Investigating Knowledge Tracing Algorithms and Learner Simulators for Training Adaptive Recommendation Agents in Education

Daniel Farrell, B.Sc.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Prof. Vincent Wade

August 2021

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Daniel Farrell

August 31, 2021

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Daniel Farrell

August 31, 2021

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my research supervisor, Professor Vincent Wade, whose knowledge, guidance, and mutual passion for education technology have made this journey enjoyable and insightful throughout. Your support is greatly appreciated.

Additionally, I would like to thank my professors and classmates at Trinity College Dublin. It has been a challenging and unprecedented year for all, but despite the difficulties faced, we all came together to see it through until the end. You should all be extremely proud.

Last but not least, a huge thank you to my friends and family for their continuous support.

Daniel Farrell

University of Dublin, Trinity College
August 2021

Investigating Knowledge Tracing Algorithms and Learner Simulators for Training Adaptive Recommendation Agents in Education

Daniel Farrell, Master of Science in Computer Science

University of Dublin, Trinity College, 2021

Supervisor: Prof. Vincent Wade

The research and implementation of intelligent agents for recommending educational content to students has long been hindered by three major necessities: a suitable knowledge tracing algorithm, the possession of extensive historic learner data, and an access to an abundance of human learners.

As there is a lack of comparative analyses investigating knowledge tracing algorithms through experimentation, engineers are often left wondering which algorithm is best suited for their educational recommender systems. This paper seeks to provide clarity by conducting an in-depth comparative analysis of modern knowledge tracing algorithms, namely: Bayesian Knowledge Tracing (BKT), Bayesian Knowledge Tracing with Forgetting (BKT+F), Performance Factors Analysis (PFA), and Deep Knowledge Tracing (DKT).

Typically, in order to develop educational recommendation systems, large amounts of historic data and human learners are required. However, this dissertation investigates a solution for training these recommendation systems without having access to these resources. To do this, this dissertation puts forward the design and implementation of a robust simulated learner with the ability to answer exercises and to build up skill levels through practice, all facilitated through knowledge tracing algorithms. As they mimic human learner behaviour, learner simulators can be used to interact with and train agents for recommending educational exercises.

To prove this works, this dissertation implements a recommendation agent trained entirely using learner simulators. It is then shown that learner simulators carrying out exercises following an adaptively recommended learning path experience larger improve-

ments in tests scores to those following a randomly assigned learning path. It is found that learner simulators following an adaptive learning path can improve their test scores by 33% using just 9 educational exercises, however this takes 15 exercises to accomplish following non-adaptive learning paths.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nowadays, more and more schools and universities are moving over to online platforms as a means of digitising student learning. We have seen this in recent years with the emergence of popular online educational platforms such as Coursera, Udemy, and even Blackboard. These platforms are the perfect place for institutions and educators to disseminate educational material, and to monitor their students' performances. Students carry out tasks, navigate educational content, submit assignments, and receive their grades all through these platforms. We have all of this learner data available to us, but how is it being used to provide a better learning experience for students?

Intelligent pedagogical systems can greatly improve educational gain and experience for both teaching staff and students by aggregating, analysing, and reporting on learner data. Using learner analytics, systems have been put in place for the early detection of students at risk of failure (Wolff et al. (2014)), for providing personalised feed-back on educational activities (Awais et al. (2019)), and for recommending educational content that students with similar learning styles found useful (Nurjanah (2016)). As shown, learner data can lead to some exceptionally intelligent educational systems. But can it be used to effectively recommend educational content to the individual student?

Due to the rising prominence of online learning platforms, adaptivity in education has become a huge topic of research. Everyone has different learning motivations, skills, and levels of knowledge, but these traits are not typically catered to. Traditionally, educational materials have been disseminated to students in a predefined fashion, not taking into account the dynamic behaviour and skill levels of the individual. Personalised learning looks at ways of adapting education to the individual in order to provide them with the best educational gain that is suited to their personal skill levels and preferred learning methods. Although it is possible to personalise learning on many axes, this dissertation focuses on personalisation concerning student skill levels.

A growing trend amongst the personalised learning research community, is the investigation of intelligent systems that can recommend educational content to students. These systems can be built for various different goals, such as recommending study content that aligns with topics that students are struggling with, or for recommending a learning path for students that will take them from a beginner level to a master of the topic at hand. They can also consist of various different underlying technologies, such as large-scale knowledge graphs or deep neural networks. But two things these systems all have in common, they all require large amounts of historic data to be implemented, and the researcher/developer will need to have access to an abundance of human resources (learners) for training and evaluation. This project focuses on this exact topic, seeking to investigate methodologies for developing educational recommendation systems that do not require any historic data or human resources. The usage of the discoveries and technology developed in this dissertation are particularly suitable for early design, development, and testing of reinforcement learning based educational recommendation systems. However, the usage of human learners in the final verification and deployment testing would still typically be required.

Going forward, this chapter will lay out the motivation behind the study, discuss the research questions at hand, and highlight the overall objectives of the research. A discussion on the potential contributions that this research may provide to the research area is also given, along with a general overview of the structure of the report.

