Spaced Repetition Learning with Reinforcement Learning

Mohit Nihalani, Sushanth Samala, Rohit Saraf
{mn2643 , ss12852, rs6785}@nyu.edu

Abstract With the recent advancements of technology there is a lot of development in online education systems such as Duolingo, Khan Academy. These platforms use various machine learning algorithms to provide high-quality personalized teaching to anyone in the world for free by modeling the knowledge of students as they interact with the content. One of the biggest challenges for these online platforms is to choose a sequence of topics and present it to users which helps them to maximize learning by retention of knowledge. One of the modes widely used technique to overcome human forgetting is spaced repetition in which periodic, spaced review of content improves long-term retention. The problem to create an automated tutoring system for this task is that scheduler has to balance the competing priorities of introducing new items and reviewing old items to maximize learning. Recently Deep Reinforcement Learning has yielded proficient agents for complex tasks and they have shown to mimic human behavior. In this paper, we used various state of the art reinforcement learning algorithm to create Automated tutoring agent for spaced repetition system which can learn teaching policies by operating on raw observation of student histories and select the next item to review. Results show that by providing better incorporating better features in the agent's observation state of the art RL algorithm such as PPO (Proximal Policy Optimization) is able to outperform widely-used heuristics like SuperMemo and the Leitner system.

1 Introduction

The ability of humans to learn and retain a large amount of information is an essential component in acquiring new knowledge. Old scientific theories on how humans learn to identify two critical variables that determine the probability of recalling an item; repeated exposure to the item, time since the item was last reviewed. In the early part of the twentieth century, education focused on the acquisition of literacy skills: simple reading, writing, and calculating. It was not the general rule for educational systems to train people to think and read critically, to express themselves clearly and persuasively, to solve complex problems in science and mathematics. Now, at the end of the century, these aspects of high literacy are required of almost everyone to successfully negotiate the complexities of contemporary life. As Nobel laureate Herbert Simon wisely stated, the meaning of "knowing" has shifted from being able to remember and repeat information to being able to find and use it.

In the more recent work in cognitive science the theories like Behaviorism[1] which explains conceptualized learning as a process of forming connections between stimuli and responses. Motivation to learn was assumed to be driven primarily by drives, such as hunger, and the availability of external forces, such as rewards and punishments. Furthermore, existing systems are tailored to specific learning objectives and student models, which limits their flexibility. Modeling student learning is difficult potentially harder than learning a good scheduling policy. The core function of any spaced repetition software is to present items to a user in an optimal way so users can learn as well as review old items to maximize learning.

The motivation for this work is that modeling the student and accurately predicting outcomes is merely a surrogate for helping the student achieve their learning objective. This is a very challenging task as human learning is grounded in complexity of both the human brain and human knowledge, and most of the work relies on first-order Markov models which learns about the student environment first but in this paper we are using recent advancements in model-free and deep reinforcement learning to directly optimize the learning objective and learn effective content selection strategies. The main objective is to create a Tutor RL agent that will interact with the student environment and choose the activities/topics that help students maximize knowledge by making them review already learned topics sequentially and effectively also known as spaced repetition learning.

In this paper we present a formulation in which we apply deep reinforcement learning methods with recurrent policies and better features to improve on the work done in [2]. Spaced Repetition is formulated as a POMDP environment in the paper. We compared the performance of various reinforcement learning algorithms on this task, and these autonomous tutors learn from scratch by interacting with student simulators which are based on cognitive models of human memory. Our primary contribution is a review scheduling algorithm using reinforcement learning which takes an observation of student answering question, takes the action of what student should review or learn using a POMDP policy where we do not have explicit information about which actions should precede others and where the student's knowledge state does not necessarily decompose into independent components, receive rewards once student achieves its learning objectives. We take a similar approach to formalizing spaced repetition as a POMDP which is explained through Fig 1.

2 Related Work

[2] solves the knowledge tracing problem using a partially observable Markov Decision Process(POMDP) model and trust region policy optimization. [3] implements POMDP as a way to model teaching, and uses a particle-based solver to arrive at the most optimal policy. [4] makes use of expectimax search to select teaching actions. [5] defines customized action selection as a contextual bandit problem. [6] trains a teacher with a policy gradient algorithm that uses a particle-based belief update mechanism to track the Bayesian student state.

[7] applies DRL to autonomous tutoring system. We used a method that is similar to DRL on autonomous tutoring systems except that we are applying these DRL algorithms and synthetic techniques in the context of spaced repetition learning. [8] takes the idea of adaptive tutoring task with a real-world gaming application, to avoid expensive online experiments on real students it makes use of an offline importance sampling-based method to select representations and hyperparameter settings.

Existing spaced repetition systems mostly rely on heuristics for review scheduling. The Leitner system [9, 10] to forcefully prioritize items by novelty and difficulty makes use of a network of FIFO queues. Pimsleur [11], SuperMemo [12], Anki [13], and Mnemosyne [14] use layers of pragmatic hard-coded set of rules to decide on when to next review and prioritize items within a session. [15] makes use of some fixed threshold to select the item with predicted recall likelihood this threshold-based policy has a threshold $\theta \in [0, 1]$.

These papers are broadly related to knowledge tracing, the problem of estimating how a student's knowledge changes over time as they interact with content [16, 17, 18, 4, 19], and machine teaching, the method of arriving at optimal training set for a learning problem [20, 21, 22].

