Riiid! Answer Correctness Prediction

Tiago Antunes[1], **Sahand Sabour**[1], **Henry Zheng**[1]
Department of Computer Science and Technology
Tsinghua University, Beijing, China
{vazama10,shanm20,zheng-jh20}@mails.tsinghua.edu.cn

Abstract

Knowledge Tracing is the process of modeling students' understanding of a knowledge concept based on their interactions with a series of exercises and learning activities. With the recent COVID-19 pandemic and rise of online education, the field of knowledge tracing is highly essential as it provides students with their own customized learning experience. In this project, we were tasked to employ a new knowledge tracing model and/or improve upon previous state-of-the-art work to accurately predict whether a student would answer a given exercise correctly, given his/her performance history. We implemented two commonly used transformer-based models using PyTorch. We further analyze the problem to understand what were our difficulties when dealing with the task, what were the steps we took and the results that were obtained.

1 Introduction

Due to the recent COVID-19 pandemic and rise of need for social distancing, numerous schools have been temporarily closed. Many organizations have transferred their events and activities to virtual environments. This has led the society to switch to online educational courses, material, and environments. As these platforms provide a significant amount of information for students' learning activities, attentiveness, and performance, they can provide valuable data for further analysis to improve on the online learning experience. Hence, the topic of Knowledge Tracing has become highly essential and has gained considerable attention from the research community. Knowledge tracing (KT) can be defined as the task of modeling students' comprehension regarding knowledge concepts (KCs) upon their interactions with a series of learning tasks, exercises, and coursework. Accordingly, an accurate KT model can be utilized to predict students' performance in future interactions, which was our goal during this project. Hence, knowledge tracing is crucial for providing students with personalized learning experiences, which allows them to choose subjects based on their needs and obtain relative knowledge with appropriate tasks and exercises.

In general, KT can be formalized as a supervised learning task, where given the exercise history $E = e_1, e_2, ..., e_t$ and the binary student response history $R = r_1, r_2, ..., r_t$ (with 1 representing a correct response), the aim is to predict the probability of the student submitting a correct response to the next exercise; i.e. $p(r_{t+1} = 1|e_{t+1})$. Early approaches focused on deep learning for constructing KT models, with Deep Knowledge Tracing (DKT) [3] and Recurrent Neural Networks (RNN) gaining more attention. Several approaches, such as Dynamic Key-Value Memory Networks (DKVMN) [6], which utilized key and value matrices to model the relationship between students' knowledge level and the provided exercises, have also gained some attention, mainly due to their superior interpretability in comparison with prior deep learning models. Recent work has built upon the success of transformers [5] and has produced state-of-the-art performance [10].

In this paper, we introduce two highly accurate, transformer-based Knowledge Tracing models that were utilized for this competition: Self-Attentive Knowledge Tracing (SAKT) [9] and Separated Self-Attentive Neural Knowledge Tracing with Temporal Features (SAINT+) [11]. Accordingly, we

Advanced Machine Learning (2020) Final Project, Tsinghua University, Beijing, China.

demonstrate and explain our implementations of these models and the employed methodology. Lastly, we provide the obtained evaluation results of our implemented models based on the *Riiid! Answer Correctness Prediction* dataset, which was provided for this competition on *Kaggle*.

2 Related Work

2.1 Bayesian Knowledge Tracing

The Bayesian Knowledge Tracing model initially proposed by Corbett & Anderson [1] has been used in several parts of the industry, which introduces the *learning parameters* and the *mediating parameters* split **S**, the probability that although the student has the knowledge he might give a wrong answer, and guess **G**, the probability of guessing it right although the student has no knowledge on the topic, building the basis for Knowledge Tracing. Nevertheless, this model has a few problems [2] regarding its parameters that can be solved as, for example, *Model Degeneracy*.

2.2 Deep Neural Network Knowledge Tracing

Recently, deep neural networks have become the most popular tool to approach Knowledge Tracing [4] due to their higher performance upon the Bayesian Knowledge Model which is widely seen as the base for this problem. The usage of Recurrent Neural Networks (RNN) is due to them being able to represent knowledge over time, relating automatically to the temporal characteristic of the problem and being able to learn from data instead of being hand-tuned similarly to previous work on the area, giving origin to Deep Knowledge Tracing (DKT). It was also introduced the concept of Long Short-Term Memory (LSTM) which solves the vanishing gradient problem and also improved the performance of the RNN in the Knowledge Tracing problem over BKT and PFA.