1.1 Motivation

When implementing any form of an adaptive learning platform, it is essential to maintain a model for all students interacting with the system. These models are essentially an encapsulation of information that describes the student. Depending on the goals of the platform, this information could be details such as their preferred learning methods, skill levels, previous grades, or topics of interest. When building systems for adaptively recommending educational content to students, particularly in a bid to maximise their learning gains, it is pinnacle that these models encapsulate the level of knowledge the student possesses in the topics available. This is so that content that is most likely to improve their level of knowledge in the course can be recommended to them.

Knowledge tracing algorithms, which will be investigated later in the report, provide a means of modelling and quantifying a student's level of knowledge in all topics being taught to them. These knowledge tracing algorithms are the backbone to many of the studies being carried out in educational recommender systems. But there is one major

problem when working with these algorithms, they require a large amount of historic data that capture student learning sequences. Datasets that intricately depict student learning paths as they navigate some educational content are quite scarce, as of course most popular platforms will not release this kind of information to the public. If the researcher or developer does not have access to this type of data, how will they test, train, or even create their own knowledge tracing algorithms?

Although much literature around knowledge tracing algorithms do exist, these tend to report the usage and application of specific knowledge tracing algorithms (Liu et al. (2021)), however there is a lack of comprehensive comparative analysis of these algorithms via experimentation. Therefore there is a need to investigate how different knowledge tracing algorithms perform across different learner datasets.

When creating an educational-based recommender system, particularly using reinforcement learning, more concerns arise. Reinforcement learning agents notoriously require a huge amount of interactions (such as students answering assignments recommended to them) in order to train and achieve an effective recommendation policy. Since most developers will not have access to a large amount of human testers, how will they train their reinforcement learning agent to assign educational content effectively? If they cannot train their agent, then they will not be able to research and create new methods for recommending educational content. And how would they evaluate their recommender systems to ensure they are functioning effectively before deploying it to a real online system?

1.1.1 How can Learner Simulators Help?

As a result of the motivational problems highlighted, this dissertation seeks to research and develop a system for adaptively recommending educational content that does not rely on any historic data or human resources.

To do this, a learner simulator will be created that simulates student behaviour. Specifically this means the learner simulator can carry out tasks such as answering assignments and building up their skill levels in the topics being taught, simulating realistic human learner behaviour. The learner simulator will not require any pre-training on historic data and therefore eliminates the need for researchers or engineers to possess large historic datasets in the early development of adaptive educational systems.

These learner simulators will also be used to train and evaluate an intelligent reinforcement learning agent for adaptively recommending educational content. This means a researcher would no longer require access to a vast amount of human resources for testing their systems as these simulators essentially imitate how a real learner would interact

with the system. This way researchers and developers can train, test, and configure their own reinforcement learning policies before deploying their recommender system to a real educational platform. We are not suggesting that the final verification of the recommendation system should not involve human trials, but rather that a significant effort in the development of the educational recommender can be greatly relieved through the use of learner simulators prior to deploying it for human learners to interact with.

1.2 Research Questions

This research paper seeks to answer the question of whether an effective intelligent agent used for the adaptive recommendation of educational content can be built without the use of historic data or human resources. Particularly, the system proposed will operate with the goal of maximising students' educational gain whilst minimising the amount of content recommended to them (information overload).

In order to investigate this research question, the following four sub-questions will be tackled:

- How do state-of-the-art knowledge tracing algorithms compare to each other when trained on historic learner sequence data? What are their strengths and weaknesses based on the analysis?
- Can knowledge tracing algorithms be used to create learner simulators for imitating student learning behaviour?
- Can these learner simulators be used to adequately train an intelligent recommendation agent for adapting student learning paths?
- How does an adaptive learning path compare to a suitable baseline learning path? Does it provide any gains in student knowledge?

1.3 Research Objectives

In order to answer the research questions defined above, the following objectives have been set out:

- Carry out a case study on current state of the art knowledge tracing algorithms. This case study will provide a further understanding of how these algorithms operate and how they perform when historic data is available to the user. An analysis and comparison of their performance will be carried out. The state of the art algorithms

used will be Bayesian Knowledge Tracing (BKT), Performance Factors Analysis (PFA), and standard Deep Knowledge Tracing (DKT).

- Implement a generalised learner simulator that can be used to imitate student learning behaviour as they carry out assignments and class tests. Bayesian Knowledge Tracing will be the algorithm used to model the simulator's knowledge state and to carry out it's decision making process.

- Using this learner simulator, implement a reinforcement learning based system for adaptively recommending educational content. The goal here is to maximise the educational gain (increase in knowledge) whilst minimising information overload (number of assignments). An actor-critic reinforcement learning method will be used for this part of the research.

- Evaluate the system by comparing the learning path and knowledge increase of the simulators as they follow an adaptive learning path versus an appropriate baseline learning path.

1.4 Contributions

By carrying out the objectives highlighted, this project will contribute to the research area for multiple reasons.

By providing a case study on multiple state-of-the-art knowledge tracing algorithms, researchers and engineers can get a sense of how these algorithms compare to each other, in terms of their implementation and their performance when historic data is available. Four popular knowledge tracing datasets are investigated during this case study along with an evaluation on how each knowledge tracing algorithm performs on them. This evaluation could prove useful to academics researching the same datasets or algorithms.

As stated previously, the implementation of a learner simulator can provide many uses to researchers carrying out work in adaptive education. The use of a learner simulator eliminates the need for large amounts of historic data and human resources. Engineers and researchers can now train and test recommendation systems much more conveniently with reduced effort, shortened development time, and a greater confidence before deploying it to a real online platform.