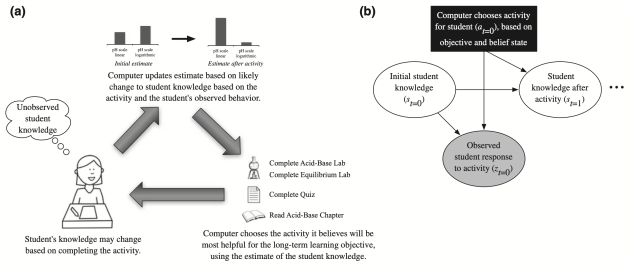

Fig 1. Mapping the POMDP model to teaching. (a) The teaching process consists of the computer choosing actions, dependent upon student knowledge. The student knowledge evolves based on the activities that she completes. By observing the student's behavior, such as the tutor can gain information about what the student understands and what misunderstandings she may have. The computer teacher can make pedagogical choices to achieve some long-term objective, such as minimizing the time for the student to reach mastery[31].

3 Models

For creating RL Agents for the spaced repetition learning task we need to have some sort of data or simulator which interacts with the agent and create trajectories on which agent can learn. Finding the appropriate data set for this task was challenging as all of the datasets didn't meet our requirements. The main problems with openly available datasets such as widely used 'ASSISTment Skill Builder Data' and 'Duolingo HLR [37]' is to create an interactive student learning environment e.g. if the agent selects a particular item to review, likely, the particular topic was not available in the dataset trajectory for that particular user, another problem is to find the probability of the user able to recall a present item and shape rewards based on the recall probability. So we decided to create the student simulator model as described in [2] which is based on the cognitive model of the human memory and this simulator interacts with our agent similar to the real-world scenario. We experimented with two different models of human memory which are listed below.

3.1 Probabilistic models of Human Memory

Ebbinghaus's exponential forgetting: This is one of the oldest forms of probabilities model of human memory. We are replicating the similar model design mentioned in [2] so we can compare our results with them easily. The model used is the binary version of the model which means either the user completely recalls an item or forgot an item. Therefore we model the probability that the learner recalls (forgets) item I at time t using the exponential forgetting curve model i.e.:

$$\mathbb{P}(r) = exp(-n_i(t)(t - t_r)) \tag{1}$$

where t_r is the time of the last review and $n_t \in \mathbb{R}^+$ is the forgetting rate at time t and this depends on many factors such as the difficulty of the item or number of successful and unsuccessful attempts. For simplicity we are only taking the difficulty of the item into the account.

Half Life Regression Model: This model became very popular after the study done by Duolingo[37]. It is an extension of the EFC model and it takes into account the log-linear model of memory strength instead of the item difficulty. Therefore the probability for the item to be recalled is:

$$S = exp(\overrightarrow{\theta} . \overrightarrow{x}) \tag{2}$$

Where $\overrightarrow{x}$ is a vector representing the features of the student history, where features include number of attempts, correct answers, incorrect answers, and identity of n items i.e. $X = \mathbb{N}^3 \times \{0,1\}^n$ similar to [37], and $\overrightarrow{\theta}$ are the model parameters.

3.2 Reinforcement Learning:

There has been a lot of advancement in Reinforcement Learning methods especially in knowledge tracing field as

a lot of new algorithms are being created which interacts with the real-world system and recommends the action which leads to the most benefit in the future. As reinforcement learning is best used for sequential decision making under uncertainty it was best suited for our problem of spaced repetition learning. Reinforcement learning works well on an MDP environment and there are broadly two different types of RL algorithms:
On the one hand there model-based RL, in which we know the dynamics of the environment such as state transition probabilities and action probabilities. This type of environment is usually not available. On the other hand, there are Model-Free RL algorithms where model learns to form memories based on interactions with the world and this makes it challenging since it is unknown which features of observation will be relevant later and associations may have to be formed over many steps, to solve this problem fully observed environment is assumed.
For both of the cases the ultimate goal of any agent is to learn policy π^* that maximizes expected future discounted return. MDP is represented with states S, actions a, transition distribution T, reward function R and discount factor γ which is used to give more weight to the recent interactions of the environment compared to the older ones. So π^* can be represented as:

$$\pi^* = \text{argmax}\,\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)|\pi] \tag{3}$$

For this project we worked with various model-free algorithms primarily on various policy optimization algorithms such as TRPO, PPO, and A2C.

3.3 Trust Region Policy Optimization (TRPO)

TRPO is one of the widely used policy gradient method which optimizes the policy by taking the largest step towards optimal policy but within a defined constrained which keeps new policy not to diverge by a huge margin from the old policy. This constrain is known as KL divergence. This helps to solve the biggest disadvantage of vanilla policy gradient methods which keeps new and old policy close in parameter space but a small difference in parameter space can lead to a bad step which collapses the policy performance and because of that it's impossible to use large step size and simultaneously hurts its sample efficiency. TRPO is effective in avoiding this type of collapse as KL divergence is a measure of distance between probability distributions which helps to monotonically improve performance.
Let π_θ denote a policy with parameters θ. The theoretical TRPO update is:

$$\theta_{k+1} = \arg\max_{\theta}\ \mathcal{L}(\theta_k,\theta)\text{s.t. }\bar{D}_{KL}(\theta||\theta_k) \le \delta \tag{4}$$

where $\mathcal{L}(\theta_k,\theta)$ is the surrogate advantage, a measure of how policy π_θ performs relative to the old policy π_{θ_k} using data from the old policy:

$$\mathcal{L}(\theta_k,\theta) = s,a \sim \pi_{\theta_k}\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), \tag{5}$$

and $\bar{D}_{KL}(\theta||\theta_k)$ is an average KL-divergence between policies across states visited by the old policy:

$$\bar{D}_{KL}(\theta||\theta_k) = s \sim \pi_{\theta_k} D_{KL}\left(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s)\right). \tag{6}$$

