Learning to Teach Using Reinforcement Learning

Michael Raffelsberger 11772903, Susanna Weinberger 11718215

University of Vienna, Vienna

Abstract. **Keywords:** Machine Learning · Reinforcement Learning · Personalized Education · Knowledge Tracing

1 Introduction

In this project, we deal with the topic of personalized education, i.e. ensuring that students acquire knowledge about some subject(s) with an individualized sequence of exercises for learning. The project contains two major parts: One part is all about fitting knowledge tracing models that take into account how hard certain skills are to learn and to apply and can maintain an estimate of the current knowledge of a student during an exercise sequence. The other part is about finding reinforcement learning agents that learn to select exercises to make the knowledge acquisition process as efficient as possible. Last but not least, we deal with the fairness aspect that arises when students with different speed of learning are to be taught.

2 Notation

- $\boldsymbol{S}$: set of skills.
- $s \in \boldsymbol{S}$: a single (atomic) skill.
- $\boldsymbol{E}$: set of exercises.
- $e \in \boldsymbol{E}$: a single exercise involving one or more skills from the skill set.
- $e_t \in \boldsymbol{E}$: the exercise worked on by some student at timestep t.
- $\boldsymbol{S}_e \subseteq \boldsymbol{S}$: the skills involved by exercise e. Note that technically speaking we just distinguish between exercise types, which means that we do not distinguish between exercises involving the exact same skills. Therefore, $\boldsymbol{S}_{e1} = \boldsymbol{S}_{e2} \Rightarrow e1 = e2$.
- $B_i \subseteq \boldsymbol{S}$: a block of associated skills.
- $\boldsymbol{B}$: the set of blocks $\{B_1, \ldots, B_{|\boldsymbol{B}|}\}$.
- $\boldsymbol{E}_i \subseteq \boldsymbol{E}$: the exercise set spanned by block B_i.
- $L_t^{(s)} \in \{0, 1\}$: the learning state of skill s at timestep t. The superscript indicating the skill is often omitted.
- $C_t \in \{0, 1\}$: indicating whether exercise e_t was answered correctly or not.
- $C_t^{(s)}$: if available, indicating whether a student performed a correct action for skill s at timestep t.
- $P(L_0), P(T), P(G), P(S)$: parameter names in the BKT model explained in the respective sections.

- $T \in \mathbb{N}$: the length of some sequence with time indices $(0, \ldots, T-1)$ with running index t.
- $i, j \in \mathbb{N}$: running indices whenever required.

3 Knowledge Tracing Approaches in Literature

Before introducing existing approaches for our problem, we first formulate our setting: In general, we assume to have a set $\boldsymbol{S}$ of (atomic) skills that a student is supposed to learn. A student learns by doing a sequence of exercises $(e_t)_{t=0}^{T-1} = (e_0, \ldots, e_{T-1})$ from set $\boldsymbol{E}$, where each exercise $e \in \boldsymbol{E}$ involves a subset $\boldsymbol{S}_e \subseteq \boldsymbol{S}$ of the skill set. As soon as the student finishes exercise e_t, we can evaluate $C_t \in \{0, 1\}$ indicating whether exercise e_t was answered incorrectly ($C_t = 0$) or correctly ($C_t = 1$).

3.1 Bayesian Knowledge Tracing

Bayesian knowledge tracing (BKT) [4] is an inference scheme for monitoring the knowledge acquisition of students when solving exercises, originally used in a programming tutor environment. Estimating the current knowledge of a student makes it possible to individualize the exercise sequence chosen by the tutor.
In BKT, we assume that $\boldsymbol{S}$ is a set of independent skills[1]. By assumption, a skill $s \in \boldsymbol{S}$ can either be in the unlearned state ($L^{(s)} = 0$) or in the learned state ($L^{(s)} = 1$). To trace the student's knowledge, we maintain a probabilistic estimate $P(L^{(s)} = 1)$ for all $s \in \boldsymbol{S}$ of whether the student can already handle skill s. For every skill $s \in \boldsymbol{S}$, there are four empirically estimated parameters. For the remainder of the subsection, we just focus on t in some subsequence $(t_0^{(s)}, \ldots, t_\tau^{(s)})$ of $(0, \ldots, T-1)$ where some skill s is involved. C_t in $(C_{t_0}^{(s)}, \ldots, C_{t_\tau}^{(s)})$ denotes whether the student performed a correct action for that skill at the corresponding timestep. For the sake of simplicity, we also omit superscripts indicating the skill:

- $P(L_0)$: The prior probability $P(L_0 = 1)$ that the skill is already learned at timestep $t = 0$, i.e. at the beginning of the learning sequence.
- $P(T)$: The transition probability $P(L_{t+1} = 1 | L_t = 0)$ that the student learns the skill when applying it in some exercise. Note that transitions are assumed to be independent of whether the skill was applied correctly. Forgetting an already learned skill is impossible, i.e. $P(L_{t+1} = 0 | L_t = 1) = 0$.
- $P(G)$: The guess probability $P(C_t = 1 | L_t = 0)$ to apply a skill correctly even though it is still in the unlearned state.
- $P(S)$: The slip probability $P(C_t = 0 | L_t = 1)$ to apply a skill incorrectly even though it is already in the learned state.

Given estimates for the four parameters, the tutor can update the probabilistic estimate $P(L_t = 1)$ for the skill at every interaction by applying the following internal transition logic:

[1] In [4], the skills are called production rules.

$$P(L_{t+1} = 1) = P(L_t = 1|C_t) + (1 - P(L_t = 1|C_t)) \cdot P(T), \tag{1}$$

$$P(L_t = 1|C_t) = \frac{P(C_t|L_t = 1) \cdot P(L_t = 1)}{P(C_t)}, \tag{2}$$

$$P(C_t) = P(C_t|L_t = 1) \cdot P(L_t = 1) + P(C_t|L_t = 0) \cdot P(L_t = 0), \tag{3}$$

$$P(L_t = 0) = 1 - P(L_t = 1). \tag{4}$$

On top of that, we can formulate the emission logic determining the probability that the student will apply the given skill correctly at timestep t:

$$P(C_t = 1) = P(L_t = 1) \cdot (1 - P(S)) + P(L_t = 0) \cdot P(G). \tag{5}$$

Figure 1 illustrates the dynamics of the BKT model for a single skill.

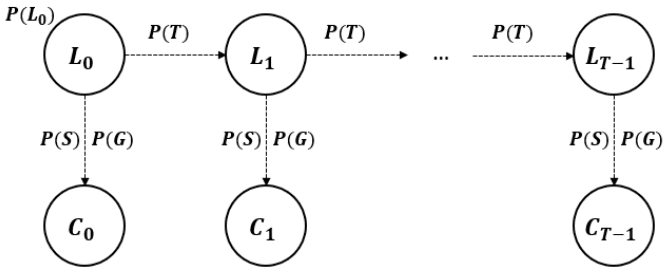


Fig. 1: Graphical representation of the transition and emission logic of the BKT model for a single skill.

If we are able to estimate the model parameters for every skill, we can also predict the probability of a correct outcome for a composed exercise e_t with $|\boldsymbol{S}_{e_t}| > 1$:

$$P(C_t = 1) = \prod_{s \in \boldsymbol{S}_{e_t}} P(C_t^{(s)} = 1). \tag{6}$$

Since the skills are assumed to be independent, it is just the product of the individual probabilities. The above formula assumes that a single skill s can only appear once per exercise. If a skill is involved multiple times, the formula has to be adjusted accordingly.

