MLP Coursework 4: RNNs for knowledge tracing

s1758150 s1771906 s1784599

Abstract

Personalised online education has the potential to improve the education of millions. In order to personalise education it is important to model a student's knowledge over time and be able to predict how successful they will be at completing future tasks. This is known as "knowledge tracing".

Recent work suggests that recurrent neural networks (RNNs) are able to achieve state-of-the-art knowledge tracing results. Our first report reimplemented the RNN model but did not match state-of-the-art. In this report we describe problems with trying to replicate the level of performance in the literature. We detail many experiments with different hyperparameters and undocumented data representation we found in code associated with published results, and then data and code experiments used to confirm that the failure to replicate the performance reported in the literature must be due to a subtle implementation error.

1. Introduction

With millions of students using online education platforms and advances in machine learning, we may now have both the data and the techniques to model a student's knowledge, and guide each individual through a personalised curriculum. Personalised tutoring has been reported to improve student performance by as much as two standard deviation above the average student in a conventional class (Bloom, 1984).

Using the student's history of questions and performance on those questions, the task of "knowledge tracing" is to model a student's knowledge over time in order to predict how successful they will be at completing future tasks. (Corbett & Anderson, 1994; Piech et al., 2015). Models of knowledge acquisition can then be combined with reinforcement learning methods or models of how tasks increase knowledge and used to build an optimal curriculum (Piech et al., 2015; Rafferty et al., 2016; Reddy et al., 2017). Knowledge tracing is also of interest to cognitive scientists trying to understand the mechanisms of human learning (Lindsey et al., 2013; Patil et al., 2014)

The current state-of-the-art in knowledge tracing uses a recurrent neural network (RNN) trained on a sequence of (question, score) pairs to predict whether or not a student will get a question right given their history. This is known as "Deep Knowledge Tracing" (DKT) ((Piech et al., 2015; Khajah et al., 2016; Xiong et al., 2016). Our original research question was whether or not learning a richer representation of student knowledge acquisition, by training a latent variable RNN (Chung et al., 2015; Fraccaro et al., 2016) improves state-of-the-art performance on knowledge tracing.

In our previous report we described experiments intended to replicate the published DKT results as the baseline with which to compare novel methods. We reported preliminary results with 10% lower AUC (our primary metric) than the published state-of-the-art for our dataset [1]. Our plan for this report was to close this gap with a more exhaustive search of the hyperparameter space, and then to investigate latent variable RNN architectures.

However, our hyperparameter search did not yield improved results. We ran on the order of 100 experiments, systematically testing every available hypothesis we could think of, but could not close the AUC gap. Given that our AUC score of 65% is slightly worse than much simpler, traditional models such as Hidden Markov Models and Performance Factor Analysis (Gong et al., 2010), it did not seem promising to consider more advanced recurrent models such as an SRNN (Fraccaro et al., 2016). Rather, it seemed more useful to perform a thorough investigation of *why* the published results differed so greatly from ours.

At the outset of this report, we conjectured three possible sources of the AUC gap: (i) hyperparameter settings, (ii) dataset differences, and (iii) code implementation. Our primary research objective was to methodically document which components actually affect performance, and which do not. We believe this objective has value beyond simply closing the AUC gap: the field of knowledge tracing would benefit from an openly available sources of practical guidance on architectural choices, hyperparameter settings, preprocessing steps and datasets. Such guidance seems especially valuable in light of the rather different code implementations of deep knowledge tracing currently available, and previous data-quality (Xiong et al., 2016) and baseline implementation (Khajah et al., 2016) issues.

Ultimately, we did not close the 10% AUC gap. The large majority of our experiments supported the view that our code had no major flaws, since we could fit very well to the training data, and simply struggled to generalise to the validation set. Moreover, various stress-testing measures

[1]This is a deduplicated version of the Assistments 2009 dataset, called Assistment-(c) in (Xiong et al., 2016)

and sanity checks did not unearth a bug. Nevertheless, we ran the publicly available code given by (Xiong et al., 2016), and obtained their reported results of 75% AUC, so we are confident that it is an error in our code, and not a genuine replicability problem.

The report is structured as follows: we first introduce the dataset and task, including evaluation method (Section 2), and then describe the model and key methodological question to be investigated (Section 3). Our experimental results are given in Section 4, and cover hyperparameter searching, different forms of data pre-processing and alternative datasets. All results are negative. That is, we are unable to close the 10% AUC gap. Still, we think that, in trying to close the gap, we ended up producing a detailed comparison of the published empirical work in deep knowledge tracing, highlighting subtle differences in methodology that are hard to infer from the published material.

