MASTER'S THESIS 2021

Applying Knowledge Tracing to Predict Exercise Response Time

Shamiran Jaf, Sepehr Noorzadeh

ISSN 1650-2884
LU-CS-EX: 2023-79

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-79

Applying Knowledge Tracing to Predict Exercise Response Time

Tillämpning av knowledge tracing för prediktion av svarstid

Shamiran Jaf, Sepehr Noorzadeh

Applying Knowledge Tracing to Predict Exercise Response Time

Shamiran Jaf
shamiran.jaf@gmail.com

Sepehr Noorzadeh
se1711no-s@student.lth.se

October 7, 2021

Master's thesis work carried out at Akribian AB.

Supervisors: Martin Hassler Hallstedt, martin@akribian.com
Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Knowledge tracing is a relatively well-studied and recently popular application of neural networks. It involves using an interaction history to predict some aspect of future interactions in an educational context. However, all research in this topic has been focused on predicting the *correctness* of future exercises. This thesis aims to adapt existing knowledge tracing models to instead predict the *response time* of future exercises instead. To the best of our knowledge, this has not been attempted before.

We found that using some adjustments to existing models, response time prediction is possible. We believe that response time prediction can be a great tool for calculating the fluency of students since fluency is strongly dependent on the speed at which students solve exercises.

Keywords: MSc, knowledge tracing, RNN, LSTM, transformer, attention, education technology

Acknowledgements

Firstly we would like to thank our supervisor, Pierre Nugues, for his invaluable supervision and guidance throughout the thesis.

We would also like to thank everyone involved from Akribian for being so supportive of our work: Martin Hassler Hallstedt, Henrik Rosvall, Cameron Green and Abbey Lewis. We are especially thankful to Cameron for enduring our weekly stand-up meetings.

Contents

Chapter 1
Introduction

Education is an important foundation for equality and societal prosperity, while teachers struggle to give each student the time and personalized learning they need. Learning is at the same time increasingly taking place on digital platforms, allowing for new opportunities to help teachers achieve the goal of providing an optimal learning experience for every student. The field of *knowledge tracing* aims to capitalize on one of these opportunities: the new-found ability to track students interactions with curricula on an interaction-to-interaction basis.

Using online platforms such as *intelligent tutoring systems*, a wealth of new information can be gathered for each interaction with an exercise. That information includes whether the student has answered correctly or incorrectly and how much time s/he took to answer. Previous attempts at quantifying student knowledge on an interaction-to-interaction basis have used laboriously constructed algorithmic models. A new branch of *knowledge tracing* has successfully applied different AI models to predict student performance, measured by the correctness of her/his response on exercises. These models don't need to be explicitly programmed, and have shown performance improvements over their algorithmic counterparts.

However, current models that only predict correctness are limited in their potential applications; making choices in how to adapt learning paths for a student's requires a deeper understanding of the student's fluency in the concepts that the learning path is composed of.

In this thesis, we show that the models used for correctness prediction can be adapted to predicting response time, which is an important component of fluency. In addition, we found that both response time and correctness prediction performance can be improved by adding handcrafted features.

To the best of our knowledge, this is the first work to apply *knowledge tracing* models to the task of response time prediction. To this end, we have managed to produce promising results, where our models have a greater predictive performance than our baseline.

We hope that our findings can contribute to a more interpretative and detailed analysis of each student's knowledge state. This can in turn be used to create more adaptive and *personalized learning paths* for each student.

1.1 Task formulation

We denote a sequence of interactions between a student and exercises as $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, where the interaction $\mathbf{x}_t$ consists of information about the exercise being answered $\mathbf{e_t}$, along with the student's response to the previous exercise, denoted as $\mathbf{r_{t-1}}$. This thesis sets out to solve two tasks pertaining to predicting the student's response, $\mathbf{r_t}$:

1. The first task we addressed in this Master's thesis is to predict the correctness of the student's response to the current exercise, denoted as $\mathbf{r_t}$, as shown in Figure 1.1.

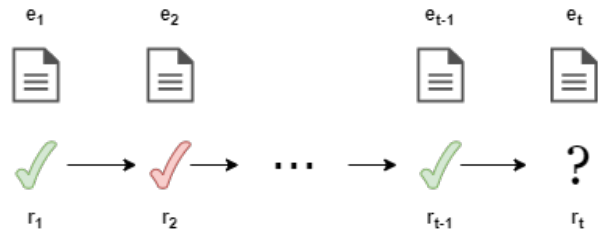


Figure 1.1: A sequence of interactions where each response is either correct or incorrect

2. The second task we addressed is to predict the response time – the time it takes for a student to respond to an exercise – for the response r_t, as shown in Figure 1.2.

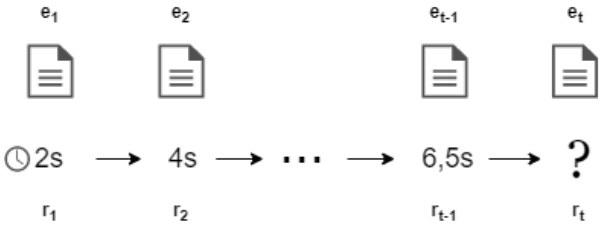


Figure 1.2: A sequence of interactions where each response has a response time measured in seconds

Chapter 2
Related works

The task of predicting student's future responses has led to an emergent field of study called *knowledge tracing*. Knowledge tracing refers to the usage of artificial intelligence or statistical models to track the knowledge levels of students across different subjects or concepts.

The goal of knowledge tracing is to, given a sequence of student interactions with a learning platform, predict some aspect of their future interactions. So far, the field of knowledge tracing has been focused on the task of predicting the correctness of a response, which can be either correct or incorrect. The related works below are thus all concerned with correctness. Whereas this thesis is dedicated to improving upon that task, it also attempts to utilize the techniques and models used for predicting correctness to predict response time.

2.1 Bayesian knowledge tracing

Bayesian models for knowledge tracing were introduced by Corbett and Anderson (1995) and have since been a popular method for predicting a student's ability to answer a question correctly. In its original form, Bayesian Knowledge Tracing (BKT) modelled the probability of a student learning a knowledge concept as an equation of probabilities and parameters, which is updated after each student interaction.

Modern BKT variants still yield comparable results to early neural network models, as reported by Khajah et al. (2016). It is also easy to interpret how the model arrives at a prediction since the parameters underlying the prediction have been manually set. However, BKT models require expert knowledge in order to both map each exercise to its corresponding knowledge concepts and to correctly configure the model parameters. Furthermore, the model is not able to automatically learn latent knowledge concepts, since the knowledge concepts are explicitly programmed and fixed.

2.2 Deep Knowledge Tracing

Introduced in Piech et al. (2015), Deep Knowledge Tracing (DKT) refers to the application of machine learning models to the task of knowledge tracing. Inspired by the success of recurrent neural networks in language processing, Long-Short Term Memory (LSTM) networks were applied to the task of correctness prediction.

The authors were able to achieve a considerable improvement in performance compared to BKT models when predicting correctness. In contrast to BKT models, the DKT model – being a neural network – will itself learn the relationships between different Knowledge Concepts. Therefore, the DKT model does not require the same degree of expert knowledge in the subject matter.

2.3 Self-Attentive Knowledge Tracing

Recurrent neural networks were augmented by a new mechanism called *attention* by Bahdanau et al. (2015). Attention allowed neural networks to learn more long-term relations by allowing any position in a time series to directly relate to an earlier position.

Eventually, it was discovered by Vaswani et al. (2017) that attention can be a powerful mechanism on its own without being applied to an RNN. The paper introduced the *multi-headed self-attention* layer which could model long term relationships between samples in time-series data.

Multi-headed attention was applied to knowledge tracing by Pandey and Karypis (2019) to create a new model called the *Self Attentive Knowledge Tracing* (SAKT).

The SAKT architecture showed an improvement in performance over earlier models. In addition, the SAKT model is much faster to train on average than the aforementioned models thanks to much higher training parallelism. This higher training parallelism is in turn enabled by the lack of recurrent architecture, and the usage of multiple independently trained attention heads.

2.4 Relation-Aware Knowledge Tracing

Pandey and Srivastava (2020) augmented the SAKT model by modeling student forgetfulness and exercise relation. The forgetting behaviour was modelled by looking at the *timestamp* fields typically found in knowledge tracing datasets. The exercise relation modelling was made by comparing the text content of the exercises using *natural language processing* methods.

2.5 SAINT

Vaswani et al. (2017) not only introduced the multi-head self-attention layer, but utilized it for natural language processing by arranging several layers of multi-head self-attention into *encoder* and *decoder* blocks.

Choi et al. (2020a) used a similar architecture to create a new knowledge tracing model called *SAINT* that outperformed the state of the art knowledge tracing models at the time.

The SAINT model was later augmented by Shin et al. (2021) by incorporating the temporal features available in the EdNet dataset to create *SAINT+*. SAINT+ uses new time-related features such as *response time* and *lag time* to more accurately predict correctness for exercises in the *EdNet* dataset (Choi et al., 2020b).

2.6 KEETAR

Jeon (2021) introduced a novel sequence-to-sequence architecture as a winning entry in the Kaggle competition *Riiid AIEd Challenge 2020*. The architecture is noteworthy for combining an attention-based model with an RNN network. In addition, the model also utilizes an engineering trick to reduce the complexity of the matrix multiplications associated with training. This reduced training time (albeit at the cost of accuracy) allows for the use of longer sequences to offset the aforementioned loss of accuracy, for a net gain in performance.

Chapter 3
Theory

In order to understand the knowledge tracing models used in this thesis, some background information and theory is required. What follows is an attempt to explain the theory behind each model that we've used.

3.1 Feedforward networks

One of the simplest components employed in neural networks is a multi-layer perceptron, also known as a feedforward network or FFN. A feedforward network works by multiplying a trainable weight matrix with the input vector and adding a bias vector:

$$\text{FFN}(x) = xW^T + b$$

where **x** is the input vector, **W** is the weight matrix and **b** is the bias vector.

By chaining several feedforward networks and using Rectified Linear Unit (ReLU) activation layers in between, complex relations can be captured in the input data. However, when dealing with time-series data, feedforward networks are not very useful because of their fixed input length.

3.2 Recurrent Neural Networks

While Recurrent Neural Networks (RNNs) employ feedforward networks, they are able to handle arbitrary input lengths and employ an internal state in order to remember meaningful information from past inputs. This makes them suitable for the task of predicting future performance, such as correctness and response time prediction.

After having first been introduced by Rumelhart and McClelland (1987), RNNs have since taken many forms. While the conventional RNN is rarely used today in favor of its

successors, it is necessary to understand the conventional RNN since its successors share its temporal features.

