DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Using deep reinforcement learning for personalizing review sessions on e-learning platforms with spaced repetition

SUGANDH SINHA

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Using deep reinforcement learning for personalizing review sessions on e-learning platforms with spaced repetition

Sugandh Sinha

Master in Computer Science
June 10, 2019
Supervisor at KTH: Pawel Herman
Examiner: Örjan Ekeberg
Principal: Sana Labs AB
Supervisor at Sana Labs: Anton Osika
School of Computer Science and Electrical Engineering,
KTH Royal Institute of Technology

Acknowledgements

I would like to thank both my supervisors, Anton Osika at Sana Labs and Pawel Herman at KTH, for providing constant guidance and support during the entire duration of my thesis work. I would also like to take this opportunity to thank Ann Bengtsson for helping me through the administrative work and providing guidance before starting the thesis. I am also thankful to Örjan Ekeberg for choosing to examine my thesis.

I would like to thank my brother and my family members for their support. Lastly, I would also like to thank my friends and the team at Sana labs for a fun environment and engaging discussions during the course of my thesis.

Abstract

Spaced repetition is a learning technique in which content to be learned or memorized is reviewed multiple times with gaps in between for efficient memorization and practice of skills. Two of the most common systems used for providing spaced repetition on e-learning platforms are Leitner and SuperMemo systems. Previous work has demonstrated that deep reinforcement learning (DRL) is able to give performance comparable to traditional benchmarks such as Leitner and SuperMemo in a flashcard based setting with simulated learning behaviour. In this work, our main contribution has been introduction of two new reward functions to be used by the DRL agent. The first, is a realistically observable reward function that uses the average of sum of outcomes in a sample of exercises. The second uses a Long Short Term Memory (LSTM) network as a form of reward shaping to predict the rewards to be used by DRL agent. Our results indicate that in both cases, DRL performs well. But, when LSTM based reward function is used, the DRL agent learns good policy smoother and faster. Also, the quality of the student-tutor interaction data used to train the LSTM network displays an effect on the performance of the DRL agent.

Sammanfattning

Spaced repetition är en inlärningsteknik där innehåll som ska memoriseras upprepas med mellanrum flera gånger för att minnets styrka ska öka. Två av de vanligaste algoritmerna som används för att ge spaced repetition på digitala utbildningsplattformar är Leitner och SuperMemo. Tidigare arbete har visat att Deep Reinforcement Learning (DRL) för schemaläggning av spaced repetition kan ge inlärning likt traditionella algoritmer i en flashcard-baserad simulering av lärande studenter. I detta arbete är vårt huvudsakliga bidrag att introducera två nya belöningsfunktioner som används av DRL-agenten. Den första är en realistisk observerbar belöningsfunktion som använder medelvärdet av summan av resultat i ett prov av övningar. Den andra använder ett återkopplat neuralt nätverk (LSTM) som en form av reward-shaping för att räkna ut de belöningar som DRL-agenten ska belönas med. Våra resultat visar att DRL i fungerar bra i båda fallen. När LSTM-baserad belöningsfunktion används lär sig DRL-agenten en bra policy snabbare. Resultaten visar också att kvaliteten på student-interaktionsdata som används för att träna LSTM-nätverket har en stor effekt på DRL-agentens prestanda.

Contents

List of Abbreviations

CAI	Computer Aided Instruction
CAL	Computer Aided Learning
CBI	Computer Based Instruction
CBT	Computer Based Training
DRL	Deep Reinforcement Learning
EFC	Exponential Forgetting Curve
FIM	Fisher Information Matrix
GPL	Generative Power Law
GRU	Gated Recurrent Unit
HLR	Half Life Regression
LSTM	Long Short Term Memory
MDP	Markov Decision Process
MOOCs	Massive Open Online Courses
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RNN	Recurrent Neural Network
TRPO	Trust Regional Policy Optimization
TNPG	Truncated Natural Policy Gradient

Chapter 1

Introduction

Students often find themselves learning or memorizing information in a limited time span. Some students tend to learn a lot of information in a very short amount of time, sometimes even overnight right before the tests. Learning in this manner is known as *massed learning* or *cramming*. Psychological studies have shown that this technique is not very effective for long-term retention of learned or memorized materials[8].

Ebbinghaus [20] first demonstrated how the learned information is forgotten over time when the learned material is not practiced. He showed his findings with a graph, now, more commonly known as forgetting curve, figure 1.1. In 1880 [19], Ebbinghaus first expressed the forgetting curve using the power function given below:

$$x = [1 - (\frac{2}{t})^{0.099}]^{0.51}$$

where x is percent retained and t is time since original learning (in min).

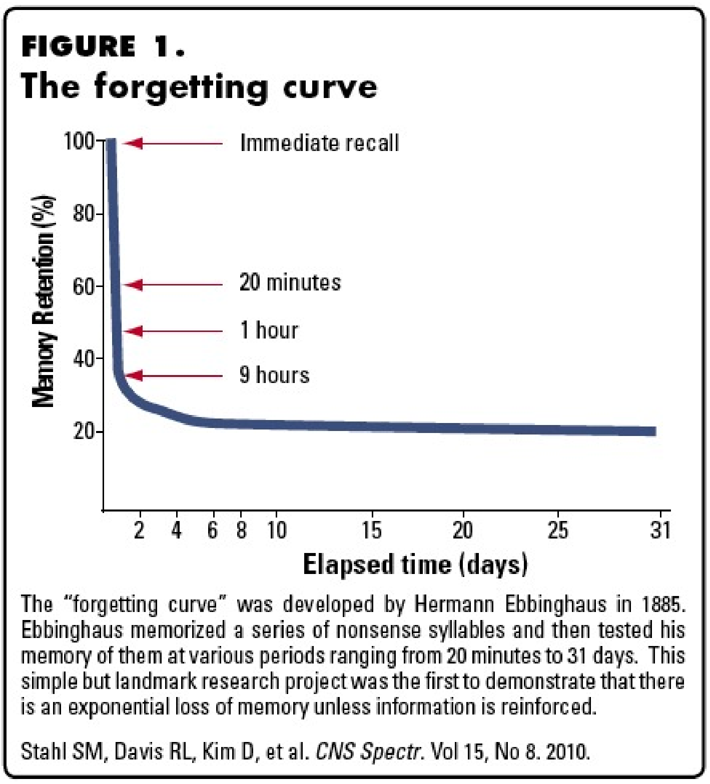
But in his work in 1885 [20], Ebbinghaus used a logarithmic function to denote the forgetting curve:

$$b = \frac{100k}{(logt)c + k}$$

where b is percent retained, t is the time since original learning, and c and k are constants with $k = 1.84$ and $c = 1.25$.

There have also been other attempts made to find a better mathematical approximation for the forgetting curve such as Heller et. al.[24] and Wickelgren [58].

Spaced repetition[34] is another learning approach in which content to be learned or memorized is reviewed multiple times with gaps in between for



The "forgetting curve" was developed by Hermann Ebbinghaus in 1885. Ebbinghaus memorized a series of nonsense syllables and then tested his memory of them at various periods ranging from 20 minutes to 31 days. This simple but landmark research project was the first to demonstrate that there is an exponential loss of memory unless information is reinforced.

Stahl SM, Davis RL, Kim D, et al. *CNS Spectr.* Vol 15, No 8. 2010.

Figure 1.1: *Forgetting Curve. Image from Stahl et.al., Play it Again: The Master Psychopharmacology Program as an Example of Interval Learning in Bite-Sized Portions. In CNS Spectr. 2010 Aug;15(8):491-504.* [21]

efficient memorization and practice. Research since the 19th century has shown that spacing repetition of study material with delays between reviews has a positive impact on the duration over which the material can be recalled [20, 27]. Different ways for performing this spacing have been argued for and investigated - the preferable method for deciding on the spacing scheme is, however, to empirically learn the optimal spacing scheme such as the framework proposed by Novikoff et. al. [40].

This could be done in an e-learning environment where an agent/model is in charge of scheduling and presenting materials to students for memorizing. Since every student might possess a different learning capability, the agent/model should preferably be able to infer the learning pattern for each student to be able to present learning material effectively. The term 'effectively' is used to denote how many times and when the material should be presented again, as spaced practice has shown to reduce this number. Most of the previous work in the knowledge tracing domain has been about predicting whether the practice exercise that the agent/model is going to present will be answered correctly or not by the student such as [42]. However, how to make decision about what exercises to present to a student and when to do it has received less attention. Such a decision making problem needs a different approach, which we will attempt to solve here with the technique of DRL.

As an example, one of the most commonly used methods in flashcards for spaced repetition is Leitner System [31]. The basic idea behind this technique is to make users interact more with items that they are likely to forget and let them spend less time on items that they are able to recall efficiently.

1.1 Motivation and problem description

The growing popularity of e-learning websites and mobile applications have made it possible for users to learn at their own convenience. Recommending content and personalizing the learning sessions is a highly sought after tasks on these platforms. This work explores opportunities for replacing the manually selected criteria for which exercise to show and when to show it to users with an end-to-end recommendation system that learns the criteria by itself.

In previous work, Reddy et al. [45] investigated a model-free review scheduling algorithm for spaced repetition systems, which learns its policy using the observations made on student's study history without actually learning a student model. Reddy et. al. used deep reinforcement learning (DRL) for performing spaced repetition. The reward function used in their work was dependent on the 'updated' student state which is the new state of student, s_{t+1}, when the agent takes an action on student with state s_t at timestep t. This work is a deeper exploration of that work and also an extension of it by using different reward functions which were independent of these 'updated' student states. We focused on finding out how efficient a reinforcement learning algorithm can be as compared to previously published heuristics like Leitner[31] and SuperMemo[5] when it comes to making spaced repetition and making students learn by using different reward functions.

1.1.1 Research Question & Objectives

We used three student learning models - Exponential Forgetting Curve (EFC), Half Life Regression (HLR) and Generalized Power Law (GPL); and four baseline policies: RANDOM, LEITNER, SUPERMEMO, and THRESHOLD. The performance of the DRL algorithm is measured using two metrics: expected recall likelihood and log-likelihood.

The research questions that this project seeks to answer are:

- *How does DRL perform compared to other baseline policies when we replace the reward from being the exact probability of recalling each item, used by Reddy et al. [45], to a realistically observable reward such as the average of the sum of correct outcomes in a sample of exercises?*
- *What is the effect of replacing the same reward function with an RNN model that predicts the reward (reward shaping)?* We used Long Short Term Memory (LSTM), a kind of recurrent neural network (RNN), which has been shown to achieve good results for sequential prediction tasks [42].

To answer the above questions, student models were allowed to interact with the recommendations of the DRL tutor and its teaching performance was compared to the baseline policies on the two performance metrics mentioned earlier.

1.1.2 Scope

We did not test our reinforcement learning (RL) agent in real world environment with students due to the time constraints of this project. Also, the student simulators do not model all the characteristics of a typical student. The parameters of the student simulators were not derived from the distribution of real student data but were set to reasonable values based on previous published works [45], [44]. For reward shaping experiments and experiments to compare the performances of Trust Region Policy Optimization algorithm (TRPO) and Truncated Natural Policy Gradient algorithm (TNPG), only one student model was used because of the time limitation considering the computation time.

1.2 Thesis Outline

Chapter 2 (Background) starts out by giving an overview of the field history and then defines the related theoretical concepts that are needed in order to get a better understanding of this work. The chapter finally, lists out some relevant work that has already been done in this problem domain or is related to the problem that we are trying to solve.
Chapter 3 (Method and Experiments) gives detail about the dataset, the experimental set up including parameter settings and model architecture.
Chapter 4 (Results) presents the results of the experiments.
Chapter 5 (Discussion and Future Work) discusses the implications of the results and also discusses ways in which this work could be extended. It also lists possible implications of this work from the point of view of sustainability and ethics.
Finally, **Chapter 6 (Conclusion)** sums up the findings of this work.

Chapter 2

Background and Related Work

2.1 History

2.1.1 Reinforcement Learning

Over the years, the area of machine learning has made some rapid progress and has become increasingly popular. Now, with the advent of deep learning, it has become the focus of research when it comes to the field of Artificial Intelligence, as artificial neural networks try to mimic the activity of the neurons in the human brain. Even though this idea is very old and can be traced back to as early as 1943 in one of the seminal papers of McCulloch and Pitts [36], it only recently became feasible to implement these complex networks on large datasets because of the advancements in hardware. Since its inception, deep learning has now found its way to almost every area in machine learning from image classification to drug discovery.

In a broader sense, we can consider learning algorithms to fall into three main categories based on the feedback that they receive from the world. On one extreme, there is supervised learning in which the algorithms are presented with a target value that they check after each iteration to set the value of hyperparameters to an optimum value such that the error is minimized. On the other extreme, there is unsupervised algorithms where no feedback is provided by the environment to the learning algorithms and the algorithm has to figure out the parameters using the patterns found in the features that are provided as input. Between these two extremes lies RL.

One of the most notable characteristics of humans and animals has always been their ability to learn from their experience by interacting with their environment. Interacting with the environment provides a great deal of information such as cause and effect, actions needed to achieve a goal and ramifications of an action. Most of our actions are taken by keeping into account the goal that we are trying to achieve and the kind of responses that we expect to receive when we take certain actions. RL is one such approach in the paradigm of learning systems which focuses on goal directed learning as compared to other machine learning algorithms [55].

RL is fast becoming another hot topic of research. One of the major reason for this could be attributed to the pioneer work of Silver et. al on Alpha Go Zero [51]. The team behind it started off with AlphaGo [50], the first program ever to defeat the world champion in the game of Go. Now, they have gone a step further than that in the sense that AlphaGo Zero does not even need the training data from thousands of games to learn to play. Instead, it learns by playing against itself and is much faster in learning to become an expert.

One of the earliest work which could be said to have formed the basis of RL is the work by psychologist Edward L. Thorndike in which he developed his law of effect [56]. This principle states that the responses which lead to pleasant outcomes are more likely to occur in similar situations while the responses that create unpleasant outcomes are less likely to occur.

One of the most challenging aspect of RL is the huge amount of data and computational power it needs, as well as the reproducability of research results [25]. So, currently, RL mostly finds its applications in non-mission critical applications of sequential decision making problems. Despite all these challenges RL is starting to make its impact and is now being used in robotics, recommending online content, in the field of medicine for optimal medication dosing [43] and recommending use of medical tools [39]; it is even being used in tuning neural networks [6], [12].

2.1.2 Intelligent Tutoring Systems

Although human tutoring has always been the de facto standard when it comes to teaching, over the past few years e-learning has become massively popular because of their ease of access that allows users to learn anywhere at anytime. For e.g. - Massive Open Online Courses (MOOCs) like Coursera, EdX, etc. and learning applications like Duolingo, Quizzlet, etc. At first, most of the learning systems just informed users of the wrong answers without considering anything of the learning process (e.g.,[4],[54]). These first generation systems were called CAI, short for computer-assisted instruction tutors. These traditional systems sometimes are also referred to as Computer-Based Instruction (CBI), Computer Aided Learning (CAL), and Computer-Based Training (CBT). Soon thereafter, researchers started to look into systems with the intent of getting more 'intelligent' approaches to learning that would oversee the learning process(e.g.,[13], [22], [52]).

As described by Shute [49], "A system must behave intelligently, not actually be intelligent, like a human". These systems were the second generation

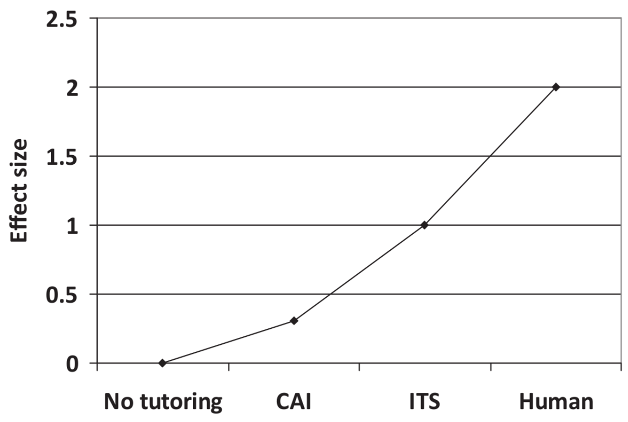


Figure 2.1: Common belief about effect sizes of types of tutoring. Image from Kurt Vanlehn, The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. in Educational Psychologist, vol. 46, p 197–221 [57]

systems and were called ITS (Intelligent Tutoring systems).

According to VanLehn[57], CAI tutors are considered to increase examination scores of students by 0.3 standard deviations over usual levels. ITSs are believed to be more effective, boosting test performance by about 1 standard deviation. Human tutors are the most effective of all, raising test scores by 2 standard deviations. Fig. 2.1 shows this trend among different types of tutoring.

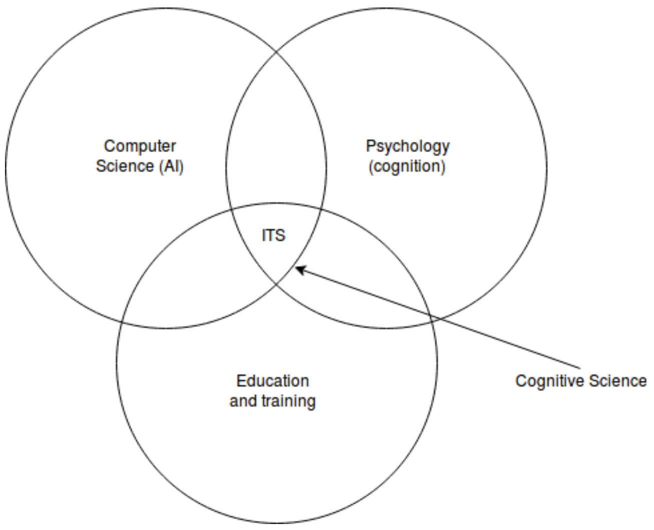


Figure 2.2: ITS Domains. Image adapted from Hyacinth S. Nwana , Intelligent Tutoring Systems: an overview in Artificial Intelligence Review, vol. 4, p 251-277 [41]

Research in ITS is a very challenging task as it is an interdisciplinary field combining AI, cognitive psychology and educational theory. Fig. 2.2 captures the relation between various fields and ITS. Because of this interdisciplinary nature, the research goals, terminology, theoretical frameworks, and emphases amongst researchers vary a lot. Although ITS have not been widely adapted yet but the research in ITS combined with machine learning is bound to grow even more.

2.2 Related Theory

2.2.1 Models of human memory

2.2.1.1 Exponential Forgetting Curve:

Ebbinghaus's [20]classic study on forgetting learned materials states that when a person learns something new, most of it is forgotten in an exponential rate within the first couple of days and after that the rate of loss gradually becomes weaker.
Reddy et al. [44] give the probability of recalling an item as:

$$P[recall] = exp(-\theta \cdot d/s), \tag{2.1}$$

where θ is the item difficulty, d is the time elapsed since the material was last reviewed and s is the memory strength.

2.2.1.2 Half life regression:

As described by Settles and Meeder [47], the memory decays exponentially over time:

$$p = 2^{-\Delta/h}, \tag{2.2}$$

In this equation, p denotes the probability of correctly recalling an item (e.g., a word), which is a function of Δ, the lag time since the item was last practiced, and h, the half-life or measure of strength in the learner's long-term memory.
When $\Delta = h$, the lag time is equal to the half-life, so $p = 2^{-1} = 0.5$, and the student is on the verge of being unable to remember. In this work, we have made an assumption that the responses could be only binary i.e. correct or incorrect.
Assuming that the half-life should increase exponentially with each repeated exposure. The estimated half life $\hat{h}_\Theta$ is given by

$$\hat{h}_\Theta = 2^{\Theta \cdot x}, \tag{2.3}$$

where x is a feature vector that describes the study history for the student-item pair and the vector Θ contain weights that correspond to each feature variable in x.