2. Data set and task

We test our knowledge tracing models on the Assistments 2009-2010 'skill-builder' public benchmark dataset.[2] Assistments is an online, automated tutor which assesses students in mathematics containing 222,332 answered problems from 3,104 different students on 146 types of problems. Unless stated otherwise, we use version c of the Assistments 2009 dataset as reported in Xiong et al. (2016), which has been carefully audited. This is important to mention, since the original Assistments 2009 data set, used by Piech et al. (2015), was corrupted, containing approximately 23% duplicate rows. Such data quality issues are especially important to mention in light of our replication problem. It is conceivable that incorrect or corrupted data could explain our poor results, and so we do an experimental comparison of our model over different data sets which have been used in the literature (Section 4.6).

We now briefly review the task setup as stated in coursework 3 and expand on the definition of our primary metric, AUC. Our data consists of sequences of (question, score) pairs $(\mathbf{x}_{1:n_i}, \mathbf{y}_{1:n_i})$ where n_i is the number of questions answered by student i, $\mathbf{x}_t$ is the identifier of a distinct type of question, such as 'trigonometry' and $\mathbf{y}_t$ is a binary label where 0 indicates incorrect and 1 indicates correct. At each time step t, our goal is to predict the label $\mathbf{y}_t$, given the question $\mathbf{x}_t$ and the student's current history of questions and answers $(\mathbf{x}_{1:t-1}, \mathbf{y}_{1:t-1})$. The loss function for this sequence of predictions can thus be written as:

$$L(\theta) = \sum_{i}^{S} \sum_{t}^{n_i} l(f(\mathbf{x}_t, \theta), \mathbf{y}_t)) \tag{1}$$

where S is the total number of students, l is the binary entropy loss between a prediction $f(\mathbf{x}_t, \theta)$ and target $\mathbf{y}_t$, and f is parameterised by a RNN with weights θ.

Our key evaluation metric is the Area Under the Curve (AUC), where 'the curve' refers to the Receiver Operator Characteristic (ROC) curve. AUC is defined as follows. For each question and answer pair $(\mathbf{x}_t, \mathbf{y}_t)$, our model outputs a probabilistic prediction: $0 \leq f(\mathbf{x}_t, \theta) \leq 1$. We then *threshold* this prediction: if it is greater than T, we predict 1 (correct), otherwise we predict 0 (incorrect). Different thresholds lead to different tradeoffs between the *true positive rate* and *false positive rate*. By true positive rate, we mean the chance that, given a randomly selected *correctly answered* question, our model outputs a prediction - call it f_1 - larger than T:

$$\mathbb{P}(f_1 > T) = \int_T^1 p_1(f)df \tag{2}$$

where p_1 is the probability density associated with f_1. Similarly, by false positive rate, we mean the chance that, given a randomly selected *incorrectly answered* question, our model outputs a prediction - call it f_0 - larger than T:

$$\mathbb{P}(f_0 > T) = \int_T^1 p_0(f)df \tag{3}$$

where p_0 is the probability density associated with f_0. AUC simply equals the probability that f_1 is greater than f_0 i.e the chance that our model would output a larger prediction to a randomly sampled positive case than negative case. Formally:

$$\int_0^1 \mathbb{P}(f_1 > T)p_0(T)dT = \mathbb{P}(f_1 > f_0) \tag{4}$$

There are good theoretical reasons for using AUC in contexts where classes are imbalanced (since the probability in 4 is not skewed by different ratios of positive to negative cases) or when the relative costs of misclassification between classes is uncertain (Hernández-Orallo et al., 2012).

3. Methodology

Our model, following Piech et al. (2015), is a recurrent neural network (Rumelhart et al., 1986), a type of neural network model that has achieved state of the art results in on a wide range of sequential classification tasks (Lipton et al., 2015). RNNs implement a form of memory by maintaining a hidden state, which is updated at each time step. This sequence of hidden states can then be unrolled to generate a directed acyclic computational graph that we can backpropagate through. This procedure lossily compresses long sequences into a single hidden state.

Formally, the output y_t and the hidden state h_t of a RNN at time t is computed using input data x_t and the state of the hidden state at time $t1$ as follows:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b_h) \tag{5}$$
$$y_t = \sigma(W_y h_t + b_y) \tag{6}$$

where the W, b terms are parameters with some random initialisation that are updated through backpropagation, and σ is the sigmoid function.