3.2 Deep Knowledge Tracing

While the BKT model basically fits a separate model for every skill and treats the skills independently, deep knowledge tracing (DKT) [9] uses an "all-in-one" neural network approach. In particular, an RNN or LSTM replaces the less flexible

Markov model from before without imposing as many assumptions. Furthermore, the model operates on the exercise level: The input sequence of the recurrent network is some encoding of the sequence $x_0, \ldots, x_{T-1}$ where $x_t = (e_t, C_t)$. At timestep t, the model outputs an estimation $P(C_{t+1} = 1 | e_{t+1} = e)$ for each possible exercise $e \in \boldsymbol{E}$. While this approach is harder to interpret than the BKT model, it is more flexible when it comes to predicting a student's performance.

4 Knowledge Tracing Implementation

4.1 Simple BKT

Our baseline knowledge tracing implementation is a simple BKT model that estimates the BKT parameters for one single skill from observed sequences. We use the forward algorithm for hidden Markov models (HMM) to compute the log-likelihood of an observed sequence $(C_0, \ldots, C_{T-1})$ for some student. The model is implemented in the PyTorch framework [8] and therefore uses automatic differentiation to compute gradients of the negative log-likelihood with respect to the model parameters on a batch of sequences. The parameters are optimized using the Adam optimizer. Unfortunately, this model is very simplistic. In particular, if we have an exercise e_t involving multiple skills at once, i.e. $|\boldsymbol{S}_{e_t}| > 1$ but just one label C_t indicating whether the whole exercise was solved correctly or not, we have to pretend that C_t also acts as label $C_t^{(s)}$ for every single skill s in $\boldsymbol{S}_{e_t}$. However, it is not obvious at all to what extent we can attribute the outcome of C_t to the individual skills involved at timestep t. To overcome this shortcoming of interactions between skills, we came up with a similar yet more flexible model, which we call block BKT model.

4.2 Block BKT

The block BKT model is an extension of the simple BKT model to approach settings where exercises can involve multiple skills at the same time. To ensure feasibility, we assume a block structure among the skills:

$$\boldsymbol{S} = \bigcup_{i=1}^{|\boldsymbol{B}|} B_i \quad \text{with } B_i \cap B_j = \emptyset \text{ if } i \neq j, \tag{7}$$

where B_i is a block of co-occurring skills and $\boldsymbol{B}$ is the set of blocks. The PyTorch implementation of the block BKT model finds estimates for the parameters for one single block B_i. There is still an individual prior $P(L_0)$ as well as an individual transition probability $P(T)$ for every skill $s \in B_i$. As before, a transition for a skill can occur if that skill is involved in the exercise of the current timestep. Transitions also happen independently for the different skills: If an exercise involves two skills, e.g. a and b, then:

- $(1 - P(T^{(a)})) \cdot (1 - P(T^{(b)}))$ is the probability that no transition happens at all.

- $P(T^{(a)}) \cdot (1 - P(T^{(b)})$ is the probability that only skill a gets learned but not skill b.
- $(1 - P(T^{(a)})) \cdot P(T^{(b)})$ is the probability that only skill b gets learned but not skill a.
- $P(T^{(a)}) \cdot P(T^{(b)})$ is the probability that both skills get learned.

However, the slip probability $P(S)$ and the guess probability $P(G)$ are shared parameters of the block:

$$P(C_t = 1) = (1 - P(S)) \cdot \prod_{s \in \boldsymbol{S}_{e_t}} L_t + P(G) \cdot \left(1 - \prod_{s \in \boldsymbol{S}_{e_t}} L_t\right), \tag{8}$$

which means that a student is capable of solving an exercise correctly (up to slipping) if all relevant skills are in the learned state. Otherwise, the student has to guess the correct answer. Apart from the outcome sequence $(C_0, \ldots, C_{T-1})$ and the exercise sequence $(e_0, \ldots, e_{T-1})$ the model also takes a *max_skills* parameter as input. It determines the maximum number of skills per exercise. Therefore, *max_skills*= 2 implies that no exercise can involve more than 2 skills.

Example Assume we want to fit the block BKT model to a block B_i with $|B_i| = 3$ skills and *max_skills*= 2. An example input for a single student could be

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

for the outcome sequence and

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

as encoding of the exercise sequence, meaning that in the first exercise only the third skill was exercised, in the second exercise the second and the third skill were exercised, etc. Note that padding multiple sequences of a batch to the same length must happen with a valid exercise, e.g. $\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$. In particular, $\begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$ is not a valid exercise and would lead to an error. The corresponding *BlockBKTCollate* class takes care of this potential danger.
In our example, the model effectively consists of 8 parameters:

- $P(L_0) = \left(P(L_0^{(1)})\ P(L_0^{(2)})\ P(L_0^{(3)})\right)^T$,
- $P(T) = \left(P(T^{(1)})\ P(T^{(2)})\ P(T^{(3)})\right)^T$,
- $P(S)$,
- $P(G)$.

Technically, there are twice as many unnormalized parameters in the PyTorch module that can be turned into probabilites by applying the softmax function.

With three skills, the hidden Markov model, which is still the theoretical foundation of our implementation, has $2^3 = 8$ possible states:

$$\begin{pmatrix} 0\,0\,0 \\ 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \\ 1\,1\,0 \\ 1\,0\,1 \\ 0\,1\,1 \\ 1\,1\,1 \end{pmatrix},$$

where the first row means that all skills are in the unlearned state and the last row means that all skills are in the learned state. Since *max_skills*= 2, the exercise set $\boldsymbol{E}_i$ of block B_i can be encoded by

$$\begin{pmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \\ 1\,1\,0 \\ 1\,0\,1 \\ 0\,1\,1 \end{pmatrix}.$$

$\big(0\ 0\ 0\big)$ is not a valid exercise and $\big(1\ 1\ 1\big)$ involves too many skills. Note that the forward algorithm in the block BKT model is significantly more complicated to implement as compared to the simple BKT model. We construct an $|B_i| \times |B_i| \times |\boldsymbol{E}_i|$ transition tensor from the parameters in every forward step, where $|\boldsymbol{E}_i|$ is the number of distinct exercises in our block B_i. *transition_tensor[:,:,3]*, i.e. the $|B_i| \times |B_i|$ logarithmic transition matrix corresponding to exercise $\big(1\ 1\ 0\big)$, is displayed in the Appendix as example.

Further Details Since we have learning sequences of variable length with sometimes considerable length differences in our data, we use the log-likelihood divided by the sequence length in our optimization (option *weighted=True*), i.e. $l((C,e)_{t=0}^{T-1})/T$. The resulting log-likelihood per timestep gives comparably much influence to shorter sequences though.
When testing the model, we also experienced that it tends to overestimate the priors $P(L_0)$. Therefore, we add regularization terms, controlled by the hyperparameter δ, to the loss function to prevent the priors from being to large. The option to regularize the transition probabilities, controlled by o, is also implemented.

5 Reinforcement Learning

After fitting a BKT model to our data, the goal is now to actually find a good learning strategy with reinforcement learning that performs better than baseline

functions like choosing randomly or greedily what exercises a student should get. We therefore implemented different environments based on Open AI gym [3] environments and then used stable baselines 2 [7] as learning agent. For real-life data, it is obviously only possible to evaluate with the fitted BKT parameters. Using simulated data offers the possibility of evaluating with the actual parameters that were used for generating the initial data.

