Encode & Decode: Generalizing Deep Knowledge Tracing and Multidimensional Item Response Theory

ABSTRACT

Knowledge tracing consists in predicting the performance of some students on new questions given their performance on previous questions, and can be a prior step to optimizing assessment and learning. Deep knowledge tracing (DKT) is a competitive model for knowledge tracing relying on recurrent neural networks, even if some simpler models may match its performance. However, little is known about why DKT works so well. In this paper, we frame deep knowledge tracing as a encoder-decoder architecture, which leads us to propose better models and promising future research directions. In particular, we show on several small and large datasets that a simpler decoder than the one used by DKT can predict student performance better.

Keywords

knowledge tracing, side information, item response theory

1. INTRODUCTION

Adaptive testing and personalized learning are precious technologies that have been enabled by tracing the knowledge of previous students. They rely on a response model of the learners: if we know how people learned in the past, we can optimize testing or learning for new students. This is the model-based approach of reinforcement learning, commonly referred to as knowledge tracing in the educational data mining literature.

Formally, knowledge tracing relies in predicting the outcomes of new students on some items, given the performance of various former students on these items. Numerous models have been proposed for knowledge tracing such as deep knowledge tracing (DKT) that relies on a recurrent neural network (RNN) [9]. However, Wilson et al. [13] have matched the performance of DKT with a unidimensional item response theory (IRT) model that can be seen as online logistic regression. In order to advance the field, we need to understand what distinguishes these models, and how to build upon them. For example, we can notice that the data to which these models had access was not exactly the same. IRT usually learns a difficulty parameter per item while DKT were initially strictly used on skill data for the sake of the comparison with Bayesian knowledge tracing (BKT).

In this paper, we show how we can bridge both categories of models using an encoder-decoder architecture. These architectures are usually encountered in sequence-to-sequence scenarios such as machine translation [1]. We open the black box of DKT and see how it relates to existing, well known models. This will lead us to provide suggestions of models according to the size of the dataset.

Our main contribution is an encoder-decoder architecture that take DKT, IRT and other models such as Performance Factor Analysis (PFA) as special cases. We show that it is better to learn unidimensional parameters for the decoder, which is not what the vanilla DKT model does. We finally demonstrate that RNNs can also provide excellent results even on small datasets.

We first expose related work, then our approach: encode & decode. We subsequently detail our datasets, experiments, and see the influence of specific components of our architecture in the results.

2. BACKGROUND AND RELATED WORK

Formally, knowledge tracing can be defined as follows. Let us denote I the set of items of some test. For each student, at each time step T, we know the sequence of items and outcomes already given to the student $(q_t, a_t)_{1 \leq t < T}$ where $q_t \in I$ and $a_t \in \{0, 1\}$, as well as some potential side information such as the knowledge components (KCs) required by q_t and denoted by $KC(q_t) \in \{0, 1\}^K$. Using this information, we need to predict a_t. We will note σ the sigmoid function : $\sigma : x \mapsto 1/(1 + \exp(-x))$ and logit the inverse function of σ. We now describe some models for knowledge tracing.

MIRT. In item response theory, we usually do not assume that the examinee's ability evolves over time. The probability that user i correctly answers item j is:

$$\text{logit}\, Pr(\text{user } i \text{ answers correctly item } j) = \langle \boldsymbol{u_i}, \boldsymbol{v_j} \rangle + \delta_j$$

where $\boldsymbol{u_i} \in \mathbf{R}^d$ is a learned representation of user i, $\boldsymbol{v_j}$ is a learned representation of item j and δ_j is a bias parameter

representing the easiness of item j. This model is popular in the psychometric literature, because it can allow live adaptation of the assessment, given the performance of the student.

DKT. Deep knowledge tracing is usually described as a black-box model that takes into input pairs $(q_t, a_t)_{1 \leq t < T}$ from some student, and outputs a vector of probabilities $\boldsymbol{y}$ such that y_k is the probability that the student will answer correctly an item requiring skill k. More precisely, in a DKT model, the actual probability that user i correctly answers an item that requires skill k at time t (event $R_{ikt} = 1$) is given by:

$$\text{logit}\, Pr(R_{ikt} = 1) = \langle \boldsymbol{h_t}, \boldsymbol{s_k} \rangle$$

where $\boldsymbol{s_k}$ is a representation of the skill learned by DKT that does not evolve over time and $\boldsymbol{h_t} = LSTM(\boldsymbol{h_{t-1}}, q_t, a_t)$ is a representation of the user that evolves over time. Variants of DKT have been proposed [6]. In [7], Montero et al. show that DKT can trace knowledge efficiently even when skills are interleaved, and that it shares information between skills.