[2]https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data/skill-builder-data-2009-2010

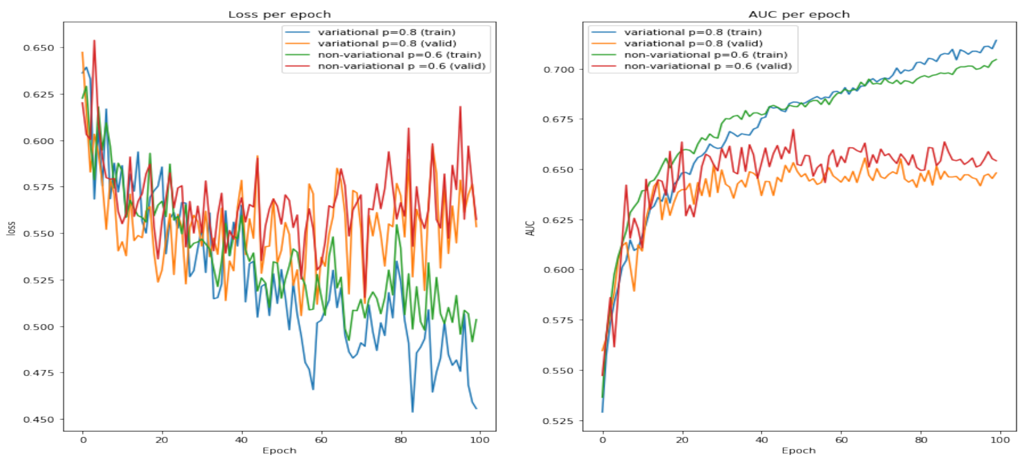


Figure 1. Loss and AUC for variational (p=0.8) versus non-variational (p=0.6) dropout

In particular we use a Long Short Term Memory architecture (LSTM), which has been found be better at learning longer-term dependencies (Hochreiter & Schmidhuber, 1997). The LSTM has a more complex computation for the hidden state, using "gate" functions $g : \mathbb{R}^n \rightarrow [0, 1]$ that remove or restore at a given time-step the information stored in the previous hidden state. There are three gates: the input gate g^{in} determines how much the input data will be combined with the hidden state, the forget gate g^{hid} adjusts the remaining value of the state, and the output gate g^{out} scales the output activation of the LSTM unit. Formally, each gate $g^\star$ is defined as

$$g_t^\star = \sigma(U^\star x_t + V^\star h_{t-1} + b^\star) \tag{7}$$

each with its own parameters $U^\star, V^\star, b^\star$, and the hidden state h_t and output y_t of the LSTM network is computed as:

$$s_t = g_t^{hid} s_{t-1} + g_t^{in} \sigma(W_x x_t + W_h h_{t-1} + b_h) \tag{8}$$

$$h_t = g_t^{out} \tanh(s_t) \tag{9}$$

$$y_t = \sigma(W_y h_t + b_y) \tag{10}$$

4. Experiments

4.1. Experimental setup

In this section, we describe our default hyperparameter settings, used in every experiment unless stated otherwise.

Our model consisted of an LSTM with 1 hidden layer and 200 hidden units. We minimise the loss function given in equation 1 using one of three optimisation methods, depending on the experiment. If using the Adam optimiser, we set the initial learning rate to 0.003, the first moment decay rate to 0.9, and second moment decay rate to 0.999 as recommended by Kingma & Ba (2014). If using stochastic gradient descent (SGD), we use an initial learning rate of 10, with a decay rate of 0.5 applied every 15 epochs. If using SGD with momentum (Polyak, 1964), we set the momentum to be 0.98. We note that our initial learning rate for SGD would be much too high for ordinary feed forward neural networks, but has empirical support for RNNs (Greff et al., 2017).

We prevent the well-known problem of 'exploding gradients' by gradient clipping with a threshold of 20. Training proceeded in mini batches of size 100, which corresponds to 100 students per batch. Initial experiments were run for 100 epochs, but due to the observation that convergence happens in 50 epochs, later experiments use this number. As explained in section 2, our primary evaluation metric is Area Under the Curve (AUC), hence we choose between models based on their AUC score on the validation set.