5.1 Environments & Algorithms

To compare the influence of different rewards and how much information an agent gets, we implemented 6 different environments. All environments are based on the assumption, that students behave according to our assumed BKT data structure. As action, the agent decides which exercise (by providing the number of the exercise) is given to the student/environment. The amount of exercises every student gets is fixed.

1. **Environment 1**: This environment returns as observation the current learning state. The reward depends on the situation, and the values for all three scenarios can be passed to the environment. The default situation is as follows:
   (a) reward = 10 if a new skill is learned in this step.
   (b) reward = 1 if the exercise includes unlearned skills or if all skills are already learned.
   (c) reward = -1 if the exercise only includes already learned skills and not all skills are learned yet.
2. **Environment 2**: This environment is almost equivalent to Environment 1, the only difference being that the reward is now $\frac{\text{number of learned skills}}{\text{total number of skills}}$.
3. **Environment 3**: Again this environment is based on Environment 1, but now the reward is only given to the agent at the end by the number of learned skills. This represents a more realistic scenario, where the learning states of the skills are evaluated through a final exam at the end. Nevertheless, getting the actual learning states is still more informative than real exam results would be.
4. **Environment 4**: As the true learning states of students are typically not available, it makes sense to also look at different scenarios. In Environment 4, we return how many exercises were solved correctly until now as observation. The reward is defined as the sum of the probabilistic learning states. Those are calculated according to our data structure and the already discussed formulas in 3.1. Based on whether an exercise was solved correctly or not for each skill s involved in the exercise, we calculate its new learning state $L_{t+1}^{(s)}$ at step $t+1$ by

$$P(L_{t+1}^{(s)} = 1) = P(L_t^{(s)} = 1|C_t) + (1 - P(L_t^{(s)} = 1|C_t)) \cdot P(T^{(s)}).$$

   For calculating $P(L_t^{(s)} = 1|C_t)$, Bayes' theorem is used:

$$P(L_t^{(s)} = 1|C_t) = \frac{P(C_t|L_t^{(s)} = 1) \cdot P(L_t^{(s)} = 1)}{P(C_t|L_t^{(s)} = 1) \cdot P(L_t^{(s)} = 1) + P(C_t|L_t^{(s)} = 0) \cdot P(L_t^{(s)} = 0)}.$$

This formula can then be further processed depending the whether the exercise was solved correctly or not. $P(C_t|L_t^{(s)} = 0)$ corresponds to $P(G)$ if $C_t = 1$, else $1 - P(G)$. Furthermore, in case of a correctly solved exercise, i.e. $C_t = 1$, it holds

$$P(C_t|L_t^{(s)} = 1) = (1-P(S)) \cdot \prod_{i \in \boldsymbol{S}_{e_t} \setminus s} P(L_t^{(i)}) + P(G) \cdot \left(1 - \prod_{i \in \boldsymbol{S}_{e_t} \setminus s} P(L_t^{(i)})\right),$$

and in case of a wrong exercise ($C_t = 0$), it holds

$$P(C_t|L_t^{(s)} = 1) = P(S) \cdot \prod_{i \in \boldsymbol{S}_{e_t} \setminus s} P(L_t^{(i)}) + (1-P(G)) \cdot \left(1 - \prod_{i \in \boldsymbol{S}_{e_t} \setminus s} P(L_{n-1}^{(i)})\right),$$

where $\boldsymbol{S}_{e_t} \setminus s$ is the set of all skills queried in the current exercise e_t excluding the skill s we are looking at right now.

5. **Environment 5**: This environment is similar to Environment 4, the difference being that the observation consists of the probabilistic learning states in order to provide a bit more information to the agent.
6. **Environment 6**: This environment uses the same observation as in environment 4. The reward is equivalent to the one used in environment 1.

Unfortunately, there was not enough time to implement own algorithms for our agents, therefore we used stable baselines 2 [7], namely A2C, DQN and PPO2. Although we first experimented with stable baselines 3 [10], we then decided to use the predecessor since during the time of our implementation, there were no recurring policies available for stable baselines 3.

5.2 Baselines

In order to compare the achieved results, we implemented 3 different baseline agents.

1. **Random**: In each step, a random action (=exercise type) is chosen.
2. **Single Greedy**: In each step, the agent looks at the "lowest skill" (concerning its number) that is not learned yet. Depending on the information the respective environment provides, not learned means truly not learned or that the probability that it is learned lies under a predefined mastery threshold (as default value for this threshold we use 0.95). The student then gets an exercise that includes only this skill. If all skills are already learned, the student gets a random exercise.

3. **Block Greedy**: Again, the agent looks at the "lowest skill" that is not learned yet in every step. The student then gets an exercise that includes mentioned lowest skill and in total *max_skills*. Which of the possible exercises is taken, is decided randomly.

6 Data

As it is often the case, it must be found a balance between the complexity of a real world scenario and the simplifications that are used for being able to implement a working model. In our project, we therefore looked at simulated data, which is very flexible and easy to control, and at real world data namely the *Assistment Skillbuilder data* [1, 6]. The easiest case is looking at a homogeneous student group that has to learn exactly one skill.
However, this scenario is quite simplistic. Therefore, it makes sense to expand to students with different abilities (learn faster, slip more easily,...) and take into account more skills as well as dependencies between those skills. As correlations between all skills (more than 50 in the case of the *Skillbuilder data*) would introduce too many variables and dependencies in our block BKT model[2], we decided to simulate/find blocks of co-occurring skills in our data. Projecting this to actual students and exercises, it would mean that the understanding of logarithmic and exponential functions might occur in the same block, while knowledge about basic descriptive statistics might be found in another block.

6.1 Assumed Data Structure

After testing some smaller cases first, the final assumptions concerning our data that we made when giving exercises to students were:

- n_skills: in total n_skills can be learned.
- n_blocks: n_skills are divided into various blocks of different size, each skill is included in exactly one block. We tried blocks up to size 6.
- *exercises*: Each exercise deals with skills from exactly one block. We get feedback if a student (identified by a user ID) solves an exercise correctly or not (0/1). The different exercises (including different skills) are called exercise types.
- max_skills: Each exercise can involve at most max_skills skills.

According to our model assumptions a student gets an exercise that deals with various skills (eg skill 1,3,4). The student has either already learned each skill or not (e.g. has learned skill 1, but not 3 and 4). We assume that a student does know how to solve an exercise, if he has learned **all** respective skills. Nevertheless, he can still slip (or guess the exercise right per accident if he does not know all skills yet) - so the probability that he solves the exercise correctly is P(S) or P(G) depending on his current state. After working on an exercise, the student

[2] Note that the number of states grows exponentially with the number of skills.

may learn new skills - for each included skill this happens with a probability P(T) (which may depend on the skill). In general we also assume that students do not forget skills.
For actually fitting our BKT models we used a data frame that consists of one exercise per row and as columns: user ID, included skills in this exercise (can be multiple columns) and if the exercise was correctly answered or not.

6.2 Simulator - Simulated Data

In total, we implemented 3 different versions of our simulator, to simulate data with our assumed characteristics. Basically, each new version extends the functionalities of the earlier ones:

- *Simulator_v*1:
  - Can handle only exercises that include exactly one skill.
  - Includes a learning array that shows for each student and skill after how many exercises it was learned.
- *Simulator_v*2:
  - Can handle exercises that include also multiple skills.
  - Each block includes the same amount of skills.
  - Each exercise includes the same amount of skills.
- *Simulator_v*3:
  - Can handle exercises that include also multiple skills.
  - Includes blocks and exercises of various size.

