connect the exercise corresponding to that row to the first two exercises ranked based on edge weight, which is proportional to the attention weights between the pair of exercises. We can see that the based on the attention weights, we are able to achieve the perfect clustering of the exercise tags based on the hidden concepts from which they are derived. An interesting observation is that two exercises which occur far apart in the sequence but belonging to the same concept can be identified by SAKT. For example, as shown Figure 3b a query on exercise 22 assigned most weight to the key with exercise 5 even when they occur far apart in the sequence.

Two exercises which are relevant to each other tend to have high attention weights as the performance on one of them impacts the performance on the other. Additionally, in the real-world scenario, the exercises which occur close in the sequence tend to belong to the same concept. Thus, we expect that the attention weights biased towards the exercises that occur recently in the interaction sequence. To illustrate this, we manually analyzed ASSIST2009 dataset to visualize the attention weights for some selected samples. Table 4 shows some of the exercises along with the past interactions and attention weights assigned to each interaction.

Ablation Study: Table 4 shows the performance of default SAKT architecture and all the variants on all the datasets (with $d = 200$).

No Positional Encoding (PE): In this variant of the default architecture, we removed the positional encoding. As a result, the attention weights assigned for predicting the performance of student on a particular exercise depends only on the interaction embedding, without being affected by its position in the sequence. In case of ASSIST2009 and ASSIST2015, the dataset is sparse and hence the impact of removal of PE is not much pronounced as is the case with the dense dataset such as ASSISTChall and STATICS.

*No Residual Connection (RC):*RCs shows the importance of low level features i.e., the interaction embedding while making the prediction. Since our architecture is not very deep,

Table 5: Ablation Study

Architecture	Synthetic	ASSIST 2009	ASSIST 2015	ASSIST Chall	STATICS
Default	**0.832**	**0.848**	0.854	**0.734**	**0.853**
No PE	0.827	0.842	0.849	0.715	0.832
No RC	0.823	0.847	**0.857**	0.709	0.834
No Dropout	0.832	0.845	0.851	0.711	0.840
Single head	0.823	0.828	0.845	0.709	0.851
0 block	0.826	0.837	0.822	0.634	0.819
2 blocks	0.827	0.840	0.853	0.724	0.845

the RC do not contribute much to the performance of the model. In fact removal of residual connection gives better performance than default for the ASSIST2015 dataset.
No Dropout: Dropout is used in neural network to regularize the model so that it can generalize better. Overfitting of the model is more effective for dataset with less number of records compared to the number of parameters of model. As a result, role of dropout is more effective for ASSIST2009 dataset and STATICS dataset.
Single head: Instead of using 5 heads as is the case in default architecture, we tried a variant of using only one head. Multiple heads help in capturing the attention weights in different subspaces. Using single head consistently drops the performance of SAKT on all the datasets.
No block: When no self-attention block is used the prediction of the next exercise depends only on the last interaction. It can be seen that without attention block the performance is significantly worse than that of default architecture.
2 Blocks: Increasing the number of blocks of self-attention increases the number of parameters of the model. However, in our case this increase of parameters does not prove to be useful in improving the performance. The reason being an important aspect of prediction of performance of student at an exercise is dependent on his performance on the past relevant exercises. Adding another block of self-attention makes the model more complex.

Training efficiency: Figure 4 demonstrates the efficiency of various methods based on their run times on GPU during the training phase. Comparing the computational efficiency, SAKT only spends 1.4 seconds in one epoch which is 46.42 less than the time taken by DKT+ (65 seconds/epoch), 32 times less than DKT (45 seconds/epoch) and 17.33 times less than DKVMN (26 seconds/epoch). We conducted the experiments on a single GPU of type NVIDIA Titan V.

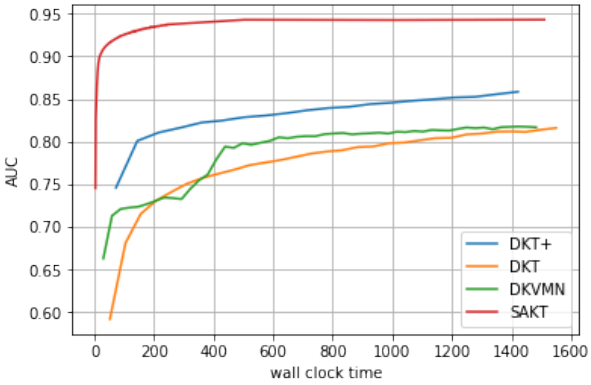


Figure 4: Training Efficiency on ASSIST2009 dataset.

5. CONCLUSION AND FUTURE WORK

In this work, we proposed a self-attention based knowledge tracing model, SAKT. It models a student's interaction history (without using any RNN) and predicts his performance on the next exercise by considering the relevant exercises from his past interactions. Extensive experimentation on a variety of real-world datasets shows that our model can outperform the state-of-the-art methods and is an order of magnitude faster than the RNN-based approaches.

6. REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[3] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. *CoRR* abs/1808.09781 (2018).

[4] Mohammad Khajah, Robert V Lindsey, and Michael C Mozer. 2016. How deep is knowledge tracing? *arXiv preprint arXiv:1604.02416* (2016).

[5] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[6] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. 2015. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*. 505–513.

[7] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065* (2016).

[8] John Self. 1990. Theoretical foundations for intelligent tutoring systems. *Journal of Artificial Intelligence in Education* 1, 4 (1990), 3–14.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[10] Chun-Kit Yeung and Dit-Yan Yeung. 2018. Addressing two problems in deep knowledge tracing via prediction-consistent regularization. *arXiv preprint arXiv:1806.02180* (2018).

[11] Jiani Zhang, Xingjian Shi, Irwin King, and Dit-Yan Yeung. Dynamic key-value memory networks for knowledge tracing. In *Proceedings of the 26th International Conference on World Wide Web*.