Conventional RNNs work by, for each step in a sequence of length T, taking in an input vector $\mathbf{x_t}$, a hidden state $\mathbf{h_{t-1}}$ and producing an output vector $\hat{\mathbf{y}}_\mathbf{t}$ and an updated hidden state $\mathbf{h_t}$.

$$\begin{aligned} \mathbf{h_t} &= \tanh(\mathbf{W_{ih}x_t} + \mathbf{b_{ih}} + \mathbf{W_{hh}h_{(t-1)}} + b_{hh}) \\ \hat{\mathbf{y}}_\mathbf{t} &= \tanh(\mathbf{W_y h_t} + b_y) \end{aligned}$$

When processing the next input in the sequence, the updated hidden state is used as an additional input into the RNN along with the next input, as can be seen in Figure 3.1. Although each step in the sequence involves the same neural network, with unchanged weight matrices, the internal state changes and the network can thus be seen as having a temporal state.

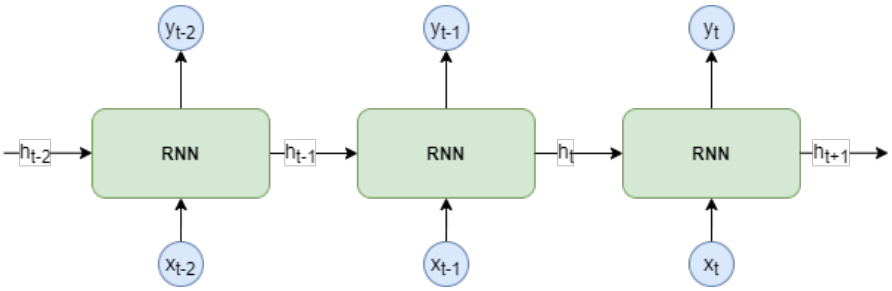


Figure 3.1: Example of an unfolded Recurrent Neural Network

3.2.1 Challenges

The conventional RNN learns how to manipulate its internal state in order to capture meaningful patterns across time. However, when considering patterns that play out during a long span of time, the standard RNN runs into two types of problems; exploding and vanishing gradients, as reported by Kolen and Kremer (2001).

Both problems stem from the fact that the back-propagation not only propagates the error through the networks layers, but does so for each time-step in the sequence. This means that the weight update for a time-step far back in time will be contingent on a long chain of multiplications of partial derivatives, whose results will either converge towards zero or diverge upwards. In the case of convergence towards zero, the update to the weight will be insignificant, which can be interpreted as the model not being able to learn anything from that time-step.

For the purpose of knowledge tracing, short-term memory is not enough for an accurate model. Students learn across long sequences of interactions with a curriculum, and interlinked knowledge concepts may not be located close to each other.

3.3 Long Short-Term Memory

Long Short-Term Memory (LSTM), proposed by Hochreiter and Schmidhuber (1997), shares the temporal connection and hidden state concepts of an RNN. It was designed to deal with

the vanishing and exploding gradient problems through assuring a constant error flow backward through time, thus allowing a longer memory than that of an RNN.

While the input and output from the LSTM network, as in the case for the RNN, is an input vector $\mathbf{x_t}$, a hidden state $\mathbf{h_{t-1}}$, it also takes as an input a cell state $\mathbf{C_{t-1}}$ and outputs an updated cell state $\mathbf{C_t}$ as well as $\mathbf{h_t}$ which acts as both the updated hidden state and the output, as can be seen in Figure 3.2. As was the case for Figure 3.1, Figure 3.2 is a visualization of a single LSTM with one set of weights which has been unrolled temporally.

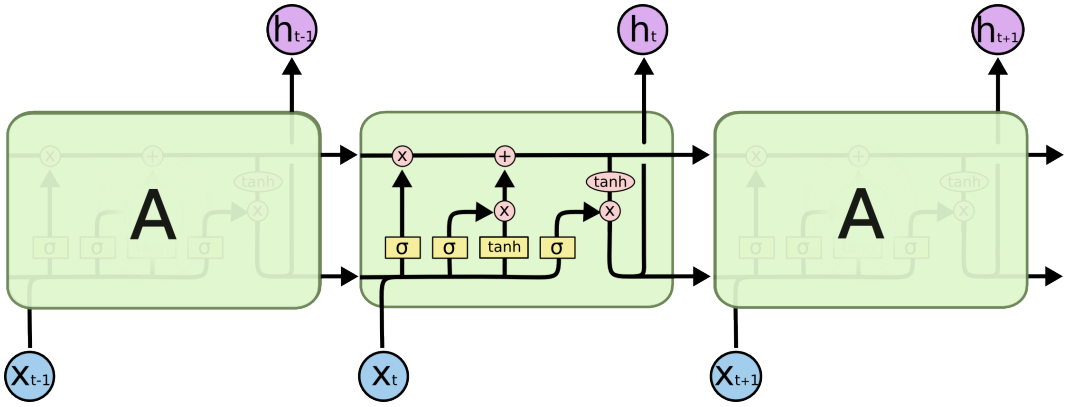


Figure 3.2: An unrolled LSTM architecture from Olah (2015)

.

The novelty which grants LSTM its improved memory compared to its predecessor is the deliberate manipulation of the cell state, which is controlled by three layers which are called gates. The gates respectively control what information should be *forgotten*, *stored* and *updated*.

Forget gate The purpose of the forget gate, which was introduced by Gers et al. in order to allow the LSTM to reset its own state, is to remove that information from the cell state which is no longer necessary to remember. (Gers et al., 2000)

The forgetting is achieved by element-wise multiplication of the cell state with the vector f_t

$$\begin{aligned} \mathbf{C_t} &= \mathbf{f_t} \odot \mathbf{C_{t-1}} \\ \mathbf{f_t} &= \sigma(\mathbf{W_f}[\mathbf{h_{t-1}}, \mathbf{x_t}] + b_f), \end{aligned}$$

where f_t is computed using the sigmoid function, and thus has elements ranging from 0 to 1. This means that the point-wise multiplication will result in forgetting the values at those indexes where the corresponding element in $\mathbf{f_t}$ is zero.

Input gate In order to update the cell state, the input gate will first decide what indexes to update by computing the vector $\mathbf{i_t}$ and also candidate values for every index, $\mathbf{\tilde{C}_t}$

$$\begin{aligned} \mathbf{i_t} &= \sigma(\mathbf{W_t}[\mathbf{h_{t-1}}, \mathbf{x_t}] + b_i) \\ \mathbf{C_t} &= \tanh(\mathbf{W_C}[h_{t-1}, \mathbf{x_t}] + b_C) \end{aligned}$$

Point-wise multiplication of $\mathbf{i_t}$ and $\mathbf{C_t}$ will then control what candidate values will be added to the cell state through point-wise addition.

Output gate In the same manner as in the previous two gates, the previous hidden state. $\mathbf{h_{t-1}}$, and the input, $\mathbf{x_t}$, will be used to control which parts of the cell state that should be passed through to the output by first passing them through a sigmoid layer. The output of the sigmoid layer will then be point-wise multiplied with the update cell state, after it has been passed through a tanh layer. The result is then passed on as the output $\mathbf{h_t}$.

3.3.1 Challenges

Although the LSTM architecture has managed to significantly reduce the problems stemming from vanishing and exploding gradients by limiting the amount of multiplications that is needed when back-propagating, it has not managed to eliminate them completely. This is because of the sequential structure being the same, which means that the same lengthy and costly back-propagation chains still exists.

3.4 Attention

Attention is a mechanism that allows for sequence-to-sequence training without being constrained by the sequentiality of the input time-series data. While it was originally proposed by Bahdanau et al. (2015) as an enhancement to RNNs, attention was later found to be fairly powerful when used on its own and became a common alternative to RNN (Vaswani et al., 2017).

In an attention model, the input data, is projected into a series of linear matrices to create three matrices named query, key and value, denoted as $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$. The *attention score* is then calculated using the *Scaled Dot-Product Attention*

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\mathbf{T}}{\sqrt{n}})\mathbf{V},$$

where n is length of the sequences in the time-series.

This allows each point in the input data sequence to attune to another part of the input data, regardless of the difference in position in the time series. However, in some models it may be necessary to prevent a data point from attuning to data points in the future, in order to enforce causality in the predictions. This can be done using *future masking*.

In a multi-headed attention model, multiple attention scores are calculated using multiple projections, called *attention heads*. Each attention head is attuned to a different part of the input, and is therefore likely to learn a different type of relation between the elements in the time series. The final output is constructed by concatenating the output from each individual attention head.

3.5 Transformer

The transformer, proposed by Vaswani et al. (2017) utilizes the multi-headed attention architecture using two different types of blocks, namely *encoders* and *decoders*, which both contain multi-headed attention and FFN components. The architecture uses one uniform dimension size, a common one being 512, which applies to the output and input of the encoder and

decoder layers as well as the embedding size of the input and output. Together, they can take variable length sequences and output variable length predictions, just as LSTM.

Encoder blocks The input sequence will first be embedded and positionally encoded using sine and cosine waves and is passed to the first encoder block. It is then passed through the encoder's multi-headed attention component and a feedforward network in order to create a representation of the sequence with the same dimension as the embedded input sequence. This representation will be passed to the decoder block, as well as the next encoder block in the stack, if it exists.

After each multi-headed attention and FFN component a residual connection and normalization takes place in order to allow for better flow of gradients and facilitate faster training.

Using several encoder blocks lets the transformer encode increasingly complex representations of the input sequence. This complexity is needed for the language related tasks that transformers are developed for, but is not as necessary for the kind of relationships between knowledge concepts, which aren't as complex as human language.

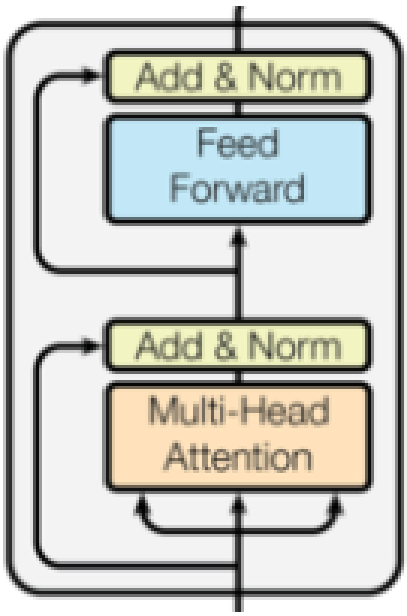


Figure 3.3: Encoder block from Vaswani et al. (2017).

Decoder blocks The decoder blocks will both take the representations generated by the encoder blocks, as well as the previous outputs from the decoder in order to generate a prediction as to what the next output should be.

This is done through first routing the previous output embeddings through a future masked multi-head attention layer, the output of which will then be used as the query value in a second multi-head attention layer. This query value can be seen as a compressed representation of the previous output. The output of the encoder blocks will then be used as key and value, together with the query from the previous multi-head attention layer, as input into a second multi-head attention layer. The output of the multi-head attention layer is ran through a feedforward network before being output.