6.3 Skillbuilder Data

The *Skillbuilder* dataset contains almost 350000 rows, where each row contains one exercise execution by some student involving up to 4 skills, i.e. $max\_skills = 4$. We also have the outcome whether the student solved the exercise correctly (C_t in our notation), a *user_id* and an *order_id* allowing us to sort the recorded learning sequence of each student by time. In order to be able to use the data in our block BKT model, we had to conduct a couple of preprocessing steps:

1. We first removed rows without any known skill. After that, we ended up with 283105 rows, 4163 students, 149 different exercise types ($|\boldsymbol{E}| = 149$ in our notation) and 123 skills ($|\boldsymbol{S}| = 123$ in our notation). Another column *problem_id* indicates the actual exercise instance, where we had 17751 distinct values. However we are not using this information since we just operate on the exercise type level.
2. Next, we removed rare skills with less than 30 occurrences in the whole dataset. Since we have to estimate a prior $P(L_0)$ and a transition probability $P(T)$ for every single skill, too little occurrences will prevent us from finding reliable estimates. This step only removed 45 rows.

3. To find all exclusive blocks of co-occurring skills, we defined an undirected skill graph $G = (V, E)$, where the node set V is given by the skill set $\boldsymbol{S}$ and an edge $e_{ij} \in E$ between the skills i and j exists if the two skills co-occur in at least one row. Further, each edge e_{ij} has a weight equal to the number of times the two skills i and j co-occur in our data. The connected components of this graph exactly correspond to the skill blocks we are looking for. The dataset contains 75 such blocks with the largest block containing 15 skills.
4. A block with 15 skills would have 2^{15} states in our block BKT model. Therefore, we decided to split some blocks to have a maximum block size of 5. We repeatedly applied spectral clustering to split the largest block until all blocks had 5 or less skills. For spectral clustering, we used the affinity matrix of the connected component corresponding to the respective block from the skill graph.
5. Eventually, we extracted a separate dataset for every block. There are 83 such blocks containing a total of 116 different skills. The datasets for the blocks have 32 to 24253 rows, include 5 to 1353 students and a row per student ratio between 1 and 30.15. Note that the result depends on the random number seed due to the randomness in spectral clustering when using k-means.

7 Knowledge Tracing Results

7.1 Simple BKT

To evaluate our simple BKT implementation, we simulated 2000 students, each with a learning sequence of length $T = 50$ for a single skill with our *Simulator_v1*. Afterwards, we tried to estimate the BKT parameters with our simple BKT implementation. By repeating this experiment 10 times, we can compute a mean prediction as well as an empirical mean squared error (see: Table 1)[3].

	$P(L_0)$	$P(T)$	$P(G)$	$P(S)$
True parameter	0.05	0.20	0.30	0.15
Mean prediction	0.04	0.20	0.28	0.15
Empirical MSE	0.0002	0.0002	0.0013	0.0001

Table 1: Results for 10 trials of fitting the simple BKT model on simulated data with *Simulator_v1*.

On top of that, we decided to compare our implementation to that of a group of researchers that made their library (pyBKT) publicly available [2]. In particular, we simulating a slow and a fast learner group and merged both datasets. Looking at Table 2, our fitting performs quite well in comparison to pyBKT and to the actual data [4]. Obviously, the results depend on the random number seed though.

[3] The results can be reproduced with the *test_simplebkt_simulator_v1* script.
[4] These results were obtained by running the scripts *fit_pybkt* & *fit_simplebkt*.

	$P(L_0)$			$P(T)$			$P(G)$			$P(S)$		
Dataset	true	pyBKT	own	true	pyBKT	own	true	pyBKT	own	true	pyBKT	own
Slow learners	0	0.02	0.06	0.10	0.08	0.08	0.10	0.10	0.11	0.15	0.14	0.14
Fast learners	0	0.09	0.09	0.30	0.30	0.30	0.20	0.20	0.20	0.20	0.18	0.17
Mixed learners	-	0.08	0.08	-	0.12	0.13	-	0.15	0.15	-	0.15	0.15

Table 2: Results for fitting on a single skill compared with pyBKT

7.2 Block BKT

To test our block BKT implementation, we simulated 2000 students, each with a learning sequence of length $T = 60$ for a block of 5 skills and *max_skills*= 4. Table 3 presents the results when comparing the true data generating parameters, the estimates found by the simple BKT model and the estimates found by the block BKT model. Note that the block BKT model exactly represents the simulation process, while the simple BKT model does not. Also note that the simple BKT model will find a slip and guess parameter for every single skill, while the block BKT model has shared slip and guess parameters for the whole block. It turns out that both models overestimate the prior probabilities. Surprisingly, this effect is even worse for the block BKT model. However, the block BKT model succeeds in precisely recovering the true transition parameters and is more reliable when it comes to slipping and guessing parameters[5].

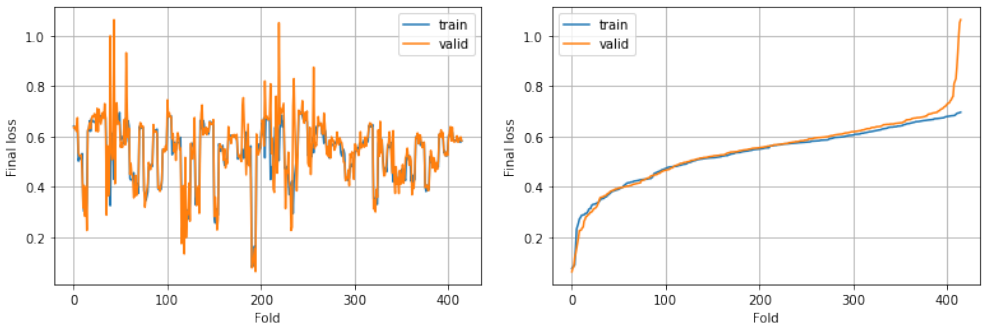
	$P(L_0)$			$P(T)$			$P(G)$			$P(S)$		
Skill	true	simple	block	true	simple	block	true	simple	block	true	simple	block
1	0	0.14	0.17	0.20	0.19	0.20		0.17			0.30	
2	0	0.13	0.18	0.30	0.14	0.30		0.22			0.28	
3	0	0.07	0.14	0.52	0.12	0.54	0.20	0.27	0.19	0.20	0.26	0.23
4	0.20	0.07	0.22	0.07	0.11	0.07		0.19			0.24	
5	0	0.14	0.13	0.12	0.10	0.12		0.21			0.26	

Table 3: Comparison of the simple BKT and the block BKT model when fitting a block of 5 skills. The simulator (true) and the block BKT model use shared values $P(G)$ and $P(S)$ for the whole block.

Cross-Validation To fit the 83 blocks of the Skillbuilder dataset, we used 5-fold cross-validation to find useful hyperparameters, especially the learning rate. Unfortunately, fitting so many models (also on quite different datasets) is a hard task. We conducted some manual inspection of training and validation learning curves to draw some conclusions. Figure 2 shows the average batch

[5] These results were obtained by running the scripts *test_blockbkt_simulator_v3* & *test_simplebkt_fitmultiple_simulator_v3*.

loss for one epoch after the last gradient step for all training and validation folds ($83 \cdot 5 = 415$ in total). The training and validation losses have a one-to-one correspondence in the left subfigure. In the right subfigure, both arrays are just sorted independently[6]. We can see that the validation loss tends to be a bit higher than the training loss, which in principle is a good sign. It might be the case however, that the model is actually underfitting on some blocks, especially if there is enough data to use a more complex model. In retrospect, we would use the forward logic of Bayesian knowledge tracing to evaluate the predictive quality on student sequences instead of just focusing on the negative log-likelihood loss, which is harder to interpret. An accuracy might be easier to grasp and would also allow for comparing different models, e.g. comparing the block BKT model to a DKT model. Another thing that we would adapt is our parameter initialization. The $U[0, 1]$ distribution we are currently using does not really accord with how we regularize parameters.