4.2. Hyperparameter experiments

Our preliminary experiments, detailed in the previous report, suffered from overfitting: the loss continued to decrease and AUC continued to increase beyond 75% on the training set but the validation metrics would plateau or start to deteriorate within 50 epochs, never achieving more than 67% AUC. Given the AUC scores >75% reported in the literature, our first set of experiments are an extensive search of hyperparameters. We investigate two alternative dropout methods, varying numbers of hidden units and three different optimisers.

Dropout

The research we tried to replicate (Xiong et al., 2016) used dropout on the output connections of the network, but not the recurrent connections. This is standard practice, since dropout on recurrent units is known to destabilise learning (Pham et al., 2014). However, results in our previous report showed that overfitting sets in at 50 epochs, using dropout probability p=0.6. We also provided preliminary results with *variational* dropout (Kingma et al., 2015; Gal & Ghahramani, 2016), which can be applied to recurrent connections. We saw some evidence of this solving overfitting and instead introducing underfitting. These baseline experiments clearly warranted a more thorough search over values of the dropout probability parameter. We grid-searched over {0.5, 0.6, 0.7, 0.8, 0.9 } for both variational and non-variational dropout.

In figure 1, we plot a comparison of normal and variational dropout for p=0.6 and p=0.8 (respectively), which were the optimal values obtained after grid-searching. We see that non-variational dropout is slightly superior, peaking at around 50 epochs and obtaining a validation AUC of 66%, which is likely not different from our baseline in a statistically significant sense. After 50 epochs, both methods converge, and afterwards their AUC validation curves very gently decrease (and the corresponding loss curves gently increase) whilst AUC and loss on the train set steadily improves, implying mild overfitting.

Neither method of dropout, for any value of the probability parameter that we searched, could stop the AUC validation curve from peaking at 50 epochs. Whilst we initially assumed this peak was the outcome of overfitting, and that a more strongly regularized model would continue to increase it's validation AUC past 50 epochs, the experimental results suggest otherwise. The results, taken by themselves, seem to imply a fundamental limit on generalization error for this combination of dataset and model. However, given that there are multiple published results with higher AUC scores than those obtained here, such a strong conclusion seems rash; we expect that some other hyper-parameter setting or flaw in our code would explain our poor results.f

Number of hidden units

One simpler approach to solving overfitting, that might result in less underfitting than variational dropout, is to just use fewer hidden units combined with early stopping. Whilst a large number of hidden units combined with dropout tends to be the most effective approach for deep fully-connected neural networks, RNNs are notoriously difficult to regularise (Zaremba et al., 2014; Gal & Ghahramani, 2016) so we wanted to try a simple approach as a sanity check. In particular, we present the results of varying the number of hidden over {100, 125, 150}.

In figure 2 we see that overfitting still occurs after somewhere in the region of 20 to 30 epochs. The blue curve, representing 100 hidden units, overfits slightly less, as can be seen from the AUC decreasing slightly less quickly on the validation set after 30 epochs, but the difference between it and 150 hidden units is marginal. If we apply an early stopping rule, all numbers of hidden units would perform similarly, attaining around 64-65% AUC on the validation set, which is slightly worse than the score we obtain using 200 hidden units, dropout, and early stopping in the last experiment.

Optimiser

One major difference between the code of Xiong et al. (2016) and the code of Piech et al. (2015), is that the former uses the Adam optimiser with learning rate 0.1, whilst the latter uses vanilla stochastic gradient descent with an unusual learning rate schedule, initialised at 30, and decaying by a factor of approximately a third every 12 epochs, with a minimum rate of 1. Our baseline model followed Xiong et al. (2016) in using Adam, but used a much smaller initial learning rate of 0.003, since we found that setting the learning rate to 0.1 destabilised learning. The fact that we could not use exactly the same initial learning rate as Xiong et al. (2016) was a source of concern, and so we wanted to investigate if using the SGD with rapid learning rate decay as used by Piech et al. (2015) would increase our model's ability to learn good parameters, boosting our validation AUC. We also tested SGD with momentum, to acquire more insight into how sensitive training is to choice of optimiser.

As can be seen from figure 3, no optimization method allows us to increase on our validation AUC ceiling of 65-66%. Whilst SGD with momentum appears to be the best choice, followed by SGD and then Adam, we note that the Adam optimizer produced results comparable to SGD in previous experiments, and so it would seem that random fluctuations account for a lot of the variation, rendering the differences we observe uninteresting.