We can already notice how similar DKT and MIRT compute the logit of the probability of a correct outcome from the student. But using skill representation or item representation should not be mutually exclusive, as shown with the following model.

KTM In [11], Vie and Kashima have shown that it is possible to learn representations for all users, items, skills in a test and combine them in a pairwise manner. Interestingly, most existing models for knowledge tracing such as IRT, PFA, MIRT are special cases of KTMs, according to the representations considered in the modeling. We note $\boldsymbol{x}$ (called *metadata* in this article) the encoding of the features of an event, for example: "user i answers correctly item j that requires skills k_1 and k_2" may be encoded by the concatenation of a one-hot vector for user i, another one-hot vector for item j, and a 2-hot vector for skills k_1 and k_2. KTM learn a bias $w_k \in \mathbf{R}$ and an embedding $\boldsymbol{v_k} \in \mathbf{R}^d$ for each feature k.

$$\text{logit}\, Pr(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i<j} x_i x_j \langle \boldsymbol{v_i}, \boldsymbol{v_j} \rangle.$$

3. OUR APPROACH: ENCODE & DECODE

We see all these models through the same lens: our architecture relies on two main components, an encoder E and a decoder D that are trained jointly. They respectively require an expression of the metadata of former actions from the student $\boldsymbol{x_t^{in}}$ and metadata of assessment at time t: $\boldsymbol{x_t^{out}}$.

$$\begin{cases} \boldsymbol{h_t} = E(\boldsymbol{h_{t-1}}, \boldsymbol{x_t^{in}}) \\ p_t = D(\boldsymbol{h_t}, \boldsymbol{x_t^{out}}) \end{cases} \quad t = 1, \ldots, T$$

and $\boldsymbol{h_0} = \boldsymbol{0}$. In this expression, $\boldsymbol{h_t}$ represents the learned representation of the user at time t, and p_t the probability that the attempt of the student at time t will be correct. Please note that this modeling is rich enough to encompass broader tasks than knowledge tracing, for example we can model the event that user u watched the video of some lesson, with the corresponding metadata. If we can take advantage of this kind of data, we can hopefully recommend new videos to some student to optimize their learning.

3.1 Encode: learn representations that evolve over time

The representation of the student at time t is given by $\boldsymbol{h_t} = E(\boldsymbol{h_{t-1}}, \boldsymbol{x_t^{in}})$. Let us see different examples of encoders.

3.1.1 Constant, or maximum likelihood estimation

In MIRT, the ability of student i does not evolve over time: $E_{MIRT} = \boldsymbol{u_i}$ which is learned. In [13], when Wilson et al. compare the performance of DKT to IRT, at test time they dynamically estimate the most likely user ability according to the previous student outcomes, and the learned item parameters. Maximum likelihood estimation (MLE) is possible with simple models such as IRT, but for more complex architectures or priors, it is not always feasible because it increases the time complexity during the test phase.

3.1.2 Counters

The famous model and hard baseline Performance Factor Analysis (PFA) [8] can also be encoded as our approach. Let us consider the binary vectors $KC(q_t)$ and $\boldsymbol{x_t^{out}} = KC(q_{t+1})$ that are the corresponding rows of items q_t and q_{t+1} in a q-matrix. We note $\boldsymbol{x_t^{in}} = (a_t KC(q_t), (1 - a_t) KC(q_t))$, where $(\cdot, \cdot)$ denotes concatenation. The first K components of $\boldsymbol{x_t^{in}}$ are 0 if the response was incorrect and the last K components of $\boldsymbol{x_t^{in}}$ are 0 if the response was correct. The estimated ability of the student over time is a linear function of their number of previous successful and unsuccessful attempts:

$$\begin{cases} E_{PFA}: & \boldsymbol{h_t} = \boldsymbol{h_{t-1}} + \underbrace{(a_t KC(q_t), (1 - a_t) KC(q_t))}_{\boldsymbol{x_t^{in}}} \\ D_{PFA}: & p_t = \sigma(\langle \boldsymbol{h_t}, \boldsymbol{w}, \boldsymbol{x_t^{out}} \rangle + \langle \boldsymbol{x_t^{out}}, \boldsymbol{\beta} \rangle) \end{cases}$$

where $\langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \rangle = \sum_k a_k b_k c_k$.