(a) Aligned (b) Individually sorted

Fig. 2: Cross-Validation results showing the average batch loss on one epoch after the last gradient update step.

Final Fit Finally, we fit block BKT models to all Skillbuilder blocks again, this time without cross-validation and therefore using all available student sequences. The final average batch loss for a single epoch (for all 83 blocks) is depicted in Figure 3. The distribution of all parameter estimate is depicted in Figure 4[7].

[6] These results were obtained by running the scripts *crossvalidate_skillbuilder_blockbkt* & *evaluate_crossvalidation*.

[7] These results were obtained by running the scripts *fit_skillbuilder_blockbkt* & *evaluate_fit*.

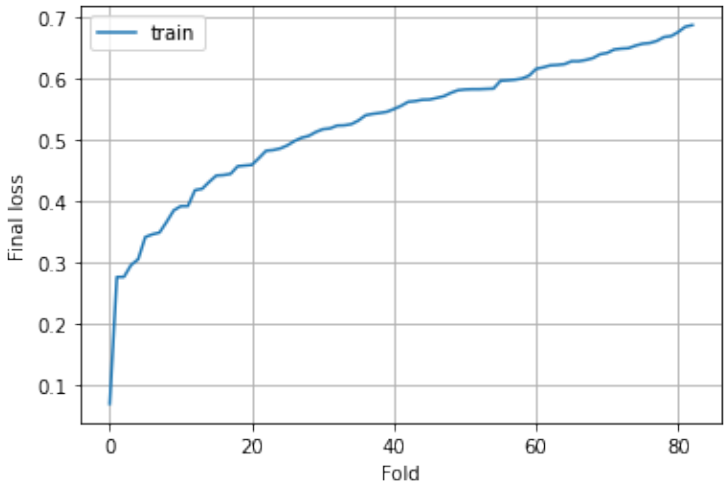


Fig. 3: The average batch loss on one epoch after the last gradient update step for all 83 final block BKT models.

8 Reinforcement Learning Results

For evaluating the performance of our reinforcement learning agent, we tested various models and saved them to be able to also access them later on. Although we tried different methods for seeding, none seemed to work in the intended way - therefore we achieved our results without seeds (but for evaluation, it was always averaged over 100 episodes).

8.1 Simulated Data

Having a lot of different algorithms and parameters, we decided on testing some general configurations, see Table 4, and then focusing for the Skillbuilder data set and further analysis on the more promising ones. For testing, we used (BKT) parameters that included 14 skills with 3 blocks and various parameters for guess, slip, transition and priors L_0. max_skills was set to 3 and students received 100 exercises. We always trained with 500000 time steps and a discount factor of 0.99. The evaluation results were averaged over 100 episodes. [8]
Looking at those first results, the block greedy strategy clearly performs quite well and is hard to beat. One reason may be, that in our data structure learning one single skill in exercises with multiple skills is equally hard as learning it in a single skill exercise. This also reflects in the bad performance of single greedy. Thinking about this topic concerning actual students and exercises, our model

[8] The results of this subsection can be achieved by running *run_RL_simulated*.
[9] Describes the percentual difference between best algorithm and best baseline

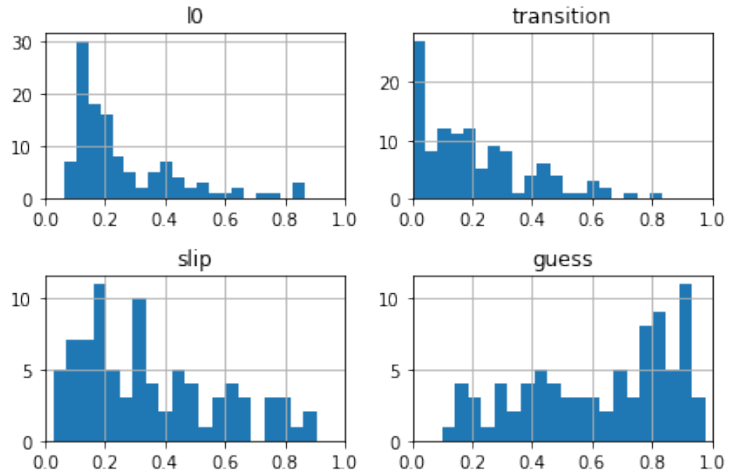


Fig. 4: Distribution of all fitted parameter estimates in the final fit.

	MLP			MLP LSTM		baselines			
Env.	A2C	PPO2	DQN	A2C	PPO2	random	single greedy	block greedy	gap [9]
1	191.07	166.25	158.86	61.1	127.37	117.2	199.37	**208.27**	8.25%
2	70.61	56.92	34.49	64.74	31.68	57.15	54.74	**73.4**	3.80%
3	12.52	13.3	3.49	12.04	12.23	11.23	11.73	**13.79**	3.55%
4	894.16	841.39	600.78	401.72	796.94	778.28	616.88	**919.15**	2.71%
5	**936.77**	796.03	901.18	494.35	490.46	780.09	610.08	918.13	-
6	137.16	142.14	144.58	130.74	104.99	113.01	121.88	**172.07**	15.98%

Table 4: Rewards for various configurations

is probably still too simplistic. On the other hand, it may also depend on the skills themselves which model makes the most sense.

Interesting to see is also that *DQN* seems to have huge difficulties with environment 3 (where we just provide a reward in the end). What is also visible, is that the LSTM policies do not perform well. The actual reason for that is hard to evaluate by just looking at the table. One cause may be the higher complexity of those models (so they would need more timesteps and tuning). Most promising seems to be environment 5, as here our agent performs already better than the baseline, and in general A2C with a MLP policy. This algorithm is the best performing agent in 4 out of 6 environments. Concerning the runtime for training, it seems only to be dependent on the policy (so MLP being clearly faster than MLP LSTM) and not on the algorithm or environment.

Considering the percentage value for "gap", the best performing configurations are *Environment 5 - A2C - MLP*, *Environment 4 - A2C - MLP* and *Environment 3 - PPO2 - MLP*. We tested if more timesteps or a different discount factor have

a positive effect, by evaluating the standard deviation of the evaluation episodes and how many skills were learned on average in the end, see Table 5.

				Reward		Skills learned		
Agent	timest.	discount fac.	Environment	Agent	Best Basel.	Agent	Best Basel.	Std
A2C,MLP	500k	0.99	5	936.77	918.13	12.4	13.4	61.40
A2C,MLP	1Mio	0.99	5	959.1	918.13	12.77	13.4	52.47
A2C,MLP	500k	0.90	5	968.71	918.13	12.93	13.4	62.34
A2C,MLP	1Mio	0.90	5	971.02	918.13	13.15	13.4	64.18
PPO2,MLP	500k	0.99	3	13.3	13.79	13.3	13.79	0.81
PPO2,MLP	1Mio	0.99	3	13.35	13.7	13.35	13.7	0.84
PPO2,MLP	500k	0.90	3	11.78	13.7	11.78	13.7	1.03
A2C,MLP	500k	0.99	4	894.16	919.15	13.27	13.43	57.36
A2C,MLP	1Mio	0.99	4	691.49	919.15	9.87	13.43	54.07
A2C,MLP	500k	0.90	4	797.65	919.15	10.92	13.33	59.07

Table 5: Test concerning different learning rates and time steps

By using different parameters, the configuration *Environment 5 - A2C - MLP* now manages to outperform the greedy block strategy concerning the reward. Unfortunately, looking at the average amount of learned skills, the greedy block baseline still outperforms all of our agents. Nevertheless, at least for 2 configurations we reached an improvement by increasing the time steps and/or using a different discount factor. Looking at the penultimate row, it does not always help to provide additional time steps, but it does in all three cases lower the standard deviation, so more time steps make the agents more stable.
We decided to use respectively the best configurations found, as visible in Table 6 for all further elaborations and named them Agent 1-3 for easier reference.