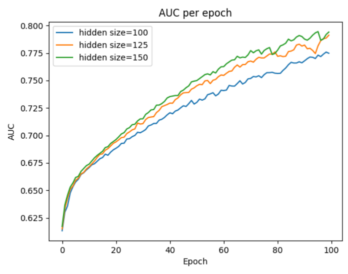
This experiment concludes a series of negative results regarding the question of whether hyperparameter tuning could solve the gap between our AUC score of 65% and the published score of 75%. This alarming gap prompted us to investigate more serious differences in methodology, such as data preprocessing.

4.3. Data preprocessing experiments

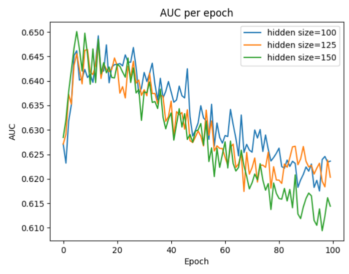
Our data preprocessing steps were informed by the written description of Xiong et al. (2016), along with their publicly available code. Given that Piech et al. (2015) and Khajah et al. (2016) have also published code, we cross-validated the three implementations to ensure consistency. The key differences between the implementations are that (i) Khajah et al. (2016) uses *sequence truncating*, which means breaking up long sequences of answers into shorter ones ii) Piech et al. (2015) uses *semi-sorted mini-batching*, meaning that the students are sorted by number of questions answered before being allocated to batches, which are then shuffled (iii) Piech et al. (2015) and Khajah et al. (2016) use a different version of the Assistments data set, containing 23% duplicate rows according to Xiong et al. (2016), as we discussed in section 2.

4.4. Sequence truncation experiment

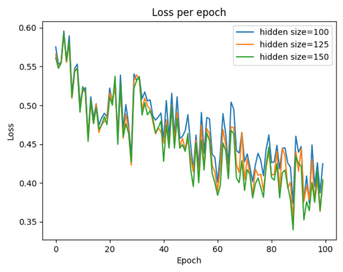
We conjecture that breaking up long sequences might assist learning in our model, since backpropagating gradients over very long sequences can cause them to vanish or explode. Specifically, we experimented with a truncation value of:



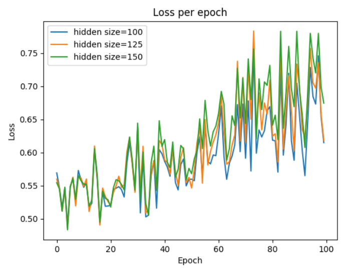
(a) Train AUC



(b) Validation AUC



(c) Train Loss



(d) Validation Loss

Figure 2. Different number of hidden units in LSTM

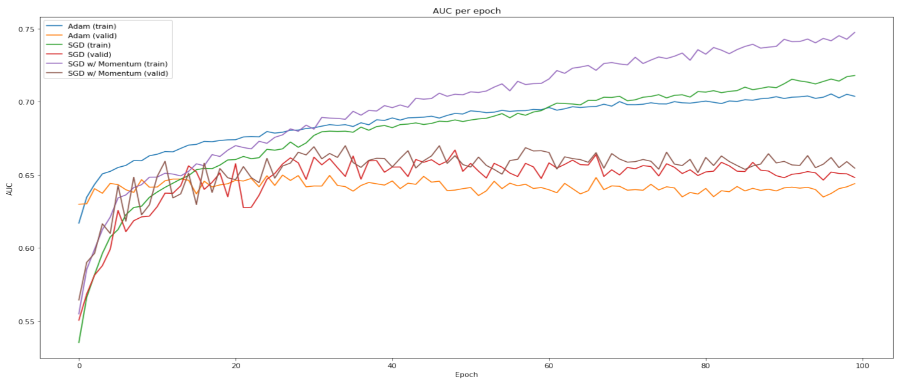


Figure 3. AUC curves on training and validation set for different optimization methods

2, 50, 100. We anticipated that the smallest value of 2 would yield poor results, since it closely approximates a non-sequential model that tries to map input question to an answer without capturing dependencies in a student's learning trajectory. However, a model incapacitated in this way actually serves as a useful baseline to compare the other models to, giving us a measure of the utility of sequential information. If the models using longer sequences are not much better, that strongly suggests a bug in our code.

As we can see from figure 4, sequence truncating appears to have an effect on learning. Truncating to a length of 2 stifles learning, resulting in an validation AUC of just 58%, only 8% higher than random guessing. Whereas, truncating sequence length at 100 appears to enhance learning, allowing our model to obtain a validation AUC of 67% when applying early stopping, which is slightly better than the non-truncated models used in previous experiments. These results, whilst not resolving our replication problem, display the kinds of trends one expects from a correct implementation. In particular AUC validation decreases when we break the data into very short sequences, whilst using 100 length sequences obtains performance that is at least as good as non-truncated models, and possibly slightly better.