Improvements can still be made upon the given RNN with LSTM model taking into account the human factor of learning something. As stated in previous research [7], each person has a different learning ability and therefore need to be considered differently than other with a different learning ability (while also taking into account the past knowledge at a time). At the same time, knowledge that has been acquired recently is much more likely to still be remembered comparing to previous one [13] and the short-term memory effect needs to be taken into account. For this, it was used Convolutional Neural Networks (CNN) which was able to include this detail into its model, ending up with Convolutional Knowledge Tracing (CKT) which, due to its convolutional nature, requires a change of the underlying data representation from one-hot encoding to a matrix format.

2.3 Graph Neural Network Knowledge Tracing

Graphical Knowledge Tracing (GKT) approach tackles the learning of high-order relation information between the knowledge and question[13, 12], the long term dependencies of the questions [13] and at the same time maintaining the interpret-ability of the model [8]. In KT scenarios, some questions may require understanding of multiple knowledge areas to answer a single question. Conversely, a single knowledge area may also be used to answer different kinds of questions. These predictive models assumes the students' mastery of knowledge means the students could answer the corresponding question correctly. However, this assumption neglects the idea that the level of difficulty may affect the accuracy to different questions answered by the student [13]. GNN approach aggregates knowledge and question embedding to mitigate these issues. Including a attention mechanism to prevent loosing of long-term dependencies information as the model training goes further [13]. Other than these benefits, GNN models can also provide better interpretable prediction results that BKT and DKT models [8].

2.4 Key-Value Memory networks

One of the main problems of Bayesian-based approaches such as BKT [2] is their lack of ability to establish the relationship between different knowledge concepts. In addition, deep models such as DKT [4] fail to track the level of comprehension for each individual knowledge concept as these models merely produce a single collective state for all KCs. Hence, in order to tackle the shortcomings of these two types of approaches, Dynamic Key-value Memory Networks (DKVMNs)[6] were introduced. In these networks, key and value matrix pairs are used to construct a more powerful

memory than LSTMs in terms of storing the history of student performances. Accordingly, at each time stamp, input features or tags are written as immutable keys, the exercise-response tuple (e_t, r_t) is written to or read from the value matrix, and the probability of a correct response $p(r_{t+1} = 1|e_{t+1})$ is produced as output. Therefore, DVKVMs are able to overcome the barriers of both BKTs and DKTs by realizing the relationship between different KCs while providing the ability to track each KC individually.

3 Methodology

3.1 Problem Definition

The main objective of our DAKT model is to accurately predict students' performance on future exercises, given a history of their behavior on the *Santa* platform. Essentially, for ***n*** number of students' behavior history can be represented as an adjacency matrix $(R, F)^n$, where $F_i \in \mathbb{R}^d$ and $R_i \in \{0, 1\}$. F_i is a feature vector for the given entries in the dataset, and R_i is the corresponding user answer; Hence, the problem can be formulated as correctly predicting the following probability

$$P(R_i = 1|F_i)$$

where F_i is the current question that the user needs to guess.

3.2 Input Representation

Our model accepts the *Riiid! Answer Correctness Prediction* dataset, which is provided on *Kaggle*, as input. This dataset is in a question-response sequence format. Its major property is that the questions are collected in bundles; that is, questions regarding a common topic are provided as a collective bundle. In addition, for this competition, the given dataset is divided into three parts: train data, questions, and lectures.

As the name suggests, the train data is the essential part of training the model and it contains the following attributes:

- Row ID: A unique ID for each entry in the dataset.
- Timestamp: The duration between the user's first event completion and this interaction.
- User ID: A unique ID for the student who is responsible for this interaction.
- Content ID: A unique ID for the interaction corresponding to this entry.
- Content Type ID: Shows whether the relative interaction was a question or a lecture.
- Task Container ID: Represents the bundle for this interaction.
- User Answer: The answer provided by the student.
- Answered Correctly: Shows whether the provided answer is correct.
- Prior Question Elapsed Time: The average time (in milliseconds) that it took the student to complete the prior bundle.
- Prior Question Had Explanation: Shows whether the student was provided with an answer or an explanation upon completing the previous bundle.

The lectures and questions portion of the dataset are utilized to provide more information regarding a specific interaction, based on the mentioned *Content ID*. These portions include the following attributes:

- ID: A foreign key for the Content ID in the train data portion.
- Bundle ID: A unique ID for the corresponding bundle (only available for questions).
- Correct Answer: The correct answer for the given interaction (only available for questions).
- Part: Top level category for this interaction based on the sections of the TOEIC test.
- Tags: Detailed tag codes for the given interaction. Questions have one or more tags while lectures can only have one tag.
- Type: a brief description of the purpose of this interaction (only available for lectures).