Agent name	Algorithm	Policy	Time steps	Discount Factor	Environment
Agent 1	A2C	MLP	1Mio	0.90	5
Agent 2	A2C	MLP	500k	0.99	4
Agent 3	PPO2	MLP	1Mio	0.99	3

Table 6: Models

As these agents (and the greedy block baseline) learn almost all skills in the till now tested scenario, it may provide useful to have a look at what happens if we look at a scenario with only 50 exercises, see Table 7.
Providing less exercises to each student, we can examine better how fast our algorithms actually learn. Our results show, that Agent 1 & 3 perform better than the best baseline now, so manage to teach a student more skills with reduced

	Reward				Skills learned	
Agent name	Agent	Random	Single Greedy	Block Greedy	Agent	Best Baseline
Agent 1	369.37	288.05	192.76	320.09	10.32	10.18
Agent 2	265.45	281.97	197.01	309.64	6.69	9.83
Agent 3	11.38	9.0	8.27	11.22	11.38	11.22

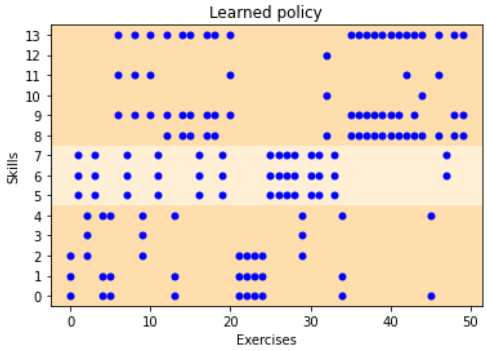
Table 7: Results with 50 exercises

exercise opportunities. Agent 2, which gets the least information concerning the observations, does not perform well, in this case even worse than a random agent. Looking at the amount of learned skills, it is also visible, that the learning progress after 50 exercises is already well advanced, so depending on the goal, it may be enough to query only 50 exercises.
Let's also have a look at the curricula of those models in comparison to the (best) baselines, see Figure 5. To understand the curricula, it is important to know, that each skill has a different transition probability - the lowest ones being 0.1 for skill 3 and 7 and 0.05 for skills 8, 10 and 12.
Interesting to see is the influence of transition probabilities, especially well visible in (b), (d) and (f). As the greedy baselines start with the naming wise lowest unlearned skill, it may happen, that they get stuck on a ´´difficult skill" like the single greedy baseline in (f) on skill 3 and the block greedy policy in (b) and (d) on skill 8. We thought therefore about implementing the greedy baselines in a way that they start by learning the easier skills, but decided against it as we do want our students to learn all skills and not preferably easy skills. Our agents do not seem to be influenced by this factor (they are also not required to query exercises in a special order).
Looking closer at the agents, they mostly seem to exploit the maximum amount of skills per exercise (3), as does the block policy, but not all the time. Agent 1 seems to vary the exercises a lot between the different blocks while agent 3 focuses more on blocks for a longer period. As was also visible in the results concerning the reward, Agent 2 does not seem to learn well and stays in block 1 almost the whole time.

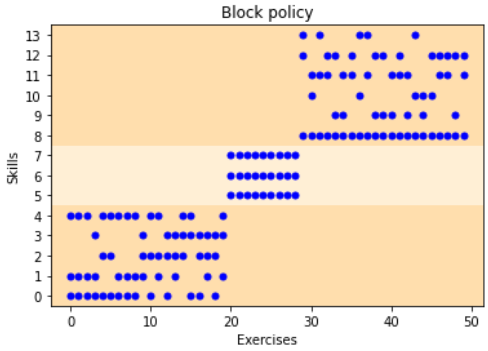
8.2 Skillbuilder Data

Testing the before agreed on 3 models on the fitted skillbuilder parameters, we achieve the following results[10]. Since the number of skills (116 skills in 83 blocks) is bigger, we propose more exercises to the students (300 exercises), see Table 8. We can look at the respective curricula in Figure 6. Again clearly visible is the disadvantage of the greedy policy of being stuck on an exercise with low transition probability (skill 24 has a transition probability of 0.003) and therefore in some cases even performing worse than the random policy. Interesting to see is

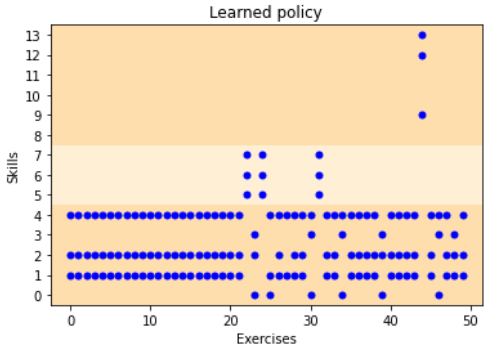
[10] The results of this subsection can be achieved by running *run_RL_skillbuilder*.



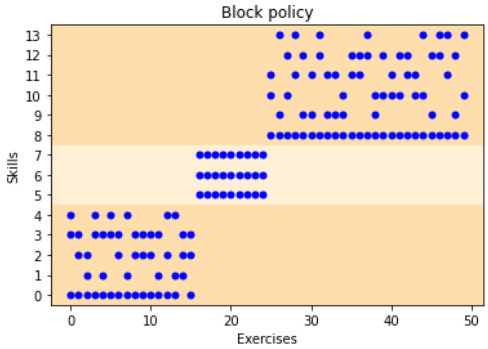
(a) Agent 1



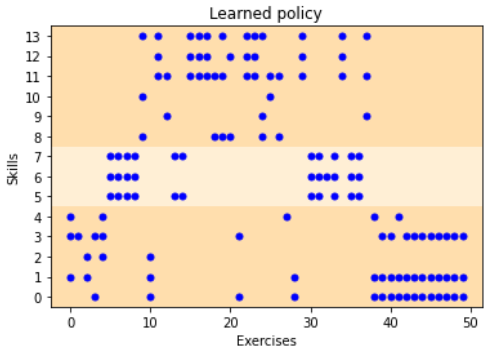
(b) Best baseline (block greedy, env 5)



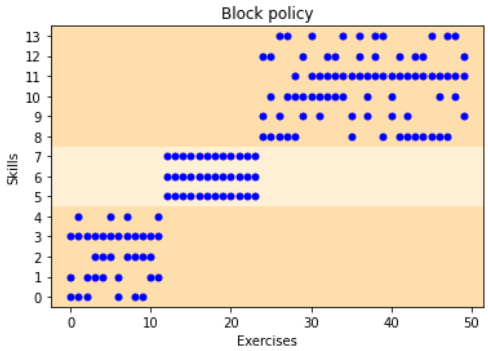
(c) Agent 2



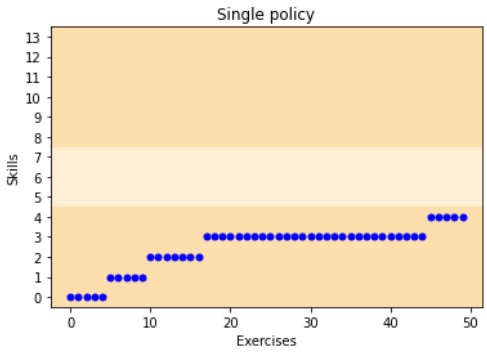
(d) Best baseline (block greedy, env 4)