An educational-based recommendation system using reinforcement learning methods is implemented and investigated as part of this report which will contribute to current research in adaptive education and recommender systems.

The contributions to the research area highlighted in this section pave ways for re-

searchers to work with or to build upon the technologies proposed as part of this dissertation.

1.5 Dissertation Outline

The rest of this dissertation will be organized as follows:

- Chapter 2 will give a background overview of the theories and technologies used in the dissertation. It will also present and discuss some state-of-the-art methodologies.
- Chapter 3 will document a case study carried out evaluating and comparing 4 state-of-the-art knowledge tracing algorithms.
- Chapter 4 will discuss the design and implementation of a learner simulator based on Bayesian Knowledge Tracing.
- Chapter 5 will describe the design and implementation of an educational content recommender system trained using learner simulators.
- Chapter 6 will discuss and evaluate the recommender system, comparing it against some baseline methods.
- Chapter 7 will conclude the dissertation with some final thoughts and potential future work.

Chapter 2

Background & State of the Art

The chapter will give some background information on the fundamental components of this dissertation, namely: knowledge tracing, reinforcement learning, and educational content recommendation. For each of these components, some state-of-the-art techniques will be presented and discussed. The information provided in this chapter will be the backbone of the system proposed later in the report, therefore it is pinnacle for the understanding of the chapters ahead.

2.1 Knowledge Tracing

In intelligent learning based systems, it is imperative to construct a model or "knowledge trace" for all students interacting with the system. A student's knowledge trace is essentially a representation of their proficiency in all skills being taught in the course. For example, in a mathematics course, a student may be extremely proficient at finding the roots of a quadratic equation, but struggle with differentiation. If you keep a knowledge trace of a student's proficiency levels, their weaknesses can be identified and educational content could be recommended to them to help them improve.

But how are these knowledge traces constructed? Knowledge tracing algorithms are used for just that. Their job is to analyse a student's responses to assignments or tests and to construct a machine readable representation of the student's proficiency levels in all skills being taught to them. These knowledge traces are then constantly updated as students progress through the course.

Going forward, 3 state-of-the-art knowledge tracing algorithms will be presented and discussed, namely: Bayesian Knowledge Tracing, Deep Knowledge Tracing, and Performance Factors Analysis.

2.1.1 Bayesian Knowledge Tracing

Bayesian knowledge tracing (BKT), originally proposed in 1994 (Corbett and Anderson (1994)), is one of the earliest and most commonly used knowledge tracing algorithms of recent times. The algorithm was first created to be deployed on an ITS for students learning programming. The purpose of the algorithm is to model the students' change in knowledge state over time as they carry out some exercises.

The goal of BKT is to map the performance of a student (observable variable) to an estimate of their level of knowledge (latent variable). In BKT, the student's latent knowledge is represented as a set of binary variables, where each of these variables represent the understanding or non-understanding of a particular skill. A skill is simply a concept that relates to the course such as addition or subtraction in mathematics. A Hidden Markov Model is then used to update these variables as a student answers questions correctly or incorrectly on a given skill.

The BKT model is a dynamic Bayesian network of two states, which can be seen in Figure 2.1 below. The model observes when a student is presented with a problem requiring a specific skill and assumes the student either knows the skill (Learned) or does not know the skill (Not Learned), and the student can either answer the question correctly or incorrectly.

A student who knows the skill behind the question will typically answer the question correctly, but in certain cases they might make a mistake and get the question wrong despite knowing the skill, this is called a slip and is represented by the probability P(S). Inversely, a student who does not know the underlying skill behind a question will typically answer the question incorrectly, but in certain cases they can make a guess and get the question correct regardless of not knowing the skill, this guess is represented by the probability P(G). Each student also has a certain probability of knowing the skill before answering the question, and this is represented by the probability P(L), which we will see the formula for later. There also exists a certain probability P(T) that the student will display their knowledge of the skill and move from the "Not Learned" state to the "Learned" state.

The four parameters used in BKT are described below. These four parameters exist for every skill the user encounters.

- P(L) or p-init, the probability of the student knowing the skill before answering the question.
- P(T) or p-transit, the probability of the student demonstrating their knowledge of the skill and moving from a "Not Learned" state to a "Learned" state.

- P(G) or p-guess, the probability of the student answering correctly despite not knowing the skill.
- P(S) or p-slip, the probability of the student answering incorrectly despite knowing the skill.

In Figure 2.1 below, the process of mapping observable variables (correct/incorrect answers) to latent variables (knowledge level) can be seen clearly. It can also be observed how a binary representation of the student's latent knowledge state will begin to form as they answer more questions on the skills in the course.