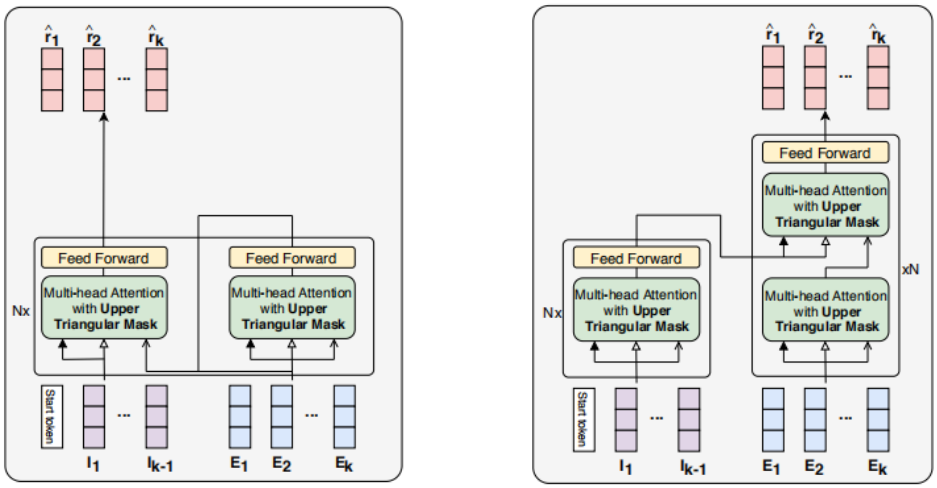
3.3 Dataset Insights

Based on initial exploratory data analysis on the provided dataset, we found several interesting relationships between the features:

- There are a total of 393656 students, from which around 4% have answered more than 1500 questions. They are not representative of the whole dataset. 60% of the users answered only up to 61 questions.
- Most of the users use the system for a short period of time and drop out after
- One third of the questions are answered incorrectly
- If the prior question had explanation, the students have higher tendency of getting the answer correctly.
- The percentage of answered questions increases with the number of questions answered, but there are many outliers (**This doesn't include same tag questions only**)
- There's no need to be careful with lectures and questions mixed in the same task container as they all have the same timestamp. It's possible to assume previous and post questions times, which makes it simpler.
- Students get more correct answers on some specific tags, since the number of answers per tag also matters.

3.4 Attention-based Knowledge Tracing

The models implemented for this Kaggle competition are Self-Attentive Knowledge Tracing (SAKT)[9] and SAINT+[10, 11]. SAKT and SAINT+ model has similar architecture, and both are created based on the transformer model [5].



(a) SAKT

(b) SAINT

Figure 1: Implemented model architectures [10]

Transformers use attention-mechanism for transforming a sequence of input into another sequence by using encoders and decoder in the architecture. Encoder and decoder in transformers are composed of stacked of multiple Multi-Headed Attention network and Feed Forward Networks (FFN). During training, the input sequences and targeted sequences are transformed into embedding representations of size S for input sequence and size T for target sequence where S is the number of the features of each entry in input sequences and T is the number of features of each entry in the output sequences. In our case, an input sequence is a sequence of questions given to a specific user plus the meta data of the characteristics of him/her answering the question. Then an output sequence is a sequence of the correctness of his/her response to the questions. Then input sequence embedding and target sequence embedding are fed into the encoder and decoder respectively.

The Multi-Headed Attention networks of the transformers are multiple attention networks that are applied multiple times to the same input sequence with difference weights. The attention network takes inputs Q_{in}, K_{in}, and V_{in} [10]. In our implementation, Q_{in} is a matrix that consist of stacked queries, which are vector representations of an single question and its meta data of a user; K_{in} is a matrix that consist of staked keys, which are vector representations of the questions done and their meta data the by the users; and finally, V_{in} is a matrix of the values, which are vector representations of the correctness of the responses of the user to the questions. The attention network projects the Q_{in}, K_{in}, and V_{in} to latent space by multiplying with learned weights W^Q, W^K, and W^V then yields Q, K, and V respectively. In calculation of attention weights, a upper triangular mask was applied to the matrix QK^T to prevent the model to attend to the future features in the sequence. This is done by setting $-\infty$ to the values to be hidden then applying a softmax operation. Then the attention network head, $head_i$, is by multiplying the attention weights to the V_i, as shown as follows,

$$head_i = Softmax(Mask(\frac{Q_i K_i^T}{\sqrt{d}}))V_i$$

Then a Multi-Headed Attention Network is a concatenation of h attention heads multiplied by learned weight W^O as denoted by,

$$MultiHead(Q_{in}, K_{in}, V_{in}) = Concat(head_1, ..., head_h)W^O$$

Both the encoder and the decoder blocks include a Feed Forward Network (FFN) which occurs after the multi-headed attention layer. This is a very thin layer that simply does