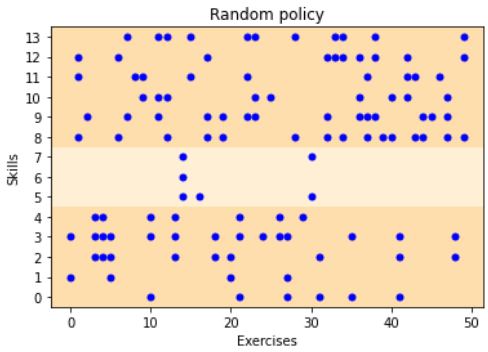
(e) Agent 3



(f) Best baseline (block greedy, env 3)



(f) For comparison: Single greedy (env 1)



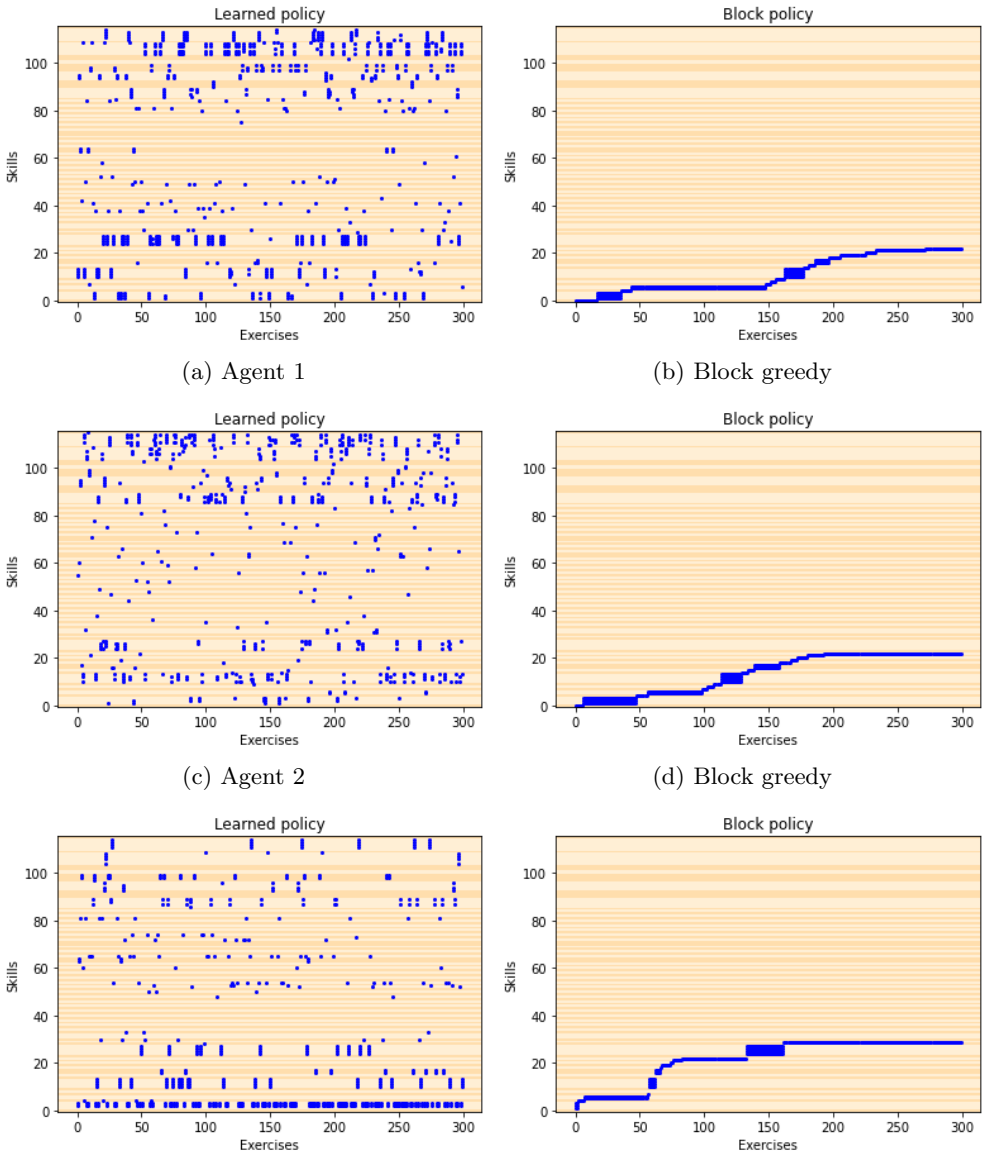
(g) For comparison: Random (env 1)

Fig. 5: Curricula of different agents

	Reward				Skills learned	
Agentname	Agent	Random	Single Greedy	Block Greedy	Agent	Best Baseline
Agent 1	15946.8	14394.34	11495.48	12102.34	62.39	46.1
Agent 2	14485.06	14417.8	11396.89	12152.49	59.33	60.67
Agent 3	57.14	59.82	47.5	49.36	57.14	59.82

Table 8: Results Skillbuilder

also that in this case, although the number of exercises is again low in comparison to the number of skills, Agent 2 performs better again. As for the simulated data, the agents again try to exhaust the maximum number of skills in most exercises. Nevertheless, the influence of this tactic is not so extreme, as most blocks are rather small. That may also be the reason, why the results of the different greedy strategies do not differ so much.



(a) Agent 1 (b) Block greedy

(c) Agent 2 (d) Block greedy

(e) Agent 3 (f) Block greedy

Fig. 6: Curricula of one evaluation run on Skillbuilder data

8.3 Fairness

All results we looked at till now actually assumed a homogeneous group of students that all have the same abilities. If we compare that to reality, this is actually not true in most cases - therefore it makes sense to also evaluate our models looking at mixed groups of students (for example slow and fast learners).[11] One idea may be, as also done in [5], to evaluate the learning gap between slow and fast learners and also take into account the number of extra opportunities that students receive after they have already learned the skill. In this experiment, data is simulated for 100 slow and 100 fast learners (defined as in Table 2). This simulated students receive a fixed amount of exercises (1 - 100). At the end of each of this 100 runs it is evaluated how many fast/slow learners have learned the skill till then and the percentual gap between those groups. Furthermore, also the number of additional exercises (=extra opportunities) that they had to solve after they had already learned the skill is shown. On the other hand, we also execute the fitted BKT model and consider a skill as learned if the probability, that the skill is learned lies over 95 %.[12] As can be seen in Figure 7, BKT clearly finds a better balance between querying additional exercises while keeping the learning gap small.