For making TRPO update easy there are some approximations made such as:

$$\mathcal{L}(\theta_k,\theta) \approx g^T(\theta-\theta_k) \tag{7}$$

$$\bar{D}_{KL}(\theta||\theta_k) \approx \frac{1}{2}(\theta-\theta_k)^T H(\theta-\theta_k) \tag{8}$$

These approximations helps to reduce objective function:

$$\theta_{k+1} = \arg\max_{\theta}\ g^T(\theta-\theta_k) \tag{9}$$

$$\text{s.t. }\frac{1}{2}(\theta-\theta_k)^T H(\theta-\theta_k) \le \delta \tag{10}$$

TRPO updates the policy using backtracking line search:

$$\theta_{k+1} = \theta_k + \alpha^j\sqrt{\frac{2\delta}{g^T H^{-1} g}}H^{-1}g \tag{11}$$

To prevent repeated calculation of H^{-1}, TRPO uses conjugate gradient descent method algorithm to solve $x = H^{-1}g$ requiring only a function which can compute the matrix-vector product Hx instead of computing and storing the whole matrix H. Here H is the Hessian of the sample average KL divergence.

3.4 Advantage Actor Critic (A2C)

The aim of designing this method was to find RL algorithms that can train deep neural network policies reliably and without large resource requirements. While the underlying RL methods are quite different, with actor-critic being an on-policy policy search method and Q-learning being an off-policy value-based method, we use two main ideas to make our algorithms practical given our design goal.

Synchronous advantage actor-critic (A2C) maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate of the value function $V(s_t;\theta_v)$. actor-critic operates in the forward view and uses the mix of n-step returns to update both the policy and the value function. The policy and the value function are updated after every t_{max} actions or when a terminal state is reached. The update performed by the algorithm can be seen as

$$\bigtriangledown_{\theta'}\log\pi(a_t|s_t;\theta)A(s_t,a_t;\theta,\theta_v) \tag{12}$$

where $A(s_t,a_t;\theta,\theta_v)$ is an estimate of the advantage function given by

$$\sum_{i=0}^{k-1}\gamma^i r_{t+1} + \gamma^k V(s_{t+k};\theta') - V(s_t;\theta_v) \tag{13}$$

where k can vary from state to state and is upper-bounded by t_{max}. As with the value-based methods we rely on parallel actor-learners and accumulated updates for improving training stability. Note that while the parameters θ of the policy and θ_v of the value function are shown as being separate for generality, we always share some of the parameters in practice[36]. We used an MLPLSTM policy which has two MLP layers of size 64 connected to an LSTM layer to produce the output for the policy $\pi(a_t|s_t;\theta)$ and one linear output for the value function $V(s_t;\theta_v)$. We also found that adding the entropy of the policy π to the objective function improved exploration by discouraging premature convergence to suboptimal deterministic policies. This technique was originally proposed by (Williams Peng, 1991), who found that it was particularly helpful on tasks requiring hierarchical behavior. The gradient of the full objective function including the entropy regularization term with respect to the policy parameters takes the form

$$\triangle_{\theta'} \log \pi(a_t|s_t;\theta')(R_t - V(s_t;\theta_v)) + \beta \triangle_{\theta'} H(\pi(s_t;\theta')) \quad (14)$$

where H is the entropy. The hyper-parameter β controls the strength of the entropy regularization term. Optimization: We investigated three different optimization algorithms in our synchronous framework – SGD with momentum, RMSProp without shared statistics, and RMSProp with shared statistics. We used the standard non-centered RMSProp update given by

$$\triangle\theta g = \alpha g + (1-\alpha) \triangle \theta^2 and \theta < -\theta - n\sqrt{g} + \varepsilon \quad (15)$$

where all operations are performed element-wise.

3.5 Proximal Policy Optimization

Proximal policy optimization (PPO) algorithms is another type of policy gradient algorithms that has shown great performance on a variety of tasks, we use the version of PPO created by Hill et al 2018. as our Reinforcement Learning algorithm. It combines the idea of A2C(having multiple workers) and TRPO as it uses the trust-region to improve the actor. PPO is based on the idea of how to make the biggest possible improvement step on the policy using the experiences collected without going outside the region which leads to performance collapse. This problem is solved by TRPO but they are not easily compatible with algorithms that share parameters between value and policy function and apart from they use second-order optimization methods. PPO on the other hand uses the first-order optimization like gradient descent and some tricks to prevent big deviations of the new policy from the old policy. According to many research PPO is claimed to be more sample efficient, less complex, and perform at least as well as TRPO.
There are two variants of PPO:
PPO Penalty: This variant is much more similar to TRPO as it solves a constrained KL constrained update but it updates the penalty coefficient throughout the training automatically by penalizing KL divergence in the objective function rather than making it a hard constrained.