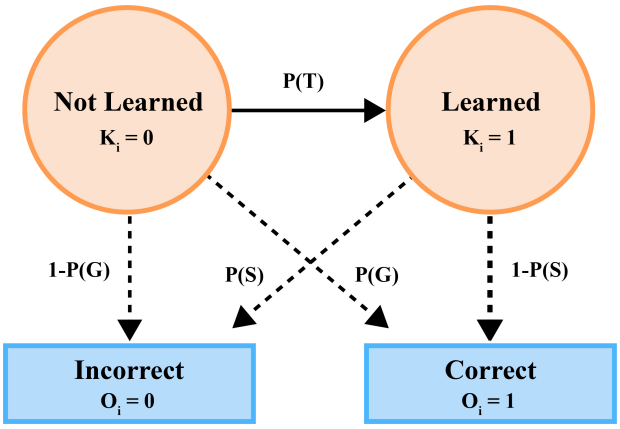

Figure 2.1: BKT Hidden Markov Model

The estimate of the level of knowledge a student possesses on a particular skill is then continuously updated every time they answer questions correctly or incorrectly on that specific skill. That process can be described by the following sequence of steps:

1. If the student gets the answer correct, the conditional probability is found using Formula A. Inversely, if the student gets the answer incorrect, the conditional probability is found using Formula B.
2. The conditional probability is then used to update the probability that the student now knows the skill p(L) and follows Formula C.
3. The probability that the student will apply their knowledge to answer the next question correctly p(C) (predict) is found using Formula D.

A
$$p(L_t|obs = correct) = \frac{p(L_t) \cdot (1 - p(S))}{p(L_t) \cdot (1 - p(S)) + (1 - p(L_t)) \cdot p(G)}$$

B
$$p(L_t|obs = incorrect) = \frac{p(L_t) \cdot p(S)}{p(L_t) \cdot p(S) + (1 - p(L_t)) \cdot (1 - p(G))}$$

C
$$p(L_{t+1}) = p(L_t|obs) + (1 - p(L_t|obs)) \cdot p(T)$$

D
$$p(C_{t+1}) = p(L_{t+1}) \cdot (1 - p(S)) + (1 - p(L_{t+1})) \cdot p(G)$$

2.1.2 Deep Knowledge Tracing

Similar to Bayesian Knowledge Tracing, the purpose of Deep Knowledge Tracing (DKT) is to effectively model students' latent knowledge as they navigate some problem space. The main difference is that DKT uses an underlying Recurrent Neural Network for modelling as opposed to a hidden Markov model. As a result, DKT can model more complex representations of human knowledge.

DKT was first proposed in 2015 and was a major improvement upon previous knowledge modelling methods (Piech et al. (2015)). It was one of the first methods to not require explicit encoding of skill tags as these can be learned from the data directly. This is because in previous methods (such as BKT), it was necessary for the creator to provide tags for each question highlighting the skill it required (e.g. slope of a line, roots of equation), but in DKT the model can automatically learn the content substructure.

To model a student's knowledge of a certain problem space, DKT first converts the student's actions into a fixed length input vector x_t. When a student answers M exercises, this input vector is of length M where each digit represents a single exercise and is 1 when answered correctly and 0 when answered incorrectly. This interaction vector is then fed into the RNN. The RNN output y_t is then a vector equal in length to the total number of exercises in the problem space where each number represents the probability of the student answering that exercise correctly. These probabilities of students answering problems correctly then becomes the knowledge model for that student as it encapsulates how well they perform in different areas. Adaptive educational platforms can then use this information to recommend problems to the student that they are struggling with, encouraging them to practice and learn.

Figure 2.2 displayed below shows a student's interaction vector x_t being inputted into

the DKT model along with their corresponding outputs y_t.

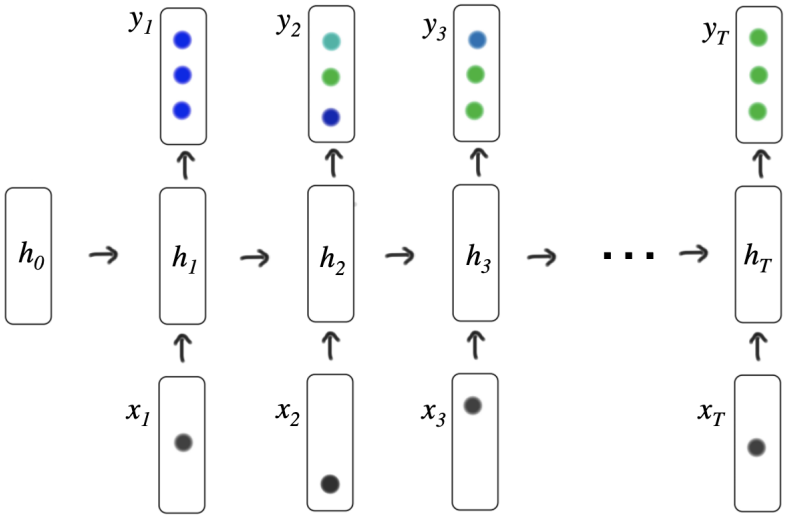

Figure 2.2: DKT predicting student knowledge

2.1.3 Performance Factors Analysis

In some knowledge modelling methods like BKT, each exercise in the problem space is considered to require only one skill. In reality, this is not normally the case, most exercises require multiple skills from the learner in order to be answered correctly. That is one problem area that Performance Factors Analysis (PFA) was created for in order to overcome. Originally proposed in 2009 (Pavlik et al. (2009)), PFA was created as an alternative to the popular knowledge tracing algorithms at the time such as BKT. As well as considering each exercise to require multiple skills, PFA also models skill mastery as a level of competency between 0 and 1.