2.2.1.3 Generalized power law:

Wixted and Carpenter [61] state that the probability of recalling decays according to a generalized power law as a function of t:

$$P[recall] = \lambda(1 + \beta.t)^{-\Psi}, \tag{2.4}$$

where t is the retention interval, λ is a constant representing the degree of initial learning, β is a scaling factor on time $(h > 0)$ and Ψ represents the rate of forgetting.

DASH model [37] is a special case of GPL and is an acronym summarizing the three factors (difficulty, ability,and study history).

2.2.2 Spaced Repetition

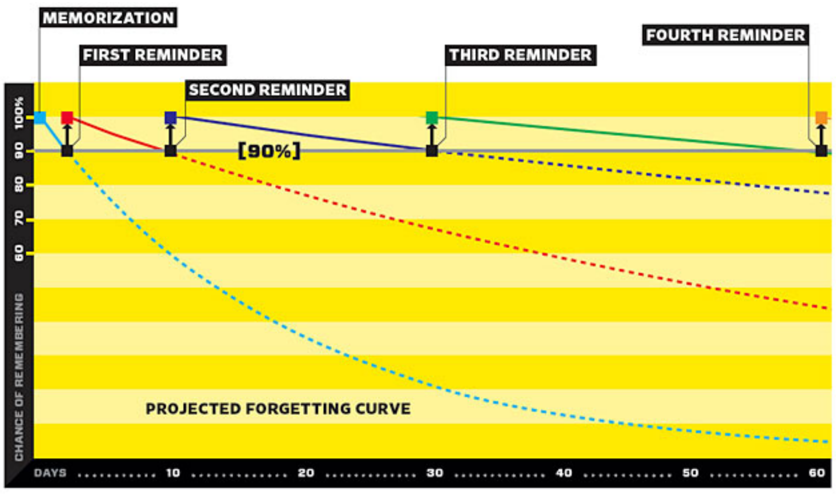


Figure 2.3: Graphical representation of spaced repetition. Image from https://www.supermemo.com/en/blog/did-ebbinghaus-invent-spaced-repetition, accessed on June 20, 2018 [1]

Kang [27] states that having the initial study and subsequent review or practice be spaced out over time generally leads to superior learning than having the repetition(s) occur in close temporal succession (with total study time kept equal in both cases). This phenomenon is called the spacing effect and this technique is referred to as spaced repetition. Fig. 2.3 shows how spaced repetition helps in memorizing materials.

2.2.3 Leitner system

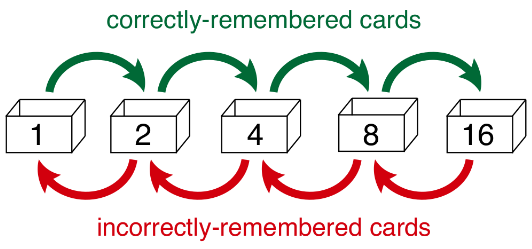


Figure 2.4: Leitner System. Image from Settles and Meeder, A Trainable Spaced Repetition Model for Language Learning in ACL 2016 [47]

Leitner System [31] is one of the most widely used method used in flash-cards for spaced repetition. The basic idea behind this technique is to make users interact more with items that they are likely to forget and let them spend less time on items that they are able to recall efficiently. Fig. 2.4 shows the working of Leitner system. This system manages a set of n decks. When the user sees an item for the first time then that item is placed in deck 1. Afterwards, when the user sees an item placed in deck i and recalls it correctly, the item is moved to the bottom of deck i+1. However, if the user answers incorrectly then it is moved to the bottom of deck i-1. The goal of the Leitner system is to make user spend more time on lower decks which contains the items that the user is not able to recall efficiently.

[44] uses a Queue based Leitner system to generate a mathematical model for spaced repetition system.

2.2.4 Supermemo System

There have been different versions of SuperMemo or SM algorithms. The first computer based SM algorithm was SM 2 and since then there have been multiple revisions. The SM 2 algorithm as described by Wozniak [5] is described as below:

1. Split the knowledge into smallest possible items.
2. With all items associate an E-Factor equal to 2.5.
3. Repeat items using the following intervals:
   I(1):=1
   I(2):=6
   for n>2: I(n):=I(n-1)*EF
   where:
   I(n) - inter-repetition interval after the n-th repetition (in days),
   EF - E-Factor of a given item which is easiness factor reflecting the

easiness of memorizing and retaining a given item in memory
If interval is a fraction, round it up to the nearest integer.

4. After each repetition assess the quality of repetition response in 0-5 grade scale:
   5 - perfect response
   4 - correct response after a hesitation
   3 - correct response recalled with serious difficulty
   2 - incorrect response; where the correct one seemed easy to recall
   1 - incorrect response; the correct one remembered
   0 - complete blackout.

5. After each repetition modify the E-Factor of the recently repeated item according to the formula:
   EF':=EF+(0.1-(5-q)*(0.08+(5-q)*0.02))
   where:
   EF' - new value of the E-Factor,
   EF - old value of the E-Factor,
   q - quality of the response in the 0-5 grade scale.
   If EF is less than 1.3 then let EF be 1.3.

6. If the quality response was lower than 3 then start repetitions for the item from the beginning without changing the E-Factor (i.e. use intervals I(1), I(2) etc. as if the item was memorized anew).

7. After each repetition session of a given day repeat again all items that scored below four in the quality assessment. Continue the repetitions until all of these items score at least four.

Since we are only considering binary responses in this work, the Supermemo algorithm has been modified to use a binary grade scale, i.e 0 for an incorrect response and 1 for a correct response.

2.2.5 Intelligent Tutoring Systems

ITS, short for Intelligent Tutoring Systems are systems that have been developed with the intention to teach students. As mentioned in [41], the architectures of ITS can vary a lot. But earlier studies identified 3 core modules present in all ITS [7], [9]

- The expert knowledge module.
- The student model module.
- The tutoring module.

But, more recent studies made researchers to agree on a fourth module as well.[60], [35], [10]

- The user interface module.

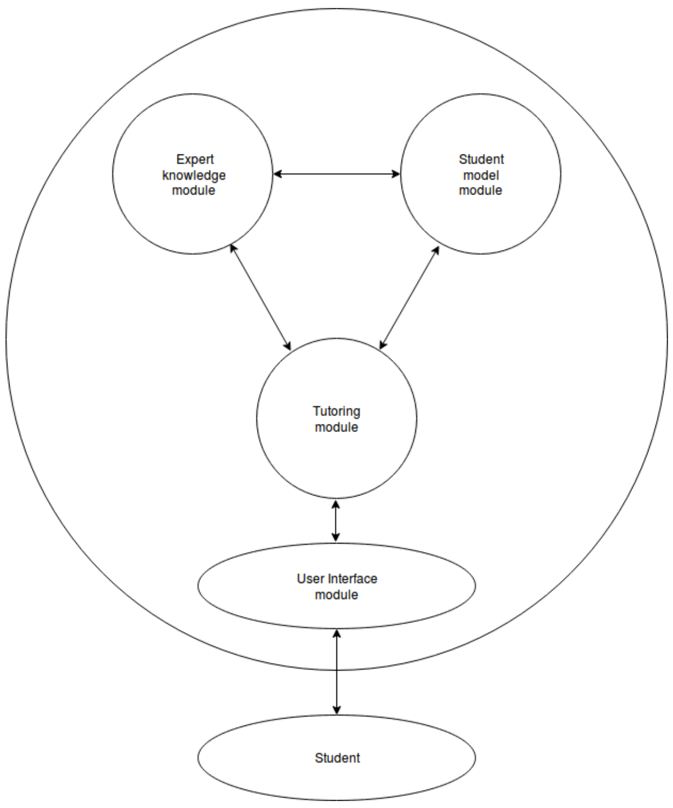


Figure 2.5: *General ITS Architecture. Image adapted from Hyacinth S. Nwana , Intelligent Tutoring Systems: an overview in Artificial Intelligence Review, vol. 4, p 251-277 [41]*

Fig. 2.5 shows the general ITS architecture. The **expert module** represents the source of the knowledge which is taught to the students. This module should be able to generate questions, answers and sometimes, even the steps to solve a problem.

The **student model** module refers to the ever-changing representation of the skills and knowledge that the student possess. In order to adapt the tutor module to the respective needs of student, it is important that the tutor module understands the current skills and knowledge level of student.

The **tutor module**, sometimes also called as the teaching strategy, guides the student through the learning process. This module is responsible for which lesson to present to the student and when to present it.

The **user interface module** is a bidirectional communication channel through which the student and the system interact. How the information is presented to the student can easily affect the effectiveness of the entire system.

2.2.6 Relation between Tutoring Systems and Student learning

Capturing a student's knowledge level is a very challenging task but at the same time it is also an imperative task that needs to be done when implementing an intelligent tutoring system. In order to tailor a good set of recommended questions to student, a system should be able to determine the current level of student's knowledge and also, should be able to predict the likelihood of the student answering the questions presented by the system correctly and while doing so, this process also modifies the level of student's knowledge. The system presents a question to the system and based on the student's answer receives a corrective feedback. Duration and the manner in which the material was learned is another factor that highly affects the estimation of student's current level of knowledge [33]. Individual differences among students also vary which is yet another factor that supports the reason for estimating the knowledge level of students.

2.2.7 Reinforcement Learning

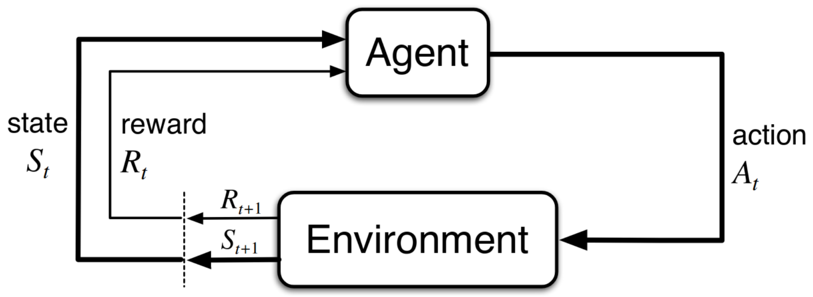


Figure 2.6: The agent–environment interaction in a Markov decision process. Image from Richard S. Sutton and Andrew G. Barto, 1998, Introduction to Reinforcement Learning (1st ed.), MIT Press, Cambridge, MA, USA. [55]

As described by Sutton and Barto [55], RL can be described as a Markov Decision Process consisting of:

- A set of states $\mathcal{S}$.
- A set of actions $\mathcal{A}$.

- A set of state-transition probability distribution, $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$,

$$p(s' \mid s, a) \doteq Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a)$$

- An immediate reward function that gives either the expected rewards for state–action pairs as a two argument function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r \mid s, a),$$

  or the expected rewards for state-action-next-state triples as a three-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$,

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in R} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}.$$

- A discount factor, $\gamma \in (0, 1)$.
- Sometimes, a distribution of initial state s_0 is also given $\rho_0 : \mathcal{S} \rightarrow \mathbb{R}$.

2.2.7.1 Elements of Reinforcement Learning

Agent and Environment:
An agent is an entity that interacts with its environment to achieve its goal by learning and taking appropriate actions. Everything surrounding the agent comprises the environment. The agent must be able to sense the state of the environment in order to take appropriate action which would affect the state of the environment. At each time step, the agent interacts with the environment. Fig. 2.6 shows the process of interaction between an agent and its environment in an MDP. MDPs usually include three aspects - sensation, actions and goals. Because of limiting itself to just these three aspects, MDPs may not be sufficient to solve all decision-learning problems.

Policy:
A policy defines how the agent will act in a particular state of the environment. Policies may be stochastic. It is simply a mapping from states to probabilities of selecting each possible action. It can be viewed as stimulus-response rules in biological systems.

If the agent is following policy π at time t, then $\pi(a \mid s)$ is the probability that $A_t = a$ if $S_t = s$.

Reward signal:

A reward is a number which is sent to the agent from the environment after each time step. The goal of the agent in a RL problem is to maximize the total reward accrued over time. The reward signal determines whether an event is good or bad for the agent. If the reward received by the agent after performing an action selected by the policy was a low reward then the policy may be changed.

Episodes and Returns:
If an agent-environment interaction can be naturally broken into subsequences then we call these subsquences as episodes and such tasks are called episodic tasks. Each episode ends in a special state called the terminal state. If the agent-environment interaction can not be naturally broken into episodes then these tasks are called continuing tasks.

The expected return, G_t, is defined as the sum of rewards. If $R_{t+1}, R_{t+2}, R_{t+3}, ...,$ denotes the sequence of rewards obtained after time step t then expected return, G_t, is given by:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_t.$$

When discounting is used, the agent select actions to maximize the sum of the discounted rewards it receives over the future.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate.

Calculating return for continuing tasks is challenging because the final time step in this case would be T = ∞, and the return could become infinite as well.

Value function:
A value of a state is defined as the total reward that is accrued by the agent when starting from that particular state and the states that follow. The value function simply determines the goodness of a state for an agent. In contrast to reward signal, which defines the desirability of an action in an immediate context, the value function describes the desirability of an action on a long term basis. For instance, an action might give a very low immediate reward but in the long run, it may produce a high value.
The value of a state s under a policy π, denoted by $v_\pi(s)$, is the expected return when starting in s and following π thereafter. For MDPs, v_π is given

by

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \forall s \in S$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. The value of the terminal state is always zero. v_π is the state-value function for policy π.

Similarly, the value of taking action a in state s under a policy π, denoted $q_\pi(s, a)$, as the expected return starting from s, taking the action a, and thereafter following policy π is given by:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right].$$

q_π is the action-value function for policy π.

Model of the environment (optional):
A model of the environment imitates the environment and its behaviour in which the agent is going to perform. It is optional as there are some RL algorithms that are model free and use trial and error methods because it is not always possible to build a model of the environment.

2.2.7.2 Rewards, Returns and Value functions

Rewards are essential component of RL as without it, it is impossible to estimate value. After every time step the action selected by the agent should be the action with the highest value not the highest reward. Also, it is easy to estimate rewards as they are given directly by the environment while estimating value is a very challenging task because they are calculated based on the actions that the agent takes and then re-calculated again after each time step.
While return gives the expected discounted sum of rewards for one episode, a value function gives the expected discounted sum of rewards from a certain state.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

2.2.7.3 Exploration/Exploitation trade-off

One of the key challenges in RL is keeping a balance between letting the agent use an action that it has used previously and has found it to be effective (exploitation) versus using a new randomly selected action and determine its effectiveness in that particular situation (exploration).

An agent that only performs exploitation can be thought of as using Greedy algorithm and it selects action with the highest value. In this case the action is selected based on the following expression,

$$a_t^* = argmax_{a \in A} Q_t(a),$$

where a^* is the selected action at time step t , $argmax_a$ denotes the action a for which the expression that follows is maximized, $Q_t(a)$ is the mean reward of action a at time step t and finally $Q(a)$ is given as:

$$Q(a) = \mathbb{E}[r \mid a],$$

An agent using only Greedy algorithm can behave sub-optimally forever. On the other hand, an agent that only performs exploration never uses the knowledge that it has gained over time.

One of the simplest algorithms that can be used for solving exploitation/-exploration trade-off is the ε-Greedy algorithm which allows agent to select a random action with small probability ε and with probability 1 - ε use the action that the agent knows that was effective previously.

2.2.8 Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) has been developed by Schulman et. al [46] and is an iterative procedure that gives guaranteed monotonic improvements for optimizing policies. Given an MDP defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0)$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the initial state-transition probability, r is the reward function, γ is the discount factor and ρ_0 is the initial state distribution. Let π be a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1])$ and let $\eta(\pi)$ represent its expected discounted reward which is given as:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right],$$

where $s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$.

Q_π represents the state-action value function and is given as:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right].$$

V_π represents the value function and is given as:

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right].$$

and A_π represent the advantage function and is given as:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s),$$

where $a_t \sim \pi(s_t \mid a_t), s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$ for $t \geq 0$.

Let π_θ represent a policy with parameter θ. If we are trying to optimize for this new policy π_θ then at each iteration, the following constrained optimization problem is solved in TRPO:

$$\begin{aligned} &\underset{\theta}{\text{maximize}} \mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim\pi_{\theta_{old}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)} A_{\theta_{old}}(s,a) \right] \\ &\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{old}}} [D_{KL}(\pi_{\theta_{old}}(\cdot \mid s) \mid\mid \pi_\theta(\cdot \mid s))] \leq \delta_{KL} \end{aligned}$$

where $\rho_{\theta_{old}}$ is the discounted state visitation frequency induced by $\pi_{\theta_{old}}$, $\pi_{\theta_{old}}(a|s)$ is the behavior policy for collecting trajectories, $A_{\theta_{old}}(s,a)$ is the advantage function, δ_{KL} is a step size parameter which controls how much the policy is allowed to change per iteration and $\mathbb{E}[D_{KL}(\pi_{\theta_{old}} \mid\mid \pi_\theta)]$ gives the average KL-divergence between policies across states visited by the old policy.

TRPO provides improvement over vanilla policy gradient methods by making it easier to define the length of the step size. It uses the distributions sampled from the old policy to optimize the new policy which also makes TRPO more sample efficient.

TRPO learns a policy with exploration by sampling actions according to the most recent policy version. The randomness in selection of action is dependent on the training procedure and the initial conditions. During the course of the training this randomness gradually decreases as it starts to exploit the rewards that have already been found.

2.2.9 Truncated Natural Policy Gradient

As described by Duan et al. [16], Truncated Natural Policy Gradient (TNPG) uses a conjugate gradient algorithm to compute the gradient direction. The conjugate gradient algorithm only needs computing $I(\theta)v$, where $I(\theta)$ represents the Fisher Information Matrix (FIM) and v represents an arbitrary vector, to compute the natural gradient direction. This is an improvement over Natural Policy Gradient which computes the gradient direction using $I(\theta)^{-1}\nabla_\theta\eta(\pi_\theta)$ where $\nabla_\theta\eta(\pi_\theta)$ is the gradient of the average reward and π_θ represents the policy $\pi(a;s,\theta)$. Due to the use of FIM inverse, Natural Policy Gradient suffers from high computational cost. TNPG is useful for applying natural gradients in policy search where parameters space is high dimensional.

2.2.10 Recurrent Neural Networks

Recurrent Neural Networks [18] are mostly used for tasks that involve sequential inputs such as speech and language. One of the major problems with traditional neural networks is that they do not consider the dependence of inputs and/or outputs to each other. This issue is addressed by RNNs as they use history to predict future outcomes. They are network with loops in it so that the information can flow from one step of the network to the other, making it persistent.

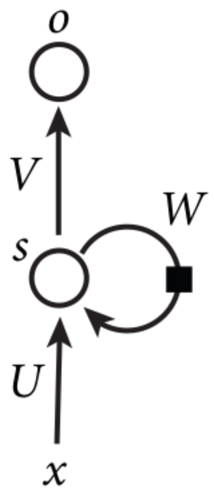


Figure 2.7: High level representation of RNN. Image from LeCun et. al. Deep learning in Nature, vol. 521 [30]

Fig. 2.7 shows a general high level representation of an RNN. Loops in the RNN can be expanded in a chain like architecture which can be seen as a very deep feedforward networks so that information from one step of the network can be fed into the next step and so on. Fig. 2.8 shows an unfolded RNN architecture.