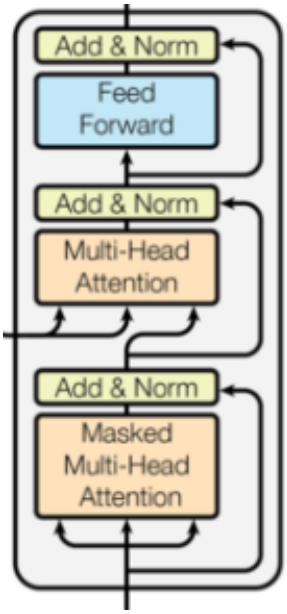


Figure 3.4: Decoder block from Vaswani et al. (2017).

Full Transformer architecture The full transformer architecture, as can be seen in Figure 3.5, shows the connection between the encoder layers and decoder layers. The output of the final decoder layer is run through a linear layer and a softmax function in order to generate an output probability, which could for example be a one-hot encoding of a word.

While RNN networks use a chain of back-propagation in order to remember meaningful relationships in a sequence, the attention mechanism lets transformers remember arbitrarily long relationships. Each part of the sequence can itself contain information about what other parts of the sequence is related to it. Because of this, important information far back in time can be remembered just as clearly as information which is closer in time.

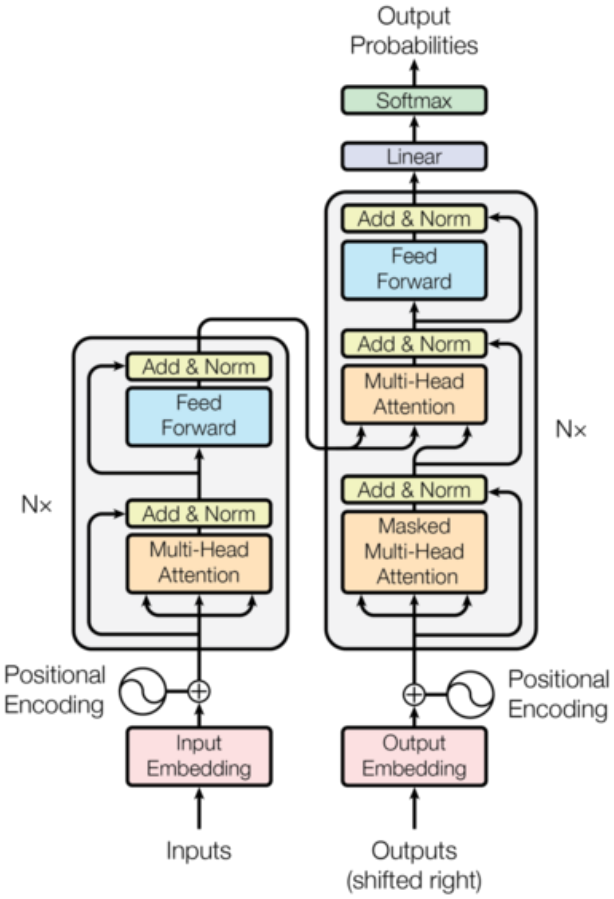


Figure 3.5: Transformer architecture from Vaswani et al. (2017), show N encoders to the left and N decoders to the right.

Chapter 4
Datasets

In order to train models, labeled training data is needed. Luckily online tutoring systems have allowed for the collection of such data, and several publicly available datasets gathered from tutoring systems have been used in this study. These datasets have been used in earlier knowledge tracing studies. In addition to these datasets, we also have access to a private dataset called the *Akribian* dataset.

Name	Rows	Categories	Unique exercises	Ratio of correct answers
Akribian	71,413	239	787	90.0%
ASSISTments 2012	2,630,080	199	50,989	69.6%
Junyi Academy	16,217,311	10	1,327	70.4%
EdNet	101,230,332	7	13,525	65.7%

Table 4.1: Summary of datasets

Table 4.1 shows the size of the datasets, along with other information. The datasets are largely similar with regards to structure and the type information they contain. Every dataset is tabular and each row describes one interaction between a student and the tutoring system. However, the datasets also differ in many aspects. One such difference, as can be seen in Table 4.1, is that the dataset of Akribian, which is a young startup company, is considerably smaller than the public datasets. This depends on the fact that the public datasets have been gathered from a larger number of users over a longer period of time.

Another difference is in the structure of the tutoring systems themselves. Akribian's game *Count on me!* is a linearly structured game, where most users follow the same pre-planned route, in order to learn basic mathematical skills. On the other hand, the other tutoring systems are closer to quiz systems for evaluating what the students have already learned in class, rather than replacements for a traditional textbook. Therefore they select questions using more complicated methods.

The datasets also differ with regards to the contents of the exercises. The Akribian dataset consists of simple maths exercises aimed towards young children, while ASSISTments and Junyi Academy dataset exercises are maths exercises meant for high school or university students. The EdNet dataset is unique in the sense that it doesn't contain any maths exercises, instead consisting of exercises for learning English.

4.1 Akribian dataset

The Akribian dataset is extracted from the game *Count on me!* which can be played on tablets. The purpose of the game is to teach 6-9 year old children basic maths. This is done in the app by interspersing story driven segments and math segments. The game is divided into a hierarchical structure where each individual exercise is a part of a learning sequence, and every learning sequence is part of a learning task. The game's progression is linear and personalization mainly consists of skipping or repeating learning sequences. As such, there isn't much variation in the sequence of questions answered by each student.

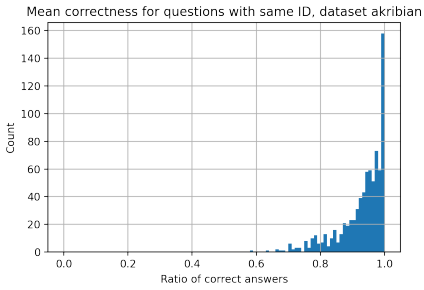
Field name	Description
`StudentId`	Unique identifier for each student
`LearningSequenceTitle`	Title of the sequence the current exercise belongs to
`ExerciseTitle`	Title of the exercise sequence being answered
`SubmissionOutcomes`	An integer representing either a correct answer, an incorrect answer or a timed out answer.
`ExerciseResponseTime`	An integer representing the response time in milliseconds
`FinishedOn`	A date-time string representing the timestamp for the interaction

Table 4.2: Relevant fields from the Akribian dataset

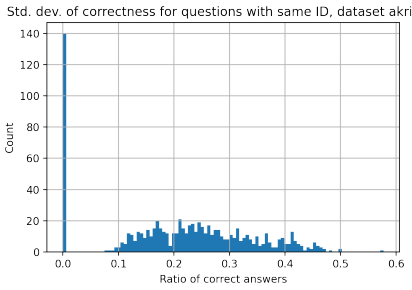
Table 4.2 shows the relevant information featured in each row of the Akribian dataset. Each row of the dataset describes one question answered by the user of the app *Count on me!*, and describes different aspects of one response to an exercise. The *SubmissionOutcomes* column consists of a series of answers given by the user, formatted as a string of integer values separated by semicolons. The values can be 0 for a correct answer, 1 for an incorrect answer, and 2 for a *time-out*. A time-out means the user did not answer the question after an extended period of time.

In order for a user to move to the next question, the current question must be answered correctly. Due to this property, a response can either be a single correct answer, or several incorrect answers followed by either a correct answer or a time-out at the end. In order to make the data suitable for usage as the label in a binary prediction problem, the response data was reformatted into a single integer value. This value is 1 if the user answered correctly on the first try, and 0 if the user answered incorrectly on the first try or timed out.

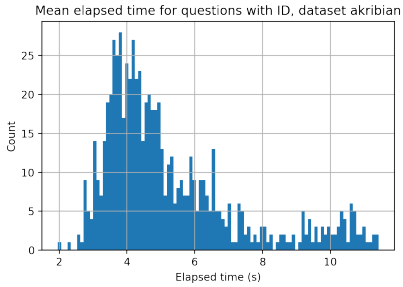
Figure 4.1 shows the distribution of the response time and correctness for exercises in the Akribian dataset. It can be seen that some exercises have very low variance for response time and correctness, which can be due to the small size of the dataset.



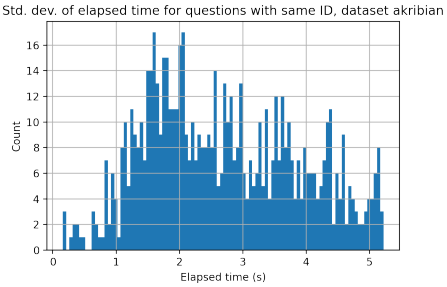
(a) Histogram of mean correctness



(b) Histogram of correctness standard deviation



(c) Histogram of mean response time



(d) Histogram of response time standard deviation

Figure 4.1: Histogram of mean and standard deviations per question ID for the Akribian dataset. Response time values above the 90th percentile have been deleted to remove outliers.

4.2 ASSISTments dataset

The ASSISTments dataset consists of user interactions with exercises in ASSISTments, an online education tool described in Feng et al. (2009). ASSISTments is an online system that allows teachers to assign exercises to students who then can use the platform to complete exercise while receiving both automated hints and help from their teachers. Because of its size and relatively long history of usage in e-learning, it is one of the most widely used datasets in the field of knowledge tracing. There are several versions of the ASSISTments dataset available publicly, and each dataset differs slightly with regards to the what type of data it contains. ASSISTments 2012 was chosen for this thesis as it contains timestamps, which can be used to engineer better features. In addition, it is the largest of the ASSISTments datasets.

Table 4.3 shows the relevant information featured in each row of the ASSISTments dataset. Unlike the Akribian dataset, ASSISTments does not have a time-out response.

Figure 4.2 shows the distribution of the response time and correctness for exercise categories in the ASSISTments 2012 dataset. Similar to the Akribian dataset, many exercises have very low variance in response time and correcntess. This is likely because the dataset has a high number of exercises relative to the total number of responses, resulting in some exercises having very few responses.

Field name	Description
`user_id`	Unique identifier for each student
`skill_name`	Category of the current exercise
`problem_id`	Unique identifier for the current exercise
`correct`	An integer representing either a correct answer or an incorrect answer
`ms_first_response`	An integer representing the response time in milliseconds
`end_time`	A date-time string representing the timestamp for the interaction

Table 4.3: Relevant fields from the ASSISTments 2012 dataset

4.3 Junyi Academy dataset

The Junyi Academy dataset consists of user interactions with exercises in Junyi Academy, an online education tool derived from Khan Academy and mainly used in Taiwan. Junyi Academy, similar to ASSISTments, is an online system for assigning and assessing maths problems for students. The dataset is hierarchical with four levels of subdivision.