The formula used for modelling the student's accumulative learning is shown below. Here β is a measure of the "easiness" of the skill on display, s tracks the prior successes for that skill shown by the student, f tracks the prior failures for that skill shown by the student, and γ and ρ scale the effect of these measures. Once this has been calculated, it is passed through a logit function to output the final probabilities.

$$m(i, j \in KCs, s, f) = \sum_{j \in KCs} (\beta_j + \gamma_j s_{i,j} + \rho_j f_{i,j})$$

In Figure 2.3 below, we can see data from a student practicing a given skill. In the final column we can also see the output from the PFA model which is an estimate of the student's strength in the practiced skill. This is how PFA models a student's competency in a given skill as a value between 0 and 1.

Student	Correct	Subgoal	Skill	Model Pred.
a51864	1	a51864C101019	S44IdentifyGCFonenumbermultipleofother	0.6058313
a51864	0	a51864C10810	S44IdentifyGCFonenumbermultipleofother	0.6351781
a51864	1	a51864C101020	S44IdentifyGCFonenumbermultipleofother	0.6089495
a51864	1	a51864C10796	S44IdentifyGCFonenumbermultipleofother	0.6382028
a51864	1	a51864C101021	S44IdentifyGCFonenumbermultipleofother	0.6664663
...	...	...	...	...

Figure 2.3: Example of input data and predictions

2.1.4 Surveys on Knowledge Tracing Algorithms

There have been some surveys carried out on knowledge tracing algorithms prior to this research that attempt to compare their different applications and technologies.

One particular survey sought to provide a comprehensive overview of the many different variants of knowledge tracing algorithms (Liu et al. (2021)). Particularly it focused on presenting a technological breakdown of the algorithms as well as providing some applications of these knowledge tracing algorithms in educational scenarios. One weakness the paper identified about BKT and PFA is how they model skills independently and therefore cannot take into consideration relationships between skills. Of course in actual learning, skills can often be somewhat related to each other, for example counting and addition. One would not be able carry out addition without being able to count. As a result, a strength of DKT is its ability to capture and model these inter-skill relationships.

In another research paper, the authors sought to compare the machine learning techniques used for knowledge tracing sourced from 51 research papers on the topic (Ramirez et al. (2021)). They parsed these papers to automatically extract details such as the machine learning technique that was used in the research, the purpose of the machine learning technique, and the rational for using the specific technique. Through this they found that the most common machine learning methods in knowledge tracing were techniques such as BKT, DKT, Bayesian Networks, and Support Vector Machines. By parsing the rationals behind using these specific machine learning methods, they found that DKT was commonly used as it handles sparse data much better than other methods. They also found that BKT often faced difficulties in situations where questions required more than

one skill or in situations where the practice of a specific skill is spread out over a long time.

Although both of these papers seek to compare knowledge tracing algorithms both in terms of their technology and applications, they do this through an extensive literature review and do not not work with the algorithms themselves. For this reason there is a need to carry out a comparative analysis of these algorithms through experimentation, which part of this dissertation seeks to do.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning technique whereby an intelligent agent can learn to carry out specific tasks by interacting with an environment and receiving feedback consequences) for the actions they take. These agents can be trained to carry out tasks such as playing games (Pac-Man, chess, checkers), avoiding collisions in self-driven cars, or for recommending educational content to students that will increase their knowledge gains. Any reinforcement learning workflow consists of 5 main entities: an Agent, an Environment, a list of Actions, a Reward function, and a continuously updated description of the environment called the State.

As an example of a typical reinforcement learning process, suppose you are attempting to train an intelligent Agent to become a master of the 1980 classic arcade game, Pac-Man. In this case, the agent (Pac-Man) must learn how to navigate around a maze eating as many dots as it can whilst avoiding ghosts that are trying to catch him. In this example, Pac-Man would be the Agent, and the maze would be the Environment. The Actions in this case would be to move left, right, up, or down. After every action, Pac-Man will receive a reward if it has eaten a dot, and receive a penalty if it does not eat a dot, or perhaps for coming too close to a ghost. The State would then be information about how the environment looks after the action has been taken, such as the locations of the dots, ghosts, and Pac-Man. Figure 2.4 below depicts this reinforcement learning process, which of course is universal to any application, not just this Pac-Man example.