In other words, $\boldsymbol{h_t} = (W_{i1}, \ldots, W_{iK}, F_{i1}, \ldots, F_{iK})$ where W_{ik} (F_{ik}) is the number of previous successes (failures) of student i over skill k: $W_{ik} = \sum_{t:k \in KC(q_t)} a_t$.

3.1.3 Recurrent neural networks (RNN)

In DKT, the ability of the student is a function of their previous actions, i.e., pairs (item q_t, outcome a_t):

$$E_{DKT}: \boldsymbol{h_t} = RNN(\boldsymbol{h_{t-1}}, q_t, a_t).$$

In their original paper, Piech et al. [9] assume that each item is only related to one skill among $1, \ldots, K$. They learn a joint representation $\boldsymbol{x_t^{in}} \in \{0, 1\}^{2K}$ for the pair (s_t, a_t) where $s_t = KC(q_t)$, because they claim that separate representations for s_t and a_t degraded performance. When there are too many pairs of (q_t, a_t), they use fixed low-dimensional representations instead of $\{0, 1\}^{2K}$, inspired by compressed sensing. It makes sense to use skill in lieu of items for encoding the actions of the student, in order to have redundancy and avoid the item cold-start problem.

In DKT-DSC, an extension of DKT, Minn et al. [6] use an encoding of the triplet (q_t, a_t, c_t) as metadata $\boldsymbol{x_t^{in}}$ where c_t is a dynamic clustering information of the student based on their vector $(W_{i1} - F_{i1}, \ldots, W_{iK} - F_{iK})$ updated at each time step. They managed to outperform DKT on several datasets.

3.1.4 Linear projection component

The encoder may need multidimensional parameters to encode learning dynamics, for example if it is a RNN. But in practice, most hard baselines for knowledge tracing such as PFA only model skill parameters with one dimension. So we also introduce a scalar $h'_t = \langle \boldsymbol{w}, \boldsymbol{h_t} \rangle + b$ which is a projection of the multidimensional latent state of the student after a linear layer ($\boldsymbol{w}$ and b are learned). This term can be interpreted as some unidimensional ability of the student at time t.

3.2 Decode: combine learned representations to predict

The objective of the decoder is to combine the learned representation $\boldsymbol{h_t}$ of the student at time t with some parameters involved in the assessment at time t such as item, skill, or some side information such as country, device used, whether the test was pretest, posttest, or practice. Generally, the decoder learns biases $\boldsymbol{w}$ and embeddings V for all features involved in the outcome of item q_t except the user. The metadata $\boldsymbol{x_t^{out}}$ can be seen as a mask to specify which biases or embeddings should contribute to the logit of probability p_t. Many examples of metadata can be seen in [11]. In order to make predictions and generalize to unseen user-item pairs, we use the learned representations to infer the outcomes. $p_t = D(h_t, \boldsymbol{x_t^{out}})$.

3.2.1 Multidimensional decoder

For example, MIRT computes a simple dot product of the user representation and the item representation, plus a bias representing the easiness of the item. So we can fix $\boldsymbol{x_t^{out}}$ as a one-hot vector representing the item q_{t+1} and:

$$\begin{aligned} D_{MIRT} : p_t &= \sigma(\langle \boldsymbol{h_t}, \boldsymbol{v_{q_{t+1}}} \rangle + w_{q_{t+1}}) \\ &= \sigma(\langle \boldsymbol{h_t}, \boldsymbol{x_t^{out}} V \rangle + \langle \boldsymbol{x_t^{out}}, \boldsymbol{w} \rangle). \end{aligned}$$

DKT computes a dot product of the user representation (latent state) and the skill representation. In other words, $\boldsymbol{x_t^{out}}$ is a one-vector representing skill k.

$$\begin{aligned} D_{DKT} : p_t &= \sigma(\langle \boldsymbol{h_t}, \boldsymbol{s_k} \rangle) \\ &= \sigma(\langle \boldsymbol{h_t}, \boldsymbol{x_t^{out}} V \rangle + \langle \boldsymbol{x_t^{out}}, \boldsymbol{w} \rangle). \end{aligned}$$

Please note that the expression of D_{DKT} and D_{MIRT} are the same: only $\boldsymbol{x_t^{out}}$ is different.

