Exploring deep knowledge tracing to predict student performance

François Chesnay, Heidi Kim, Cameron Mohne
{fchesnay, hyunsunk, mohnec1}@stanford.edu

Abstract

As online education expands, posing a potential socioeconomic balancer, the quality of learning and feedback becomes increasingly important to aid people on their educational journey. We apply a SAINT-based model to EdNet, a large dataset of 100 millions student interactions, to predict student performance and also apply interpretability to identify key improvement areas. The AUC of our best performing transformer-based model is 0.81109. The ablation analysis of the model inputs provide insights, though our model would have benefited from also integrating the lectures as input.

1 Introduction

In recent years, rising costs in learning have reduced access to quality education, deepening systemic inequity in this sector. Lack of access to education has long been recognized as a key limiter of economic mobility and generational wealth in the long term. In order to broaden access to quality instruction, powerful but cost-effective learning solutions, such as computer-assisted technology, must be made available at scale.

To date, education research for online learning has focused on understanding students' knowledge state to predict the probability of correct responses, based on historical learning activity. Less work has applied explainability techniques to extract the information embedded in these trained deep learning models to grant personalized feedback. We believe adding this under-explored mechanism of tailored instruction has the potential to significantly improve the effectiveness of online learning platforms.

To approach the task of identifying personalized feedback, we built a model inspired by the SAINT knowledge tracing model, using a RNN+transformer (RNNT-KT). Given an input data set of online student learning activity (questions answered correctly/incorrectly, time elapsed per question, etc.), our model predicts students' likelihood of answering the subsequent question correctly. Finally, we perform an input ablation analysis to capture which input features carry the most weight when students are predicted to answer incorrectly. This work serves as a starting point for offering students tailored feedback at scale, especially if future improvements include lectures seen/unseen by a given student.

2 Related work

Liu et al., 2015 [1] presented a survey of knowledge tracing. With the advent of deep learning, an RNN-based Deep Knowledge Tracing (DKT) model has been proposed (Piech et al., 2015)[2], relying on long short-term memory (Hochreiter and Schmidhuber, 1997)[3]. Other KT using RNN includes [4][5].

,

Transformer-based models[6], such as AKT (Ghosh et al. 2020)[7], SAKT (Pandey and Karypis 2019)[8], SAINT (Choi et al. 2020a)[9], (Shanghui et al. 2020)[10] and SAINT+ (Shin et al. 2020)[11], have been proven to outperform other approaches.

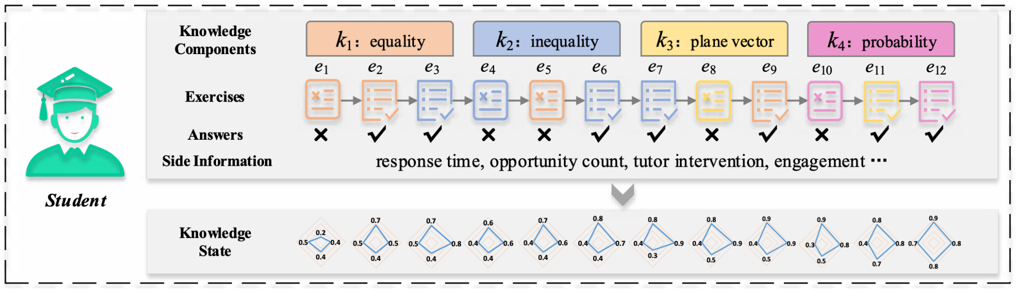
Sundararajan et al. (2017) presented an axiomatic attribution for deep networks with Integrated Gradients [12]. Clark et al. (2019) analyzed bert's attention [13], while looked at the issue of attention not being explanation[14].