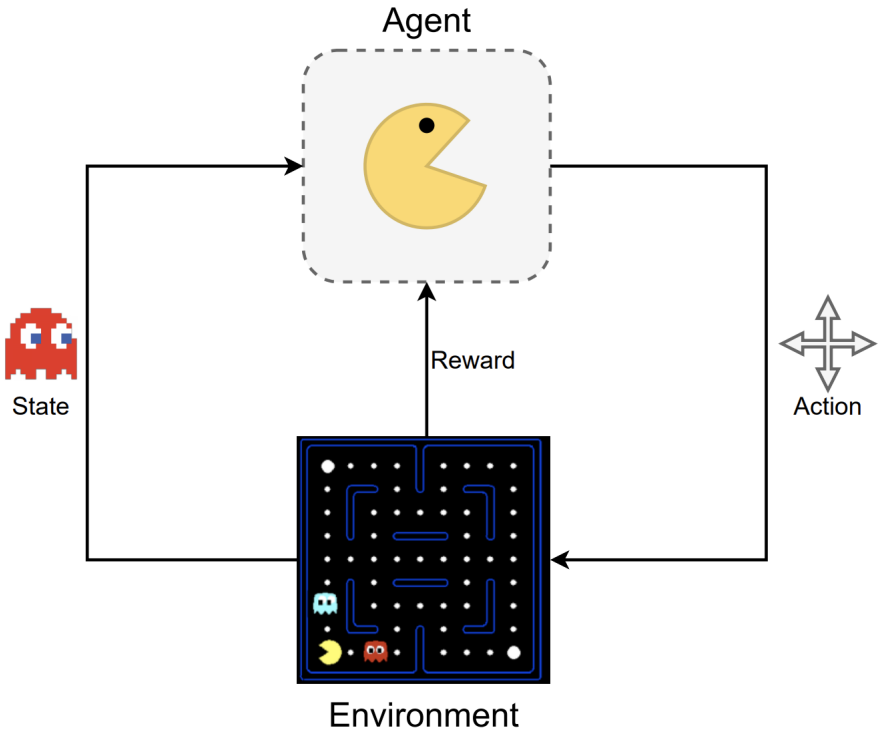

Figure 2.4: Reinforcement Learning process

After the Agent repeats this process of taking an action and receiving a reward or a penalty for said action, it will begin to realise the strategies it should take in order to receive the best reward. This is how the Agent become more intelligent and proficient at carrying out the tasks it has been designed to do. These strategies that an Agent learns in reinforcement learning are referred to as policies.

2.2.1 Actor-Critic Methods

In this dissertation, the reinforcement learning method used for recommendation will be an Actor-Critic method. In Actor-Critic reinforcement learning methods, an Agent is made up of two components. These two components are the policy function, and the value function. The role of the policy function, known as the Actor, is to select what Action to take next. In terms of the Pac-Man example, these actions would be to go left, right, up, or down. The role of the value function, known as the Critic, is to critique the actions that the Actor is selecting. Specifically, this means the Critic calculates the total Reward the agent can expect to receive if the Actor continues to follow its current policy. Since the Critic is constantly learning about and critiquing the policy currently being followed by the Actor, this is considered on-policy learning. On-policy learning

algorithms are essentially algorithms that constantly attempt to evaluate and improve upon the policy that is currently being used to choose actions.

After an action is selected, the Critic evaluates the new state and determines whether the result is better or worse than expected. The critique given by the Critic takes the form of Temporal Difference (TD) error. The TD error is calculated using the formula below:

$$\delta_t = r_{t+1} + \gamma V_{(S_{t+1})} - V_{(S_t)}$$

Where r_{t+1} represents the observed reward for the action taken, $V_{(S_{t+1})}$ represents the estimate of total reward that will be received from the new state until the end of the cycle following the current policy, γ represents a discount rate, and $V_{(S_t)}$ was the estimate of the total reward the Critic calculated in the previous state. From this you can see that the TD error at each time is the error in the value estimate made by the Critic.

The typical workflow of an Actor-Critic reinforcement learning scenario has been depicted below in figure 2.5.

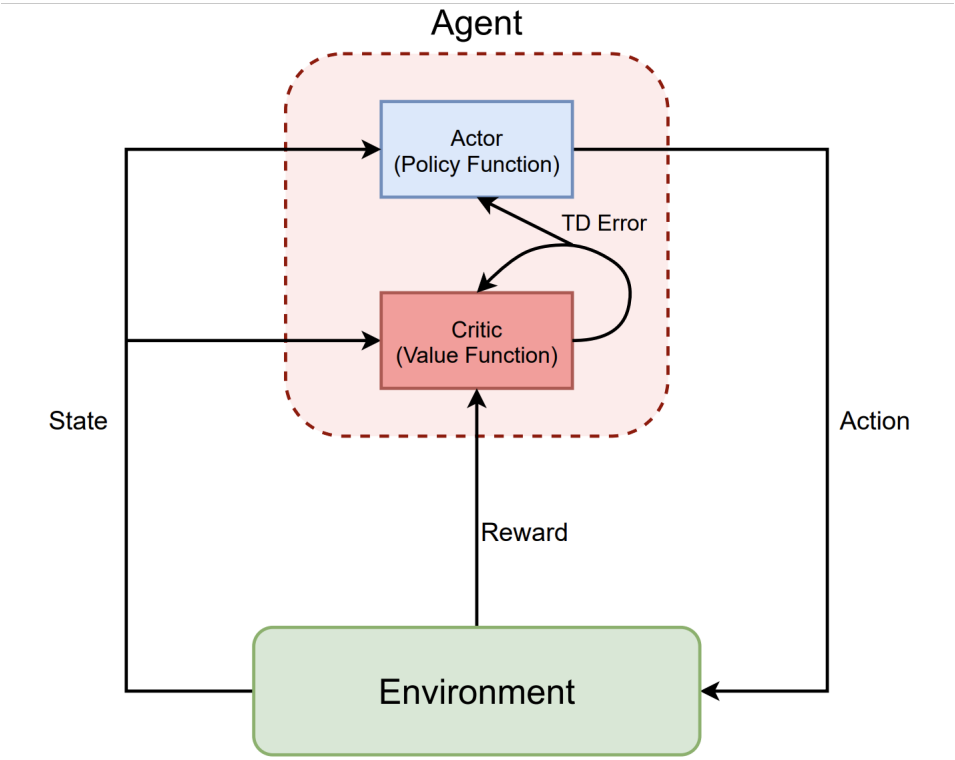

Figure 2.5: Actor-Critic Workflow

The internal workings of an Actor and Critic are achieved using neural networks. Specifically, the Critic is a neural network that takes in a list of Actions and States, and as mentioned previously it attempts to estimate the total reward that will be received from the current state to the end of the cycle following the current policy. This is depicted in figure 2.6 below. Here we can see the outputs are the value (or Q-Value) estimates.