Field name	Description
`uuid`	Unique identifier for each student
`level2_id`	Category of the exercise sequence being answered
`ucid`	Unique identifier for being answered
`is_correct`	An integer representing either a correct answer or an incorrect answer
`total_sec_taken`	Response time in seconds, truncated down to an integer
`timestamp_TW`	A date-time string representing the timestamp of the response

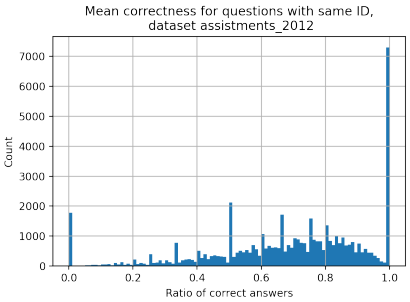
Table 4.4: Relevant fields from the Junyi Academy dataset

Table 4.4 shows the relevant field in the Junyi Academy dataset. It is worth mentioning that the timestamp field in this dataset has a resolution of 15 minutes, unlike the other datasets which have second precision.

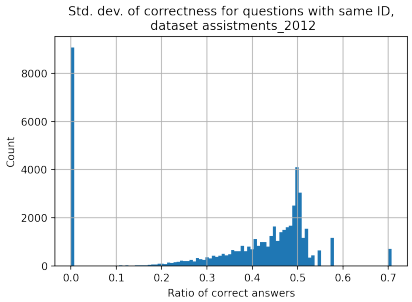
Figure 4.3 shows the distribution of the response time and correctness for exercises in the Junyi Academy dataset. Unlike the previous two datasets, Junyi Academy dataset has few exercises and many responses.

4.4 EdNet dataset

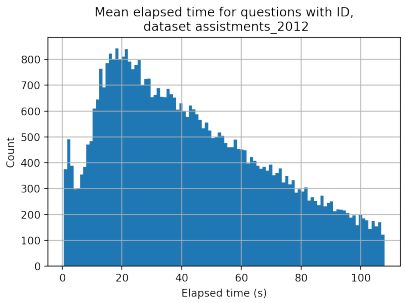
The EdNet dataset as described by Choi et al. (2020b) is a large-scale hierarchical dataset collected from *Santa*, a mobile app developed by *Riiid!* that tutors more than 780,000 South Korean students in English. It has recently become a popular dataset in benchmarking the performance of knowledge tracing models, largely thanks to a *Kaggle* competition where users submitted models that competed on the dataset.



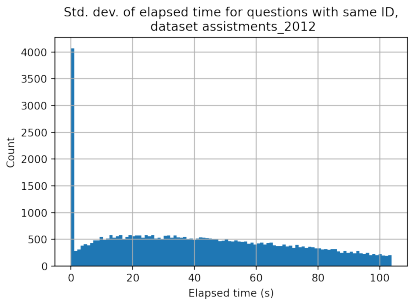
(a) Histogram of mean correctness



(b) Histogram of correctness standard deviation



(c) Histogram of mean response time

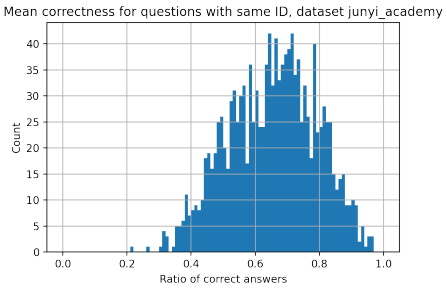


(d) Histogram of response time standard deviation

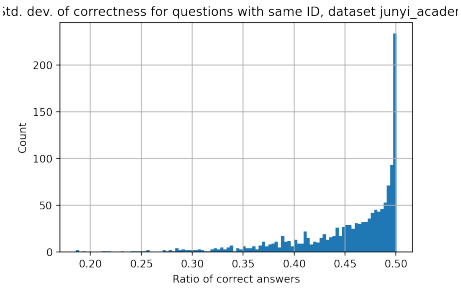
Figure 4.2: Histogram of mean and standard deviations per question ID for the ASSISTments 2012 dataset. Response time values above the 90th percentile have been deleted to remove outliers.

Table 4.5 shows the relevant fields from the EdNet dataset. The EdNet dataset is unique in its representation of temporal data. The response time field represents the response time of the previous response, rather than the current response. In addition, the timestamp field is represented in seconds, rather than a date-time string, and is relative to the first exercise.

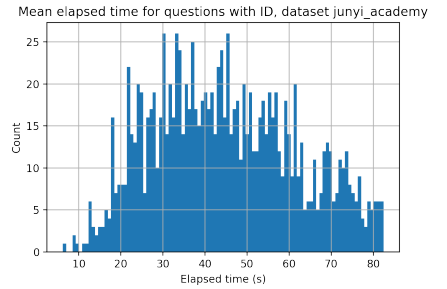
Figure 4.4 shows the distribution of the response time and correctness for exercises in the EdNet dataset. Similar to the Junyi Academy dataset, EdNet has few exercises and many responses.



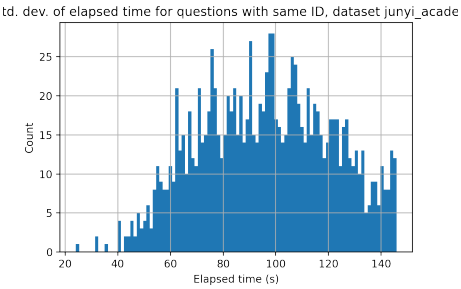
(a) Histogram of mean correctness



(b) Histogram of correctness standard deviation



(c) Histogram of mean response time

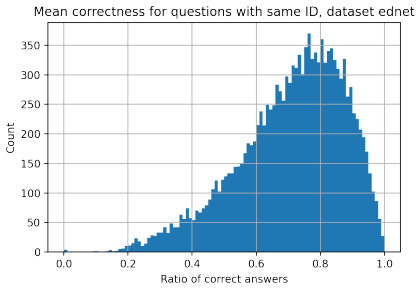


(d) Histogram of response time standard deviation

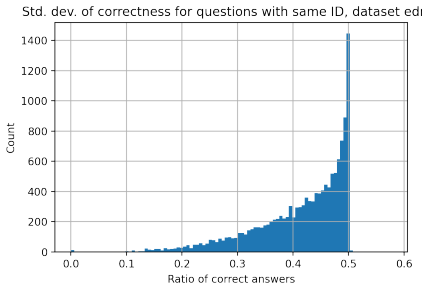
Figure 4.3: Histogram of mean and standard deviations per question ID for the Junyi Academy dataset. Response time values above the 90th percentile have been deleted to remove outliers.

Field name	Description
`user_id`	Unique identifier for each student
`part`	Category of the exercise sequence being answered
`content_id`	Unique identifier for the exercise being answered
`answered_correctly`	An integer representing either a correct answer or an incorrect answer
`prior_question_response_time`	A floating point value representing the response time for the *previous* question, in milliseconds
`timestamp`	An integer representing the total number of seconds passed since the user answered their first exercise

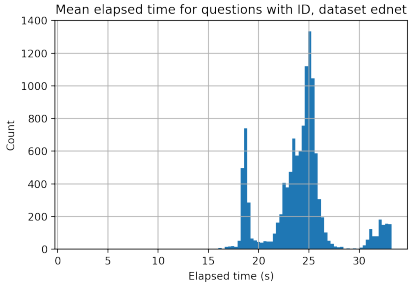
Table 4.5: Relevant fields from the EdNet dataset



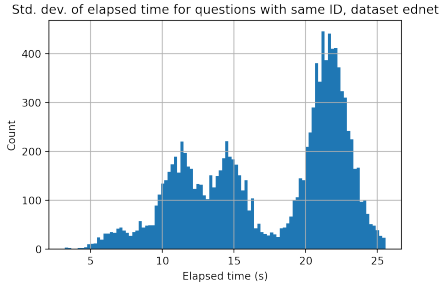
(a) Histogram of mean correctness



(b) Histogram of correctness standard deviation



(c) Histogram of mean response time



(d) Histogram of response time standard deviation

Figure 4.4: Histogram of mean and standard deviations per question ID for the EdNet dataset. Response time values above the 90th percentile have been deleted to remove outliers.

Chapter 5
Approach

To solve the two tasks defined in the introduction, we have used several of the sequence-to-sequence models that have previously been applied for correctness prediction. The models are, in chronological order of introduction, DKT, SAKT, SAINT and KEETAR. In addition, we used data processing to augment the datasets with additional features that can be used by the models to model more complex relations.

5.1 Data processing

The datasets used in this study differ in the quantity, types and format of their features. Therefore it was necessary to implement a pipeline for processing the datasets so that they all adhere to the same format and specifications. This allows every dataset to be used with every model without the need for modifications to the model.

5.1.1 Formatting the data

The first modification was using integer indices for exercise and category identifiers. Normally these fields would either be represented by strings, representing either a human-readable title, or a *unique identifier* used internally by a database. Since strings are not able to be turned into embeddings for inputting into the model, these values were converted into integers. This was done by creating lists of unique exercise and category identifiers and converting the strings into integers representing their indices in these lists.

In addition, all timestamps were converted to the *seconds since Unix Epoch* format in order to make it possible to calculate time differences in seconds. Rows lacking data were filtered out, as well as rows containing response time values that were larger than the 90th percentile of response time values. The reason for this was to filter out outliers resulting from users leaving their computer devices without answering the exercise.

5.1.2 Sorting the datasets

Some of the datasets contain millions of entries and require several gigabytes of RAM memory to load. As such, they could not be read in their entirety in one pass. To address this issue, we developed a framework to read the datasets a predetermined number of rows at a time. The interactions in the datasets are stored in chronological order of the responses, since that is the norm in most databases used in online services. This means that when reading the dataset in parts, there would be no guarantee that the entire interactions history of the users would be loaded into the memory at once. This could lead to the extraction of several disjoint user exercise histories that are actually fragments of the same user history. This issue could damage the model's performance since it wouldn't have access to the entire exercise interactions history of each user.

To solve this problem, we first sorted the database by user ID so that every user's interactions would be in continuous order. Each user's interactions was then sorted by timestamp in an increasing order to make sure that the exercises are in chronological order.

5.1.3 Padding and windowing

After sorting, we cut the exercise history for each user into windows of a specific size. Exercise histories that were below the window size were padded to be the correct size, and those that were over the window size were cut into several windows.

At first, we used the mean exercise length as the window size. However, following further experimentation with the window size, it was concluded that a higher window size can capture the entire interaction history of most users and therefore has better performance. However, high window sizes proved to be slow to train owing to larger matrices which were slow to multiply.

After some experimentation with windows of varying lengths, a window size close to the 90th percentile user history length proved to be good enough with regards to training time and model performance for most datasets.

Initially, the windows were shifted one step at a time to maximize the number of time sequences that could be used for training. This meant that users with long exercise histories would end up creating many overlapping windows, providing more data for training. However, this proved infeasible for larger datasets like EdNet due to limited hard disk space and long training times.

In addition, this would cause users with a very long exercise history to have disproportionate representation in the data and introduce unwanted bias to the model. Therefore a *window stride* was introduced which decided how many steps the window would be shifted each time when windowing. After some experimentation the window stride was set to half the window size to create sequences with 50% overlap as a compromise between size and performance.