3 Dataset and Features

3.1 Data Background

To date, EdNet is the world's largest open dataset for AI education, containing upwards of 100 million student-system interactions. The data was collected over 2 years by *Santa*, an AI tutoring service with more than 780K users in Korea, (Choi et al. 2020)[15] preparing students for the Test of English for International Communication (TOEIC). This data was made available for the Riiid! AIEd Challenge 2020 on Kaggle. For each student, we have the list of their past questions, answers, the relative timestamps, time spent for each question and whether they viewed the explanation after answering the question. We also have a list of lectures watched by users as they progress in their education. The metadata of questions and lectures can also be helpful in predicting the correctness of students' answers.

2



3.2 Challenges Related to our Dataset

The first challenge of the dataset is its size as `train.csv` has over 100 million rows and 10 columns, and the execution of the plain vanilla panda command **pd.read_csv** will result in an out-of-memory error. Thus, it is necessary to set the data type for the DataFrame columns with the argument dtype.

The second challenge is how the data is divided up. There are three files: `train.csv`, `questions.csv`, and `lectures.csv`. While joining `train.csv` with one of the others was doable, it proved difficult to join it with both. Despite trying several work arounds, even after we completed the join, the new structure for our data broke the existing model with much time wasted, for a fix to no avail. This means we had to cut out a large segment of potentially useful data.

3.3 Data Preprocessing

To feed the transformer, raw training data in the form of one activity per row, needs to be preprocessed. The first step is to encode categorical features into integer indices for embedding layers of the transformer. This step is also applied to the metadata tables (for questions and lectures).

We did some feature-engineering by creating extra features: (i) time lag (time interval from the last question bundle/container) - since all questions in the same bundle share a timestamp, they also share the same time lag; (ii) question popularity; and (iii) question difficulty, computed from the whole train table.

The train data table is then grouped by user id into one row per student, so that we have sequences representing the history of activities for each student. These sequences are the inputs of the transformers' embedding layers.

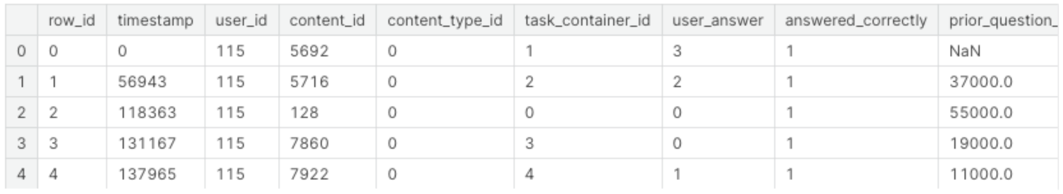
3.4 Data Characteristics and Split

We have 393,656 unique users in our train set. We have 13,782 content ids, codes for the user interaction in our train set, of which 13,523 are questions. User interactions represent answering questions or watching videos. We have 10,000 unique batches of questions or lectures.

Our training set is composed of 100 million user interactions. The cross-validation set is composed of 2.5 million interactions. Finally, the test set is also composed of 2.5 million interactions.

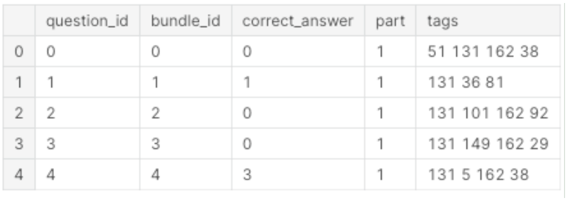
3.5 Example of Data

example_test.csv



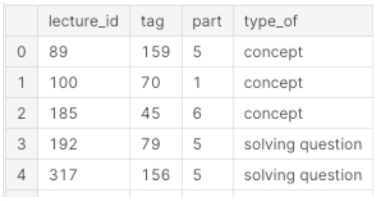
	row_id	timestamp	user_id	content_id	content_type_id	task_container_id	user_answer	answered_correctly	prior_question_
0	0	0	115	5692	0	1	3	1	NaN
1	1	56943	115	5716	0	2	2	1	37000.0
2	2	118363	115	128	0	0	0	1	55000.0
3	3	131167	115	7860	0	3	0	1	19000.0
4	4	137965	115	7922	0	4	1	1	11000.0