PPO Clip: This is a new variant of PPO which minimizes a novel objective function and this variant is used by us in our experiments. It doesn't have any KL divergence which makes it easy to implement with the Stochastic Gradient descent method as it doesn't need to make adaptive updates. PPO updates the policy via:

$$\theta_{k+1} = \underset{x}{\operatorname{argmax}} E[L(\theta_k, \theta)] \quad (16)$$

and usually makes steps to minimize the given objective function:

$$L^{clip}(\theta) = \hat{E}_t[\min r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}^t] \quad (17)$$

Here: θ is a policy parameter, $\hat{E}^t$ denotes the empirical expectation over timesteps, r_t is the ratio of the probability under the new and old policies, respectively, $\hat{A}_t$ is the estimated advantage at time t, and ε is the hyperparamter.

3.6 Deep Recurrent Q Network

Deep Q learning has shown remarkable success in training agents in video games, since it beat world champions at go. Q learning works for MDP environment (S,A,P,R) where an agent receives at a given timestep a state s_t and chooses action which determines s_{t+1}. The main objective of Q learning is to maximize long-term expected return of executing an action from a given state which are known as Q values, and these Q values are learned iteratively.

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'}(Q(s^{'}, a^{'}) - Q(s,a)) \quad (18)$$

For the large state space it is not feasible to estimate separate Q values, so instead of that neural networks parameterized by weights and biases collectively denoted as θ is used (Mnih et al. 2015) to approximate Q values.). Instead of updating individual Q-values, updates are now made to the parameters of the network to minimize a differentiable loss function.

$$L(s, a\theta_i) := Q(s,a) + \alpha(r + \gamma \max_{a'}(Q(s^{'}, a^{'}|\theta_i) - Q(s,a|\theta_i))^2 \quad (19)$$

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta L(\theta_i) \quad (20)$$

Due to same network is generating next state target Q-values that are used in updating its current Q-values, updates are very unstable. To make it stable two tricks are used: First, experiences are recorded in a replay memory and then sampled uniformly at training time. Secondly: separate target network $\hat{Q}$ is used where its parameters θ^- are updated to match θ every predefined iterations.

This type of learning is not effective in deciphering the underlying state of POMDP, so we are using a variation of Q network which Deep Recurrent Q network as defined in [39,40] which replaces the fully connected layer with the LSTM layer. Though we used a GRU layer for all of our networks. Instead of receiving single observation, the model received a history of observations and this history of

observations will be stored in experience replay buffer, and during training sample of histories will be used from replay buffer. First two layers of the network will be fully connected layer with 512 neurons and then the fully connected GRU layer and finally, a linear layer outputs a Q Value for each action.

4 Agent and Environment Formulation

4.1 Environment Formulation

As our student environment is setup according to the [2], we are formulating out problem similarly mentioned in the paper, so we have some benchmark results to compare to. Most of the work related to knowledge tracing has been formulated as POMDP (Partially Observable Markov Decision Processes). POMDP are extremely flexible framework but they have exponentially large state space making exploration a big issue. Solving POMDP using reinforcement learning is itself a challenge as all algorithms tend to assume the fully-observed case and is often solved by hand-crafting a solution.

We model our environment as POMDP with environment states S and set of actions A. In this agent only indirectly observes the underlying state of the MDP through the observations, and in principle it receives history of observation $h_t = (o_1, a_1, o_2, a_2, ...a_{t-1}, o_t)$ rather than a single observation. The ultimate goal of the agent is to learn the policy $\pi(h_t)$ which maps history of observations and learn policy with actions to maximize discounted rewards over time.

The state space S for the agent depends on the student model. For EFC, $S = R^{3n}$ encodes the item difficulty, + delay, and memory strength for n items. For HLR, $S = \theta(R_+ * X)^n$ encodes the model parameters, delay, and memory strength for n items. The action space A = [n] consists of n items that the agent can show to the student, where [n] = 1, 2, ..., n. Static hyper-parameters like student ability, item difficulty, and log-linear model coefficients are held constant, and dynamic quantities like number of attempts and correct answers are deterministically incremented. The reward function depends on the learning objective. If the student's goal is to maximize the expected number of items recalled, then [2]

$$R(s,.) = \sum_{i=1}^{n} P[Z_{i=1}|s] \tag{21}$$

To maximize the likelihood of recalling all items,

$$R(s,.) = \sum_{i=1}^{n} \log P[Z_{i=1}|s] \tag{22}$$

The student model behaves as an MDP environment and the agent doesn't have direct access to the student state space and it only receives observations $o \in O$ which are conditioned on the underlying state.The observation distribution $O(z|s,a) = P[Z_a = z|s]$ is given by the recall likelihood specified by the student model.

4.2 Agent Formulation

We implemented every algorithm mentioned in section3 of the paper from scratch and also compare the performance from the off-shelf implementation of garage RL lib[39] to verify the results. As mentioned earlier we tried to solve a POMDP environment that has an exponentially large state space so we have to use function approximate such as a neural network for estimating Q-values and Policy. To solve the large POMDP environment [39,40] have used recurrent networks, similarly, we will be using a gated recurrent neural network policy architecture. These types of networks have a high dimensional, continuous, representation of latent state and they preserve information for many time steps which have proven to be very helpful in solving the POMDP environment.

At each timestep, the policy takes an input of observation which is:

$$o := (i, t, r, s, \theta) \tag{23}$$

which means that the learner reviewed item i at time t and either recalled it $(r = 1)$ or forgot it $(r = 0)$, θ represents item difficulty, and s represents the student's strength of each item. During our experiment we found these sets of features appropriate as adding extra features such as several correct and incorrect attempts, was just making state-space larger and decreased the performance. The policy takes the history of these sequences and output the next item presented to the user. Strength is just the ratio of total attempts and correct answers for each question. Many would argue that student strength and item difficulty are latent parameters and should not be used in the agent's observation, but we found that using these drastically improved the performance, when compared to the features used in [2] such as item, recall and time.