3.2.2 Biases

The decoder does not necessarily use embeddings V to compute the predictions. A vector of biases $\boldsymbol{w}$ can be enough, as we will see now.

In PFA, p_t depends on a weighted sum of the counters $\boldsymbol{h_t}$, according to the knowledge components that are assessed at time t and specified in $\boldsymbol{x_t^{out}} = KC(q_{t+1})$:

$$\text{logit}\, p_t = \underbrace{\sum_{k \in KC(q_{t+1})} \gamma_k W_{ik} + \delta_k F_{ik}}_{\langle \boldsymbol{h_t}, \boldsymbol{w}, \boldsymbol{x_t^{out}} \rangle} + \underbrace{\sum_{k \in KC(q_{t+1})} \beta_k}_{\langle \boldsymbol{x_t^{out}}, \boldsymbol{\beta} \rangle}.$$

Parameters $\boldsymbol{w} = (\boldsymbol{\gamma}, \boldsymbol{\delta})$ and $\boldsymbol{\beta}$ are learned, respectively biases for wins, fails and skills. Please note that this formulation inspires us to provide models handling multiple skills. We will denote this kind of encoder as "swf" in the experiments.

We can now see that it is easy to derive more complex decoders by specifying a set of metadata in output $\boldsymbol{x_t^{out}}$, in a similar way to KTM. For example, weights can be learned for the item q_{t+1} as well, and added to the logit during decoding.

3.3 Training and Testing

We optimize the log-loss, also known as mean negative log-likelikood:

$$L(a, p) = \sum_{t=1}^{T} \log(1 - |a_t - p_t|).$$

This log-loss can be computed for a batch of students, but we have to care about the fact that all students did not attempt the same number of items.

Once the parameters of the encoder E and the decoder D have been trained, for a new student it is easy to unroll the encoder on the sequence of outcomes (q_t, a_t) encoded as $\boldsymbol{x_t^{in}}$, get the corresponding latent states $\boldsymbol{h_t}$, and feed them to the decoder with the metadata $\boldsymbol{x_t^{out}}$ to get the predictions, and compute performance metrics.

4. EXPERIMENTS

We report all results according to their accuracy (ACC) and area under the curve (AUC).

4.1 Models

We tried different combinations of encoders and decoders. We excluded MIRT from the baselines because it is not evolving over time.

4.1.1 Encoders

- RNN of latent dimensionality d where the encoding of actions (s_t, a_t) is sampled from a Gaussian multivariate distribution of dimension d and fixed. We used GRU (Gated Recurrent Unit) because it has fewer parameters than LSTM (Long Short-Term Memory), so it is less prone to overfitting [**?**].
- counter: like PFA described above.
- no encoder: there is no latent state learned for the student. Predictions only rely on $\boldsymbol{x_t^{out}}$ and some learned biases and/or embeddings.

4.1.2 Decoders

- multidimensional skills: like in DKT, where embeddings $\boldsymbol{s_k}$ are learned for each skill $k = 1, \ldots, K$ for decoding the latent state $\boldsymbol{h_t}$ of the student. We will refer to it in the experiments as d, and "s" as $\boldsymbol{x_t^{out}}$.
- bias swf: learning biases for skills, wins, fails, like in PFA.
- bias iswf: learning biases for items, skills, wins, fails, like in KTM.
- no decoder: there is no metadata of the test.