Here, x_t represents the input at time step t, s_t is the hidden state at time step t which can be assumed to be memory of the network which contains information about the history of all the past elements of the sequence and is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1} + b_s)$;the function f usually is a non linear activation function such as $tanh$ or $ReLU$ and b_s is the bias for latent units, o_t is the output at time step t and may be calculated as $o_t = \sigma(Vs_t + b_o)$ where b_o is the bias for the readout unit. The same parameters (matrices U, V and W) are used at each time step.

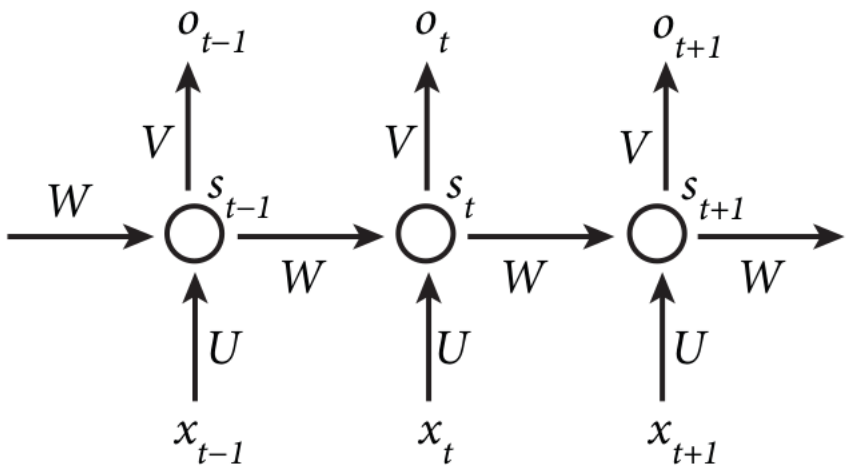


Figure 2.8: A typical unfolded representation of RNN. Image from LeCun et. al. Deep learning in Nature, vol. 521 [30]

RNNs are different from feedforward networks as the RNNs have the ability to make use of sequence of inputs by using their memory. These sequences can vary in size and RNNs can adapt to them dynamically. Also, since the RNNs have output at each time step, there is a cost function associated with the output of each time step as compared to feedforward network where the cost function is applied to its final output.

Even though RNNs can learn to use relevant information from the past, in practice, it becomes very difficult for them to capture long term dependencies. Long Short Term Memory (LSTM)[26] and Gated Recurrent Unit (GRU)[14] try to solve this problem and are better suited to learn these long term dependencies.

The overall chain-like structure of LSTM is very similar to an RNN but the internal structure of units or repeating modules within LSTM are slightly more complex than the ones that are present in an RNN. In order to add or remove information from being passed from one state to the next, LSTM employ logical gates. Internal structure of a unit in an LSTM can be modelled as follows:

$$i_t = \sigma(U^{(i)}x_t + W^{(i)}s_{t-1}) \qquad \text{(input gate)} \qquad (2.5)$$

$$f_t = \sigma(U^{(f)}x_t + W^{(f)}s_{t-1}) \qquad \text{(forget gate)} \qquad (2.6)$$

$$o_t = \sigma(U^{(o)}x_t + W^{(o)}s_{t-1}) \qquad \text{(output gate)} \qquad (2.7)$$

$$\tilde{c_t} = tanh(U^{(\tilde{c_t})}x_t + W^{(\tilde{c_t})}s_{t-1}) \qquad \text{(new candidate values)} \tag{2.8}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c_t} \qquad \text{(new cell state)} \tag{2.9}$$

$$s_t = o_t \cdot tanh(c_t) \qquad \text{(hidden state)} \tag{2.10}$$

Each gate serves a purpose in LSTM. The forget gate determines what fraction of the previous memory cell should be carried over to the next step whereas, the input gate decides which values will be updated. The new candidate values creates a vector of information that has to be added to the new state. New cell state updates the old cell state, c_{t-1}, into the new cell state c_t by combining new candidate values with the input gate. Finally, the output gate determines which parts of the cell state should be provided as output.

GRU is another type of RNN and also has a gated architecture. It has an update gate which is formed by combining the forget and input gates into a single gate. There are some additional changes as well including the integration of cell state and hidden state. The architecture of a GRU can be described mathematically as follows:

$$z_t = \sigma(U^{(z)}x_t + W^{(z)}s_{t-1}) \qquad \text{(update gate)} \tag{2.11}$$

$$r_t = \sigma(U^{(r)}x_t + W^{(r)}s_{t-1}) \qquad \text{(reset gate)} \tag{2.12}$$

$$\tilde{h_t} = tanh(U^{(\tilde{h_t})}x_t + r_t \cdot W^{(\tilde{h_t})}s_{t-1}) \qquad \text{(new candidate values)} \tag{2.13}$$

$$s_t = (1 - z_t) \cdot \tilde{h_t} + z_t \cdot s_t \qquad \text{(hidden state)} \tag{2.14}$$

2.3 Related Work

Modelling human learning or representing memory with a mathematical function has been a focus of research since the late 18th century. Ebbinghaus [20] was the first to study one of the simplest memory models called the Exponential Forgetting Curve. It models the probability of recalling an item as an exponentially decaying function of the memory strength and the time elapsed since it was last practised. Since then, attempts have been made to generalize all kinds of human memory with retention functions such as Rubin and Wenzel[11]. Piech et. al.[42] used Recurrent Neural Networks to model student learning. Bayesian Knowledge Tracing (BKT)[15] is another approach that has often been used to model students, for e.g. in [3, 38].

Understanding these models could be the key to discovering solutions for more efficient learning protocols. Settles and Meeder[47] proposed Half-life regression for spaced repetition, which was applied to the area of language learning, and compared it against traditional methods such as Leitner, Pimsleur and Logistic regression. Spaced repetition is a learning technique that has been shown to produce superior long-term learning including memory retention, problem solving, and generalization of learned concepts to new situations [27].

More recently, RL has found its way to newer domains. Abe et. al.[2] evaluated various types of RL algorithms in the area of sequential targeted marketing by trying to optimize cost sensitive decision. They concluded that indirect RL algorithms work best when complete modelling of the environment is possible and also that their performance degrades when complete modelling is not possible. They also found that Hybrid methods can reduce computation time to attain a given level of performance and give decent policies. Zhao et. al.[62] introduced DRL to the area of recommendation systems by building upon actor-critic framework. They proposed a system capable of continuously improving its strategy while interacting with the users as compared to traditional recommendation systems that make recommendations by following fixed strategies. However, their work did not mention anything about the effectiveness of their proposed system and how well it performed in comparison to other recommendation systems. Some works have also combined deep neural networks with RL such as Su et. al.[53]. They have used various types of RNNs to predict rewards which they termed as reward shaping for faster learning of good policies. Due to the usage of RL in numerous domains of varying nature, comparison between the performance of different RL algorithm has also been conducted. Dual et. al.[16] designed a suite of benchmark tests consisting of continuous control tasks and tested various RL algorithms on this suite. They concluded that both TNPG and TRPO algorithms outperform all other algorithms on most tasks with TRPO being slightly better than TNPG.

RL has also found its application in the domain of ITS. Reddy et. al. [45] presented a way in which DRL can learn effective teaching policies for spaced repetition that can select contents that should be presented next to students without explicitly modelling students. They formalized teaching for spaced repetition as Partially Observable Markov Decision Process (POMDP) and solved it approximately using TRPO on simulated students. Antonova[3] investigated the problem of developing sample-efficient approaches for learning adaptive strategies in domains, including ITS, where the cost of determining the effectiveness of any policy was too high. Mu et. al.[38] attempted to create an ITS by combining two works together for the problem domain of addition; automate creation of curriculum from execution traces and progressing of students using multi-armed bandit problem. Specifically, they used ZPDES, which is a multi-armed bandit algorithm, for problems selection on their underlying student models.

Chapter 3

Method and Experiments

3.1 Dataset

The dataset used for the experiments was synthetic data based on the work of Reddy et. al. [45]. Each of the three student simulators, i.e. GPL(DASH), HLR and EFC, were used to generate student interaction data. This data was represented by a quadruple consisting of $\{q, a, t, d\}$ where q is the most recent item shown to the student and $q \in \mathbb{Z}_{\geq 0}$, a is the answer given by the student for the most recent item and $a \in \{0, 1\}$ with 0 being incorrect and 1 being correct, t is timestamp of when the most recent item was presented and $t \in \mathbb{Z}_{\geq 0}$ and d is the delay between the most recent item and the item before that and it was set to 5 seconds.

For training the LSTM, interaction data was generated using EFC student simulator for 10,000 students and for 200 steps per student for 30 items (cards/exercises). For this case, in order to get different quality of interaction data relying on the tutoring policy used, we generated the data in 3 different ways:

- Using random sample: Both the exercises and responses by students to those exercises were selected randomly. Both the exercises and the responses were independent of the previous exercises presented. Our assumption was that there might be some sequence of exercises and responses that would represent an ideal tutor presenting exercises to students depending on the student's condition.
- Using random policy tutor: We used random tutor to generate this set of data so the exercises were presented by a policy that picked exercises at random to the student but the responses by student were not random. In other words, this data represented a student who was learning while the exercises were presented randomly to him/her.
- Using Supermemo tutor: We used a tutor using Supermemo algorithm.

Interaction data in this scenario would represent a student learning using Supermemo algorithm.

3.2 Systems and Tools

The experiments were run on an Ubuntu system having Intel i7 CPU with 12 cores clocked at 3.30 GHz with a 15360 KB L2 Cache, having 48133 MB RAM and an NVidia TitanX Graphics card.

The implementation was done using Python and Jupyter Notebook. Rllab[1] was used for the implementation of DRL agents while Keras[2] was used for the implementation of LSTM network. Some standard Python libraries were also used, for e.g. Numpy[3]. EFC, HLR, and GPL memory models were implemented using OpenAI Gym[4]. Statistical analysis was done using both R and Python.

[1]http://rllab.readthedocs.io/en/latest/
[2]https://keras.io/
[3]http://www.numpy.org/
[4]https://gym.openai.com/

3.3 Experiments

3.3.1 Experimental Setup

The default parameter settings were kept as proposed by Reddy et. al. [45]. Following were the default parameters:

- Number of items was set to 30,
- Number of runs was set to 10,
- Number of episodes per run was set to 100,
- Number of steps per episode was set to 200 and
- Constant delay of 5 seconds between steps was used.
- There are 4 baseline policies also being used to compare the performance of DRL tutor against:
    - Leitner, which is using Leitner algorithm with arrival rate λ, infinitely many queues, and a sampling distribution $p_i \propto 1/\sqrt{i}i$ over non-empty queues $1, 2, ..., \infty$.
    - SuperMemo, which is using a variant of SM3 algorithm.
    - Random, which selects an item at random.
    - Threshold, which picks the item with predicted recall likelihood closest to some fixed threshold $z* \in [0, 1]$, conditioned on the student model.
- For the EFC student model, item difficulty, θ, was sampled from a log-normal distribution such that $\log \theta \sim \mathcal{N}(\log 0.077, 1)$.
- For the HLR student model, memory strength model parameters were set to be $\vec{\theta} = (1, 1, 0, \theta_3 \sim \mathcal{N}(0, 1))$ and the features for item i to be $\vec{x}_i = ($num attempts, num correct, num incorrect, one-hot encoding of item i out of n items$)$.
- For the GPL student model, student abilities $a = \vec{a} = 0$, sample item difficulties $d \sim \mathcal{N}(1, 1)$ and log $\vec{d} \sim \mathcal{N}(1, 1)$, sample delay coefficient log $r \sim \mathcal{N}(0, 0.01)$, window coefficients $\theta_{2w} = \theta_{2w} - 1 = 1/\sqrt{W - w + 1}$, and number of windows $W = 5$.
- For TRPO, batch size was set to 4000, discount rate, γ, was set to 0.99, and step size was set to 0.01. The recurrent neural network policy used a hidden layer of 32 units.

The settings for TNPG were kept the same as TRPO so that it can be compared with TNPG. γ controls the kind of learning, keeping γ small will induce cramming while keeping it large will produce long-term learning.

The LSTM implementation had 20 LSTM units. 1 dense unit with sigmoid activation function was used. Adam optimizer was also used. LSTM network had 2 hidden states to keep track of question history and observation history of student.

Figures 3.1, 3.2 amd 3.3 show how the interaction between the DRL agent and environment takes place when using different reward functions.

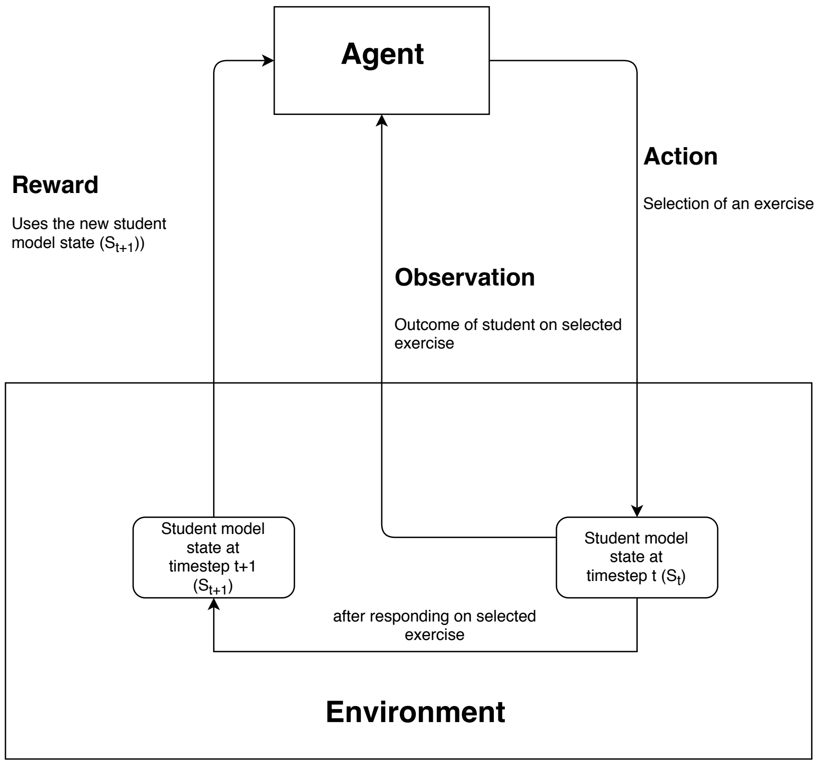

Agent
Reward
Uses the new student model state (S_{t+1}))
Action
Selection of an exercise
Observation
Outcome of student on selected exercise
Student model state at timestep t+1 (S_{t+1})
Student model state at timestep t (S_t)
after responding on selected exercise
Environment

Figure 3.1: Schematic representation of agent-environment interaction when using likelihood and log-likelihood based reward functions.

3.3.2 Reward functions and performance metrics

The reward functions used by Reddy et. al. [45], which also served as performance metrics in their work, depended on the learning objective of the student. They used likelihood as reward function if the goal was to maximize the expected number of items recalled. It was defined as follows:

$$R(s, \cdot) = \sum_{i=1}^{n} P[Z_i = 1|s] \tag{3.1}$$

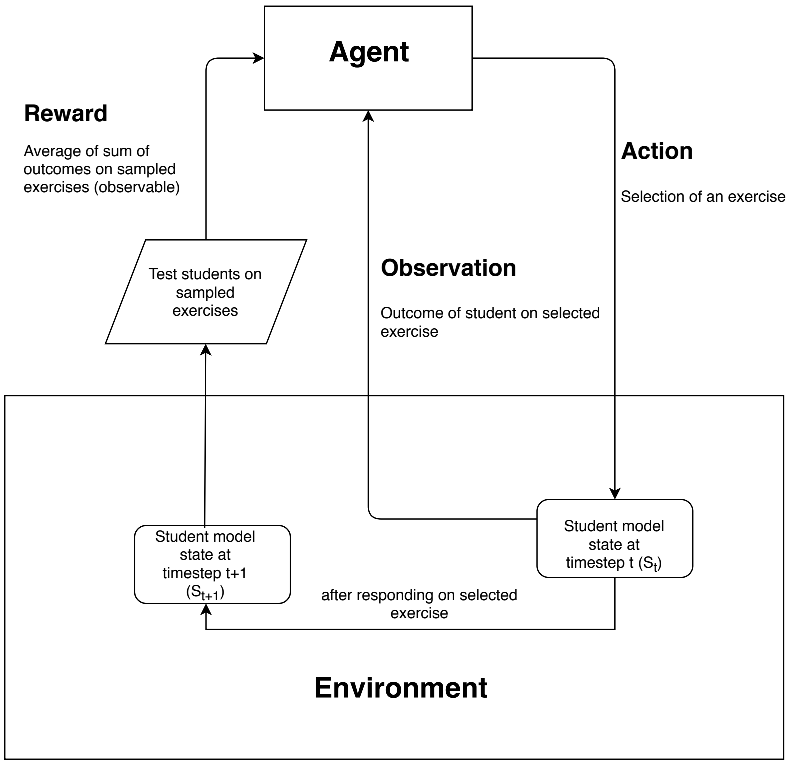

Agent
Reward
Average of sum of outcomes on sampled exercises (observable)
Action
Selection of an exercise
Test students on sampled exercises
Observation
Outcome of student on selected exercise
Student model state at timestep t+1 (S_{t+1})
Student model state at timestep t (S_t)
after responding on selected exercise
Environment

Figure 3.2: Schematic representation of agent-environment interaction when using average of sum of outcomes based reward function.

and if the goal was to maximize the likelihood of recalling all items, then

$$R(s, \cdot) = \sum_{i=1}^{n} \log P[Z_i = 1|s] \tag{3.2}$$

where where Z_i is the response of the student on the exercise shown to them, $i \in \mathbb{Z}_{\geq 0}$ is the exercise index and s is the student state.

In this work, we have defined a reward function which is given by the average of the sum of correct outcomes for every time step and can be denoted as:

$$R(s, \cdot) = \sum_{i} Z_i, \tag{3.3}$$

$$Z_i \sim P_i(\cdot \mid s)$$

where $Z_i \in \{0, 1\}$ depending on whether the response of the student was correct or incorrect on the exercise shown to them, $i \in \mathbb{Z}_{\geq 0}$ is the exercise index, s is the student state and P is the probability of student's response on a given question conditioned upon the state of the student.

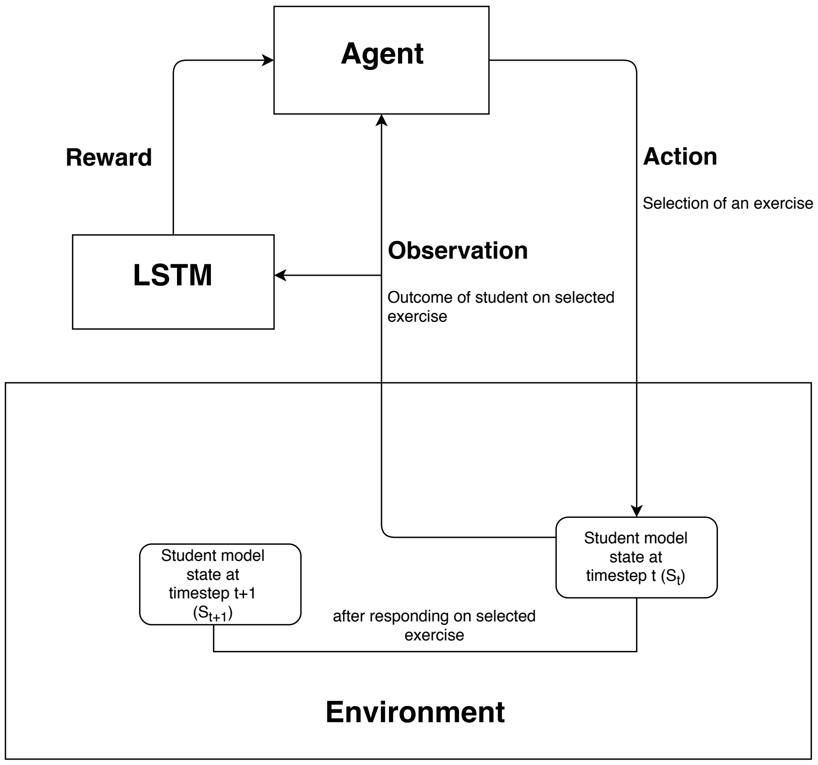

Agent
Reward
Action
Selection of an exercise
LSTM
Observation
Outcome of student on selected exercise
Student model state at timestep t+1 (S_{t+1})
Student model state at timestep t (S_t)
after responding on selected exercise
Environment

Figure 3.3: Schematic representation of agent-environment interaction when using LSTM for reward prediction.