questions.csv



	question_id	bundle_id	correct_answer	part	tags
0	0	0	0	1	51 131 162 38
1	1	1	1	1	131 36 81
2	2	2	0	1	131 101 162 92
3	3	3	0	1	131 149 162 29
4	4	4	3	1	131 5 162 38

lectures.csv

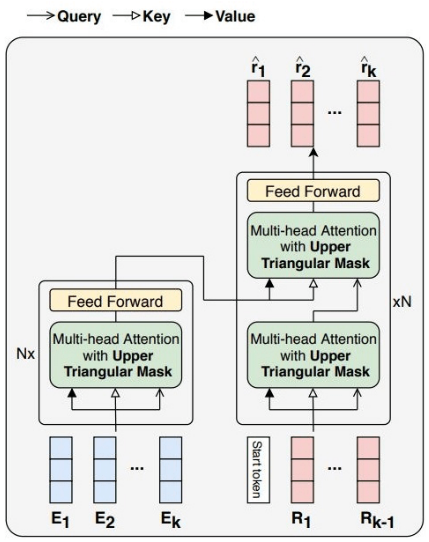


	lecture_id	tag	part	type_of
0	89	159	5	concept
1	100	70	1	concept
2	185	45	6	concept
3	192	79	5	solving question
4	317	156	5	solving question

4 Model Architecture

4.1 Description of our KT Model: a SAINT-based Model

Our model is inspired by the Saint and Saint+ based models. We intended initially to add the data for lectures seen by the students in our model. This addition proved to be cumbersome given the time constraints of the project, so the lectures were not included.


Query
Key
Value
$\hat{r}_1$
$\hat{r}_2$
$\hat{r}_k$
Feed Forward
Multi-head Attention with Upper Triangular Mask
Feed Forward
Multi-head Attention with Upper Triangular Mask
Multi-head Attention with Upper Triangular Mask
Nx
xN
Start token
E_1
E_2
E_k
R_1
R_{k-1}

The model consists of an encoder and a decoder which are stacks of several identical layers composed of multi-head self-attention and pointwise feedforward networks, as shown in Figure 2. The encoder takes the sequence of exercise embeddings as queries, keys and values, and produces output through a repeated self-attention mechanism. The decoder takes the sequential input of response embeddings as queries, keys and values; it then alternately applies self-attention and attention layers to the encoder output.

To feed the transformer, raw train data must be preprocessed, in the form of one activity per row. The first step is to encode categorical features into integer indices for embedding layers of the transformer. This step is also applied to the metadata tables (question metadata and lecture metadata). On the train table, we also added time lag (time interval from the last question bundle/container). Since all question of the same bundle share a timestamp, they also share one time lag. Two statistics on the questions - question popularity and question difficulty - are also computed from the whole train table. Adding these two features was proven (experimentally) to be helpful, at least in terms of convergence speed of the training process. The train data table is then grouped by user id into one row per student, so that we have sequences representing the history of activities for each student. These sequences act as the inputs of the transformers' embedding layers. As shown in the diagram, attention had to be masked with an upper triangle matrix, so that future samples can not be used to make predictions. We also include some feature engineering.

The main idea of using this transformer model is to learn the relationship between questions, and using LSTM compels the model to learn the sequential characteristic of the problems, giving more importance to recent activities. Relative to the original SAINT model, extensions in our model includes taking into account 'inter-container' leakage, an issue for datasets where questions are served/bundled together.

4.2 Implementation Details

Our model was developed using PyTorch 1.13.0 and Hugging Face's transformers library running on Ubuntu 20.04.5 LTS with Cuda 11.6. We ran the model for 10 Epochs, which took around 19.5 hours using one Nvidia Titan XP GPU.