To reduce the large feature space because of one-hot encoding, we used the compressed sensing trick as described in [2,4], where a random vector $n_{q,a} \quad N(0,I)$ is assigned to each input tuple. So each item representation $i_t = 0, 1$ depending on the exercise was recalled or not can be exactly encoded by assigning it to a fixed random Gaussian input vector of length $log2M$, where M is the total number of topics to learn, and delay t and strength is also encoded similarly with the outcome by taking a log.

5 Experiments and Results

The main idea behind these experiments is to find which reinforcement learning methods can be used for spaced repetition learning tasks and help students to obtain the learning objectives. We are modeling two student learning environments EFC and HLP and both of these environments are made using OPEN AI gym and we also created a custom environment wrapper to vectorize the observation so it can be used as input to policy network. We will be using two evaluation metric expected likelihood and log-likelihood. We will also be comparing the performance of each of the algorithms mentioned in section3 and will also compare their performance with baseline policies: Random, Threshold.

Apart from this as most of the inspiration of this project has been taken from [2] we will be comparing our model performance with their results.

Baseline: (1) Random policy always selects the next item for review randomly. (2) Threshold policy always selects items with predicted likelihood Z^* where it is selected to maximize the reward d using a uniformly-spaced hyper parameter sweep over the unit interval. This will serve as an upper-bound of our experiment as it has direct access to the latent parameters of the student simulator to compute recall likelihoods.

Implementation: Our implementation setup is also similar to [2], number of items n = 30 and number of steps per episode T = 200, where steps is number of times, the agent takes an action and present item to the student simulator, the constant delay is set to D = 5 seconds. We sample item difficulty θ from a log-normal distribution $\log\theta$ $N(\log 0.077, 1)$, and memory strength model parameters to be $\vec{\theta} = (1, 1, 0, \theta_3 N(0, 1))$.

For all the models, we set a batch size of 4000, with a discount rate γ = 0.99. Total 10 iterations were run for each experiment and average results were used to perform analysis. The recurrent neural network policy uses a hidden layer of 32 units, we experiment with more layers but didn't found any improvement. Other parameters respected to individual algorithms are:
TRPO: KL divergence limit = 0.01, learning rate for value function optimizer = .001, epochs = 100, optimizer = Conjugate Gradient optimizer
PPO: Clip Ration: 0.01, epochs = 100, optimizer = Adam
A2C: Value function coefficient = 0.25, entropy coefficient = 0.01, learning rate = 1e-3. Agent was trained for 500 episodes, optimizer = Adam
DRQN: Experience replay: 5000 trajectories of observations, Update Frequency of main model: 4 time steps, Update Frequency of Target Network: 100 time steps. For this network, the replay buffer was first initialized using trajectories of the Threshold baseline agent. The model was trained for 500 episodes, optimizer = Adam

Exponential Forgetting Curve Simulator Analysis
As most of our work is based on [2] we will be using their results as one of our metrics. Using the EFC as the student simulator we can see TRPO agent outperforms all the baseline and even other RL agents, achieving 35% better performance than the random agent and even surprisingly better than our threshold agent which was assumed to be our upper bound as the threshold has unfair access to the student simulator latent parameters. Comparing our performance with [2] which didn't use student strength and item difficulty as features we got an improvement of about **25%**, as in [2] TRPO agent performs worse than the threshold and only achieved 15% better performance than random.

Table 1 EFC Results

Agents	% better than Random: Reward Likelihood	% better than Random: Reward Log Likelihood
TRPO [2]	17 %	12 %
THRESHOLD	27 %	55 %
LEITNER [2]	-10 %	10 %
SuperMnemo [2]	17.5 %	42 %
TRPO*	35 %	30 %
PPO*	20 %	30 %
A2C*	5 %	-30 %
RDQN*	5 %	-40 %
PPO Tuned*	35 %	32 %

*This Paper

Our model also outperformed [2] for log likelihood by achieving **35%** better than random while [2] only achieved 11%, which shows that memory strength and item difficulty are rich features and it makes sense to incorporate them into the agent's observations

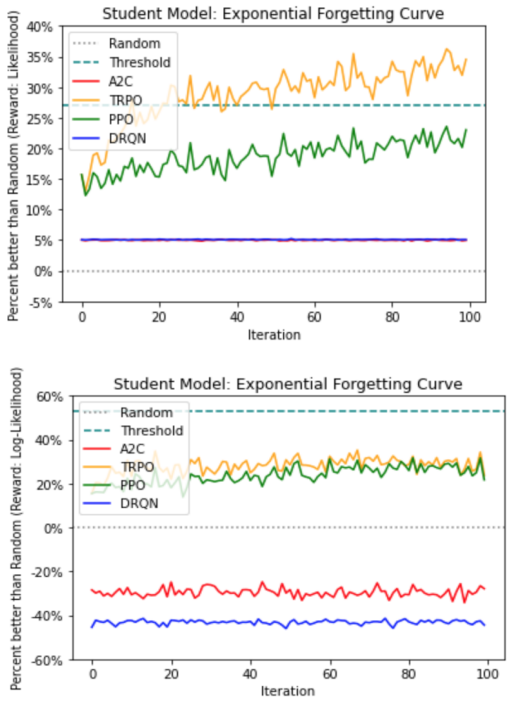

Figure 1: EFC simulator results

Performance of RDQN was expected and it didn't work well as usually in this type of task policy gradient works well while Q learning is best games and robotics environments. This performance could also be due to lack of training iterations as previously it has also shown that Q networks take a lot of time to converge and we only trained for 500 episodes. While we also found the importance of initializing the memory buffer with good trajectories helps to get better performance, as when replay buffer was first initialized with trajectories of random agent and then with the threshold agent, we saw the increase of **10%** in performance. During training, we experimented with different learning rate for

stable learning as by analyzing loss we saw the model was very unstable which was expected from the Q network and despite using modern techniques for making stable Q learning such as replay buffer and separate target network we didn't saw any improvement in performance.