The reward function when using the LSTM network can be denoted as:

$$R_{rnn} = \sum_{i=0}^{n} P_{rnn}(Z_i^j \mid o_{0:j-1}) \tag{3.4}$$

where n is the number of items, j is the current interaction step, P_{rnn} is the probability that the student will be able to correctly answer an exercise i and $o_t = (Z_i^j, i)$ where $Z_i^j \in \{0, 1\}$ depending on whether the student correctly answered the exercise shown to them or not and $i \in \mathbb{Z}_{\geq 0}$ is the exercise index.

In the experiments where we have used the new reward functions for training the DRL agent, we have evaluated the performance of trained DRL agents using only the likelihood as the performance metric given by eq. 3.1.

3.3.3 Training the LSTM

LSTM is used for predicting the rewards for the DRL agent and is a kind of reward shaping. The data sets used for training the LSTM, consisting of 10000 interaction data, was divided into training and validation sets. The training set contained 8000 entries while the remaining 2000 interaction data was used for validation to control overfitting. When training the LSTM, the aim was to optimize for binary cross entropy.

3.3.4 Relation between rewards and thresholds

Since using a black box model seems to have some way to go until it is ready to completely replace human engineered recommendation selection strategies, it is of interest to draw qualitative insights from student simulators on what is the best decision policies given the probability of student being correct on each exercise.

A recommendation strategy was evaluated that picked exercises where the student model's probability of passing was as close as possible to some 'target probability' (threshold benchmark) and the reward for different targeted probability was visualized.

10 runs were conducted for this experiment using each of the three student simulators.

3.3.5 Performance of RL agent when the number of items are varied

For this experiment, the number of items were varied to see if it has any affect on the performance of DRL agent since increasing or decreasing the number of cards affects the state-action space to some extent.

Only the number of items were changed while all other parameters were kept as the same. The value of n, the number of items, was set to 10, 20 and 30. The reward functions used for this experiment are given by eq. 3.1 and 3.2.

3.3.6 Performance of TRPO vs. TNPG algorithms

As mentioned in chapter 2, Dual et. al. [16] tested various RL algorithms on a suite of continuous control tasks and concluded that both TNPG and TRPO algorithms outperform all other algorithms on most tasks with TRPO being slightly better than TNPG. Since this problem can also be considered as one belonging to the same problem domain, the performances of both TRPO and TNPG were examined with respect to other benchmark algorithms.

Default parameters were used for this experiment and settings for TRPO and TNPG were kept as described above. Only EFC student environment was considered and Leitner algorithm was not used when performing this experiment. For this experiment, we used likelihood and log-likelihood based reward functions given by eq. 3.1, 3.2.

3.3.7 Comparison between likelihood and average of sum of outcomes based reward functions (research objective)

For this experiment, DRL agents were first trained using likelihood based reward function(eq. 3.1), and average of sum of correct outcomes based reward function(eq. 3.3). We evaluated the performance of the trained DRL agents using only likelihood as the the performance metric since likelihood is the expected value of the average of sum of correct outcomes. Likelihood was also the performance metric used to compare the performance in the results of Reddy. et al. [45]. All the parameters were set to default values.

3.3.8 Performance of TRPO with reward shaping (research objective)

Again, as mentioned in chapter 2, Su et. al. [53] have used RNNs to predict reward in the context of spoken dialogue systems. For this experiment, LSTM was used for predicting the rewards during the training of DRL agent and was given by eq. 3.4. We evaluated the performance of trained DRL agent using likelihood as the performance metric given by eq. 3.1.

For this experiment, number of episodes per run was set to 40 while keeping all other settings unchanged and only EFC student model was considered. LSTM was trained in 3 different ways mentioned earlier- using data from random sample, using data from random policy tutor and using data from Supermemo tutor.

3.3.9 Evaluation

To plot the graphs, the reward values were stored in a multi-dimensional list $R[i, j, k, l]$ where i represents student model with reward function, for e.g. EFC with likelihood would represent $i = 0$ and so on, j represents tutoring method, k represents reward per episode and l represents the number of run. We then calculated each tutoring method's percentage gain in reward over the random policy's reward.

The choice of statistical analysis test is highly dependent on the type of data being used for statistical analysis. We used different statistical methods for different experiments.

3.3.9.1 Statistical analysis when number of items is varied

To overcome the effects of non-normality and to statistically compare the performances between the tutors, we first calculated pairwise difference of means between DRL tutor's reward and random policy's reward with bootstrap sampling[28]. We used the data of reward values from the last 10 episodes from all of the 10 runs. For both DRL tutor and random policy tutor, we sampled with replacement to get 100 reward samples. We computed their mean and then calculated the difference between DRL tutor mean and random policy tutor mean. We repeated this process 5000 times to create a distribution representing difference of the means of DRL reward and random policy reward. We applied the same process on all student models and all the variation in number of items.

After getting a distribution of relative performance of DRL agents w.r.t. random policy, we used Shapiro-Wilk Normality test [48] to determine if the resulting distributions were normal or not. The homogeneity of variance assumption was not met between the resulting distributions, which was confirmed by applying Levene test [32]. Therefore, we applied Welch two-sample test [59] to determine if there existed any significant differences among the means of the difference in rewards of DRL agents and random policy with different items. We performed this test in pairs of two - (20, 30), (30,40) and (20,40). We finally performed posthoc Games-Howell test [23] to evaluate pairwise difference.

3.3.9.2 Statistical analysis of comparison between TRPO and TNPG

Rewards from all 100 episodes for all 10 runs during evaluation were used to statistically compare the performance of TRPO with TNPG. We performed Kruskal-Wallis test [29], a non-parametric test, to ascertain if there existed any significant differences among the groups.

3.3.9.3 Statistical analysis of comparison between likelihood and average of sum of outcomes

To statistically analyze the performance of DRL agent using average of sum of outcomes as reward function to the DRL agent using likelihood as reward function, we performed Kruskal-Wallis test [29] followed by posthoc Dunn's test [17] with Bonferroni correction to ascertain the dominance of one group over the other. We used rewards from all 100 episodes for all 10 runs when we evaluated DRL agents (trained using different reward functions) on likelihood performance metric.

3.3.9.4 Statistical analysis of the performance of TRPO with LSTM

For analyzing the performance of DRL agents with LSTM for reward shaping, we used rewards from all 40 episodes for all 10 runs when we evaluated DRL agent on likelihood performance metric. We first conducted test for DRL agent using LSTM relative to other methods. Later, we compared all three DRL agents using LSTM (trained using different interaction data) to each other. Since, the distribution of rewards was not normally distributed, we performed Kruskal-Wallis test [29] to determine if there were any significant differences among the various tutors. Finally, we performed Dunn's test [17] with Bonferroni correction to determine stochastic dominance between the groups.

Chapter 4

Results

In this work, we began by attempting to gain a deeper understanding of the recent literature. Since it is important to understand the strategies to pick exercises which would be recommended to students, we investigated how the reward changes relative to the selection of exercises that have predicted recall likelihood close to some threshold value. We also conducted experiments to see if the number of exercises have any effect on the performance of the DRL agent. Some recent literature [53] proclaim that TRPO and TNPG outperform other learning algorithms on continuous decision problems with TRPO being slightly better than TNPG, so we also investigated how they fair against each other in the context of this problem.

To answer the research objectives of the thesis, we conducted experiments to evaluate the performances of the DRL agents trained on the new reward functions. We compared the performance of DRL agent using the average of sum of outcomes based reward function against DRL agent using likelihood based reward function and other baseline policies on recall likelihood performance metric. Finally, we trained LSTM network on tutor-student interaction data to output the reward and then compared the performance of DRL agent using LSTM for rewards against other baseline policies on recall likelihood performance metric. We also investigated if an LSTM, that has been trained on better recommendation strategy, can give even better performance.

4.1 Relation between rewards and thresholds

We started by investigating strategies which picked exercises that generated maximum rewards. We conducted 10 runs for all the three student models viz. EFC, HLR and GPL, for the reward functions given by eq. 3.1 and 3.2. We varied the threshold value between 0 and 1, allowing the tutor to pick exercises close to the threshold value and recorded the reward value. The reward values relative to the threshold values were then represented by a plot.

Figures 4.1, 4.2 and 4.3 show the generated plots. We note that for low threshold values, the rewards were generally high in almost all the cases.

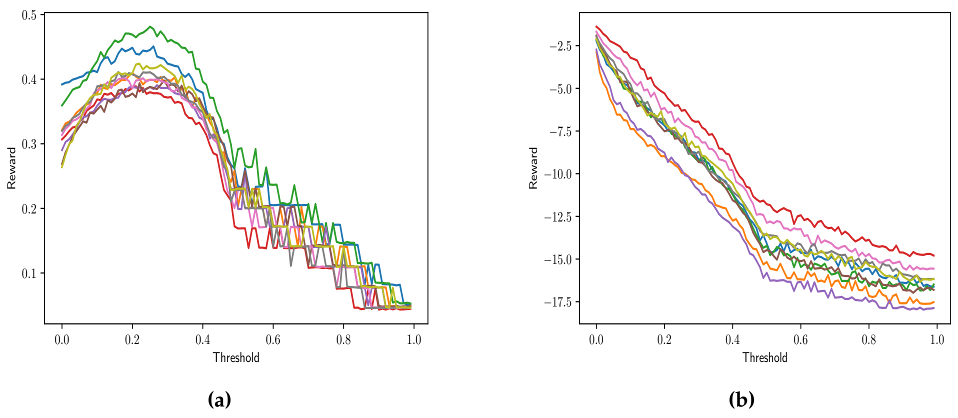


Figure 4.1: *Threshold vs. reward for EFC student model. a) likelihood used as reward function and b) log-likelihood used as reward function. Each colored line represents a different run.*

Figure 4.1(a) shows the plots for EFC student model when likelihood based reward function was used. The plots display that the reward was high for low threshold values and the reward value reaches its peak when the threshold value reaches between 0.2 and 0.4. This implies that in order to make students learn more efficiently, it is best to select exercises that a student has 20% to 40% chance of answering correctly. When the reward function was based on log-likelihood, figure 4.1(b), the reward value starts to decrease as the threshold value increases.

In the case of HLR student model the plots looked similar to a step function, figure 4.2. In both cases, when the reward function was based on likelihood and log-likelihood, the trend was similar as in the case of EFC student model. A motivation for this is that the probability of correctly recalling an item,p, for HLR model is given as $p = 2^{-\Delta/h}$ where Δ is the lag time since the item was last practiced and h the half-life or measure of strength in the learner's long-term memory. So, we can say that threshold values in the range (0, 0.5) occurs when $\Delta \gg h$ which signifies that the exercise has probably already been forgotten. Therefore, selecting exercises within the range (0, 0.5) should generate high reward. Selecting exercises close to threshold values in the range (0, 1.0) could be the case when $\Delta \ll h$ and indicates that the exercise has recently been practiced and thus, student should be able to recall it and thus generate low rewards.

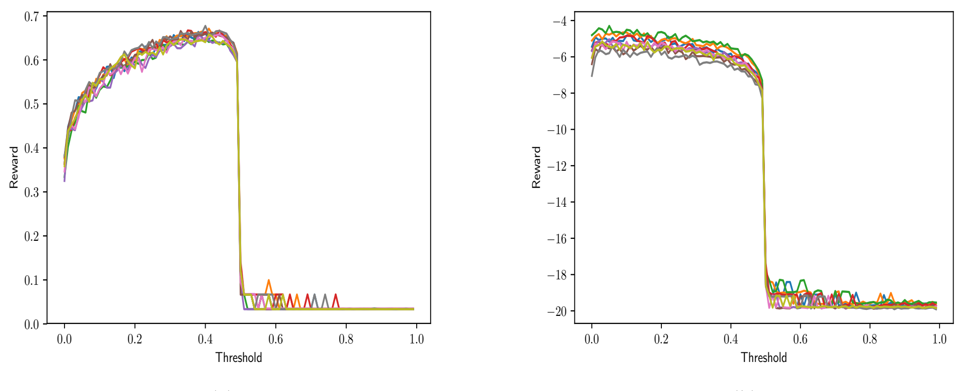


Figure 4.2: *Threshold vs. reward for HLR student model. a) likelihood used as reward function and b) log-likelihood used as reward function. Each colored line represents a different run.*

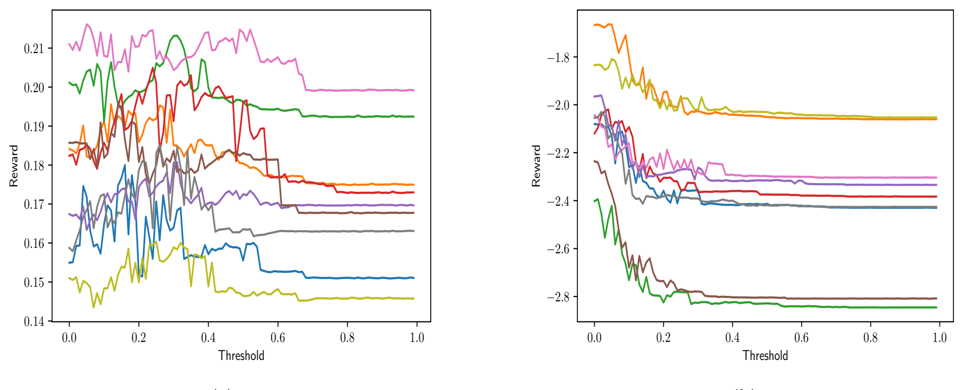
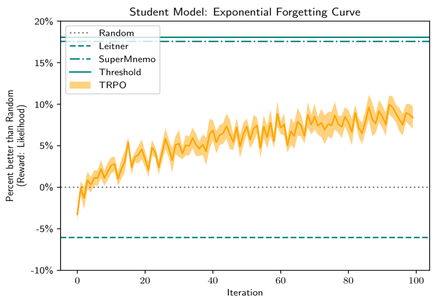


Figure 4.3: *Threshold vs. reward for DASH (GPL) student model. a) likelihood used as reward function and b) log-likelihood used as reward function. Each colored line represents a different run.*

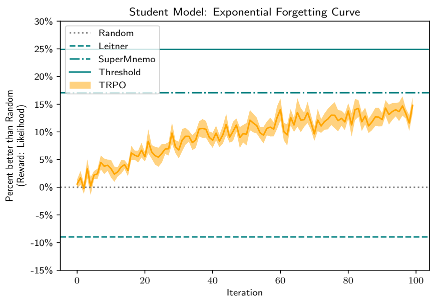
For DASH(GPL) student model when the reward function used was based on log-likelihood, figure 4.3(b), it can be seen that the reward value decreases with the increase in threshold value and follows the above trend when the reward function used was based on log-likelihood. However, in the case of likelihood based reward function, figure 4.3(a), it can be noticed that the change in reward value contains a lot of noise as the threshold value increases.

4.2 Performance of DRL agent when the number of items are varied

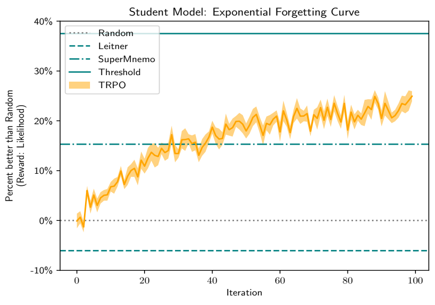
4.2.1 With EFC student model



(a)



(b)

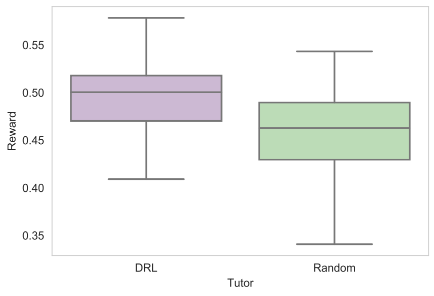


(c)

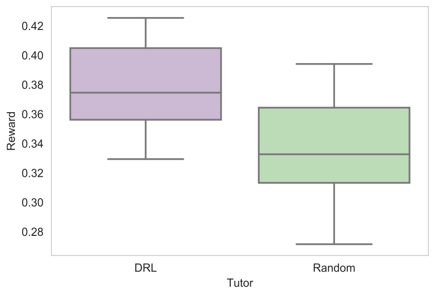
Figure 4.4: *EFC student model with likelihood based reward function used for training DRL agent. a) number of items set to 20 b) number of items set to 30 c) number of items set to 40.*

As can be seen in figure 4.4, there is a minor increase in the performance of the DRL agent relative to random baseline policy as the number of items increases when the likelihood reward function is used during training. Even though the actual reward values are decreasing as the number of items increase as can be seen in the boxplots of the reward distributions in figure 4.5, which demonstrates the distribution of rewards for the last 10 episodes for all 10 runs. We computed pairwise difference of means between DRL reward and random policy reward (DRL − Random) with bootstrap sampling. Figure 4.6 displays the boxplots for the resulting distributions with various numbers of items which indicates that there is an increase in the difference between DRL reward and random policy reward.

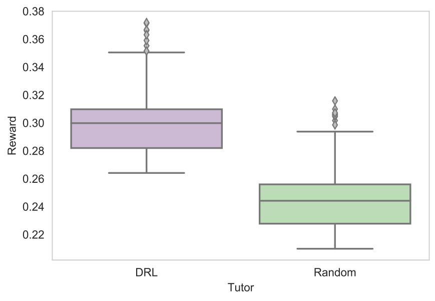
All of the three distributions obtained from the pairwise difference of means between DRL reward and random policy reward with bootstrap sampling were found to be normally distributed. For distribution generated when



(a)



(b)



(c)

Figure 4.5: *Boxplot showing distribution of rewards for the last 10 episodes for all 10 runs during the training with Random policy and DRL agent using likelihood based reward function on EFC student model. a) No. of items = 20 b) No. of items = 30 c) No. of items = 40.*

number of items were 20, 30 and 40, p was $0.56, 0.54$ and 0.41 resp. (Shapiro-Wilk Normality test, N=5000) but without homogeneous variance at $p < .001$ (Levene test, N=5000 per distribution). There were also significant differences present among the three distributions with $p < .001$ (Welch two-sample tests, N=5000). The results of the posthoc Games-Howell test are listed in table 4.1. We can see that there is a minor increase when the number of items were increased.