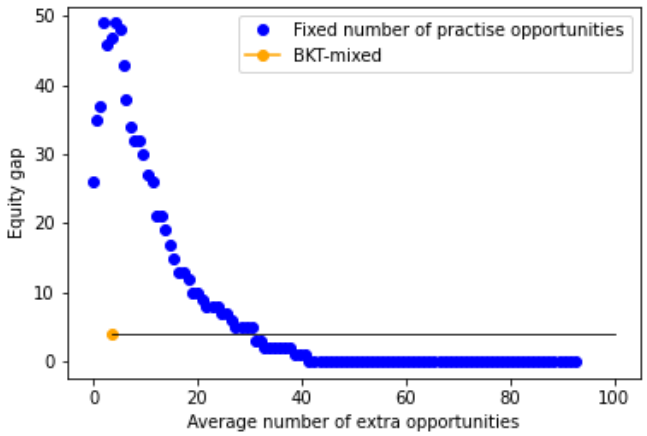


Fig. 7: Comparing the equity gap and extra number of opportunities to a fixed number of extra opportunities

Taking this a bit further, it is possible to extend this idea to multiple skills and our reinforcement learning agents. We therefore defined again two groups of

[11] The results of this subsection can be achieved by running *run_RL_evaluatewithtruedata,produce_figure* and *run_RL_slowfast*.

[12] This works similarly as shown for environment 4.

students - slow & fast learners - where fast learners had a transition probability of +0.1 for every skill compared to the slow learners and were less likely to slip.[13] We fitted on the combined data set and used these parameters to train our agents. In the end, we can have a look at how well the agent performs when dealing with both groups separately, see Figure 9. For comparison, we also show the results of an agent trained on the actual students, denoted by Agent*. As our greedy baselines do not run their own BKT model, but rather take the values from the environment, the baselines are provided with more information than the Agent in Environments 4&6 (for all other environments the observation consists of the (probabilistic) learning states).

		Reward					Skills learned		
Agentname	Students	Agent	Agent*	Random	Single Greedy	Block Greedy	Agent	Agent*	Best Baseline
Agent 1	Slow	**367.02**	359.73	282.42	189.88	315.78	**10.6**	9.74	10.0
Agent 1	Fast	**488.07**	480.82	387.31	268.49	436.56	13.2	12.44	**13.4**
Agent 1	Fitted	**482.98**	-	395.8	293.66	424.47	**12.7**	-	12.5
Agent 2	Slow	**335.74**	279.4	278.85	186.52	307.61	**10.2**	7.9	10.1
Agent 2	Fast	442.82	**444.44**	387.11	268.29	439.46	12.4	12.46	**13.4**
Agent 2	Fitted	**443.06**	-	395.57	287.5	434.01	11.8	-	**12.8**
Agent 3	Slow	11.35	**12.09**	8.7	8.67	11.65	11.35	**12.09**	11.65
Agent 3	Fast	13.79	**13.91**	11.25	12.11	**13.91**	13.79	**13.91**	**13.91**
Agent 3	Fitted	13.71	-	10.87	11.17	**13.74**	13.71	-	**13.74**

Table 9: Results on different student groups

Looking at those numbers, although clearly the slow students learn less (concerning learned skills and reward), in general agents that learned on an environment with fitted parameters performed well. Comparing the results of *Agent* to the *Agent**, especially for Agent 1 it proves true, that the Agent trained on the fitted data (environment) actually performs better than the Agent trained on with the same data that was used for evaluation. As the margins between the respective values are relatively small, we cannot prove the Agent trained on fitted data to be significantly better, but also it is not worse (as could have been suspected). One hypothesis may also be, that the fitted data helps the algorithm not too focus too much on the extreme cases (very low/very high transition), as the fitting leads to some kind of averaging between the slow and the fast parameters.

9 Conclusion

In conclusion, we implemented simple to medium flexible knowledge tracing models, which worked more or less as expected on data simulated according to the assumed data generating process. It turns out that fitting real-life data is a challenging task, typically involving multiple preprocessing steps, and is potentially difficult to evaluate and supervise. Using a BKT model for reinforcement

[13] Again 14 skills were used and 50 exercises

learning provided itself quite useful and effective, as especially our *Agent 1* can handle various configurations concerning exercises or the data it is fitted onto quite well. Nevertheless the block greedy baseline is a strong opponent that also performs really well (and maybe also shows some weaknesses of our model assumptions). In general, there is still a lot of room for improvement by testing different configurations for the agents, elaborating more why recurrent policies did not work, using environments were it is learned till all skills are supposed to be known and so on. Unfortunately this was time and computational ressource-wise out of the scope for this project.

References

1. Skill-builder data 2009-2010, https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data/skill-builder-data-2009-2010
2. Badrinath, A., Wang, F., Pardos, Z.: pybkt: an accessible python library of bayesian knowledge tracing models. arXiv preprint arXiv:2105.00385 (2021), https://github.com/CAHLR/pyBKT, url of repository included
3. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016), https://gym.openai.com/
4. Corbett, A.T., Anderson, J.R.: Knowledge tracing: Modeling the acquisition of procedural knowledge. User modeling and user-adapted interaction **4**(4), 253–278 (1994)
5. Doroudi, S., Brunskill, E.: Fairer but not fair enough on the equitability of knowledge tracing. In: Proceedings of the 9th International Conference on Learning Analytics & Knowledge. pp. 335–339 (2019)
6. Feng, M., Heffernan, N., Koedinger, K.: Addressing the assessment challenge with an online system that tutors as it assesses. User modeling and user-adapted interaction **19**(3), 243–266 (2009)
7. Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y.: Stable baselines. https://github.com/hill-a/stable-baselines (2018)
8. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
9. Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L.J., Sohl-Dickstein, J.: Deep knowledge tracing. Advances in Neural Information Processing Systems **28**, 505–513 (2015)
10. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021), http://jmlr.org/papers/v22/20-1364.html

A Transition Tensor Slice from Block BKT Implementation

to/from	000	100	010	001	110	101	011	111
000	$\log((1-P(T^{(1)}))(1-P(T^{(2)})))$	$-\inf$	$-\inf$	$-\inf$	$-\inf$	$-\inf$	$-\inf$	$-\inf$
100	$\log(P(T^{(1)})(1-P(T^{(2)})))$	$\log(1-P(T^{(2)}))$	$-\inf$	$-\inf$	$-\inf$	$-\inf$	$-\inf$	$-\inf$
010	$\log((1-P(T^{(1)}))P(T^{(2)}))$	$-\inf$	$\log(1-P(T^{(1)}))$	$-\inf$	$-\inf$	$-\inf$	$-\inf$	$-\inf$
001	$-\inf$	$-\inf$	$-\inf$	$\log((1-P(T^{(1)}))(1-P(T^{(2)})))$	$-\inf$	$-\inf$	$-\inf$	$-\inf$
110	$\log(P(T^{(1)})P(T^{(2)}))$	$\log(P(T^{(2)}))$	$\log(P(T^{(1)}))$	$-\inf$	0	$-\inf$	$-\inf$	$-\inf$
101	$-\inf$	$-\inf$	$-\inf$	$\log(P(T^{(1)})(1-P(T^{(2)})))$	$-\inf$	$\log(1-P(T^{(2)}))$	$-\inf$	$-\inf$
011	$-\inf$	$-\inf$	$-\inf$	$\log((1-P(T^{(1)}))P(T^{(2)}))$	$-\inf$	$-\inf$	$\log(1-P(T^{(1)}))$	$-\inf$
111	$-\inf$	$-\inf$	$-\inf$	$\log(P(T^{(1)})P(T^{(2)}))$	$-\inf$	$\log(P(T^{(2)}))$	$\log(P(T^{(1)}))$	0

This is one slice of the logarithmic transition tensor from the example in Section 4. To be precise, it is *transition tensor[:,:,3]*, i.e. the transition matrix for exercise $\begin{pmatrix}1 & 1 & 0\end{pmatrix}$ when fitting a block B_i containing $|B_i| = 3$ skills.