5.1.4 Feature engineering

To gain as much information as possible from the relevant data in each dataset, we engineered a series of features for each user interaction in users' exercise histories.

Time difference

The first engineered feature is the time that has passed between each exercise and the exercise before it. This feature can effectively track how long it has been since the user last answered a question.

The motivation for including this feature is that students tend to forget information over time. By looking at how long it has been since a student last solved exercises, the model could learn whether a student had forgotten some information and would likely perform worse.

The feature was created by calculating the difference between the submission timestamps of each question and the question before it. In addition, the current question's response time was subtracted from this value to get the actual time difference between submitting the answer to a question and starting the next question. This is similar to the *lag time* feature used by Shin et al. (2021).

Mean response time per exercise

While response time is individual and dependent on the fluency of the student, it is also highly dependent on the characteristics of the exercise itself. In other words, some exercises take a longer or shorter time to solve, regardless of the user's fluency.

This can depend both on the question's difficulty and the time it takes to formulate the question to the student. An example of the latter would be a question, which has a long text description that takes a long time to read. By supplying the model with the mean response time for each unique exercise, the model should be able to take this into account.

Standard scored (z-scored) response time

Response time has often been used as a feature in previous knowledge tracing studies. Past studies simply used the response time in seconds as a new feature.

We hypothesized that if the model is given information about how the user performed *relative to other users*, important information about the overall fluency of the user can be inferred. In order to provide such information to the model, we decided to introduce a sort of more *relative* response time feature.

One way to implement such a feature would be to use standard scoring over all response latencies in the dataset. However, as mentioned before, different exercises take different amount of time to complete on average. Since the mean response time is different for each exercise, the response time was normalized based on responses to the same exercise ID. We started by first calculating the mean response time μ and standard deviation σ for each exercise. Afterwards, the standard score z for each response was calculated using

$$z = \frac{x - \mu}{\sigma},$$

where x is the response time in seconds.

This standard score is equivalent to how many standard deviations a single observed response time is from the mean, when considering every other response time to the same question. Therefore, a positive value would mean a slower than average response to the question and a negative value would mean a faster than average response time.

Ratio of correct answers per exercise

By calculating the ratio of correct answers to the total answers per exercise, expressed as a continuous value between 0 and 1, the model can be supplied with a measure of each unique exercise's difficulty.

5.1.5 Types of features

The input features used in every model are split into two types, depending on what type of information they hold about a response to an exercise.

One type corresponds to information about the current exercise e_t and consists of the unique identifier and the category of the exercise currently being answered, in addition to the mean response time and correctness ratio for questions with the same identifier. This category of features is called the *query features*.

The other type of input features consists of information about the past response r_{t-1}, and consists of the correctness, response time and the timestamp difference of the response and is called the *memory features*.

In addition, attention-based models benefit from *positional embeddings*, which include information on the positions of the data points in a sequence. This is useful because attention mechanism, unlike a recurrent neural network, does not operate sequentially on time-series data and therefore does not keep track of the order of data points in a sequence. Therefore, positional embeddings are added to both the query features and the memory features in attention-based models.

5.1.6 Feature embeddings

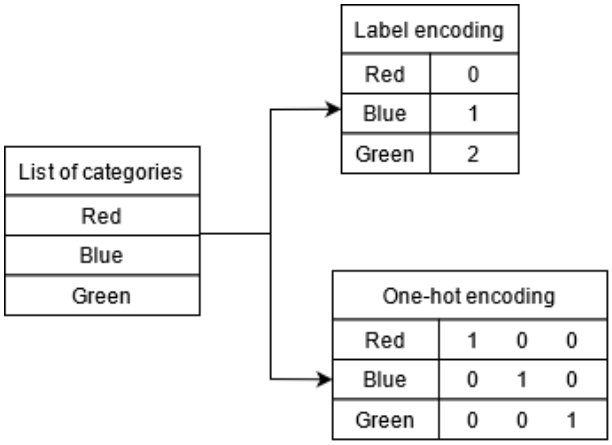


Figure 5.1: Types of categorical encoding

Categorical data refers to data that belongs to a category, rather than being a continuous measurement. This type of data is normally represented in one of two forms, both utilizing a list of categories.

The first method is to represent each category as an integer corresponding to an index of the category in the list. The second method is to use a vector with the same size as the list

of categories, where all the values are 0 except for the value that has the same position as the represented category in the list. The former method is known as *label encoding* and the latter method is known as *one-hot encoding*. A visual representation of the two types of encoding can be seen in Figure 5.1.

Problems with categorical data representation

There are problems with directly using categorical data as a feature, and they depend on the type of encoding used to represent the data. In the case of label encoding, unrelated labels that may follow each other will be interpreted as being related by the model when fed into a model with linear components (for example, a multi-layer perceptron). An example of a situation that can give rise to this problem is when using label encoding for words in an alphabetically sorted list, where words with no semantic relation could have labels that are very close to each other simply because of how they're spelled.

One-hot encoding avoids this problem by using a long indexing vector instead of an integer label, avoiding similarity between unrelated but subsequent categories. However, one-hot encoding gives rise to a new problem regarding the length of the vectors. Long category lists could be too large to be used in practice for model training. For example, the ASSISTments 2012 dataset contains over 50,000 different unique exercises, and using the one-hot encoding of the exercise IDs as a feature will lead to models that are unfeasible to train on the current generation of computers because of the large size of the input vector. In addition, one-hot encoding does not allow for any measure of similarity between categories, which could otherwise be beneficial to the performance of the model.

Embedding solves these two problems by using a relatively smaller vector with continuous values. These values can be randomized to clearly distinguish between categories, or pre-calculated to create meaningful relations between categories.

An example of pre-calculated embeddings is the GloVe embeddings (Pennington et al., 2014) for words, which are calculated using an iterative method that ensures that words with similar semantics will have embeddings that are similar, measured by *cosine similarity*. Unfortunately, such sophisticated embeddings are not available for the exercise IDs and exercise categories that are used in knowledge tracing. Therefore, categorical embeddings with random values are used to ensure that each exercise and category is transformed into a continuous-valued vector that is distinct from other categories. These embeddings are trainable, and the model can possibly discover relations between exercises while training.

Continuous features

There are also features in this study that are not categorical, namely response time, timestamp difference, mean response time and mean correctness. Normally, these features could be concatenated to the rest of the embeddings along the feature dimension. However, when using additive embeddings similar to those described in Pandey and Karypis (2019) and Choi et al. (2020a), this is not possible, as the features that are added need to have the same dimensions.

These features could be cast into integers and turned into embedding vectors with correct dimensions using random embeddings. However, the random nature of the embeddings would prevent the model from utilizing the continuous nature of the values. For example, the random categorical embeddings for an response time value of 4 seconds would on average

have the same similarity to the embeddings for 5 seconds and the embeddings for 50 seconds. In reality, 4 seconds and 5 seconds are much closer to each other than to 50 seconds.

Continuous embeddings

In order to create embeddings that preserve this fundamental continuity in continuous values, a special type of embeddings known as *continuous embeddings* similar to those used by Shin et al. (2021) were employed. These embeddings are trainable vectors with a length equal to that of the categorical embeddings. These vectors are multiplied by their respective continuous-valued features to create embeddings with the same length as the categorical embeddings. Thanks to this property, the continuous nature of the features are preserved since they are multiples of the same embeddings vector. Shin et al. (2021) has shown that for the SAINT model and for the response time feature, which takes on continuous values, continuous embeddings result in better performance compared to categorical embeddings.

In addition to the categorical and continuous embeddings, the attention-based models in this study use *positional encoding* as described by Vaswani et al. (2017).

Feature name	Embeddings type
Exercise ID	Categorical embedding
Exercise category	Categorical embedding
Past correctness	Categorical embedding
Response time	Continuous embedding
Timestamp difference	Continuous embedding
Mean response time	Continuous embedding
Mean correctness	Continuous embedding
Positional embeddings	Positional Encoding

Table 5.1: Embedding types for different features

Table 5.1 shows the types of embeddings used for each feature in the models employed in this study.

5.2 Models

To find the best architecture for predicting question correctness, a series of different knowledge tracing models were implemented, namely Deep Knowledge Tracing (DKT), Self-Attentive Knowledge Tracing (SAKT), Separated Self-Attentive Neural Knowledge Tracing (SAINT) and Last Query Transformer RNN (KEETAR). The first model is purely RNN-based, the next two are attention-based models, and the last model is a hybrid model that utilizes both attention mechanism and an RNN layer. We implemented all models from code provided by the authors or, in case such code is unavailable, by attempting to implement the model architectures as described by the authors.

We ran each model according to the hyperparameters given by their authors, when available. Otherwise, we used manual hyperparameter tuning. Since most of the models had not been run on most of the datasets in previous studies, it was necessary to run many tests to manually optimize the hyperparameters for each model-dataset pair.

For every model, the input features were transformed into a series of embeddings and concatenated. These embeddings were then fed into a feedforward network to reduce the total dimension of the input to the dimension of a single embedding, before being input into the models.

5.2.1 DKT

DKT (*Deep Knowledge Tracing*) is an LSTM-based model first proposed by Piech et al. (2015). It is the simplest model used in the study and consists of an LSTM layer followed by a feedforward network.

While the original DKT model uses an encoded set of tuples representing each posssible exercise tag and correctness value (correct or incorrect) as input features, the version of DKT used in this thesis is modified to also incorporate temporal features (response time and timestamps), as well as additional features (mean correctness and mean response time). All of the the query and memory embeddings are concatenated and passed through a feedforward network before being passed to the LSTM layer.

The output of the LSTM layer is then routed through the feedforward network whose output is the final output of the model. The model is trained using a binary cross entropy loss function for correctness prediction and a mean squared error loss function for response time prediction.

5.2.2 SAKT

SAKT (Self-Attentive Knowledge Tracing) is an attention-based knowledge tracing model first proposed by Pandey and Karypis (2019). It uses one or more multi-headed attention layers, and an output multilayer perceptron layer. Since attention models are not constrained by sequentiality during training, the SAKT model is very fast to train. The implemented variant of SAKT also uses additional temporal features similar to DKT. The query for the multi-head attention model consists of the query feature embeddings passed through a DNN and the key and value consist of the memory features passed through a different DNN for key and value.

5.2.3 SAINT

SAINT (Separated Self-Attentive Neural Knowledge Tracing) is a Transformer-based model developed by *Riiid!* and decribed by Choi et al. (2020a). It uses a transformer model with a variable number of encoder and decoder layers, and an output multilayer perceptron layer, as seen in Figure 5.2. The encoder and decoder layers have different inputs: The encoder uses the memory embeddings as query, key and value, and the decoder uses the query embeddings as query and the encoder output as the key and value. Our variant of SAINT is more similar to the proposed SAINT+ model described by Shin et al. (2021) owing to its use of temporal features.