Table 4.1: *Results from posthoc Games-Howell test on distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward). Performance of DRL agents was compared relative to random policy when number of items were changed for EFC student model using likelihood based reward function.*

No. of items pair	Difference	p-value
$30 - 20$	.0062	<.01
$40 - 30$	.0107	<.01
$40 - 20$	0.0168	<.01

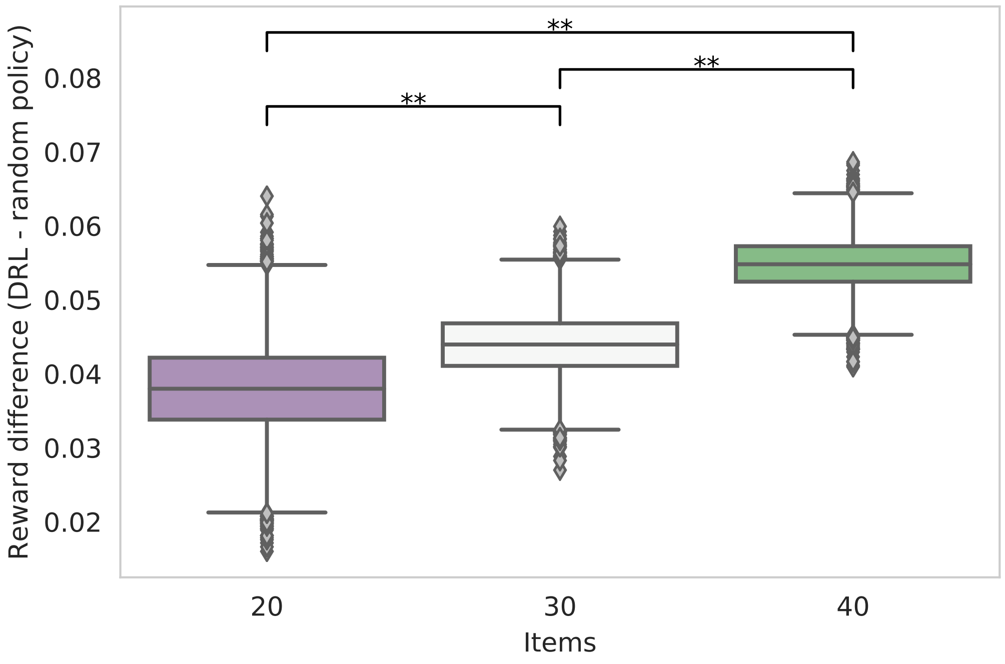
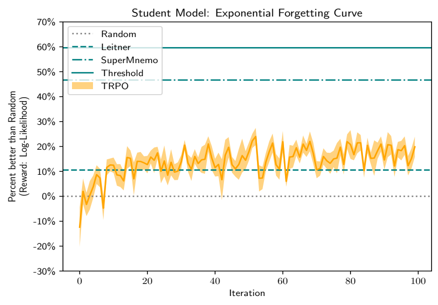
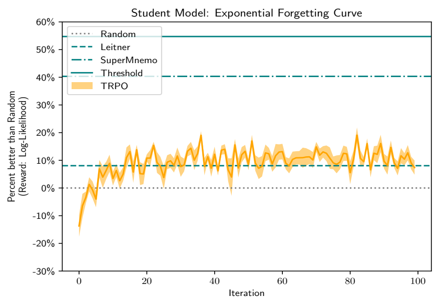


Figure 4.6: *Boxplot showing distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward) for EFC student model using likelihood based reward function for different number of items.*

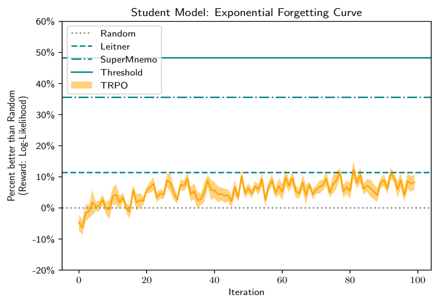
On the other hand, when log-likelihood was used for training, given in figure 4.7, as the number of items increase, a small decrease in the performance of the DRL agent relative to the random policy can be noticed. The actual rewards decrease in this case also as can be seen in figure 4.8. However, distributions resulting from the pairwise difference of means between DRL reward and random policy reward with bootstrap sampling showed no general trend. We can see in figure 4.9, which shows the boxplots for the distributions representing the pairwise difference of means between DRL reward and random policy reward with bootstrap sampling, that the difference between the reward values drops from $n = 20$ to $n = 30$ and then increases again for $n = 40$.



(a)



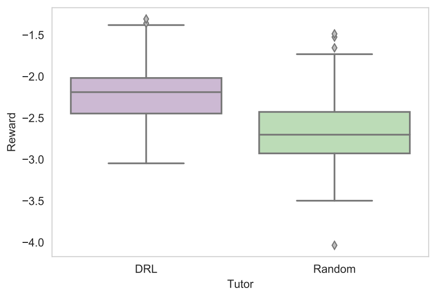
(b)



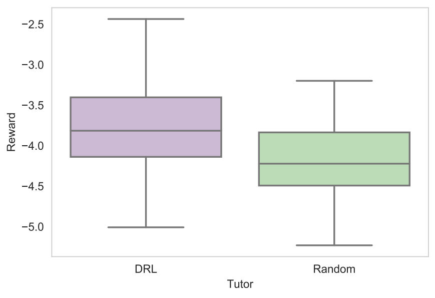
(c)

Figure 4.7: *EFC student model with log-likelihood based reward function used for training DRL agent a) number of items set to 20. b) number of items set to 30 c) number of items set to 40.*

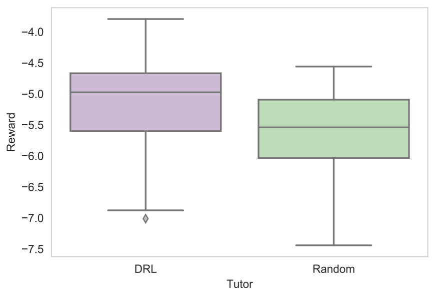
The results of the posthoc Games-Howell test, in table 4.2, did not reveal a general trend in the differences as the number of items are varied.



(a)



(b)



(c)

Figure 4.8: *Boxplot showing distribution of rewards for the last 10 episodes for all 10 runs during the training with Random policy and DRL agent using log-likelihood based reward function on EFC student model a) No. of items = 20 b) No. of items = 30. c) No. of items = 40.*

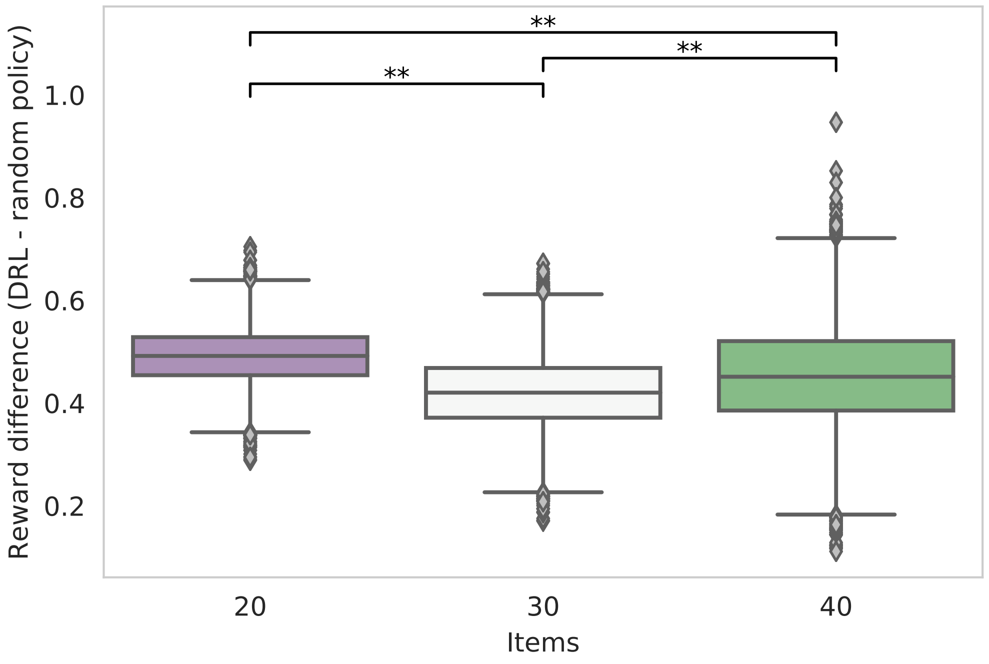
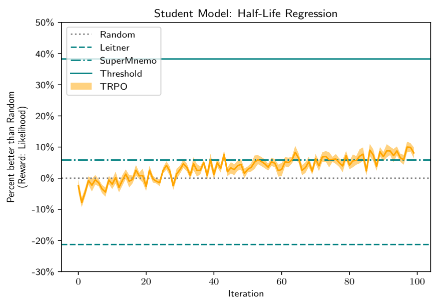


Figure 4.9: *Boxplot showing distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward) for EFC student model using log-likelihood based reward function for different number of items.*

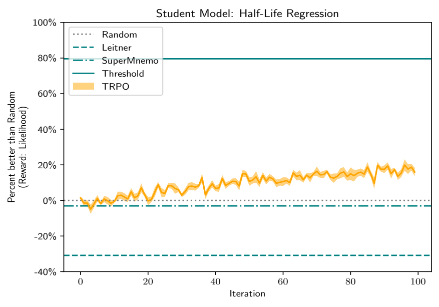
Table 4.2: *Results from posthoc Games-Howell test on distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward). Performance of DRL agents was compared relative to random policy when number of items were changed for EFC student model using log-likelihood based reward function.*

No. of items pair	Difference	p-value
$30 - 20$	$-.072$	$<.01$
$40 - 30$	$.032$	$<.01$
$40 - 20$	$-.040$	$<.01$

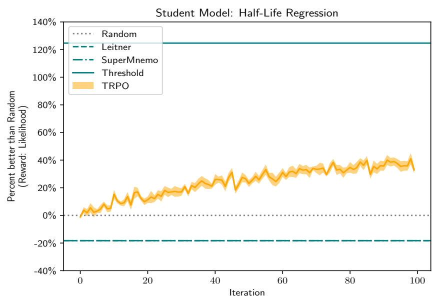
4.2.2 With HLR student model



(a)



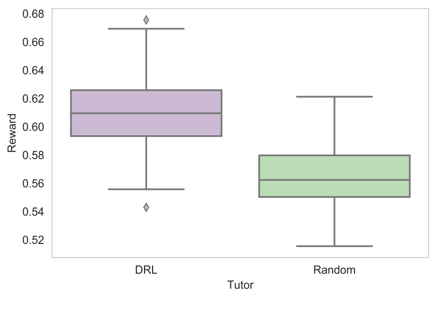
(b)



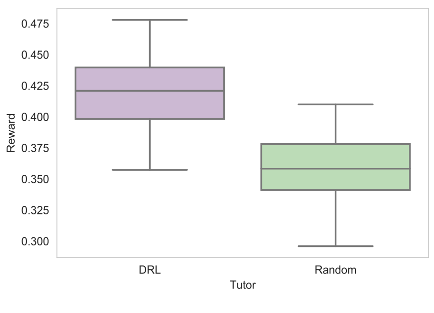
(c)

Figure 4.10: *HLR student model with likelihood based reward function used for training DRL agent. a) number of items set to 20 b) number of items set to 30 c) number of items set to 40.*

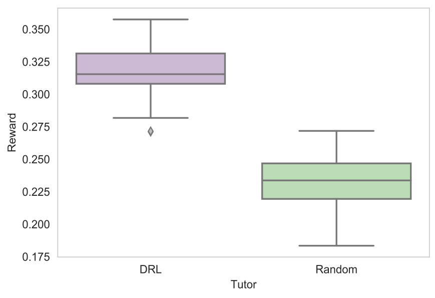
With HLR model, DRL agent using likelihood based reward function shows the same trend as in EFC; the performance of DRL agent relative to the baseline random policy shows a little improvement as the number of items increase, figure 4.10. In figure 4.11, which shows the boxplots for the distribution of the rewards, the actual reward values can also be seen to decrease as the number of items increases. In figure 4.12 which shows the boxplots for the distributions resulting from pairwise difference of mean with bootstrap sampling (DRL reward $-$ Random policy reward), we can see that there is a definite increase in the reward values for DRL agent relative to the reward values for random policy as the number of items increase.



(a)



(b)



(c)

Figure 4.11: *Boxplot showing distribution of rewards for the last 10 episodes for all 10 runs during the training with Random policy and DRL agent using likelihood based reward function on HLR student model a) No. of items = 20 b) No. of items = 30 c) No. of items = 40.*

The results of the posthoc Games-Howell method are listed in table 4.3 proving that there is a minor increase in the performance of the DRL agent relative to random policy as the number of items increase.

Table 4.3: *Results from posthoc Games-Howell test on distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward). Performance of DRL agents was compared relative to random policy when number of items were changed for HLR student model using likelihood based reward function.*

No. of items pair	Difference	p-value
$30 - 20$	.014	$<.01$
$40 - 30$	.026	$<.01$
$40 - 20$	.040	$<.01$

When log-likelihood based reward function was used with HLR model, the plots in figure 4.13 do not reveal any noticeable variation in the relative performances of the DRL agents compared to the random policy. From figure 4.14, we can see that the actual rewards decrease for the DRL agent. The boxplots for distributions resulting from the pairwise difference of means

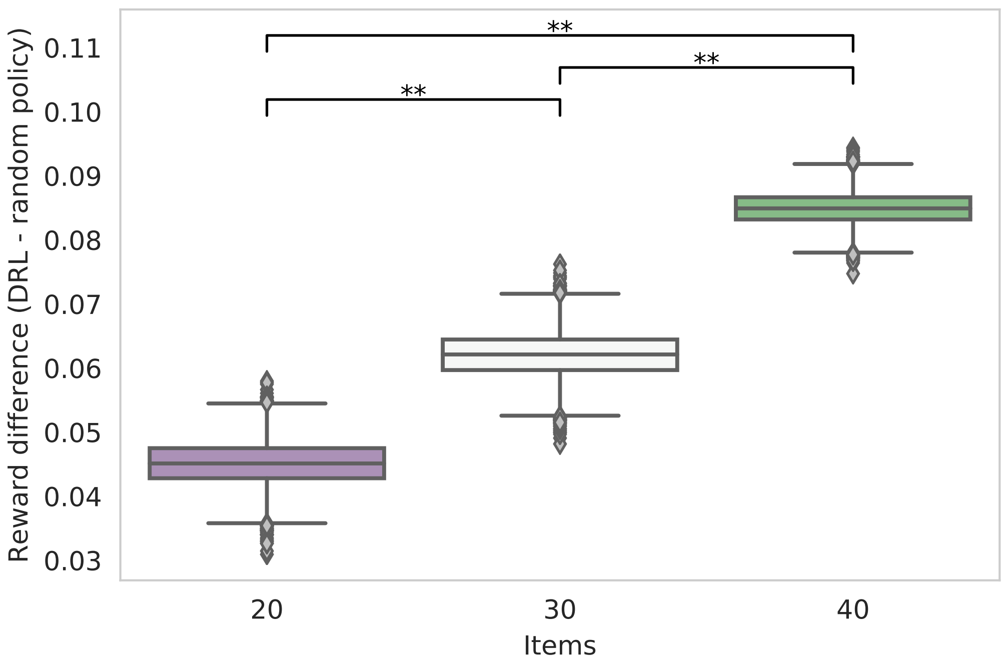
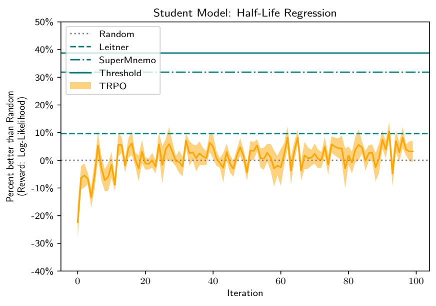


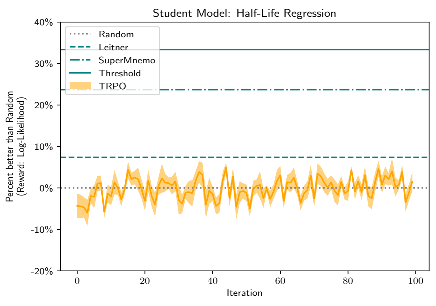
Figure 4.12: *Boxplot showing distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward) for HLR student model using likelihood based reward function for different number of items.*

between DRL and random policy with bootstrap sampling are shown in figure 4.15. They do not reveal any trend either when number of items, n, increases from 20 to 40. There are some minor variations when n increases but a general trend is missing.

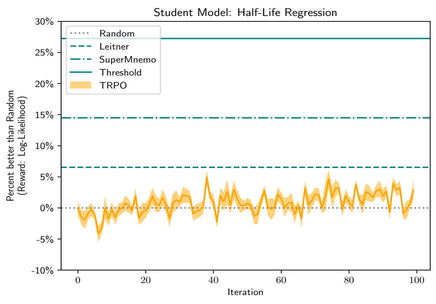
From the results of posthoc Games-Howell test, we were not able to find a general trend in the performance of the DRL agent relative to random policy as the number of items increases.



(a)

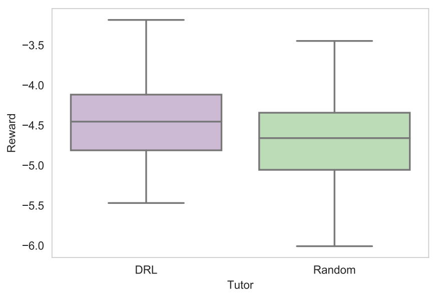


(b)

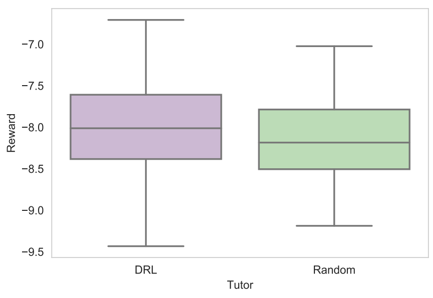


(c)

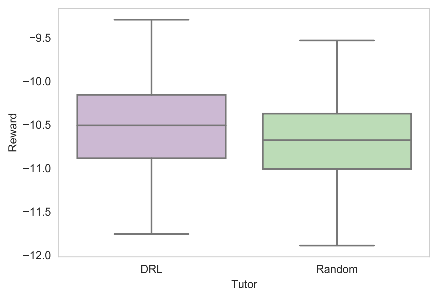
Figure 4.13: *HLR student model with log-likelihood based reward function used for training DRL agent. a) number of items set to 20 b) number of items set to 30 c) number of items set to 40.*



(a)



(b)



(c)

Figure 4.14: *Boxplot showing distribution of rewards for the last 10 episodes for all 10 runs during the training with Random policy and DRL agent using log-likelihood based reward function on HLR student model a) No. of items = 20 b) No. of items = 30 c) No. of items = 40.*

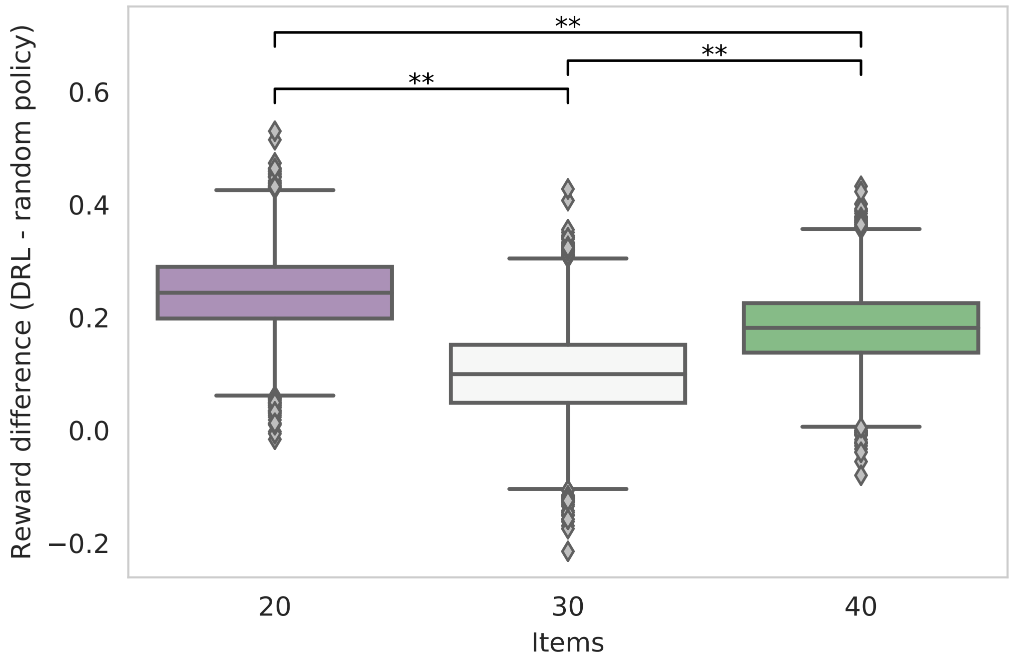
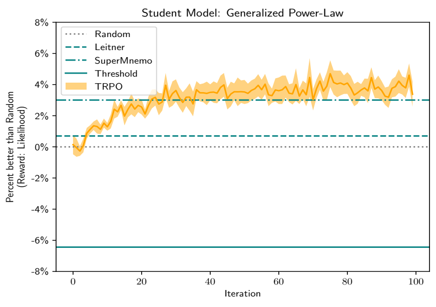


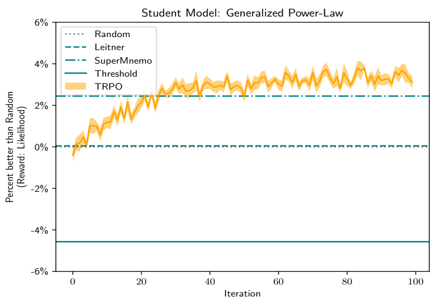
Figure 4.15: Boxplot showing distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward) for HLR student model using log-likelihood based reward function for different number of items.