4.5. Semi-sorted batches experiment

Pre-sorting students before allocating them to batches was not mentioned in the published work of Piech et al. (2015), but was used in their code implementation. It is thus difficult to know without running an experiment whether or not this technique has a significant impact on learning. We conjecture that it could have an effect for the following reason. The length of student's sequences follow a heavy-tailed distribution: a few "productive" students have answered close 1000 questions, whilst the majority have answered fewer than 100. Thus, if we mix a productive student into the same batch as less productive students, the back propagated errors from their sequences will dominate the gradient update for that batch. This is not necessarily a problem: after all, we ultimately care about correctly predicting as many individual questions as possible; we are not optimising an equally weighted average over students. But, it seems plausible that if the few productive students are highly atypical, then good parameter settings for these students are not good parameter settings for the less productive students. Hence, whenever, a batch contains a productive student, we would see large parameter updates towards values that are bad for the majority of students, hurting performance overall, and making learning less stable.

This experiment returned a negative result, producing learning curves that were indistinguishable from non-sorted models. Given that these graphs did not differ from those already presented, we conserve space by not including them here. Despite not increasing our AUC score, it is beneficial for future research to know that semi-sorting has little added-value and thus is unlikely to explain any differences in the results obtained in the literature.

4.6. Different Data sets experiment

As described in section 4.1, all experiments above use the raw data from the github of Xiong et al. (2016). To check that our replication problem is independent of dataset, we also downloaded a version of the Assistments dataset used by Piech et al. (2015) (which is known to contain duplicates), and a totally different synthetic dataset, also used by Piech et al. (2015), which is designed to be simple, with all students answering 50 questions.[3]

The best reported AUC on the duplicated Assistments dataset is 86%, which is 11% higher than the best reported non-duplicated score. We therefore expect, if our model is working correctly, to increase upon our best non-duplicated score of 67% when training on the duplicated data. We also expect a correctly implemented model to get close to the highest reported AUC of 75% on the synthetic dataset, since it is intended to be a simple version of the knowledge tracing problem. Failure to get a good score on the synthetic dataset would suggest our model has a bug.

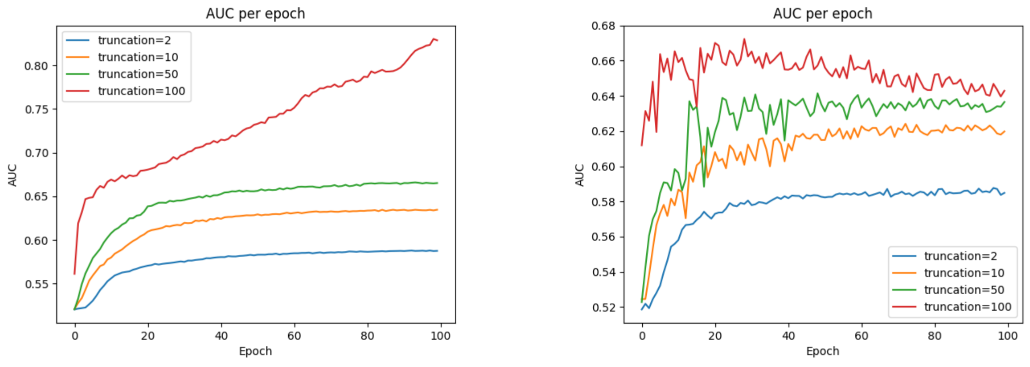
As can be seen in figure 5, training on the duplicated data only slightly improved results, to 69% AUC, which is 17% lower than the best published result for this dataset. The fact that our score increased is a positive sign, but the magnitude of increase is disconcertingly small. Even more disconcerting is the very low validation AUC score on the synthetic dataset, which plateaus at 62%. This low validation score is particularly surprising in light of the high training score, which plateaus at 78%, which is in line with published results.

These results, particularly those for the synthetic dataset, provide a very compelling case for the view that our implementation is incorrect, and likely contains a bug involving either the processing of the validation set, or the computation of AUC for the validation set. We consider such implementation issues in the next section.