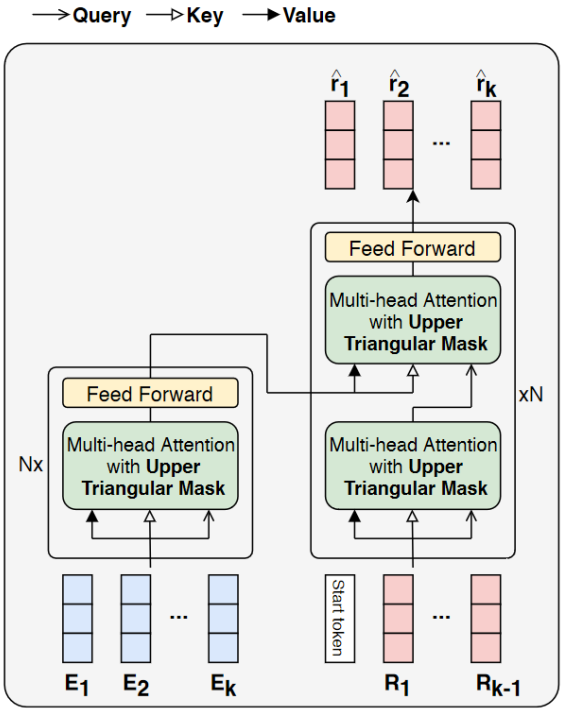


Figure 5.2: SAINT architecture from Choi et al. (2020a)

5.2.4 KEETAR

KEETAR (Last Query Transformer) is a hybrid model consisting of a SAKT-like multi-headed attention layer connected to an LSTM layer and then an output multilayer perceptron layer. It was proposed by Jeon (2021) and won the *2020 Riiid AIEd Challenge* on Kaggle. The architecture, as seen below, takes a sequence of L interactions, I_1, I_2, ..., I_L, which are then input into a single Transformer encoder. The output of the encoder passes through an LSTM into a DNN (Deep Neural Network) which outputs the prediction. A visualization of the architecture can be seen at Figure 5.3. The KEETAR model is noteworthy for a few innovations in knowledge tracing architecture design, described in the sections below.

The Q-trick

The main novelty of the architecture pertains to how attention is calculated. Traditionally, attention is calculated by matrix multiplication,

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{QK^T}}{\sqrt{n}})\mathbf{V},$$

where the query matrix, Q, is multiplied with the transposed key matrix, K^T. The operation has a complexity of $O(L^2)$ where L is the sequence length. The author realized that using only the last query vector, instead of a matrix containing each query vector in the sequence, only yielded minor performance losses while reducing the complexity of the matrix operation to $O(L)$. This optimization, called the Q-trick, allowed the author to substantially increase the sequence length which yielded a net performance increase, even counting the loss resulting from the optimized matrix multiplication.

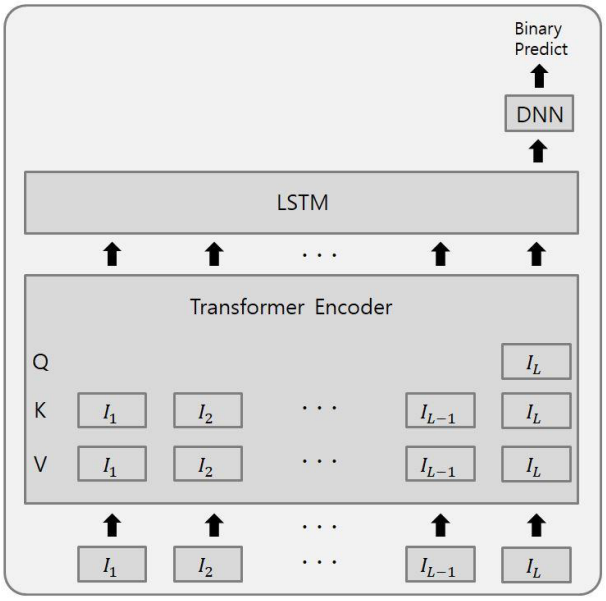


Figure 5.3: Last Query Transformer architecture from Jeon (2021).

Capturing sequence-related patterns with an LSTM

Whereas the output of transformer-based models are usually routed through a DNN before being output as a prediction, the Last Query Transformer routes the output of the transformer through an LSTM before routing it through a DNN. Thus, the transformer encoder is responsible for capturing relationships between questions, and the LSTM is responsible for finding sequence-related patterns in the data. Intuitively this can be understood as the encoder supplying the LSTM network with its understanding which is based on the relationships between every question and the LSTM network will use this information to find meaningful sequential patterns, which the encoder cannot do by itself. In this sense, the LSTM layer effectively has a role similar to the decoder in a Transformer neural network.

5.2.5 Relation matrix

In order to improve upon the results of the KEETAR model, we used an *exercise relation matrix* similar to the one described by Pandey and Srivastava (2020). Since the datasets did not contain the text contents of the questions, we were unable to calculate textual relations and our relation matrix relied exclusively on the so-called ϕ*-coefficients* between the correctness of each question based on the questions that came before it. Our motivation for adding a relation matrix was that by adding this value to the attention scores of a transformer model, information about the long term correlation between the correctness of different questions can be obtained. This information would then complement the attention model's more local, short-term information, resulting in better predictions.

Chapter 6
Experimental settings

6.1 Code and implementation

When available, we used the author's code for the models. However, some of the papers lacked publicly available code. In some other papers, the code was written in old versions of frameworks which could not be run on current Python environments or did not work with current CUDA versions. In these cases, we re-implemented the models in PyTorch 1.7.1, following the description of the model given in the respective model's paper as closely as possible.

6.2 Training

For training, we used several datasets in addition to the Akribian dataset, as described in chapter 4. We ran each model was run on each dataset and logged the results for each run. All models were trained on one Nvidia RTX 2070 GPU using cuDNN 8.0.4. The optimizer used for training was Adam with a step size of 10^{-3} and β values of $(0.9, 0.999)$.

For models whose hyperparameters are described in their paper, we used the mentioned hyperparameters. Otherwise, we used manual hyperparameter tuning to find hyperparameters that offered accuracy close to the author's reported accuracy, while also taking care to not increase training time too much.

In some cases, we weren't able to match the authors' reported results due to a lack of transparency regarding the code, the hyperparameters and the datasets for the models. In addition, some models had high variance in their final accuracy for each dataset, which implies that better results are likely possible given a large enough number of training attempts. This was not investigated due to time constraints, electing instead to run each model only a few times.

6.3 Loss functions

When predicting correctness, binary cross entropy defined as

$$L = -\frac{1}{N}\sum_{i=1}^{N} \ell(y_i, \hat{y}_i)$$

is used as a loss function, where the loss L is dependent on the ground truth correctness y_i and the predicted correctness $\hat{y}_i$ and

$$\ell(y, \hat{y}) = y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i),$$

was used as a loss function.

When predicting response time, mean square error (MSE) defined as

$$L = \frac{1}{N}\sum_{i=1}^{N} (\hat{y}_i - y_i)^2,$$

was instead used as a loss function.

6.4 Dropout

To prevent overfitting to the training data, we used dropout. After testing different values, a dropout rate of 0.2 showed the largest increase in validation AUC and was therefore chosen as the default value.

6.5 Evaluation

We split each dataset into a training and a validation dataset with a 95:5 ratio. Instead of shuffling the datasets, each dataset was split into 20 sequential segments, and each segment was split and concatenated into the training and validation datasets. This showed to be faster and give better results for most of our datasets.

After training, each model was evaluated on the validation data sequences using the correct criterion for each task.

6.5.1 Evaluation Criterion

In order to measure and compare models within the tasks of correctness and response time prediction suitable evaluation criterion are needed. Whereas correctness prediction is a binary classification problem with two classes (correct and incorrect), response time prediction is a regression problem with a continuous range of possible values. Because of this inherent difference between the tasks, two different approaches have been used.

Evaluating correctness prediction

Since correctness prediction is a binary classification problem, when comparing two models the better model is able to better separate correct and incorrect answers. Area under

curve (AUC) effectively measures this separation by using a Receiver Operating Characteristic (ROC) curve which plots the ratio between the True Positive Rate (TPR) and False Positive Rate (FPR) for each possible threshold value.

Whereas a model's predictions range in a continuous range from 0-1, a perfect model would, for some arbitrary threshold value such as 0.5, be able to correctly output a value below 0.5 for incorrect answers and output a value over 0.5 for correct answers. This perfect separation would yield an AUC score of 1, whereas a model that is not able to achieve any separation at all would yield a score of 0.5.

Evaluating response time prediction

For regression problems mean square error (MSE) and R^2 are among the most common evaluation metrics.

MSE is an intuitive metric since it can be measured in actual time units and is thus suitable for comparing the performance of different models on the same dataset. Since MSE is dependent on the response time distribution of each dataset, it is not as suitable for evaluating models across different datasets.

The R^2 score, which measures the squared correlation of the predicted value and the ground truth, is on the other hand more suitable for comparing different datasets.

Chapter 7
Results and discussion

7.1 Correctness prediction

The best results for each model and dataset can be seen in Table 7.1. In addition to the listed models, a simple statistical baseline was created for comparing model performance.

The baseline consists of a *dictionary* that simply returns the mean correctness (the ratio of correct answers per unique exercise) of each unique exercise in the training dataset, without using any machine learning methods.

	ASSISTments 2012	Junyi Academy	EdNet	Akribian
Baseline	0.720	0.688	0.717	0.830
DKT	**0.980**	0.793	0.770	**0.965**
SAKT	0.758	0.757	0.753	0.917
SAINT	0.741	0.767	0.764	0.919
KEETAR	0.948	**0.794**	**0.792**	0.940

Table 7.1: Table of models' AUC score for correctness prediction across datasets

While DKT and KEETAR clearly outperform SAKT and SAINT on the ASSISTments and Akribian datasets, the models' performance is more equal on the other datasets. When analyzing Figures 4.1 and 4.2, a significantly higher degree of exercises with a standard deviation of zero can be observed in comparison to the other datasets. That is, for a large set of exercises in both ASSISTments and Akribian every student has answered uniformly according to the mean for that exercise. The reason for this is that the ASSISTments and Akribian datasets have comparatively fewer entries per unique exercise than the Junyi and Ednet datasets; ASSISTments and Akribian have on average 52 and 91 entries per unique exercise while Junyi and Ednet have on average 12221 and 7485 entries per exercise, respectively.

With so few entries per exercise, it might be the case that SAKT and SAINT, both being models purely based on attention, do not have enough data to meaningfully model relationships between exercises. DKT and KEETAR, on the other hand, may be able to leverage the nature of their LSTM components to forego relationships between exercises and instead focus on input features and the short-term performance of each student.

This phenomenon is most evident in the Akribian dataset, which is also by design heavily skewed towards correct answers as seen in Figure 4.1(a). It is thus not surprising that Akribian is the dataset with highest overall AUC scores.