5 Learning Method

5.1 Model Training and Hyperparameter Tuning

The window size, dropout rate, and batch size are set to 100, 0.1, and 128 respectively. We use the Adam optimizer with $lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e - 8$. The hyperparameters' values are presented in appendix 1. We tuned several hyperparameters, as presented below by ablation and assessed the impact on the AUC:

Table 1: hyperparameters tuning.

	Value A	AUC score on cross-validation set
First optimal hyperparameters		0.81027
without feature engineering		0.81004
↘ learning rate	3.0e-5	0.80990
↘ dropout probability	5%	0.80994
↗ dropout probability	15%	0.80989
↗content_embedding_size	512	0.81043

We noted that changes in learning rate and dropout had limited impact on the AUC, while increasing further the impact of content_embedding_size brought a minimal improvement in AUC prediction, at the expense of large computing costs.

6 Results and Analysis

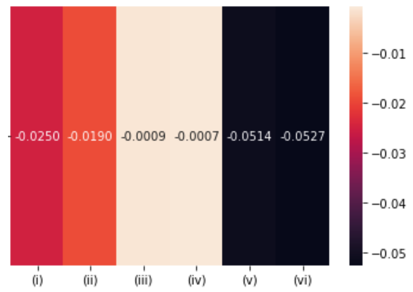
6.1 Evaluation Metrics and Baselines

We evaluated our RNNT based on the Area Under the Curve (AUC), a primary evaluation metric for KT modeling, which can be interpreted as the probability that a system will rank a randomly-chosen error above a randomly-chosen non-error, i.e. the AUC of a model that guesses 0 or 1 randomly should be 50%.

We computed two baselines, based on a random forest classifier and on LightGBM, a gradient boosting framework that uses tree based learning algorithms. The AUC of the baselines were 0.707 for the random forest classifier and 0.712 for LightGBM. The best baseline would be ranked 2,778 in the Kaggle competition out of 3,406.

6.2 Ablation Study

We had planned to provide results of model interpretability using the Captum library (importing IntegratedGradients, LayerConductance and NeuronConductance) , but it proved quite challenging due to the complexity of our model.We decided to instead perform an ablation study by systematically eliminating sections of the input, to see which elements of the input are more significant for the output.



Given our time and AWS budget constraints (our model for 10 Epochs takes around 20 hours to run), we decided to perform the ablation based on only 2 Epochs and on a reduced sample of the whole dataset. The rationale was that after 2 Epochs, the improvements in AUC very minimal. We are fully aware of the limitation of our approach, and that further work is required to produce more reliable analyses.

The graph above shows the impact of ablation study on the change AUC for (i) prior_question_elapsed_time, (ii) prior_question_had_explanation, (iii) difficulty, (iv) popularity, (v) part and (vi) tags.

The ablation study shows that the most significant factors explaining the quality of the prediction are part and tags. Prior_question_elapsed_time and prior_question_had_explanation contribute somewhat, and difficulty and popularity's contributions are nearly negligible.

7 Conclusion/Future Work

We were able to reproduce and hone a Knowledge Tracing model through feature engineering. The transformer models were the highest performing and have a clear predictive advantage over tree-based learning algorithms, such as random forest and LightGBM, as observed by the improvement measured by the AUC (+0.1). The transformer correctly predicts student's correctness 81% of the time. The AUC on our first optimal hyperparameters was 0.81027 on the cross-validation set and on the test set 0.81109, thus showing a similar level of performance - no overfitting, thanks to the use of dropout.

To illustrate the performance improvement between our tree-based baseline and our final model, it is best to look at the Kaggle ranking. Our LightGBM baseline would be ranked 2,778, while our transformer-based approach would have been ranked 23rd.

We fell short of adding the lectures watched by students in the prediction process, as the dataset did not make it possible to link specific videos to related questions (unlike on Coursera). Our methodology was constrained by the format of the data available. Future work could include working on richer data, integrating lectures and questions in order to build more comprehensive recommendations for a student when they are expected to fail a question (eg "Review Lecture 3 of Module 2").