4.7. Implementation checks

For completeness we now describe some of the checks we have carried out to ensure our code implementation is correct. Firstly, pre-processing of the data is done before the training-validation set split, and the training set is a different random sample for each experiment, therefore the model's ability to easily fit the training data suggests that data preprocessing is correct. We check this further using the fact that the target at time-step t should become the input at time-step $t + 1$. By translating the targets right one time-step, we achieve a very high AUC score, again as one would expect if the original processing was correct and the target is now included in the input data. Finally, we computed AUC using two different software libraries, since the method of approximation used to compute it can vary. We found that `tf.metrics` produces identical answers to `sklearn.metrics`.

[3]We thank Chris Piech, lead author of Piech et al. (2015), who in correspondence pointed us to the synthetic dataset.



(a) Train AUC (b) Validation AUC

Figure 4. AUC curves for different truncation lengths

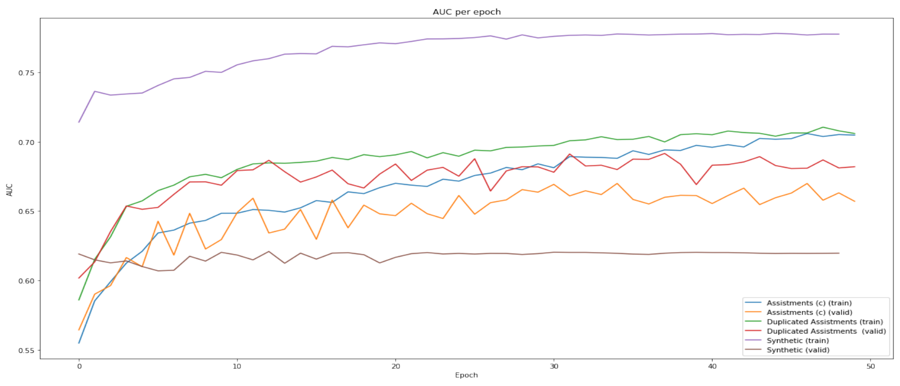


Figure 5. AUC curves on training and validation set for different data sets

5. Related work

Our work is based on replicating the three papers: Piech et al. (2015); Khajah et al. (2016); Xiong et al. (2016). These are the papers we contrast our results to in our conclusion. Each use the LSTM model described in Section 3 and the Assistments data described in Section 2, but with different preprocessing steps which were described in Section 4.3.

Knowledge tracing more generally goes back to at least the introduction of Bayesian Knowledge Tracing (BKT) (Corbett & Anderson, 1994). BKT uses a Hidden Markov Model, with a Bernoulli hidden variable representing a student's knowledge of each skill, and binary observed variables representing a student's performance (correct or incorrect) answering a question involving knowing that skill. The model has four parameters per skill: the initial probability of knowing a skill before answering a question, the probability of transitioning from knowing to not knowing a skill after each question, and the probabilities that the student get the question wrong (right) given they do (or do not) know the skill. Many variants to this basic BKT model have been suggested, for example having student-specific initial probabilities of knowledge (Pardos & Heffernan, 2010), an additional parameter for whether a skill was learned at the last question (Baker et al., 2011), contextual estimation of the probability that a student performs differently to their knowledge state (d Baker et al., 2008), and of the difficulty of each skill (Pardos & Heffernan, 2011). BKT models have traditionally been favoured because their parameters are more interpretable than DKT methods (Baker & Inventado, 2014).

In the context of personalised online education, knowledge tracing models have been used to train reinforcement learning models to generate curricula, with the set of actions being the set of possible questions and the rewards based on improvements to the knowledge of the "student" as represented by the knowledge tracing model (Rafferty et al., 2016; Reddy et al., 2017). In our previous report we also proposed the untested idea that by reversing the order of sequences used to train knowledge tracing models, we could specify the knowledge state we wish a student to get to and run the model forward to generate plausible sequences of how a student gets there.

6. Conclusion

Ultimately, we did not close the AUC gap between our experiments and the result reported in the literature, and suspect this is due to a subtle implementation error, where previously evidence suggested only further hyperparameter search was needed. Crucially, the results above show that the training data is easily fit, validation metrics improve when expected, such as when trained on longer sequence lengths and data known to contain duplicates, but that the validation AUC does not achieve the levels of previously published results on even the synthetic data.[4] This suggests that both the data processing and the model are implemented correctly, but that there is some implementation error - possibly in how the validation metrics are handled, even though the implementation of AUC itself is correct - which we have struggled to identify, and not a problem with previously published results. However, we do think that this report demonstrates due diligence in attempting to replicate results, and so for the benefit of the research community we would like to see more thorough reporting of methodology and hyperparameter settings in published papers.