4.2.3 With DASH (GPL) student model

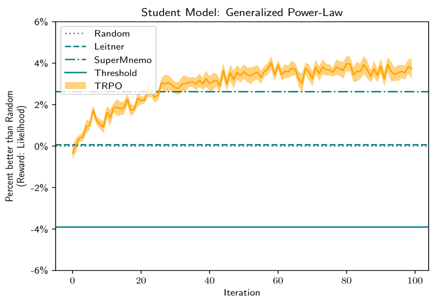
When likelihood based reward function was used with DASH student model, no variation in the performance of the DRL agent relative to the random policy can be noticed, figure 4.16. The actual reward distributions given by the boxplots in figure 4.17 showed minor variations. However, the boxplots for the pairwise difference of means distributions, given in figure 4.18 showed minor decrements in the difference of reward values. From the results of the posthoc Games-Howell test, table 4.4, we can notice that there is a very small decrease in the relative performance of DRL agent as the number of items increase.



(a)

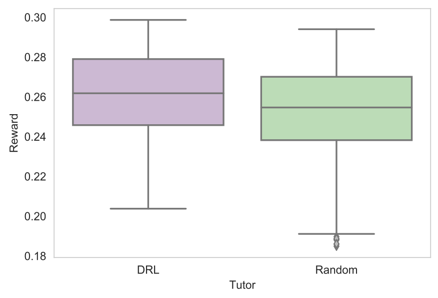


(b)

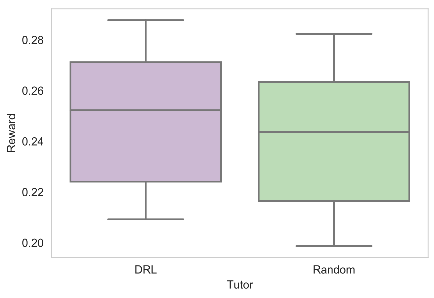


(c)

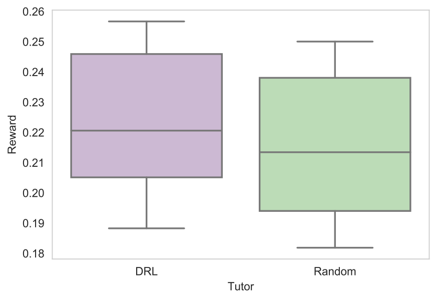
Figure 4.16: *DASH(GPL) student model with likelihood based reward function used for training DRL agent. a) number of items set to 20 b) number of items set to 30 c) number of items set to 40.*



(a)



(b)



(c)

Figure 4.17: *Boxplot showing distribution of rewards for the last 10 episodes for all 10 runs during the training with Random policy and DRL agent using likelihood based reward function on DASH student model a) No. of items = 20 b) No. of items = 30 c) No. of items = 40.*

Table 4.4: *Results from posthoc Games-Howell test on distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward). Performance of DRL agents was compared relative to random policy when number of items were changed for DASH student model using likelihood based reward function.*

No. of items pair	Difference	p-value
30 − 20	−.00082	<.01
40 − 30	−.00028	<.01
40 − 20	.−00111	<.01

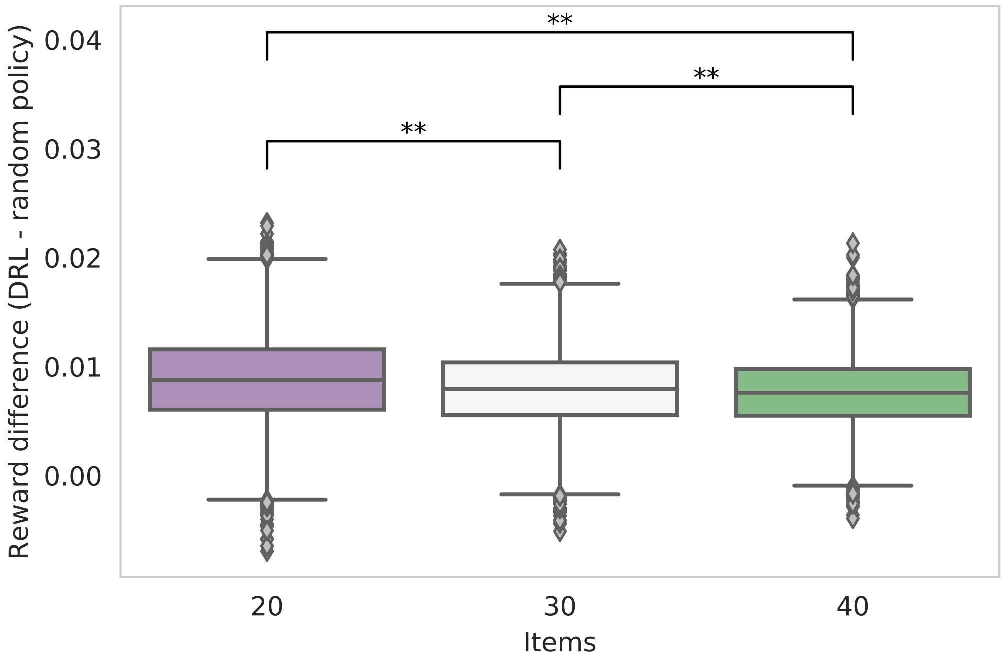
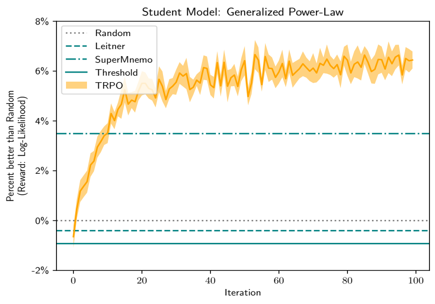


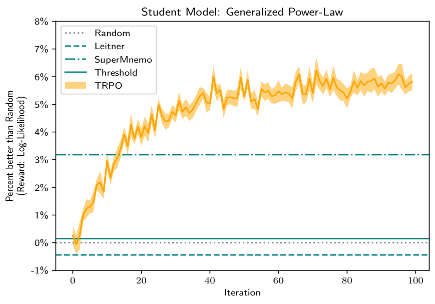
Figure 4.18: *Boxplot showing distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward) for DASH student model using likelihood based reward function for different number of items.*

For DASH student model with log-likelihood based reward function, again, no noticeable variation in the performance of the DRL agent relative to the random policy can be seen in the plots, figure 4.19 and the actual reward values have very small variations as the number of items change 4.20. The boxplots for the resulting distributions from the pairwise difference of means between DRL and random policy rewards with bootstrap sampling are shown in figure 4.21.

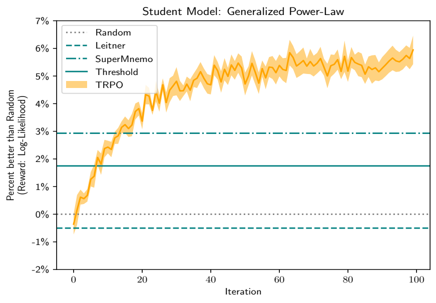
Posthoc Games-Howell test did not reveal a pattern as we increased the number of items.



(a)



(b)



(c)

Figure 4.19: *DASH(GPL) student model with log-likelihood based reward function used for training DRL agent. a) number of items set to 20 b) number of items set to 30 c) number of items set to 40.*

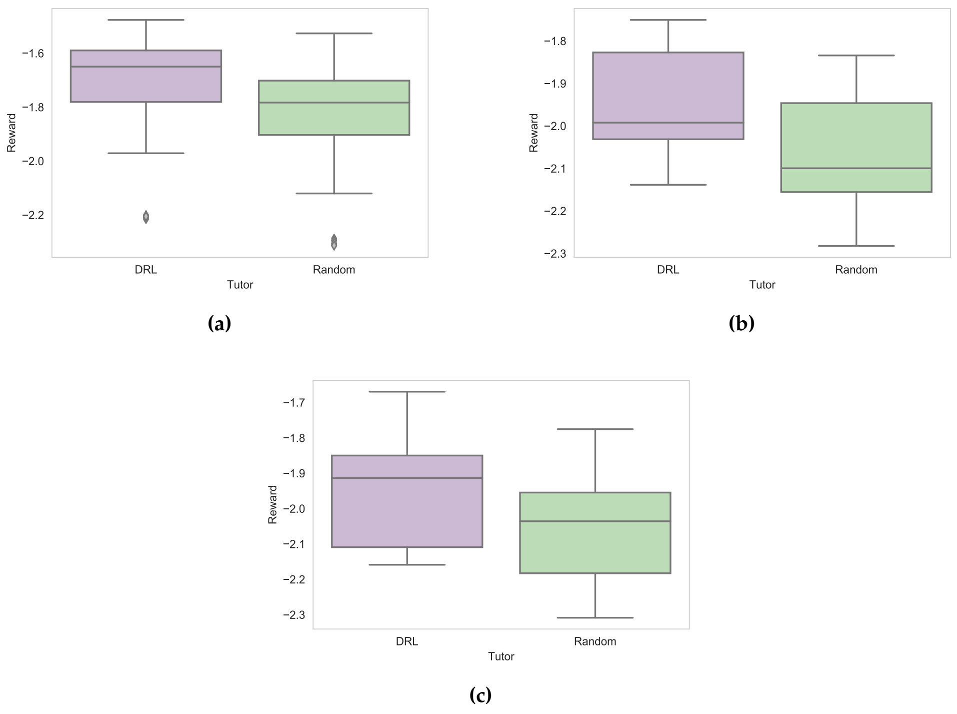


Figure 4.20: Boxplot showing distribution of rewards for the last 10 episodes for all 10 runs during the training with Random policy and DRL agent using log-likelihood based reward function on DASH student model a) No. of items = 20 b) No. of items = 30 c) No. of items = 40.

4.3 Comparison of Performance of TRPO and TNPG algorithms

Recent work has shown that TRPO and TNPG perform quite well on continuous benchmark problems with TRPO showing a slight superiority in performance over TNPG.

We conducted experiments to compare the performances of the two algorithms in the context of our problem domain. We only tested their performance on EFC student model. As can be seen from figures 4.22 and 4.23, both TRPO and TNPG algorithms perform fairly similar in both our experiments. However, we were not able to notice any significant gain of TRPO over TNPG.

No significant difference among the reward distributions for TRPO and TNPG were found for both likelihood with $p = 0.221$ and log-likelihood with $p = 0.884$ (Kruskal-Wallis test, N=1000 per policy).

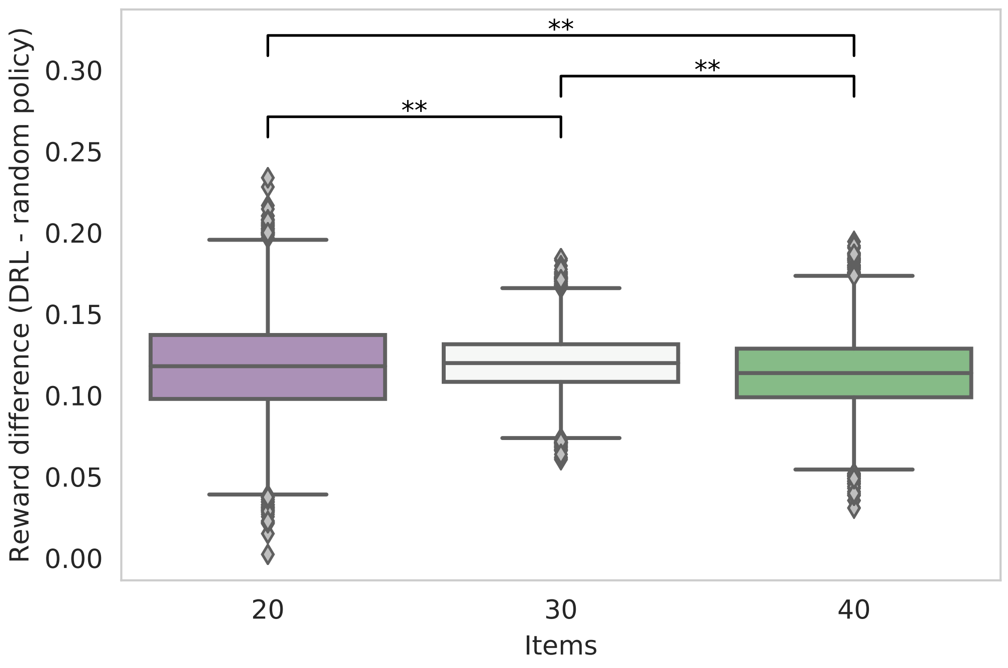


Figure 4.21: *Boxplot showing distributions resulting from pairwise difference of means with bootstrap sampling (DRL reward − Random policy reward) for DASH student model using log-likelihood based reward function for different number of items.*

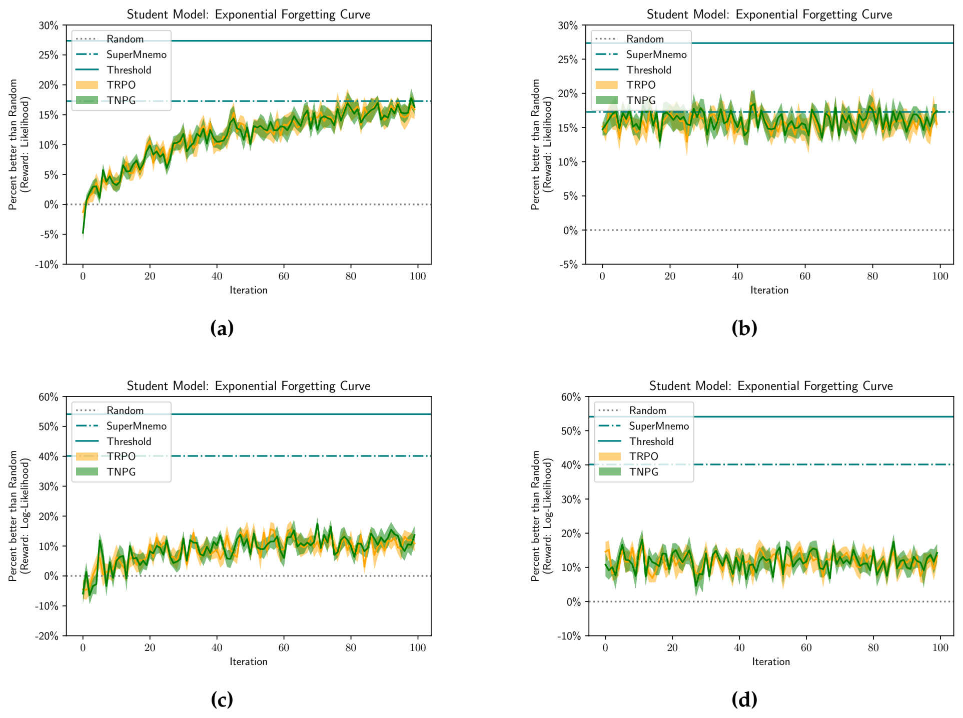
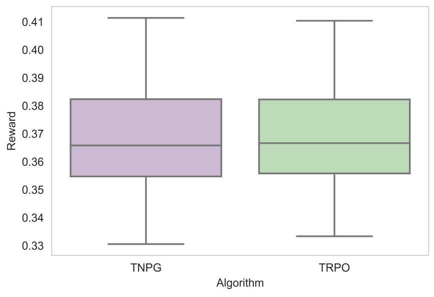
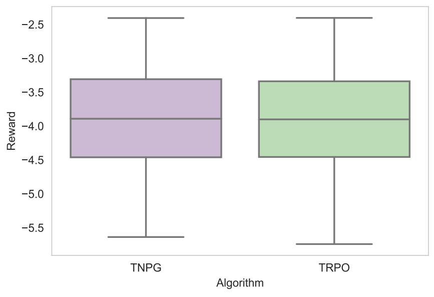


Figure 4.22: *Comparison of performance of TRPO and TNPG using likelihood and log-likelihood based reward functions on EFC student model. For the both the algorithms 10 runs were conducted with 100 episodes each. a) training using likelihood based reward function b) evaluation on likelihood performance metric c) training using log-likelihood based reward function d) evaluation on log-likelihood performance metric.*



(a)



(b)

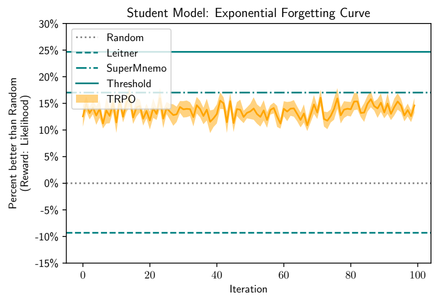
Figure 4.23: *Boxplots comparing generated rewards for TRPO and TNPG when evaluated on likelihood and log-likelihood performance metric on EFC student model. Both the algorithms were trained on likelihood and log-likelihood based reward functions prior to their evaluation. For the both the algorithms 10 runs were conducted with 100 episodes each. a) likelihood b) log-likelihood.*

4.4 Comparison between likelihood and average of sum of outcomes based reward functions (research objective)

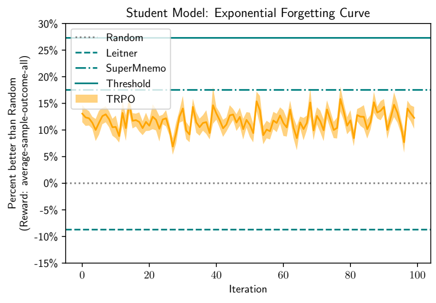
To remove the dependency of student states from the reward function, we came up with the average of sum of outcomes as the reward function. We expected that it would roughly give us the same results as likelihood. With average of sum of outcomes, we are also trying to maximize the expected number of items recalled but by limiting ourselves to observations which are student responses to exercises.