Model	Encoder	Decoder	x^{out}	ACC	AUC
Ours	GRU $d = 2$	bias	iswf	**0.880**	**0.944**
Ours	GRU $d = 2$	bias	iswf	0.862	0.929
KTM	counter	bias	iswf	0.853	0.918
PFA	counter	bias	swf	0.854	0.917
Ours	∅	bias	iswf	0.849	0.917
Ours	GRU $d = 50$	∅		0.814	0.880
DKT	GRU $d = 2$	$d = 2$	s	0.772	0.844
Ours	GRU $d = 2$	∅		0.751	0.800

Table 1: Results on the Fraction dataset. The top model was trained during 400 epochs.

For the last three cases, as no multidimensional embedding is learned on the decoder side, predictions rely on h'_t the linear projection of $\boldsymbol{h_t}$ as described in Section 3.1.4. For the last case, the prediction $p_t = \sigma(h't)$ and no bias is added to the logit, because there is no decoder.

4.1.3 Corresponding baselines

DKT has a RNN as encoder and multidimensional parameters within its decoder.

PFA and KTM use as encoder a simple counter of successful and unsuccessful attempts from the student. What distinguishes them is that PFA only models unidimensional parameters (biases) for skill, wins, fails and KTM also considers a bias for item. In all our experiments we did not consider pairwise interactions between embeddings for KTM. So PFA and KTM here are particular cases of logistic regression.

4.2 Datasets

We tried our approach on the following datasets that have diverse characteristics.

Fraction 536 middle-school students attempting 20 fraction subtraction exercises requiring 8 KCs such as, being able to put fractions at the same denominator. All students attempted all items, so this dataset is fully specified. The dataset and description of KCs can be seen in [2]. This dataset is particularly interesting because it is small scale, so complex models may overfit.

Assistments 2009 346860 attempts of 4217 students over 26688 math-related items requiring 123 KCs [3]. Some of the items require multiple KCs, up to 4. Students attempted between 1 and 1382 exercises.

Berkeley 562201 attempts of 1730 students over 234 CS-related items in a MOOC provided by Berkeley. Items can be of 29 categories, that we used as KCs. Each item belong to a single category. Students attempted between 2 and 1020 exercises.

4.3 Preparing the data

We would love to make the reader believe that combining these models is easy, but in practice there are many challenges, fortunately already known in the natural language processing community.

Model	Encoder	Decoder	x^{out}	ACC	AUC
KTM	counter	bias	iswf	0.714	**0.748**
Ours	GRU $d = 50$	bias	iswf	0.711	0.726
Ours	GRU $d = 100$	∅		0.702	0.704
DKT	GRU $d = 50$	$d = 50$	s	0.691	0.701
Ours	∅	bias	iswf	0.681	0.691
PFA	counter	bias	swf	0.682	0.686
DKT	GRU $d = 2$	$d = 2$	s	0.531	0.511

Table 2: Results on the Assistments dataset.

Batching Whenever there are too many data points, in order to save memory we need to sample a batch of users. So the log-loss is computed on a batch.

Sequences of uneven lengths Within a batch, different users may have attempted a different number of questions. However to take advantage of parallel matrix computations, it is better to compute the log-loss over matrices. So we use a mask to know where the sequences end for each student.

Sequences of long lengths On long sequences, DKT is hard to train. First, it may forget information on a long sequence, second the computation of the gradient may take time, vanish to zero or conversely get arbitrary high. As a remedy, we train on windows of fixed size called BPTT parameter: backpropagating through time [12]. The latent state of the previous batch is fed to the next batch.

We used 5-fold cross-validation: we split each dataset into 5 folds, predict any of them using the remaining ones, and average the results.

4.4 Implementation

We optimize our model using Adam [4] with learning rate $\gamma = 0.005$ and weight decay $\lambda = 0.0005$ (equivalent to ℓ_2 regularization). We train using 100 minibatches and use time windows of size 100. For the RNN, we used GRU with one layer, no dropout and latent dimensionality 2, 50 or 100. Our implementation in PyTorch will be available on GitHub after your review. We ran the Fraction dataset experiments on CPU and the Assistments and Berkeley dataset experiments on GPU. Training was stopped after 200 epochs, except for the Fraction dataset where for one of the models (see Table 1), we stopped the training after 400 epochs.

5. RESULTS AND DISCUSSION

Results are shown in Tables 1 to 3.

In all datasets, our method outperformed the vanilla DKT model. More precisely, any model with a multidimensional decoder is outperformed by a model with a unidimensional decoder. Even completely removing the decoder out of DKT can improve the prediction performance, which means the encoder is expressive enough to track the knowledge of the student.