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2$$

where W_1, W_2 are the weights and b_1, b_2 are biases. [10]

The SAKT and SAINT models have similar transformer network architecture. Their only difference in our implementations are the features and embedding used and the number of Multi-Headed Attention layers in encoder and decoder modules.

3.5 Input Preprocessing

The dataset used in the competition was *Riiid! Answer Correctness Prediction*, which has been described in detail in subsection 3.2. In order to have an efficient model, pre-processing of the provided data is highly necessary as it allows us to extract and highlight the important features. It also enables us to remove unwanted features (i.e. noise) that could severely harm the model.

We transformed the dataset from a per-row interaction into a user-data format. To this end, we grouped all user data into a single object for ease of access. This allows for increased data locality and speed when looking for the user's past interactions. When doing this transformation, we also limited the number of user interactions to at a maximum of the last $windowsize$ elements. As stated in subsection 3.3, many users replied to a low amount of questions. This means that given a sufficiently large $windowsize$, many users would stop having an enough amount of interactions, thus requiring us to *pad* the data as to guarantee that the data server had the same length. This also included generating a mask for informing the models of what was padded or not.

In SAINT+ [11], a number of embeddings are used, which are mostly derived from the input data. However, one of implemented embeddings (*lagtime*), which refers to the time between the submission of the previous exercise and the start of the current exercise, does not exist as an attribute of the dataset and must be calculated individually. Our calculation was as follows: given the timestamp of a new interaction T_i and the timestamp of the previous user interaction T_{i-1} we can then set the *lagtime* of our new interaction L_i as

$$L_i = \min((T_i - T_{i-1})/1000, 300)$$

thus getting the *lagtime* in terms of seconds as a value between $[0, 300]$.

The dataset also provided meta-data about lectures and questions. We found it challenging to include the lectures in our models. Moreover, the addition of lecture did not seem to have a major impact on the obtained results. Hence, we simply ignored these interactions. With respect to the questions, we extracted the information relative to the *part* of each questions and then assigned this part to each

interaction by using a dictionary with $(question\_id, part)$ pairs as a way of efficiently mapping the relationship.

In summary, our processed data contains the following attributes:

1. *user_id*
2. a collection of queues with elements:
    - *content_id*
    - *is_correct*
    - *task_container_id*
    - *lagtime*
    - *prior_question_elapsed_time*
    - *part_id*
    - *padding_mask*

When doing the inference, to improve the accuracy of the model, we would keep the information from before loaded in memory and use it to fetch the past user interactions. We would then update it, by removing the oldest interaction and adding the new one with the corresponding information, thus maintaining the $windowsize$ of the data.

4 Results and Discussion

Upon successful submission of our implemented models' predictions to Kaggle, our obtained Area Under Curve (AUC) scores were calculated and are provided respectively in the table below.

For SAINT+ we used a number of layers, window size, model dimension, dropout and batch size of, respectively, 4, 100, 512, 0, 64. For SAKT we used window size and dropout of 128 and 0.2, respectively.

Model	Public score	Private score
SAINT+ (3 epochs)	0.626	0.606
SAINT+ (20 epochs)	0.592	0.604
SAINT+ (encoder only)	0.681	0.683
SAKT	**0.774**	**0.773**

Table 1: Competition Results

The model that performed the best was SAKT with a big difference from SAINT+. We believe this is due to an error in our SAINT+ implementation and also due to overfitting.

Both our SAINT+ and SAKT implementations are not using the padding mask, as it was mentioned in subsection 3.5. Because we used PyTorch as the framework to build our model, in the end we stumbled upon an open issue on the *MultiHeadAttention* module which resulted in our results coming back as NaN. This further influenced our approach to use the padding mask, which reduced the accuracy of the model.