As can be seen in figure 4.24, in the cases of EFC and HLR student models, DRL agents trained on average of sum of outcomes was able to give comparable performance to DRL agent trained using likelihood based reward function. However, in the case of GPL student model, the performance of DRL agent trained on average of sum of outcomes varies a lot.

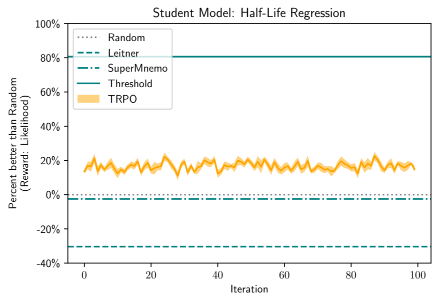
Significant difference existed between the reward distributions obtained for likelihood based reward function and average of sum of outcomes based reward function for all three student learning models with $p<.001$ (Kruskal-Wallis test, N=1000 per policy). For both EFC and HLR student model, sum of average of outcomes based reward function performed fairly close to the likelihood based reward function with likelihood based reward function performing slightly better with $p<.001$ (Dunn's test, N=1000 per policy), Z-statistic for average of sum of outcomes − likelihood based reward function on EFC was -9.293 and on HLR was -10.926. However, for DASH student model, sum of average of outcomes based reward function performed better than likelihood with $p<.001$ (Dunn's test, N=1000 per policy) and Z-statistic for average of sum of outcomes − likelihood based reward function was 3.319.



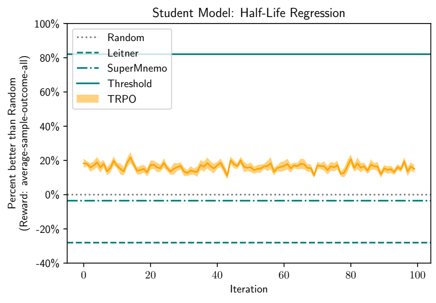
(a)



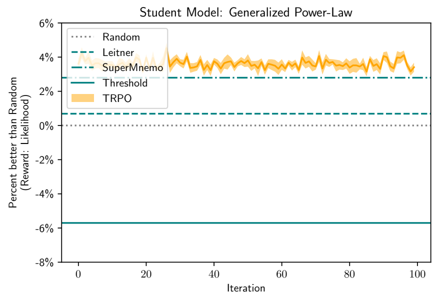
(b)



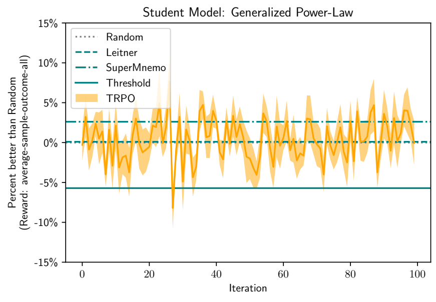
(c)



(d)

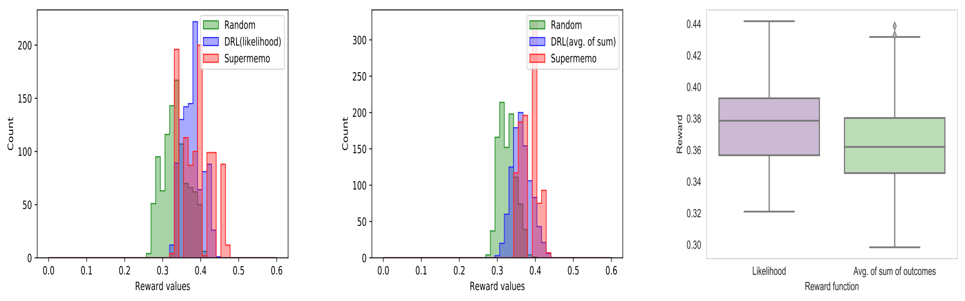


(e)



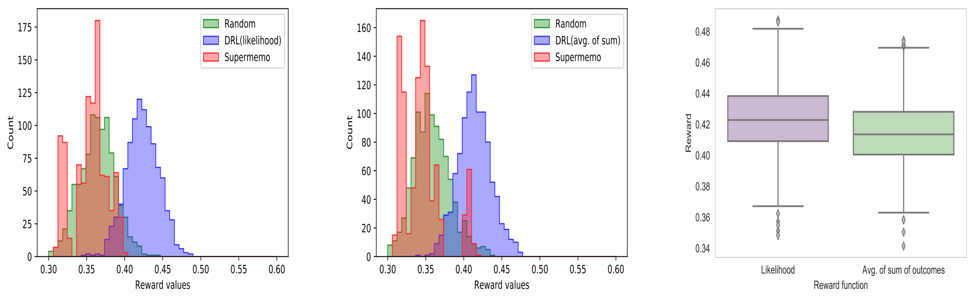
(f)

Figure 4.24: *Comparison between likelihood and average of sum of outcomes based reward functions on various student models when evaluated on likelihood performance metric. The DRL agents were trained prior to evaluation using respective reward functions a) uses likelihood based reward function for training on EFC student model b) uses average of sum of outcomes based reward function for training on EFC student model c) uses likelihood based reward function for training on HLR student model d) uses average of sum of outcomes based reward function for training on HLR student model e)uses likelihood based reward function for training on DASH(GPL) student model and f) uses average of sum of outcomes based reward function for training on DASH (GPL) student model.*



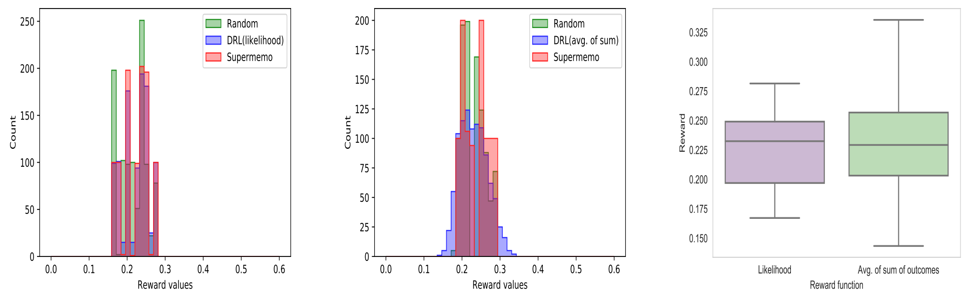
(a) **(b)** **(c)**

Figure 4.25: *Comparison of distribution of rewards for the all 100 episodes for all 10 runs for various tutors when evaluated on likelihood performance metric on EFC student model. a) DRL using likelihood as reward function w.r.t. other policies b) DRL using average of sum of rewards as reward function w.r.t other policies c) Boxplots showing rewards from DRL using likelihood as reward function and DRL using average of sum of outcomes as reward function.*



(a) **(b)** **(c)**

Figure 4.26: *Comparison of distribution of rewards for the all 100 episodes for all 10 runs for various tutors when evaluated on likelihood performance metric on HLR student model. a) DRL using likelihood as reward function w.r.t. other policies b) DRL using average of sum of rewards as reward function w.r.t other policies c) Boxplots showing rewards from DRL using likelihood as reward function and DRL using average of sum of outcomes as reward function.*

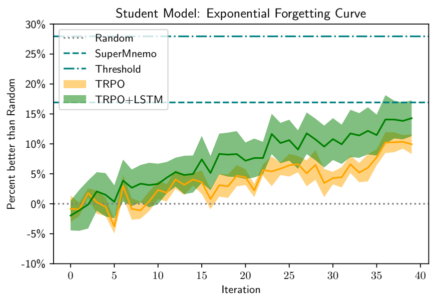


(a) **(b)** **(c)**

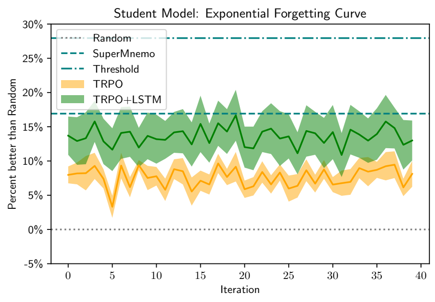
Figure 4.27: *Comparison of distribution of rewards for the all 100 episodes for all 10 runs for various tutors when evaluated on likelihood performance metric on DASH student model. a) DRL using likelihood as reward function w.r.t. other policies b) DRL using average of sum of rewards as reward function w.r.t other policies c) Boxplots showing rewards from DRL using likelihood as reward function and DRL using average of sum of outcomes as reward function.*

4.5 Performance of TRPO with reward shaping (research objective)

Another way to make the DRL agent independent of student state was to make use of an RNN. We ended up choosing LSTM, a kind of RNN, for this task which has been shown to work well with timeseries data. We conducted three sets of experiments to gauge the performance of LSTM. We trained LSTM networks in three ways - using interaction data generated from a random sample policy, using interaction data generated when using random policy tutor and using interaction data generated when using supermemo tutor. Training and validation plots for LSTMs on these datasets can be seen in Appendix A. Figures 4.28, 4.29 and 4.30 show the performance of DRL agents during training and when later evaluated on likelihood performance metric. We noticed that the DRL agents using LSTM learned smoother and faster than the DRL agents using average of sum of outcomes based reward function which conforms to the results of Su et. al. [53]. We also noticed that the performance of the DRL agent got better as the quality of the interaction data got improved.

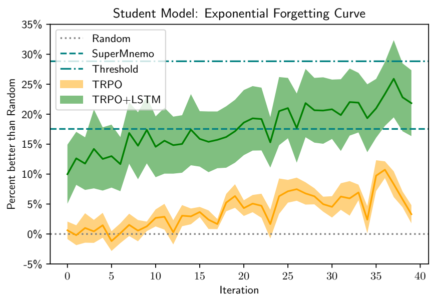


(a)

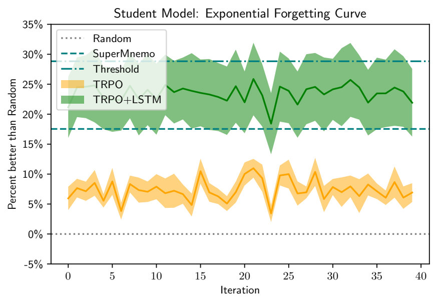


(b)

Figure 4.28: *Comparison of DRL agent using reward shaping for reward prediction against other baseline policies and DRL agent using average sum of outcomes based reward function on EFC student model. LSTM was trained on data generated using random sample policy. a) shows the performance of DRL agent while training b) shows the performance of trained DRL agent on likelihood performance metric.*

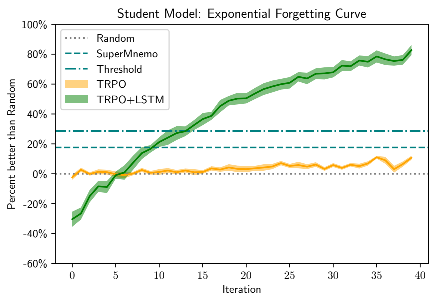


(a)

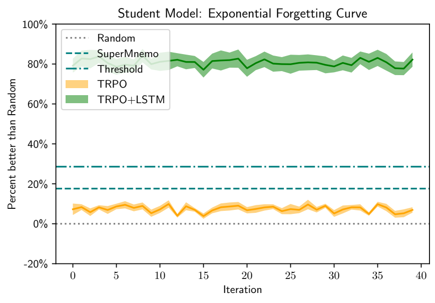


(b)

Figure 4.29: *Comparison of DRL agent using reward shaping for reward prediction against other baseline policies and DRL agent using average sum of outcomes based reward function on EFC student model. LSTM was trained on data generated using random tutor. a) shows the performance of DRL agent while training b) shows the performance of trained DRL agent on likelihood performance metric.*

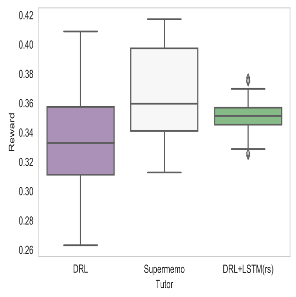


(a)

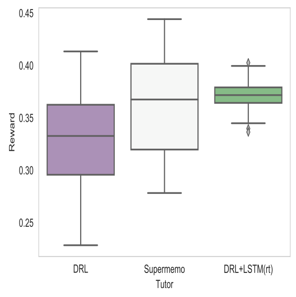


(b)

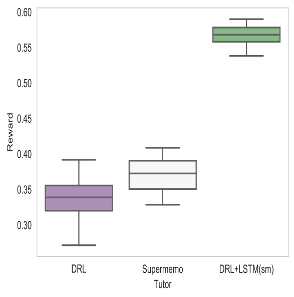
Figure 4.30: *Comparison of DRL agent using reward shaping for reward prediction against other baseline policies and DRL agent using average sum of outcomes based reward function on EFC student model. LSTM was trained on data generated using supermemo tutor. a) shows the performance of DRL agent while training b) shows the performance of trained DRL agent on likelihood performance metric.*



(a)



(b)



(c)

Figure 4.31: *Boxplots showing distribution of rewards for all 40 episodes for all 10 runs for various tutors when evaluated on likelihood performance metric on EFC student model. a) DRL with LSTM trained on random sample data b) DRL with LSTM trained on random tutor data c) DRL with LSTM trained on supermemo data.*

In figure 4.31 which shows the rewards for DRL agents when evaluated on likelihood performance metric for all 40 episodes and all 10 runs, we can see that there is an increase in the reward values as the LSTM gets trained on better interaction data. It can also be noticed that the reward value distribution from DRL agent using LSTM does not spread as much as the distribution from supermemo policy or average of sum of outcomes.

There existed significant differences among DRL agents using LSTM and other policies with $p < .001$ (Kruskal-Wallis test, N=400 per policy). The results from the post-hoc Dunn's test are listed in tables 4.5, 4.6, 4.7 and 4.8 comparing DRL agents using LSTM trained on different sets of interaction history w.r.t. other policies and then later among different DRL agents using LSTM. We can see that the superiority of DRL agent is increasing as it gets trained on better data.

Table 4.5: *Results from posthoc Dunn's test for comparing performance of DRL agent using LSTM (trained on random sample data) to other learning policies.*

Difference	Z statistic	p-value
Random policy − DRL	−8.691978	<.001
Random policy − DRL+LSTM(rs)	−17.08536	<.001
DRL − DRL+LSTM(rs)	−8.393386	<.001
Random policy − Supermemo	−19.70649	<.001
DRL − Supermemo	−11.01451	<.001
DRL+LSTM(rs) − Supermemo	−2.621130	.004

Table 4.6: *Results from posthoc Dunn's test for comparing performance of DRL agent using LSTM (trained on random tutor data) to other learning policies.*

Difference	Z statistic	p-value
Random policy − DRL	−5.988509	<.001
Random policy − DRL+LSTM(rt)	−20.25692	<.001
DRL − DRL+LSTM(rt)	−14.26841	<.001
Random policy − Supermemo	−16.16250	<.001
DRL − Supermemo	−10.17399	<.001
DRL+LSTM(rt) − Supermemo	4.094420	.001

Table 4.7: *Results from posthoc Dunn's test for comparing performance of DRL agent using LSTM (trained on supermemo data) to other learning policies.*

Difference	Z statistic	p-value
Random policy − DRL	−7.638950	<.001
Random policy − DRL+LSTM(sm)	−32.97399	<.001
DRL − DRL+LSTM(sm)	−25.33504	<.001
Random policy − Supermemo	−17.82128	<.001
DRL − Supermemo	−10.18233	<.001
DRL+LSTM(sm) − Supermemo	15.15270	<.001

Table 4.8: *Results from posthoc Dunn's test for comparing performance of DRL agents using LSTM with LSTMs being trained on various interaction data*

Difference	Z statistic	p-value
DRL+LSTM (supermemo) − DRL+LSTM (rand. tutor)	17.42208	<.001
DRL+LSTM (supermemo) − DRL+LSTM (rand. sample)	31.54730	<.001
DRL+LSTM (rand. tutor) − DRL+LSTM (rand. sample)	14.12522	<.001

Chapter 5

Discussion

5.1 Summary of findings

The objective of the thesis was two-fold: first, to get a deeper understanding of the work done by Reddy et. al. [45] and second, to use new reward functions and evaluate their performance on 'likelihood' performance metric given by eq. 3.1. Here, it is important to restate that 'likelihood' maximizes the expected number of items recalled. We considered reward function using likelihood to be a more realistic performance measure since when students prepare for exams, they study with the intention of memorizing as much information as they can.

We studied the variation in reward value as we changed the threshold value, a 'target probability' that the student model's probability of passing was as close as possible to, and we found that the reward value is usually higher at lesser threshold values implying that exercises with less predicted recall likelihood get higher reward. We also conducted experiments to find out if varying the number of exercises has any effect on the performance of the DRL agent on various student models. We found some marginal differences in the performance of DRL agent, not necessarily an increase in performance. In all the cases, the actual reward values were noticed to be reduced. An increase was also noticed for the performance of DRL agent relative to random policy when used on EFC and HLR student model with likelihood based reward function whereas DASH(GPL) student model with likelihood based reward function shows a very small decrease in performance as the number of items increase. We think more tests need to be conducted in order to get more concrete evidence. It could be done by varying the number of exercises by an even larger margin like 50 or 100 and in conjunction with simultaneously increasing the number of steps per episode. Finally, we compared the performance of TRPO algorithm with TNPG. Both algorithms have been shown to give tremendous performance as compared to other batch algorithms on continuous control problem domains with TRPO learning slightly faster than TNPG [16]. However, our results indicate that both of the algorithms learned in a similar fashion in our experiments.

The main objective of this work has been to compare the performance of DRL agent against baseline policies based on the average of sum of outcomes and exploiting LSTMs for reward predictions. We conducted experiments where we trained our DRL agent using average of sum of outcomes for reward prediction and then evaluated the performance of the trained agent on the likelihood performance metric. We observed that the performance of the DRL agent which used average of sum of outcomes for reward was very similar to the case when reward function based on likelihood was used for training of DRL agent which is also what we expected. For our experiments involving LSTMs for reward prediction, we noticed that DRL agent learns good policy smoother and faster when using LSTM for reward shaping as compared to DRL agent using average of sum of outcomes which is congruent with the findings of Su et. al. [53].

5.2 How useful is average of sum of outcomes based reward function?

We believe that the reward functions given by eq. 3.1 and 3.2 used by Reddy et. al. [45] cannot be used in a realistic environment without modelling the student first, using approaches such as BKT[15] or DKT[42], as the predicted reward value is dependent on the student state. We postulate that the reward function based on average of sum of outcomes achieves the same goal as the reward function based on likelihood but the reward function using average of sum of outcomes withdraws from its dependency on the student state as it would be based solely on an observable quantity - correctness of student response. However, using the average of sum of outcomes as a reward function might become computationally expensive as the time it will take to compute reward will grow linearly with the increase in the number of exercises. One solution to counter this problem could be to manually handpick some sample exercises from each category of exercises that could act as representative exercises for that particular category and we can compute the average of sum of outcome on these sample exercises.