7.1.1 Feature study

In order to assess the impact of different features on model performance, a feature study consisting of two phases has been conducted. In the first phase, the impact of adding individual to a baseline is measured. In the second phase, the most promising features will be combined in order to find the feature combinations that yield the best performance.

Individual features

Table 7.2: Effect of individual features on performance of KEETAR correctness prediction model on the ASSISTments 2012 dataset.

	Baseline	**Baseline + Q**	**Baseline + R**	**Baseline + Z**
AUC	0.910	0.801	0.909	0.929
AUC gain	–	-0.109	-0.001	+0.019
	Baseline + T	**Baseline + S**	**Baseline + MR**	**Baseline + MC**
AUC	0.837	0.910	0.940	0.948
AUC gain	-0.073	0.000	+0.030	+0.038

[Q] Q-trick.
[R] Response time.
[Z] Z-scored response time.
[T] Timestamp difference.
[MR] Mean response time.
[MC] Mean correctness.
[S] Sine positional embeddings.

Table 7.2 shows the effects of applying additional features to a baseline model which consists of exercise ID embeddings, exercise category embeddings, random positional embeddings and past correctness embeddings. Response time and sinusoidal positional embeddings had minimal effects on the final performance of the model. The application of the Q-trick, and timestamp difference embeddings resulted in lower performance. However, Z-scored response time, mean response time and mean correctness had a positive effect on model performance.

***Mean* features** The *mean response time* and *mean correctness* features have shown a considerable increase in the performance of the model. Both of them may be a sensible starting point for predicting correctness, as they provide a measure of the average difficulty of an exercise. This should be most evident in datasets with low variance in correctness per exercise, as the model could simply predict a value close to the mean and be quite accurate.

Z-scored response time The performance increase from the Z-scored response time could be due to the model learning to deduce the fluency level of the student. After all, students who struggle with knowledge concepts might respond very slowly on the exercises related to the knowledge concept. On the other hand, students with higher mastery of the knowledge concept might respond quicker.

Q-trick The Q-trick, while providing some training time speed-up, had too much of a negative impact on performance to warrant using it. It should be noted however that the code for the KEETAR model code was not available. Therefore we had to implement it from scratch by following the description given by Jeon (2021). This means that it is entirely possible that our implementation of the KEETAR model is wrong, but we have no way to confirm or deny this since we do not have access to the code.

Multiple features

In the second phase of the features study, the features that by themselves yielded improved performance when added to the baseline model will be combined to study their synergy. There is of course a possibility that features that by themselves did not improve performance could do so when combined with other features. However, because of time-constraints we have chosen to focus on the most promising features.

Table 7.3: Feature study for KEETAR correctness prediction model on the ASSISTments 2012 dataset.

	Z+MR+MC	MR+MC	Z+MC	Z+MR
AUC	0.934	0.934	**0.951**	0.936

[Z] Z-scored response time.
[MR] Mean response time.
[MC] Mean correctness.

Table 7.3 shows the results of the multiple feature study. We see that any combination of additional features is an improvement over the baseline. However, only one combination of features has a higher performance than the maximum of its component features, namely the combination of z-scored response time and mean correctness.

Z-scored response time and mean correctness Combining z-scored response time and mean correctness yielded the best results when added to the baseline. Intuitively this does not seem surprising. Z-scored response time measures the students speed compared to other students, which in turn correlates with fluency. Using this measure of relative fluency, the model should be able to better compare the student to others than when only using the student's past correctness. It is possible that the understanding of the student's relative fluency can improve the model's prediction, when coupled with the mean correctness of the question, which can be seen as a measure of difficulty.

	Baseline	Baseline and relation matrix
AUC	0.796	0.789

Table 7.4: Effect of relation matrix on performance of KEETAR correctness prediction model on Junyi Academy dataset.

7.1.2 Effect of relation matrix

The relation matrix used in this study is somewhat different from that used by Pandey and Srivastava (2020). The main difference is that our relation matrix implementation does not use textual relation between exercises, since we did not have access to the text for the exercises. Our relation matrix consists of only the ϕ correlation coefficients, implemented by following the equations in Pandey and Srivastava (2020).

The application of relation matrix to the KEETAR model did not result in an increase in performance on any dataset. The specific test run shown in Table 7.4 uses the authors' recommended thresholding hyperparameter $\theta = 0.8$. Since the paper did not specify the specific λ coefficient which is used for blending, we used a value of 0.5. Additional trials were run on different datasets with different values of θ and λ, but none showed an increase in AUC compared to running the same model without a relation matrix.

In addition, certain practical constraints prevented the effective use of the relation matrix on certain datasets. The ASSISTments 2012 dataset, for example, contains over 50,000 unique exercises, and as such produced a relation matrix that was over 20 GB in size. This effectively prevented us from being able to use the relation matrix in our model since it was far too large to fit on the GPU memory or the RAM.

7.2 Response time prediction

The best results for each model and dataset can be seen in Tables 7.5 and 7.6. A baseline was created by calculating the mean response time for each exercise in the training set. For each exercise in the test set, the calculated mean response time is used as the baseline's prediction.

	ASSISTments 2012	Junyi Academy	Ednet	Akribian
Baseline	10.65	13.91	6.40	1.30
DKT	10.67	13.95	**3.465**	**0.572**
SAKT	10.56	12.96	4.56	0.60
SAINT	12.21	**12.951**	5.35	0.74
KEETAR	**9.350**	13.68	4.03	0.58

Table 7.5: Table of models' MAE score (in seconds) for response time prediction across datasets

It can be seen that the results vary between datasets, with two of the datasets achieving best results using the simpler DKT model. This is in contrast to correctness prediction where KEETAR performed better on average. We believe this could mean that response time is more dependent on short-term patterns in user behaviour, since RNN-based models are designed

	ASSISTments 2012	Junyi Academy	Ednet	Akribian
Baseline	0.264	0.176	0.128	0.574
DKT	0.494	0.156	**0.639**	**0.994**
SAKT	0.533	0.227	0.474	0.888
SAINT	0.459	**0.235**	0.361	0.856
KEETAR	**0.580**	0.187	0.610	0.920

Table 7.6: Table of models' R^2 score for response time prediction across datasets

to find short-term patterns in data, as opposed to attention which is better at modeling long-term relations.

Furthermore, the AI-models are able to achieve a significant performance increase compared to baseline for the EdNet and Akribian datasets. This means that the models are learning meaningful information from the data. While the other datasets have not outperformed the baseline models to the same degree, the results show that there actually is meaningful patterns in the data that can be used for response time predictions.

7.2.1 Feature study

We ran two separate features studies using different datasets on the KEETAR model to investigate how different features affect performance on different datasets.

The effects of adding individual features to the KEETAR model when predicting the Junyi Academy and ASSISTments datasets can be seen in Tables 7.7 and 7.8, respectively. Since these feature studies were performed at different times during the thesis, there are differences in the premise of the studies. The baseline for Junyi Academy consists of ID embeddings, exercise category embeddings and random positional embeddings. The baseline for ASSISTments also includes past correctness embeddings.

Lending to the differences between the datasets, as well as the fact that the baselines are different, some features may improve performance in one of the datasets, but not the other. For example, mean response time and mean correctness give a major performance increase for the ASSISTments dataset, while actually decreasing performance for the Junyi Academy dataset. This may be caused by the higher response variance of the Junyi Academy dataset.

A feature that has yielded a major performance increase for both datasets is z-scored response time. Interestingly, it works better than response time itself.

Time-stamp difference, which is often used to model forgetfulness, also gives a minor performance increase. It is however likely that the model is actually learning how to deduce response time from it when used as as single feature. Its suitability as a feature should thus be measured in its capacity to increase performance when added to a model already using z-scored and normal response time.

Sine positional embeddings also give a minor performance increase since the embeddings are designed to be preserve some notion of continuity between subsequent positions. This makes it easier for the attention-based KEETAR model to understand the order of the input data.

Table 7.7: Effect of features on performance of KEETAR response time prediction model on the Junyi Academy dataset.

	Baseline	**Baseline + C**	**Baseline + R**	**Baseline + Z**
MAE (s)	14.27	14.18	13.27	13.26
MAE difference	–	-0.09	-1.00	-1.01
	Baseline + T	**Baseline + S**	**Baseline + MR**	**Baseline + MC**
MAE (s)	14.20	14.04	14.41	14.51
MAE difference	-0.07	-0.23	+0.14	+0.24

Table 7.8: Effect of features on performance of KEETAR response time prediction model on the ASSISTments 2012 dataset.

	Baseline	**Baseline + Q**	**Baseline + R**	**Baseline + Z**
MAE (s)	13.92	14.24	14.64	12.47
MAE difference	–	+0.32	+0.72	-1.45
	Baseline + T	**Baseline + S**	**Baseline + MR**	**Baseline + MC**
MAE (s)	12.82	12.75	11.06	12.06
MAE difference	-1.10	-1.17	-2.86	-1.86

[C] Past correctness embeddings.
[Q] Q-trick.
[R] Response time.
[Z] Z-scored response time.
[T] Timestamp difference.
[MR] Mean response time.
[MC] Mean correctness.
[S] Sine positional embeddings.

Table 7.8 shows the results of the feature study for response time prediction. Most features have a positive impact on the results, apart from response time and Q-trick. We also tried different combinations of features, but no combination yielded better performance than the best individual features.

Response time and z-scored response time

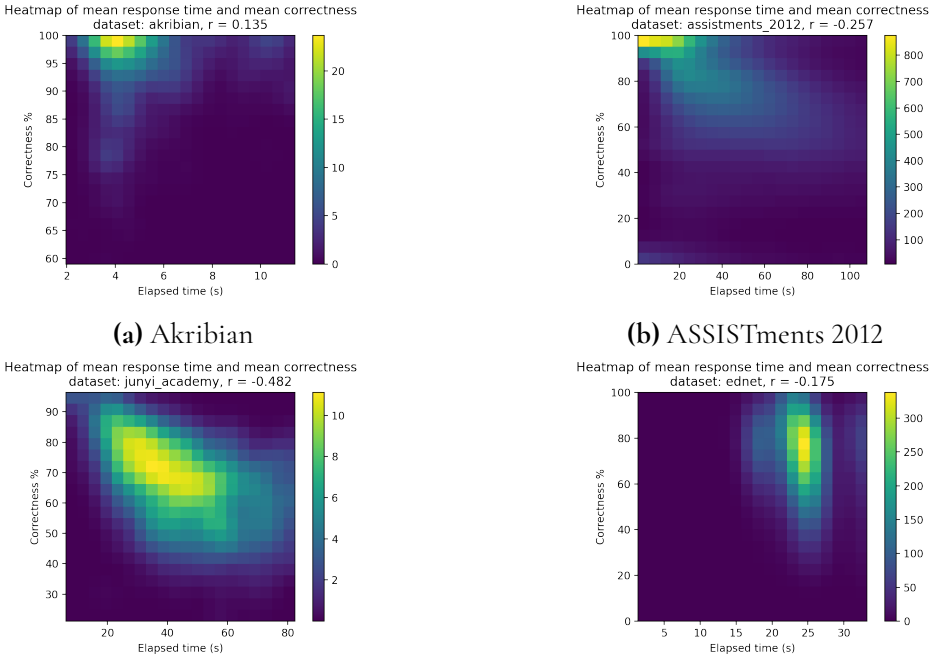
A noteworthy result of the feature study is that when predicting on the ASSISTments dataset, including the response times of the previous exercises as a feature seems to have a negative effect on response time prediction for the current exercise. However, including the z-scored response times of the previous exercises seems to have a positive effect.