Another thing to be mentioned was that our SAKT model used a *Feed Forward Network* in the end of its output before applying a linear transformation to output the result. Our SAINT+ is using an encoder-decoder paradigm (in which both encoder and decoder have a Feed Forward Network within them) and in the end is followed by a linear transformation. We believe that if before this linear transformation we included a Feed Forward Network we might be able to preserve more non-linear relationships, thus improving the result of the SAINT+ model, which is different from the original paper itself. We base this idea after analyzing a public notebook from the competition which is very similar to our approach with the exception of that detail.

Another thing to be noticed from Table 1 is our encoder only SAINT+. We initially submitted an incomplete version of SAINT+ that did not have a decoder. It consisted of just a few encoder blocks stacked on each other followed by a linear transformation. It also missed some extra details such as the *lagtime* embeddings and the attention mask in comparison to the actual SAINT+ model. This model scored higher than our final model, from which we can derive two conclusions:

- Data is noisy and the extra embeddings we used were not correctly calculated. On the previously mentioned public notebook, the author split the *lagtime* into multiple *lagtimes*, which correspond to seconds, minutes, and days respectively. We believe that this is a beneficial implementation for our future work.
- Following papers to implement their proposed models may not a good idea. We should work on establishing baselines to the given problem and try to improve them instead of directly implementing a paper without realizing it might not produce similar results in the given dataset.

5 Conclusion

To conclude, we analyze the project from the point of view of beginners to the field of Machine Learning. We see this competition as a great first experience working on a big project where we were able to apply much of the knowledge we learned during the semester. We believe it greatly contributed for us to improve our model building and paper reproduction skills and understanding more about the current area of ML. This competition allowed us to understand more how temporal relationships are built and we believe this new knowledge to be very valuable. In the future, we hope to build upon our lessons learned and achieve higher rankings in similar competitions.

References

[1] A T Corbett and John R Anderson. *Knowledge Tracing*. 1995. URL: https://link.springer.com/article/10.1007/BF01099821.

[2] Ryan S.J.D. Baker, Albert T. Corbett, and Vincent Aleven. "More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing". In: *Lecture Notes in Computer Science* 5091 LNCS (2008), pp. 406–415. DOI: 10.1007/978-3-540-69132-7-44.

[3] Chris Piech et al. "Deep knowledge tracing". In: *Advances in Neural Information Processing Systems* 2015-Janua (2015), pp. 505–513. ISSN: 10495258. arXiv: 1506.05908.

[4] Xiaolu Xiong et al. "Going deeper with deep knowledge tracing". In: *Proceedings of the 9th International Conference on Educational Data Mining, EDM 2016* (2016), pp. 545–550.

[5] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[6] Jiani Zhang et al. "Dynamic key-value memory networks for knowledge tracing". In: *26th International World Wide Web Conference, WWW 2017* (2017), pp. 765–774. DOI: 10.1145/3038912.3052580. arXiv: arXiv:1611.08108v2.

[7] Sein Minn et al. "Deep Knowledge Tracing and Dynamic Student Classification for Knowledge Tracing". In: *Proceedings - IEEE International Conference on Data Mining, ICDM* 2018-Novem (2018), pp. 1182–1187. ISSN: 15504786. DOI: 10.1109/ICDM.2018.00156. arXiv: 1809.08713.

[8] Hiromi Nakagawa, Yusuke Iwasawa, and Yutaka Matsuo. "Graph-based knowledge tracing: Modeling student proficiency using graph neural network". In: *Proceedings - 2019 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2019* 2016 (2019), pp. 156–163. DOI: 10.1145/3350546.3352513.

[9] Shalini Pandey and George Karypis. "A self-attentive model for knowledge tracing". In: *EDM 2019 - Proceedings of the 12th International Conference on Educational Data Mining* (2019), pp. 384–389. arXiv: 1907.06837.

[10] Youngduck Choi et al. "Towards an Appropriate Query, Key, and Value Computation for Knowledge Tracing". In: (2020), pp. 341–344. DOI: 10.1145/3386527.3405945. arXiv: 2002.07033.

[11] Dongmin Shin et al. "SAINT+: Integrating Temporal Features for EdNet Correctness Prediction". In: (2020), pp. 1–10. arXiv: 2010.12042. URL: http://arxiv.org/abs/2010.12042.

[12] Hanshuang Tong, Yun Zhou, and Zhen Wang. "HGKT : Introducing Problem Schema with Hierarchical Exercise Graph for Knowledge Tracing". In: (2020). arXiv: 2006.16915. URL: http://arxiv.org/abs/2006.16915.

[13] Shanghui Yang et al. "Deep Knowledge Tracing with Convolutions". In: (2020). arXiv: 2008.01169. URL: http://arxiv.org/abs/2008.01169.