One of the unexpected outcome from this experiment was for PPO, as we can see there is a big difference between the TRPO and PPO agent and we were expecting similar performance as recently it is one of the widely used algorithm analyzing further we looked at the training patterns for both of the algorithms, and we found out that entropy of TRPO model is decreasing at a very optimal rate and entropy of PPO model, was getting stuck, and when we trained for more epochs such as 500, it reached the similar entropy level as of TRPO model but average return got stuck around 69.5 - 70, which was the classical exploration issue as entropy was decreasing but rewards were not increasing which shows that PPO agent got stuck in some local optimal policy. To increase the exploration we increased the clip range to **0.3** to give the agent freedom so it can make a big jump from its old policy and explore new policy. Though in practice TRPO is more computationally expensive but while training it was the fastest agent to train as it took around 2 sec for each epoch while PPO with Adam optimizer took 1.5 minutes for each epoch. Conjugate gradient optimizer code for TRPO agent was taken from [38].

As we can see from figure 3, the average reward per episode reached around 79, while there was an expected drop in entropy, and simultaneously performance of PPO agent in the EFC environment is also improved as shown in Figure 4. So using hyperparameter tuning, better policy architecture, and better features we were able to outperform TRPO and threshold model described in [2] and in our paper by a significant margin.

For the A2C model, we experimented with various hyperparameters but we were not able to fine-tune and get any significant improvement in performance.

Half Life Regression Simulator Analysis:

Table 2 HRL Results

Agents	% better than Random: Reward Likelihood	% better than Random: Reward Log Likelihood
TRPO [2]	20 %	0 – 2 %
THRESHOLD	78 %	35 %
LEITNER [2]	-30 %	10 %
SuperMnemo [2]	0 %	22 %
TRPO*	30 %	60 %
PPO*	0 %	45 %
PPO Tuned*	30 %	55 %

*This Paper

For this experiment we decided not to test A2C and DRQN agents, as during our trial run we found both of them were performing worse compared to the random model, so for this simulator we only used PPO and TRPO model. The results of this experiment for the HLR simulator can be seen in figure 6. Both of the model were better than random for both objective

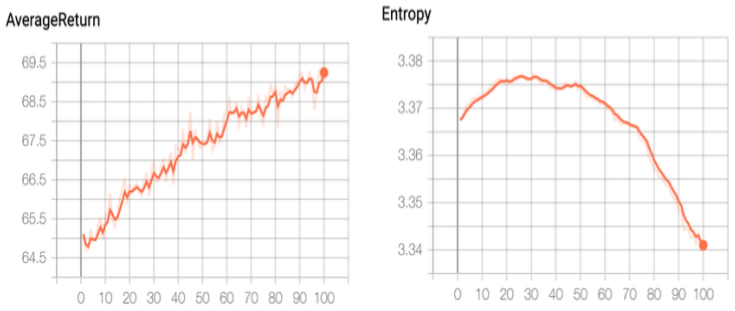

Figure 2: PPO Average Return and Entropy per episode

Figure 3: TRPO Average Return and Entropy per episode

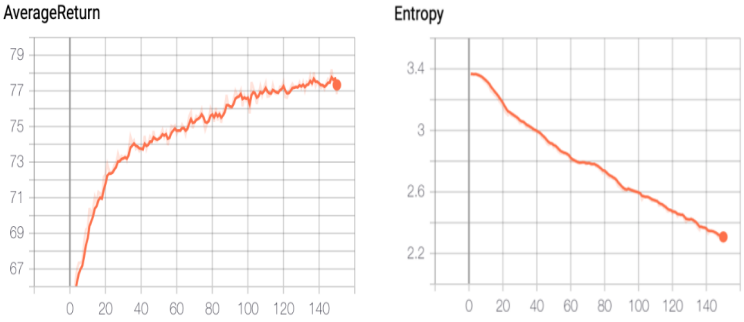

Figure 4: Tuned PPO Average Return and Entropy per episode

in maximizing expected recall and likelihood of recalling all items, and both these models outperform the THRESHOLD tutor on log likelihood objective function which is even better as the agent is presenting students topics in an optimal way such that they can remember most of the items, rather than the single item. As compared to [2] both of our agents were able to outperform their TRPO model by approximately **20%** and we kept the parameters similar to [2], the increase in performance is because of new features we used which again shows that introducing strength and item difficulty in the observation provides necessary information for an agent to choose next item.