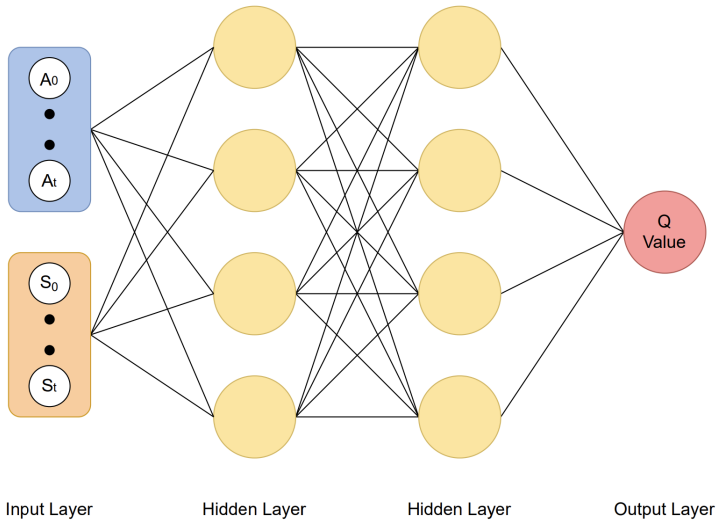

Figure 2.6: Structure of Critic

Similarly, the Actor is also a neural network, it takes in the current state and the neural network returns a probability distribution over the possible actions that the Actor can take based on the input state. The structure of the Actor is depicted in figure 2.7 below.

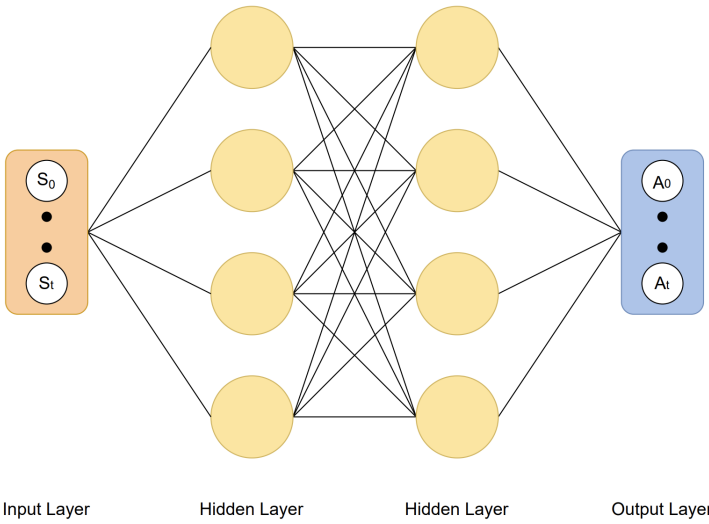

Figure 2.7: Structure of Actor

2.3 Reinforcement Learning in Education

Over the years there have been multiple research projects carried out investigating the use of reinforcement learning for educational content recommendation.

One such research project was the implementation of RILS a reinforcement learning system that could analyze learner information and in turn recommend appropriate educational content that is best suited for the specific learner (V.R et al. (2014)). RILS derives learner information through both explicit feedback (ratings, performance, etc), and implicit feedback (context of recommended content). Following this, the learner information stored about each student increases as they are recommended more content. Using this learner information, the reinforcement learning agent then recommends some educational content to the user. The agent does this by predicting the feedback that it will get from the student for all possible content it can recommend. The agent chooses the content that it believes will get the best feedback from the user, in terms of their performance and their explicit rating of the content. After the user carries out the content, their actual performance is calculated along with the explicit rating the student gives the content. The difference between the feedback predicted by the RL agent and the actual feedback then becomes the reward for the agent. This process repeats until the agent derives at an effective recommendation policy.

2.4 Summary

The components discussed as part of this chapter will by highly relevant and form the basis of the technology used in the dissertation.

The state-of-the-art knowledge tracing algorithms discussed at the start of this chapter, namely: Bayesian Knowledge Tracing(BKT), Deep Knowledge Tracing(DKT), and Performance Factors Analysis(PFA), will be prominent throughout the rest of this report, starting with a comparative analysis of their performance on some historic data.

Actor-critic based reinforcement learning methods will be used later in the project when building a recommender system for educational content. This recommender system will follow a procedure similar to that of RILS as discussed previously.

Chapter 3

Knowledge Tracing Comparative Analysis

For the first part of this dissertation, the goal is to develop and compare knowledge tracing algorithms that are suitable for use on historic data. Firstly, this chapter will give an overview of the experiment set-up, highlighting the design choices, datasets, and the tools and technologies used.

3.1 Overview

As there is currently a lack of comprehensive comparative analyses carried out on knowledge tracing algorithms, this part of the dissertation focuses on just that. The goal of this experimentation is to discover how knowledge tracing algorithms perform when trained on a range of historic data and to investigate how different algorithms compare to each other when trained on the same data. What are the strengths and weaknesses of these algorithms? How does the type of training data affect performance? Which algorithms typically perform the best?