We identified methodological issues not discussed in the literature, and have documented them here to help future work on DKT. In particular, it is important how raw data is processed into a form that can be passed to a RNN, and why such a representation is chosen, and our previous report is the first to give details of why and how data is preprocessed. as we describe in 4, we also identified in published code, not reported in papers, that past work has used different optimisers and hyperparameters, and perhaps more significantly semi-sorted the data (with no apparent effect) and truncated sequence lengths (with some evidence of it effecting performance).

Finally, the number of different experiments run across hyperparameters, preprocessing, datasets and implementations demonstrates the challenges of identifying for what reason a model is not performing well. In our experience, writing multiple implementations of some components, having multiple different datasets, and corresponding with the authors of published work has been key to us concluding the failure to replicate is due to an error in implementation, and we suggest prioritising these aspects when trying to replicate work.

References

Baker, Ryan Shaun and Inventado, Paul Salvador. Educational data mining and learning analytics. In *Learning analytics*, pp. 61–75. Springer, 2014.

Baker, Ryan SJD, Goldstein, Adam B, and Heffernan, Neil T. Detecting learning moment-by-moment. *International Journal of Artificial Intelligence in Education*, 21 (1-2):5–25, 2011.

Bloom, Benjamin S. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher*, 13(6):4–16, 1984.

Chung, Junyoung, Kastner, Kyle, Dinh, Laurent, Goel, Kratarth, Courville, Aaron C, and Bengio, Yoshua. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pp. 2980–2988, 2015.

[4]Obviously since we did not identify a satisfactory model and may want to work on this project further, we have kept from using the test set.

Corbett, Albert T and Anderson, John R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

d Baker, Ryan SJ, Corbett, Albert T, and Aleven, Vincent. More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In *International Conference on Intelligent Tutoring Systems*, pp. 406–415. Springer, 2008.

Fraccaro, Marco, Sønderby, Søren Kaae, Paquet, Ulrich, and Winther, Ole. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pp. 2199–2207, 2016.

Gal, Yarin and Ghahramani, Zoubin. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pp. 1019–1027, 2016.

Gong, Yue, Beck, Joseph E, and Heffernan, Neil T. Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In *International conference on intelligent tutoring systems*, pp. 35–44. Springer, 2010.

Greff, Klaus, Srivastava, Rupesh K, Koutník, Jan, Steunebrink, Bas R, and Schmidhuber, Jürgen. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

Hernández-Orallo, José, Flach, Peter, and Ferri, Cèsar. A unified view of performance metrics: translating threshold choice into expected classification loss. *Journal of Machine Learning Research*, 13(Oct):2813–2869, 2012.

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

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

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, Diederik P, Salimans, Tim, and Welling, Max. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Lindsey, Robert V, Mozer, Michael C, Huggins, William J, and Pashler, Harold. Optimizing instructional policies. In *Advances in Neural Information Processing Systems*, pp. 2778–2786, 2013.

Lipton, Zachary C, Berkowitz, John, and Elkan, Charles. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

Pardos, Zachary A and Heffernan, Neil T. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *International Conference on User Modeling, Adaptation, and Personalization*, pp. 255–266. Springer, 2010.

Pardos, Zachary A and Heffernan, Neil T. Kt-idem: introducing item difficulty to the knowledge tracing model. In *International Conference on User Modeling, Adaptation, and Personalization*, pp. 243–254. Springer, 2011.

Patil, Kaustubh R, Zhu, Xiaojin, Kopeć, Łukasz, and Love, Bradley C. Optimal teaching for limited-capacity human learners. In *Advances in neural information processing systems*, pp. 2465–2473, 2014.

Pham, Vu, Bluche, Théodore, Kermorvant, Christopher, and Louradour, Jérôme. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pp. 285–290. IEEE, 2014.

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

Polyak, Boris T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Rafferty, Anna N, Brunskill, Emma, Griffiths, Thomas L, and Shafto, Patrick. Faster teaching via pomdp planning. *Cognitive science*, 40(6):1290–1332, 2016.

Reddy, Siddharth, Levine, Sergey, and Dragan, Anca. Accelerating human learning with deep reinforcement learning. 2017.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

Xiong, Xiaolu, Zhao, Siyuan, Van Inwegen, Eric, and Beck, Joseph. Going deeper with deep knowledge tracing. In *EDM*, pp. 545–550, 2016.

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.