Similarly we again analyzed the poor performance of

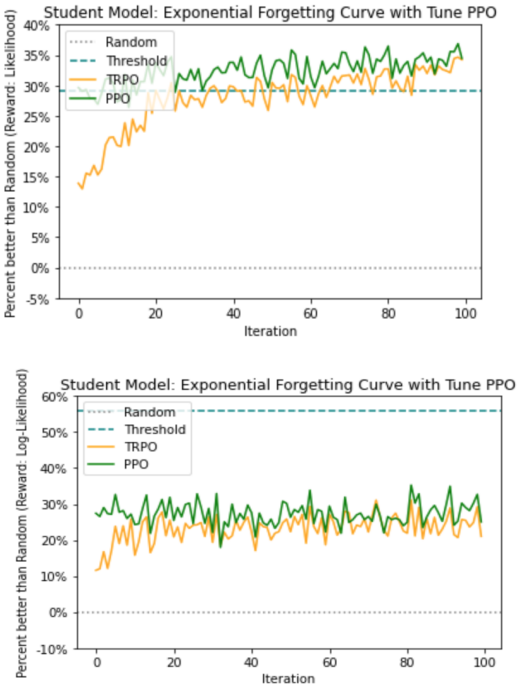

Figure 5: EFC simulator with Tuned PPO

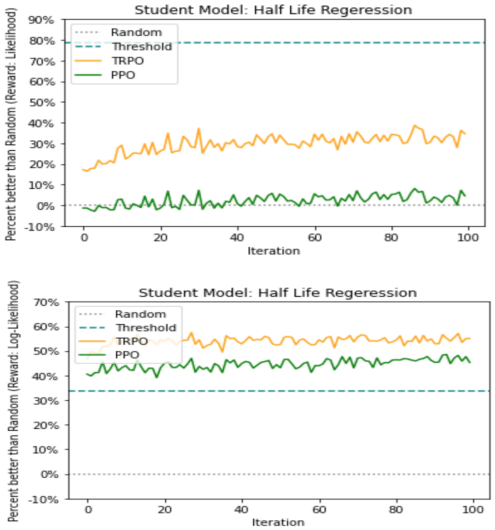

Figure 6: HRL simulator results

PPO model, compared to TRPO and the problem was again the exploration issue, we could see this time there was a significant decrease in entropy and rewards were not increasing which shows that policy was becoming deterministic very quickly and was not exploring new policies. This time we experimented with two different clip range 0.2 and 0.3, both of them were able to get good results the only difference we found that with clip range of 0.3 it took little longer to converge to optimal policy but for both of them there was a significant increase in performance, and we were able to improve the performance of PPO model from around **8%** to **40%** as shown in figure 5. Most surprising result for HLR model was to outperform THRESHOLD tutor for log-likelihood rewards as we didn't expect this to happen which shows with fine tuning RL model it can also beat complex heuristic schedulers like LEITNER and SUPERMNEMO as RL agents can in principle are capable of learning arbitrary teaching policies.

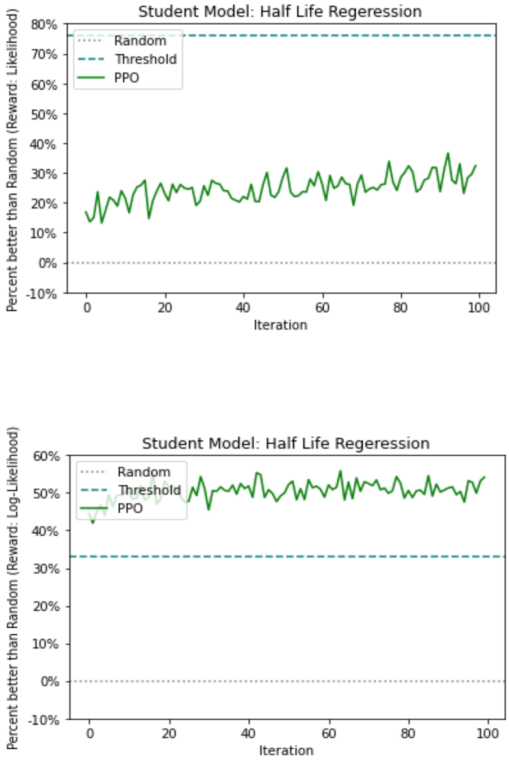

Figure 7: HRL simulator results with Tuned PPO

6 Conclusion and Discussion

In this paper, we demonstrated how various model-free approaches with neural network approximator can be extended to solve the problem of Spaced repetition teaching and also showed improvement over the prior work [2] by incorporating better features in the agent's observation, and it shows that RL agents were capable of extracting the latent structure of student memory. Both of our agents also performed better than THRESHOLD agent which had unfair access to all model parameters (and only had to fit the latent student knowledge variables). By combining the results from [2], we can also see that our Tutoring agent outperforms various heuristic scheduler algorithms such as Leitner [9] and SuperMemo [12]. The above results provide an interesting demonstration of the capacities of deep reinforcement learning methods on the knowledge tracing task. The advantage of these methods is that it can learn on their own but they require a large amount of training

data or interactions with the environment which makes them well suited to an online education environment, but not a small classroom environment. Further research could incorporate more features such as time taken to solve a question, and priority for learning an item, item text, images in the agent's observation these RL methods can even be extended to different knowledge tracing domain such as hint generation and choosing topic based on individual needs such as content which is predicted to be too easy or too hard can be skipped or delayed. Finally, most critical follow-up work is to make our student simulator more realistic by using data sets such as Duolingo [36] and then train the model policy through imitation learning from previously deployed scheduling algorithms, but before that, we have to figure out how to estimate latent parameters of the student model using these datasets.