Additionally, the interpretability library Captum could be coupled with scoring students by a short-term rolling average (relative to their peers and and calibrated by question difficulty), to help predict performance. To improve on our interpretability approach, algorithmic means (eg integrated gradients through Captum) could also be an area for future investigation as well.

8 Contributions

All group members contributed equally to the project in terms of scope, but focused on different areas. We would like to thank CS230 Project TA Davey Huang for guidance throughout the quarter.

Heidi investigated several of the SOTA KT models in the Kaggle competition by deploying available code locally. For the interpretability work, she focused on applying captum.ai's `IntegratedGradients` library to our SAINT model. Due to its incompatibility with our hand-crafted architecture, we opted for an ablation approach in the end.

Francois also reviewed a number of KT models to select the best code to be used for our baseline. He set up the Ubuntu server and the libraries to work with the model, tuning hyperparameters to improve AUC and applying ablation to gauge interpretability.

Cameron worked on various methods to implement and merge `lectures.csv` into the model. SAINT proved to be overly resistant to these changes so we excluded this feature in the interest of time. He also attempted to set up AWS, which happened to be resistant against SAINT. Lastly, he helped debug various implementations of the model.

References

[1] Qi Liu, Shuanghong Shen, Zhenya Huang, Enhong Chen, and Yonghe Zheng. A survey of knowledge tracing. *arXiv preprint arXiv:2105.15106*, 2021.

[2] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. *Advances in neural information processing systems*, 28, 2015.

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

[4] Youngnam Lee, Youngduck Choi, Junghyun Cho, Alexander R Fabbri, Hyunbin Loh, Chanyou Hwang, Yongku Lee, Sang-Wook Kim, and Dragomir Radev. Creating a neural pedagogical agent by jointly learning to review and assess. *arXiv preprint arXiv:1906.10910*, 2019.

[5] Qi Liu, Zhenya Huang, Yu Yin, Enhong Chen, Hui Xiong, Yu Su, and Guoping Hu. Ekt: Exercise-aware knowledge tracing for student performance prediction. *IEEE Transactions on Knowledge and Data Engineering*, 33(1):100–115, 2019.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[7] Aritra Ghosh, Neil Heffernan, and Andrew S Lan. Context-aware attentive knowledge tracing. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2330–2339, 2020.

[8] Shalini Pandey and George Karypis. A self-attentive model for knowledge tracing. *arXiv preprint arXiv:1907.06837*, 2019.

[9] Youngduck Choi, Youngnam Lee, Junghyun Cho, Jineon Baek, Byungsoo Kim, Yeongmin Cha, Dongmin Shin, Chan Bae, and Jaewe Heo. Towards an appropriate query, key, and value computation for knowledge tracing. In *Proceedings of the Seventh ACM Conference on Learning@ Scale*, pages 341–344, 2020.

[10] Shanghui Yang, Mengxia Zhu, and Xuesong Lu. Deep knowledge tracing with learning curves. *arXiv preprint arXiv:2008.01169*, 2020.

[11] Dongmin Shin, Yugeun Shim, Hangyeol Yu, Seewoo Lee, Byungsoo Kim, and Youngduck Choi. Saint+: Integrating temporal features for ednet correctness prediction. In *LAK21: 11th International Learning Analytics and Knowledge Conference*, pages 490–496, 2021.

[12] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[13] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[14] Sarthak Jain and Byron C Wallace. Attention is not explanation. *arXiv preprint arXiv:1902.10186*, 2019.

[15] Youngduck Choi, Youngnam Lee, Dongmin Shin, Junghyun Cho, Seoyon Park, Seewoo Lee, Jineon Baek, Chan Bae, Byungsoo Kim, and Jaewe Heo. Ednet: A large-scale hierarchical dataset in education. In *International Conference on Artificial Intelligence in Education*, pages 69–73. Springer, 2020.