5.3 How useful is LSTM for reward prediction?

Using probability of student being correct on an exercise predicted by an LSTM is another approach that we have taken to make reward function independent of underlying student model. Recall that the likelihood based reward function used by Reddy et. al. [45] gives the probability of student being correct on an exercise while being dependent on the student model state. With LSTM reward function, we are predicting the probability of student being correct on an exercise, which could then directly be plugged in replacing the likelihood based reward function. Our results show that when LSTM is used for generating rewards, the DRL agent learns faster which is in accordance with the findings of Su et. al. [53]. However, it comes as a surprise to us that the DRL agent that uses LSTM which was trained on Supermemo interaction data performs so much better than the other two DRL agents trained on random sample data and random tutor data. We speculate the cause for this observation to be reduction in the error of predicting the probability of student being correct on exercises. So, the better the spaced repetition interaction data used for training the LSTM, the lesser the error in predicting the probability of student being correct and hence, better the performance of the DRL agent.

Our results have shown that the average of sum of outcomes based reward function gives comparable performance to the likelihood based reward function. We have also observed that the DRL agent using LSTM, trained on Supermemo data, outperforms DRL agent using the average of sum of outcomes by a large margin. Even though the difference is surprisingly large, the particular setup used is not yet regarded viable for implementation in production systems because of the limitations of computing power.

5.4 Limitations

We did not test our RL agent in real world environment with students as we expect that it will take a lot of time. The parameters of the student simulators were not derived from the distribution of real student data but were just set to reasonable values based on previous published works and we did not look at generalizing to other randomized student parameters. The hyperparameter settings for TRPO has also been very minimally tuned. Another limiting factor could be the use of 200 steps per episode as it also puts limitation on the number of exercises.

5.5 Sustainability and Ethics

This work does not pose any major sustainability or ethical concerns for the society in general.

The only concern from a sustainable point of view that could be pointed out is that the computation time for training both neural networks and reinforcement learning could be huge which in turn would consume more energy resources.

One concern from the ethical perspective is that the data used for training the neural network and RL should be general enough, free from any sort of bias. As with any data-driven application, when there is data there is always a concern for data privacy. If these systems are deployed in production then it becomes very important that the learning progress of students be kept private which could also help in preventing bullying incidents where students, who take longer time to grasp concepts, from being made fun of. Performance of students on tests also influences their motivation and self-esteem, so it becomes imperative that the designed system should be able to consider all these factors into account when being tasked with teaching or reviewing students. In this work, we dealt with only synthetic data so there were no issues pertaining to data privacy and we generated data from student memory models according to previously published works. Another major ethical issue could be about Artificial Intelligence taking tutoring jobs from humans. But, the possibility of that happening will take a long really long time.

ITS might be of great service to future generations in memorizing or mastering the subjects but lots of efforts have to be made in order for them to reach that state. For instance, human tutors can also make use of other factors into consideration when taking tests like facial expressions, tone of the answer, etc. to come to a decision if the student was sure about their answer or not. Achieving something similar would mean utilizing either individual AI systems to read facial expressions, voice, etc. or a very complex system that achieves all of those tasks together to come to a decision. Human examiners also take steps to ensure that the students don't cheat while taking the exam. On the other hand, sometimes human tutors tend to show biases towards certain students which ITS would not be capable of exhibiting. If ITS' are indeed going to be used for reviewing students and we plan on making the whole process automated then we would also have to come up with systems to perform the aforementioned tasks.

5.6 Future Work

One of the most promising extension of this work would be to test the systems in live environments with real student interaction data. It would be really interesting to see how RL with reward shaping performs when compared to Leitner or Supermemo algorithms that have been known to be used on e-learning platforms and give decent performances. In order for DRL agent to not give suboptimal performance during the training phase with real students, student interaction log data from previously used scheduling algorithms such as Leitner [31], Supermemo [5] etc. can be used to pre-train the DRL agent before deploying it into production, which was also mentioned by Reddy et. al.[45].

The other task that could be done in the future would be to use a more complex LSTM network architecture or a GRU network. Also we did not experiment much with the hyperparameters of the LSTM network because of time limitations, tuning the hyperparameters to achieve a better efficiency is another direction. Additionaly it would also be interesting to explore why LSTM trained on Supermemo data makes DRL agent perform so well. In this work, we used RNN for reward shaping, the other extension would be to use neural networks to predict next exercises directly which are to be reviewed by students based on their interaction history. Since we did a minimal hyperparameter tuning for TRPO, as pointed out by Reddy et. al. [45], tuning hyperparameters for TRPO might be able to produce even better results.

Also, this work only deals with a fixed number of items being learned by students but in reality the number of items are dynamic. Sometimes students add items to the collection when they find new information and when students feel that they have fully grasped a particular item then it can also be deleted from the collection. For instance, Quizlet[1] allows user to both add and delete words and their meanings that the user wants to memorize.

[1]https://quizlet.com/

Chapter 6

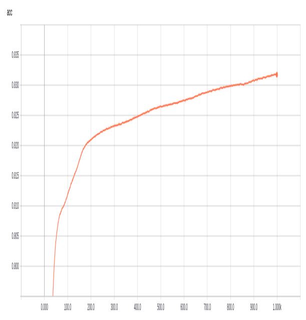
Conclusion

In this work, we demonstrate the utility of DRL in the domain of ITS using synthetic data. Utilizing multiple mathematical human learning models and reward functions, we successfully trained DRL agents and showed that they can perform comparably to traditional benchmarks.

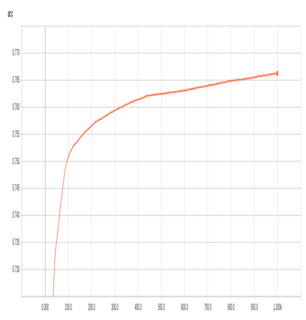
When using a realistically observable reward for DRL agents such as the average of sum of outcomes in a sample of exercises, we were able to get performance on par with traditional benchmarks. Furthermore, this work has shown that LSTM networks can be trained to predict the reward as a kind of reward shaping that also makes DRL agents learn good policies smoother and faster, thus leading to better performance. We have also observed that higher quality of interaction data used to train the LSTM enhances the performance of the DRL agent.

Appendix A

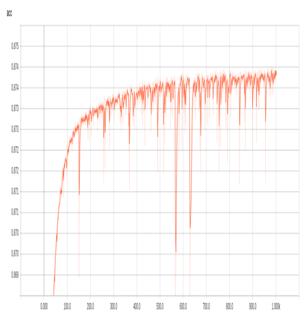
Training plots for LSTM



(a)

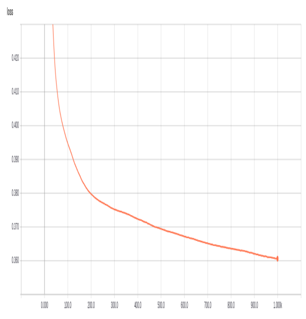


(b)

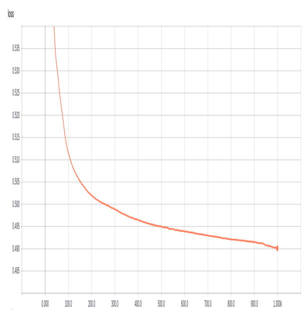


(c)

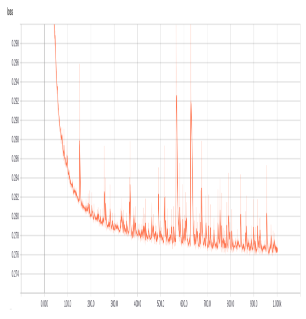
Figure A.1: *Training accuracy for LSTM. a) when trained on random sample data b) when trained on random tutor data c) when trained on supermemo sample data.*



(a)



(b)



(c)

Figure A.2: *Training loss for LSTM. a) when trained on random sample data b) when trained on random tutor data c) when trained on supermemo sample data.*

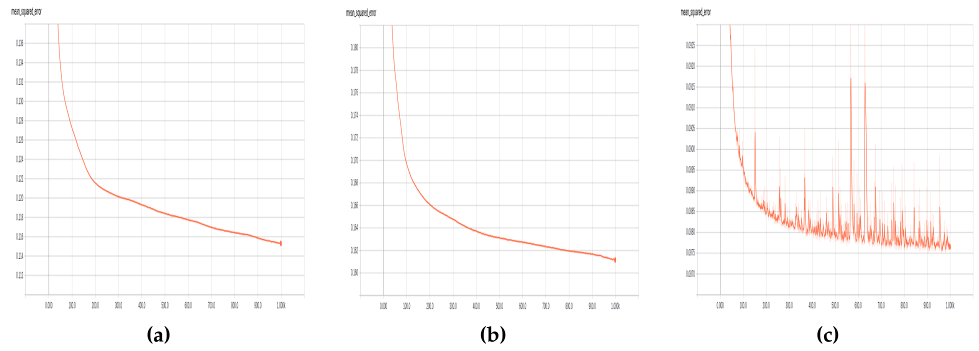


Figure A.3: *Mean squared error for LSTM during training. a) when trained on random sample data b) when trained on random tutor data c) when trained on supermemo sample data.*

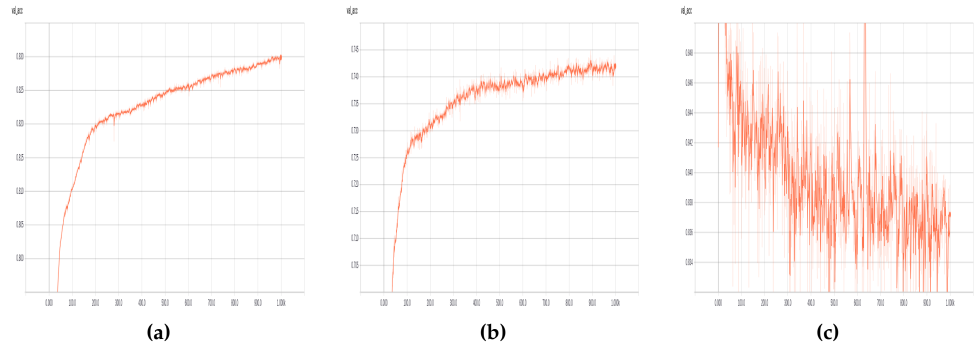


Figure A.4: *Validation accuracy for LSTM. a) when trained on random sample data b) when trained on random tutor data c) when trained on supermemo sample data.*

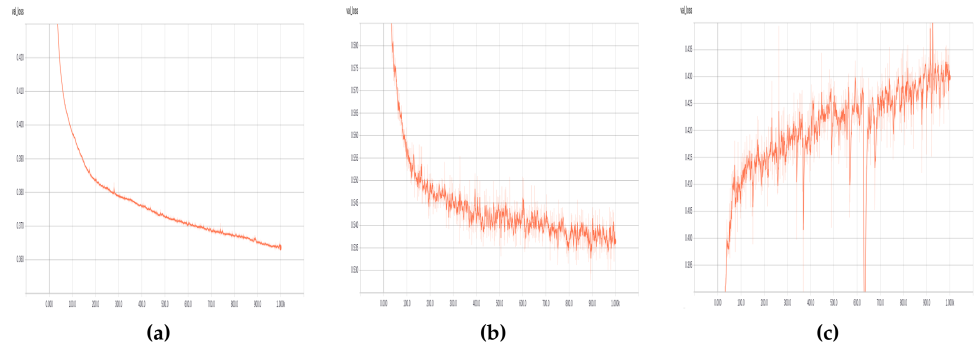
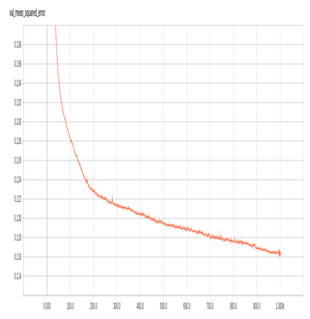
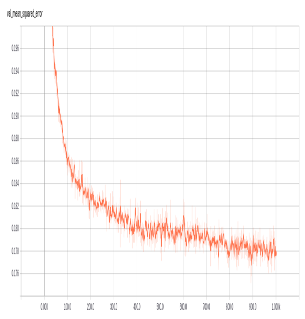


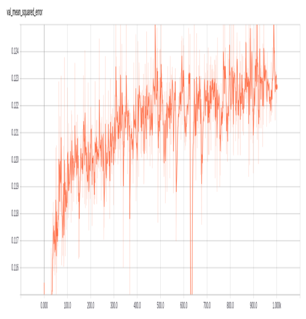
Figure A.5: *Validation loss for LSTM. a) when trained on random sample data b) when trained on random tutor data c) when trained on supermemo sample data.*



(a)



(b)



(c)

Figure A.6: *Mean squared error for LSTM during validation. a) when trained on random sample data b) when trained on random tutor data c) when trained on supermemo sample data.*

Bibliography

[1] https://www.supermemo.com/en/blog/did-ebbinghaus-invent-spaced-repetition. Accessed: June 20, 2018.

[2] Naoki Abe, Edwin P. D. Pednault, Haixun Wang, Bianca Zadrozny, Wei Fan, and Chidanand Apté. Empirical comparison of various reinforcement learning strategies for sequential targeted marketing. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 3–10, 2002.

[3] Rika Antonova. Sample efficient bayesian optimization for policy search: Case studies in robotics and education. Master's thesis, Carnegie Mellon University, 2016.

[4] R. C. Atkinson. Computerized instruction and the learning process. *American Psychologist*, 23:225–239, 1968.

[5] Piotr A.Wozniak. Optimization of learning. Master's thesis, University of Technology in Poznan, 1990.

[6] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016.

[7] A. Barr, P.R. Cohen, and E.A. Feigenbaum. *The handbook of artificial intelligence*. Number v. 1 in The handbook of artificial intelligence. HeurisTech Press, 1982.

[8] Kristine C. Bloom and Thomas J. Shuell. Effects of massed and distributed practice on the learning and retention of second-language vocabulary. *The Journal of Educational Research*, 74(4):245–248, 1981.

[9] A. Bonnet. Artificial intelligence: Promise and performance. 1985.

[10] H. L. Burns and C. G. Capps. *Foundations of intelligent tutoring systems: an introduction*. Lawrence Erlbaum, London, 1988.

[11] David C. Rubin and Amy E. Wenzel. One hundred years of forgetting: A quantitative description of retention. 103:734–760, 10 1996.

[12] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Reinforcement learning for architecture search by network transformation. *CoRR*, abs/1707.04873, 2017.

[13] J. R Carbonell. AI in CAI: An artificial-intelligence approach to computer assisted instruction. *IEEE Transactions on Man-Machine Systems*, 11:190–202, 1970.

[14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[15] 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, Dec 1994.

[16] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016.

[17] Olive Jean Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, 1964.

[18] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back propagating errors. *Nature*, 323:533–536, 10 1986.

[19] Hermann Ebbinghaus. *Urmanuskript 'Ueber das Gedächtniß'*. Passau: Passavia Universitätsverlag, 1880.

[20] Hermann Ebbinghaus. *Über das Gedächtnis*. Leipzig: Dunker, 1885.

[21] Stahl et.al. Play it again: The master psychopharmacology program as an example of interval learning in bite-sized portions. *CNS Spectr.*, 15(8):491–504, 2010.

[22] J. D Fletcher. Intelligent instructional systems in training. *Applications in artificial intelligence*, page 427–451, 1985.

[23] P. A. Games and J. F. Howell. Pair wise multiple comparison procedures with unequal n's and/or variance. *Journal of the American Statistical Association*, Vol 1.:Pp 13–125., 1976.

[24] Seitz J Heller O, Mack W. Replikation der ebbinghaus'schen vergessenskurve mit der ersparnis-methode: 'das behalten und vergessen als function der zeit'. *Zeitschrift für Psychologie*, 199: 3–18, 1991.

[25] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[27] Sean H. K. Kang. Spaced repetition promotes efficient and effective learning: Policy implications for instruction. *Policy Insights from the Behavioral and Brain Sciences*, 3(1):12–19, 2016.

[28] H. J. Keselman, Robert A. Cribbie, and Rand R. Wilcox. Pairwise multiple comparison tests when data are nonnormal. *Educational and Psychological Measurement*, 62(3):420–434, 2002.

[29] William H. Kruskal and W. Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[30] Yann LeCun, Y Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

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

[32] H. Levene and Edited by: I. Olkin. Robust tests for equality of variances. *In Contributions to Probability and Statistics*, page 278–92, 1960.

[33] Robert Victor Lindsey. *Probabilistic Models of Student Learning and Forgetting*. PhD thesis, University of Colorado Boulder, 2014.

[34] C. A. Mace. *The psychology of study*. Oxford, England: Mcbride, 1962.

[35] H. Mandl and A. Lesgold. *Learning Issues for Intelligent Tutoring Systems*. Springer, New York, NY, 1988.

[36] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[37] Michael C. Mozer and Robert V. Lindsey. Predicting and improving memory retention : Psychological theory matters in the big data era. 2016.

[38] Tong Mu, Karan Goel, and Emma Brunskill. Program2tutor: Combining automatic curriculum generation with multi-armed bandits for intelligent tutoring systems. 2017.

[39] Shamim Nemati, Mohammad M. Ghassemi, and Gari D. Clifford. Optimal medication dosing from suboptimal clinical examples: A deep reinforcement learning approach. In *38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2016, Orlando, FL, USA, August 16-20, 2016*, pages 2978–2981, 2016.

[40] Timothy P. Novikoff, Jon M. Kleinberg, and Steven H. Strogatz. Education of a model student. *Proceedings of the National Academy of Sciences*, 2012.

[41] Hyacinth S. Nwana. Intelligent tutoring systems: an overview. *Artificial Intelligence Review*, 4(4):251–277, Dec 1990.

[42] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 505–513. Curran Associates, Inc., 2015.

[43] Niranjani Prasad, Li-Fang Cheng, Corey Chivers, Michael Draugelis, and Barbara E Engelhardt. A reinforcement learning approach to weaning of mechanical ventilation in intensive care units. *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 1–9, 2017.

[44] Siddharth Reddy, Igor Labutov, Siddhartha Banerjee, and Thorsten Joachims. Unbounded human learning: Optimal scheduling for spaced repetition. *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2016.

[45] Siddharth Reddy, Sergey Levine, and Anca Dragan. Accelerating human learning with deep reinforcement learning. *Conference on Neural Information Processing Systems*, 2017. Workshop paper.

[46] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. *ICML*, 2015.

[47] Burr Settles and Brendan Meeder. A trainable spaced repetition model for language learningac. *ACL(1)*, 2016.

[48] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[49] V. Shute. Regarding the I in ITS: Student Modeling. *Proceedings of ED-MEDIA 94*, 1944.

[50] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[51] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–, October 2017.

[52] D. Sleeman and J. S. Brown. *Intelligent tutoring systems*. Academic Press, 1982.

[53] Pei-hao Su, David Vandyke, Milica Gasic, Nikola Mrksic, Tsung-Hsien Wen, and Steve J. Young. Reward shaping with recurrent neural networks for speeding up on-line policy learning in spoken dialogue systems. *CoRR*, abs/1508.03391, 2015.

[54] Patrick Suppes and Mona Morningstar. Computer-assisted instruction. *Science*, 166(3903):343–350, 1969.

[55] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[56] E.L. Thorndike. *Animal Intelligence: Experimental Studies*. Animal behavior series. Macmillan, 1911.

[57] Kurt Vanlehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46:197–221, 10 2011.

[58] Wickelgren WA. *Learning and memory.* Englewood Cliffs, NJ: Prentice-Hall, 1977.

[59] B. L. Welch. The generalization of "student's" problem when several different population variances are involved. *Biometrika*, 34 (1–2):28–35, 1947.

[60] E. Wenger. *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmanne, Los Altos, CA, 1987.

[61] J. T. Wixted and S. K. Carpenter. The wickelgren power law and the ebbinghaus savings function. *Psychological Science*, 18(2):33–134, feb 2007.

[62] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. Deep reinforcement learning for list-wise recommendations. 12 2017.

TRITA -EECS-EX-2019:217