7 References

[1] Behaviorism Wikipedia page. https://en.wikipedia.org/wiki/Behaviorism

[2] Siddharth Reddy, Sergey Levine, Anca Dragan. Accelerating Human Learning with Deep Reinforcement Learning

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

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

[5] Andrew S Lan and Richard G Baraniuk. A contextual bandits framework for personalized learning action selection. In EDM, pages 424–429, 2016.

[6] Jacob Whitehill and Javier Movellan. Approximately optimal teaching of approximately optimal learners. IEEE Transactions on Learning Technologies, 2017.

[7] Christopher James Piech. Uncovering Patterns in Student Work: Machine Learning to Under- stand Human Learning. PhD thesis, Stanford University, 2016.

[8] Travis Mandel, Yun-En Liu, Sergey Levine, Emma Brunskill, and Zoran Popovic. Offline policy evaluation across representations with applications to educational games. In Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, pages 1077–1084. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[9] Sebastian Leitner. So lernt man lernen. Herder, 1974.

[10] S. Reddy, I. Labutov, S. Banerjee, and T. Joachims. Unbounded human learning: Optimal scheduling for spaced repetition. In ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2016.

[11] Paul Pimsleur. A memory schedule. Modern Language Journal, pages 73–75, 1967.

[12] PA Wozniak and Edward J Gorzelanczyk. Optimization of repetition spacing in the practice of learning. Acta neurobiologiae experimentalis, 54:59–59, 1994.

[13] Damien Elmes. Anki. http://ankisrs.net, 2015.

[14] The mnemosyne project. http://mnemosyne-proj.org, 2006.

[15] Michael C Mozer and Robert V Lindsey. Predicting and improving memory retention: Psycho- logical theory matters in the big data era, 2016.

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

[17] Andrew S Lan, Christoph Studer, and Richard G Baraniuk. Time-varying learning and content analytics via sparse factor analysis. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 452–461. ACM, 2014.

[18] Chaitanya Ekanadham and Yan Karklin. T-skirt: Online estimation of student proficiency in an adaptive learning system. Machine Learning for Education Workshop at ICML, 2015.

[19] Kevin H Wilson, Yan Karklin, Bojian Han, and Chaitanya Ekanadham. Back to the basics: Bayesian extensions of irt outperform neural networks for proficiency estimation. arXiv preprint arXiv:1604.02336, 2016.

[20] Xiaojin Zhu. Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In AAAI, pages 4083–4087, 2015.

[21] Kaustubh R Patil, Xiaojin Zhu, Łukasz Kopec', and Bradley C Love. Optimal teaching for limited-capacity human learners. In Advances in neural information processing systems, pages 2465–2473, 2014.

[22] Adish Singla, Ilija Bogunovic, Gabor Bartok, Amin Karbasi, and Andreas Krause. Near- optimally teaching the crowd to classify. In ICML, pages 154–162, 2014.

[23] Spaced repetition Wikipedia page. https://en.wikipedia.org/wiki/Spaced_repetition

[24] Hermann Ebbinghaus. Memory: A contribution to experimental psychology. Number 3. University Microfilms, 1913.

[25] Leslie Pack Kaelbling, Michael L. Littman, Anthony R. Cassandra. Planning and acting in partially observable stochastic domains

[26] Emma Brunskill and Stuart Russel. Partially Observable Sequential Decision Making for Problem Selection in an Intelligent Tutoring System

[27] Jeremiah T. Folsom-Kovarik, Sae Schatz, Denise Nicholson. Plan Ahead: Pricing ITS Learner Models

[28] Theocharous, G., Bechwith, R., Butko, N., and Philipose, M. 2009. Tractable POMDP planning algorithms for optimal teaching in "SPAIS." In Proceedings of the 21st International Joint Conferences on Artificial Intelligence Workshop on Plan, Activity, and Intent Recognition, C. Boutilier, Ed., AAAI Press, Menlo Park, CA.

[29] Lebiere, C., Anderson, J. R. (1993). A connectionist

Implementation of the ACT-R production system. In Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society (pp. 635–640). Mahwah, NJ: Lawrence Erlbaum Associates.
[30] Joshua B. Tenenbaum. Bayesian modeling of human concept learning
[31] Anna N. Rafferty, Emma Brunskill, Thomas L. Griffiths, Patrick Shaftod. Faster Teaching via POMDP Planning
[32] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pages 1889–1897, 2015.
[33] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.
[34] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In International Conference on Machine Learning, pages 1329–1338, 2016.
[35] Volodymyr Mnih1, Adria Puigdomenech Badia1 Mehdi Mirza1, Alex Graves1, Tim Harley1, Timothy P. Lillicrap1, David Silver1, Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning
[36] B. Settles, and B. Meeder 2016. A Trainable Spaced Repetition Model for Language Learning. In Proceedings of the Association for Computational Linguistics (ACL) (pp. 1848–1858). ACL.
[37] John Schulman and Filip Wolski and Prafulla Dhariwal and Alec Radford and Oleg Klimov (2017). Proximal Policy Optimization AlgorithmsCoRR, abs/1707.06347.
[38] The garage contributors. (2019). Garage: A toolkit for reproducible reinforcement learning research. https://github.com/rlworkgroup/garage.
[39] Nicolas Heess and Jonathan J. Hunt and Timothy P. Lillicrap and David Silver (2015). Memory-based control with recurrent neural networksCoRR, abs/1512.04455.
[40] Matthew Hausknecht, Peter Stone (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. In AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15).