This can be explained by the fact that the z-scored response times of the previous question can tell the model about the speed at which the user is solving exercises, since the information it contains is relative to other responses to the same exercise. For example, a user who has solved the past few exercises quickly is likely to solve the current question at a similarly fast rate. The response time value in seconds, however, needs additional information such as the mean response time of the question which was answered, which is contained in the *previous* time step. As such, it is difficult for the model to infer information about the user's fluency from this feature.

However, on the Junyi Academy dataset, both the response time and the z-scored response time features seemed to have an equivalent positive effect on the performance of the model.

Mean response time and mean correctness

Another set of features that improved the performance of the response time prediction for the ASSISTments dataset was the *mean* features – namely, mean response time and mean correctness. These features are *query* features, which means that they describe the question currently being answered. As such, mean response time provides the model with a hint about what the magnitude of response time for the current question.



(a) Akribian

(b) ASSISTments 2012

(c) Junyi Academy

(d) EdNet

Figure 7.1: 2D histogram of mean response time per question and correctness ratio. Mean response time values above the 90th percentile have been removed for clarity.

The performance gain from mean correctness however, is not as intuitive to explain for the response time prediction model. We speculate that it might be related to the fact that mean correctness and mean response time are not independent of each other. In other words, we believe it is likely that easy questions would take less time to answer and more difficult questions would take longer to answer. Statistically, this would imply an inverse correlation between response time and correctness. To confirm or deny this hypothesis, we conducted a statistical analysis, the results of which are visible in Figure 7.1. It can be seen that exercises in most datasets, especially Junyi Academy, have a slight negative correlation between mean response time and mean correctness. The exception is Akribian, which instead has a very slight *positive* correlation.

This difference could be thanks to the difference in the nature of the data in the Akribian dataset. The data in the Akribian dataset comes from an educational game for young children, whereas the data for the other datasets comes from e-learning platforms for older demographics.

On the Junyi Academy dataset, these features decreased model performance. This could be because these features may not provide useful information, thanks to the high variance in

the response times in the Junyi Academy dataset, as seen in Figure 4.3d. Contrast this with the low response time variance found in the ASSISTments datasets, seen in Figure 4.2d.

7.3 Comparison between response time and correctness prediction

When comparing Tables 7.1 and 7.5, it can be seen that different models have varying performance on different dataset and there is no model that performs best on every dataset.

When predicting performance, KEETAR performs better on larger datasets while DKT performs better on the smaller ones. However, when predicting response time, DKT – which is an LSTM-based model – performs better on the large EdNet dataset, as well as the small Akribian dataset. KEETAR performs better on the ASSISTments 2012 dataset, and SAINT performs better on Junyi Academy dataset.

This disparity could be caused by the fact that the task of response time prediction is more dependent on short-term connections than the task of correctness prediction. When predicting correctness, it is after all important to know whether a student has answered correctly on past questions with related concepts. On the other hand, response time may be more closely tied to the current pace of the student than their knowledge of related concepts.

If this is the case, then the comparatively better performance of DKT is not surprising, considering that the advantage of attention in comparison to LSTM is the ability to model long-term relationships.

Chapter 8
Conclusion and future work

This thesis aimed to adapt existing models within the field of knowledge tracing to the task of predicting the exercise response time of students using online learning platforms. By benchmarking several models and datasets and comparing to simple statistical baselines, we achieved promising initial results.

By studying the effect of different features on the performance of the KEETAR model in the task of knowledge tracing, we showed that certain features yielded a significant performance increase. In particular, two features that we engineered, namely mean correctness and z-scored response time, yielded a major performance increase when predicting correctness.

Furthermore, a comprehensive comparison of the largest datasets and most common models yielded insights into why certain models may work better than others for a particular dataset. In particular, we found that the differences in performance were higher across datasets than models. Bigger datasets that have a lot of answers for each unique exercise were harder to predict than smaller datasets. The characteristics of the dataset, in particular the variance in the predicted variables, must be taken into account. When dealing with datasets with low response time variance in correctness and response times, for example, mean correctness and mean response time features provide large performance increases. Furthermore, LSTM has performed better on datasets with low variance whereas KEETAR, an attention-based model, performed better on the datasets with high variance.

Due to the decision making process of machine learning models being inherently difficult to interpret, we don't actually know whether or not the model's are learning meaningful representations of student knowledge. Because of this, we believe that further research should be conducted in order to interpret the decision making process of current algorithms.

In the end, knowledge tracing predictions will have to be integrated with educational models in order to actually be applied in the real world. When doing so, we believe that the addition of response time prediction will allow for more nuanced models, especially for quantifying fluency which is highly correlated with speed. We hope that our models can find applications in learning path recommendations in e-learning platforms.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*.

Choi, Y., Lee, Y., Cho, J., Baek, J., Kim, B., Cha, Y., Shin, D., Bae, C., and Heo, J. (2020a). Towards an appropriate query, key, and value computation for knowledge tracing. In *Proceedings of the Seventh ACM Conference on Learning @ Scale*, L@S '20, page 341–344, New York, NY, USA. Association for Computing Machinery.

Choi, Y., Lee, Y., Shin, D., Cho, J., Park, S., Lee, S., Baek, J., Bae, C., Kim, B., and Heo, J. (2020b). Ednet: A large-scale hierarchical dataset in education. In Bittencourt, I. I., Cukurova, M., Muldner, K., Luckin, R., and Millán, E., editors, *Artificial Intelligence in Education*, pages 69–73, Cham. Springer International Publishing.

Corbett, A. T. and Anderson, J. R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modelling and User-Adapted Interaction*, 4(4):253–278.

Feng, M., Heffernan, N., and Koedinger, K. (2009). Addressing the assessment challenge with an online system that tutors as it assesses. *User Modeling and User-Adapted Interaction*, 19(3):243–266.

Gers, F., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural computation*, 12:2451–71.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Jeon, S. (2021). Last query transformer RNN for knowledge tracing. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI-21)*.

Khajah, M., Lindsey, R. V., and Mozer, M. C. (2016). How deep is knowledge tracing? In *Proceedings of the 9th International Conference on Educational Data Mining*, volume abs/1604.02416.

Kolen, J. F. and Kremer, S. C. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. In *A Field Guide to Dynamical Recurrent Networks*, pages 237–243.

Olah, C. (2015). *Understanding LSTM Networks*. Github. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Pandey, S. and Karypis, G. (2019). A self-attentive model for knowledge tracing. In *Proceedings of The 12th International Conference on Educational Data Mining (EDM 2019)*.

Pandey, S. and Srivastava, J. (2020). Rkt: Relation-aware self-attention for knowledge tracing. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. ACM.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Piech, C., Spencer, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L. J., and Sohl-Dickstein, J. (2015). Deep knowledge tracing. *CoRR*, abs/1506.05908.

Rumelhart, D. E. and McClelland, J. L. (1987). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 6–7. MIT Press, Cambridge, MA.

Shin, D., Shim, Y., Yu, H., Lee, S., Kim, B., and Choi, Y. (2021). Saint+: Integrating temporal features for ednet correctness prediction. In *Proceedings of the 11th International Learning Analytics and Knowledge Conference (LAK21)*, page 490–496, New York, NY, USA. Association for Computing Machinery.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

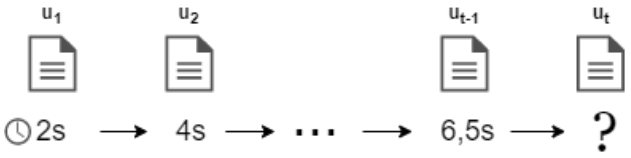
EXAMENSARBETE Applying Knowledge Tracing to Predict Exercise Response Time
STUDENTER Shamiran Jaf, Sepehr Noorzadeh
HANDLEDARE Pierre Nugues (LTH)
EXAMINATOR Jacek Malec (LTH)

Förutspå elevers svarstid med hjälp av maskininlärning

POPULÄRVETENSKAPLIG SAMMANFATTNING **Shamiran Jaf, Sepehr Noorzadeh**

Att kunna förutspå elevers framtida prestation kan vara ett kraftfullt pedagogiskt verktyg. Tidigare forskning har varit fokuserad på att förutspå korrektheten av framtida svar, men pedagogikforskare tycker att tempo är lika viktigt när man bedömer kunskapsnivå. Vårt arbete visar att det går att använda maskininlärning för att förutspå svarstid.

Att modellera elevers kunskap utifrån hur de interagerar med ett kursmaterial är ett väldigt svårt problem att lösa. Om det löstes skulle digitala läroplattformar kunna anpassa sig till varje elev och ge precis det stöd som eleven behöver. Man har försökt lösa problemet med hjälp av träna maskininlärningsmodeller till att lära sig modellera elevens kunskap, och använda denna interna modell för att förutspå elevens framtida prestation.



Hittills har man enbart tränat maskininlärningsmodeller för att förutspå korrekthet, det vill säga om en elev kommer svara rätt eller fel på en ännu obesvarad uppgift. Däremot vet man inom pedagogiken att tempot hos en student är en lika viktig faktor som korrekthet för att kunna bedöma en elevs kunskapsnivå. I vårt examensarbete har vi därför applicerat maskininlärningsmodeller, som tidigare använts till att förutspå korrekthet, till att förutspå svarstid.

De modeller som har använts till att förutspå korrekthet har från början skapats för språkbehandling. Dessa modeller är bra på att modellera samband mellan olika ord i meningar. De passar bra till vår uppgift i och med att de kan också modellera samband mellan uppgifter som följer efter varandra.

Modellerna testades på data från flera olika digitala inlärningsplattformar. Eftersom en maskininlärningsmodell är ytterst beroende av vilken information den tillhandahålls så studerade vi effekterna av att köra våra modeller på olika sorters information. Bland annat märkte vi att prestandan ökade avsevärt när vi gav modeller information om korrekthetens och svarstidens genomsnitt för den uppgift vars resultat ska förutspå.

Vårt resultat visar att det går att träna modeller som relativt noggrant kan förutspå elevers svarstid. De här modellerna kan kompletteras med tidigare modeller som förutspår korrekthet för att göra en mer helhetlig bedömning om elevers kunskapsnivå och skräddarsy kursmaterialet för varje elev.