Removing the encoder is equivalent to a KTM, which is the case on all datasets except Assistments, where we conjecture

Model	Encoder	Decoder	x^{out}	ACC	AUC
Ours	GRU $d = 50$	bias	iswf	0.707	**0.778**
KTM	counter	bias	iswf	0.704	**0.775**
Ours	$\varnothing$	bias	iswf	0.700	0.770
DKT	GRU $d = 50$	$d = 50$	s	0.684	0.751
Ours	GRU $d = 100$	$\varnothing$		0.682	0.750
PFA	counter	bias	swf	0.630	0.683
DKT	GRU $d = 2$	$d = 2$	s	0.637	0.656

Table 3: Results on the Berkeley dataset.

that as there are many items, we might suffer from the item cold-start problem by assuming an item bias. Also, our initialization of the biases is different from KTM, which may explain the discrepancy on this dataset.

On the large datasets Assistments and Berkeley, KTM, that uses a simple counter of successful and unsuccessful attempts at skill level as encoder, is among the top models. It seems to indicate that when the information is abundant, logistic regression may be enough, because the number of successes and failures is good enough as predictor. It also means that we do not need to train a deep neural network. What is even more surprising, is that on the small dataset Fraction, the top performing model uses a recurrent neural network as encoder. It may be because the sequences are small: every student only attempts 20 items. So the counter of successes are low and the sequences are small enough to be considered by the RNN.

Considering an item bias does not improve a lot the quality of the predictions on the Fraction dataset, maybe because the fraction subtraction task is particularly easy to describe using KCs, so the KCs are enough to characterize the items. However, on Assistments, one can see the improvement when considering an item bias, see the difference between PFA and KTM (0.06 AUC). Same goes for Berkeley, where the KCs are actually mere categories of items.

6. CONCLUSION

Most models in the educational data mining literature have been presented as modeling both users and items (or skills) with multidimensional parameters. In this study though, we showed that while it is indeed important to model the dynamics of the students with multidimensional parameters, the items do not necessarily need to be multidimensional to come up with really strong models. In particular, KTM in its logistic regression form is among the top models. This is encouraging because we come up with better models that have fewer parameters.

As future work, we plan to try to use other kinds of side information. For example, for the Duolingo dataset [10], contestants had to predict whether a learner would get a word correct, and the best approaches were combinations of DKT with word embeddings [5]; using the content of the word (such as bigram features) allow to learn what phonemes are harder for which categories of people. Also we will try to encode richer actions for the encoder, because devising q-matrices is costly for humans, and the RNN may be able to recover the q-matrix without supervision.

7. REFERENCES

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

[2] L. T. DeCarlo. On the analysis of fraction subtraction data: The DINA model, classification, latent class sizes, and the Q-matrix. *Applied Psychological Measurement*, 2010.

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

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[6] S. Minn, Y. Yu, M. C. Desmarais, F. Zhu, and J.-J. Vie. Deep knowledge tracing and dynamic student classification for knowledge tracing. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1182–1187. IEEE, 2018.

[7] S. Montero, A. Arora, S. Kelly, B. Milne, and M. Mozer. Does deep knowledge tracing model interactions among skills? In *Proceedings of the Eleventh International Conference on Educational Data Mining. Educational Data Mining Society Press*, 2018.

[8] P. I. Pavlik, H. Cen, and K. R. Koedinger. Performance factors analysis–a new alternative to knowledge tracing. In *Proceedings of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*, pages 531–538. IOS Press, 2009.

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

[10] B. Settles, C. Brust, E. Gustafson, M. Hagiwara, and N. Madnani. Second language acquisition modeling. In *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 56–65, 2018.

[11] J.-J. Vie and H. Kashima. Knowledge Tracing Machines: Factorization Machines for Knowledge Tracing. Proceedings of the 33th AAAI Conference on Artificial Intelligence, 2019.

[12] P. J. Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[13] K. H. Wilson, Y. Karklin, B. Han, and C. Ekanadham. Back to the basics: Bayesian extensions of IRT outperform neural networks for proficiency estimation. In *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*, pages 539–544, 2016.