Hyperparameters of the Encoders and Decoders

Table 2: Generic Hyperparameters

	Value
window_size	100,256
batch_size	8, 128
lr	3.0e-4
betas	0.9, 0.99

Table 3: Hyperparameters of the encoder

	Value
vocab_size	13525
category_size	9
content_embedding_size	256
embedding_size	64
hidden_size	256
num_hidden_layers	12
num_attention_heads	8
relative_attention_num_buckets	32
lag_time_scale_alpha	2.5
intermediate_size	1024
hidden_act	gelu
hidden_dropout_prob	0.10
attention_probs_dropout_prob	0.10
initializer_range	0.02
layer_norm_eps	1.0e-12
pad_token_id	0

Table 4: Hyperparameters of the decoders

	Value
response_embedding_size	128
embedding_size	128
hidden_size	256
num_hidden_layers	12
num_attention_heads	8
relative_attention_num_buckets	32
lag_time_scale_alpha	2.5
intermediate_size	1024
hidden_act	gelu
hidden_dropout_prob	0.10
attention_probs_dropout_prob	0.10
initializer_range	0.02
layer_norm_eps	1.0e-12
pad_token_id	0

Appendix 3 - Detailed description of the data

train.csv

- row_id: (int64) ID code for the row.
- timestamp: (int64) the time in milliseconds between this user interaction and the first event completion from that user.
- user_id: (int32) ID code for the user.
- content_id: (int16) ID code for the user interaction
- content_type_id: (int8) 0 if the event was a question being posed to the user, 1 if the event was the user watching a lecture.
- task_container_id: (int16) Id code for the batch of questions or lectures. For example, a user might see three questions in a row before seeing the explanations for any of them. Those three would all share a task_container_id.
- user_answer: (int8) the user's answer to the question, if any. Read -1 as null, for lectures.
- answered_correctly: (int8) if the user responded correctly. Read -1 as null, for lectures.
- prior_question_elapsed_time: (float32) The average time in milliseconds it took a user to answer each question in the previous question bundle, ignoring any lectures in between. Is null for a user's first question bundle or lecture. Note that the time is the average time a user took to solve each question in the previous bundle.
- prior_question_had_explanation: (bool) Whether or not the user saw an explanation and the correct response(s) after answering the previous question bundle, ignoring any lectures in between. The value is shared across a single question bundle, and is null for a user's first question bundle or lecture. Typically the first several questions a user sees were part of an onboarding diagnostic test where they did not get any feedback.

questions.csv: metadata for the questions posed to users.

- question_id: foreign key for the train/test content_id column, when the content type is question (0).
- bundle_id: code for which questions are served together.
- correct_answer: the answer to the question. Can be compared with the train user_answer column to check if the user was right.
- part: the relevant section of the TOEIC test.
- tags: one or more detailed tag codes for the question. The meaning of the tags will not be provided, but these codes are sufficient for clustering the questions together.

lectures.csv: metadata for the lectures watched by users as they progress in their education.

- lecture_id: foreign key for the train/test content_id column, when the content type is lecture (1).
- part: top level category code for the lecture.
- tag: one tag codes for the lecture. The meaning of the tags will not be provided, but these codes are sufficient for clustering the lectures together.
- type_of: brief description of the core purpose of the lecture

example_test_rows.csv

Three sample groups of the test set data as it will be delivered by the time-series API. The format is largely the same as train.csv. There are two different columns that mirror what information the AI tutor actually has available at any given time, but with the user interactions grouped together for the sake of API performance rather than strictly showing information for a single user at a time. Some users will appear in the hidden test set that have NOT been presented in the train set, emulating the challenge of quickly adapting to modeling new arrivals to a website.

Appendix 3 - Github

Our code is available on Github at `https://github.com/chesnay/cs230_projectv2`