The objective here is to compare 4 state-of-the-art knowledge tracing algorithms when trained on 4 well researched datasets. The four algorithms being compared as part of this analysis are: Bayesian Knowledge Tracing(BKT), Bayesian Knowledge Tracing with Forgetting(BKT+F), Performance Factors Analysis(PFA), and Deep Knowledge Tracing(DKT). Each knowledge tracing model will be trained on each of the datasets and their ability to predict student performance in the future will be evaluated.

3.2 Datasets

To provide an analysis of the performance of all 4 knowledge tracing algorithms, 4 suitable datasets were first chosen. The criteria for the selection of these datasets included:

- Public availability
- Wide use in research literature
- Range of different contexts and subjects
- Inclusion of necessary variables for knowledge tracing such as student ID's, skill tags, question ID's, and correctness of responses

Following this criteria, the datasets selected for this comparative analysis were: ASSISTments 2009, ASSISTments 2015, Spanish 2013, and Statics 2011. The reason 4 datasets were selected for this analysis was so that the knowledge tracing algorithms could be compared across datasets with different contexts, different students partaking, and following different teaching methods. Other knowledge tracing datasets do exist, but 4 was considered sufficient to compare the performance of all knowledge tracing algorithms.

The following chapter will discuss each of the datasets used in the comparative analysis.

3.2.1 ASSISTments 2009

ASSISTments is an online math tutoring platform aimed at grade school students in the United States. Each question in the dataset is considered to require one skill component e.g. subtracting integers, Venn diagrams, area of a rectangle, etc. In total the 2009 ASSISTments dataset consists of 3,241 students, 17,709 questions, 278,868 student interactions, and 124 skill components.

3.2.2 ASSISTments 2015

Like the 2009 ASSISTments dataset, the 2015 version was compiled in a similar manner, using their online math tutoring platform. This course did not contain the same students as the 2009 dataset. This dataset is much larger than the 2009 version and comprises of 19,917 students, 53,086 questions, 708,631 student interactions, and 100 skill components.

3.2.3 Spanish 2013

The Spanish dataset was aggregated for research on student knowledge retention in 2013 (Lindsey et al. (2014)). For this, 182 eighth grade students at a middle school in Denver were tasked with answering various Spanish questions on chapters they had been learning in class. These questions could require the student to translate sentences, conjugate verbs, list sets of vocabulary, etc. The course creator grouped these questions by the skill required to answer them correctly. In total, this dataset consists of 221 skill components such as regular conjugation, masculine nouns, plural nouns, etc. In total there are 409 exercises and 578,726 interactions from the students.

3.2.4 Statics 2011

The Statics 2011 dataset was obtained from PSLC DataShop (PSLC DataShop (2021)) and was compiled from students partaking in an engineering statics course (the study of quantifying forces between bodies) at Carnegie Mellon University. This dataset comprises of 333 students, 1,233 exercises, 189,297 interactions, and 98 skill components.

3.2.5 Overview of Datasets

In Table 3.1 below, an overview of the contents of all 4 datasets used as part of this experiment has been provided.

Overview of Contents

Datasets	Students	Questions	Skills	Responses
ASSIST 2009	3,241	17,709	124	278,868
ASSIST 2015	19,917	53,086	100	708,631
Spanish	182	409	221	578,726
Statics	333	1,233	98	189,297

Table 3.1: Overview of dataset contents

An example of the skill components contained within each dataset are provided below in Table 3.2. Unfortunately, the skill tags for the Statics 2011 dataset are not publicly available, the creater instead labels each skill in the dataset with a unique ID.

Skills in Datasets

ASSIST 2009	ASSIST 2015	Spanish
Area of rectangle	Addition of Integers	Days of the Week
Venn diagrams	Multiplying Integers	Numbers
Volume of Cylinder	Percent Of	Present Conjugation

Table 3.2: Examples of skill components in each dataset

3.2.6 Dataset Structure

Figure 3.1 below shows the structure for all datasets after pre-processing. As shown, this includes a student_id column to identify each student in the dataset and their responses, all students have a unique integer ID. The column question_id contains the unique ID for all questions in the dataset. Skill_id provides the ID of the skill required to answer that specific question (addition of integers, dividing fractions, etc.). And finally, the correct column records whether the student got that question correct (represented by 1), or incorrect represented by 0.

student_id	question_id	skill_id	correct
5	1619	41	0
5	1616	41	1
5	1636	41	0
5	1622	41	1
5	1615	41	1
5	1493	39	1
5	1714	40	1
5	1507	39	1
5	1671	77	1

Figure 3.1: Dataset structure for all 4 datasets

3.3 Models

For this part of the experiment, 4 knowledge tracing models were created: Bayesian Knowledge Tracing (BKT), Bayesian Knowledge Tracing with Forgetting (BKT+F), Per-

formance Factors Analysis (PFA), and Deep Knowledge Tracing (DKT).

These four models were chosen for this comparative analysis as they are deemed staple knowledge tracing algorithms in the field of adaptive education. There have been many extensions of these algorithms created in recent years but they all go back to these 4 algorithms. Also, knowledge tracing algorithms are considered to belong to one of three model types: probabilistic models, logistic models, and deep learning-based models. In our case, BKT and BKT+F belong to the probabilistic model type, PFA belongs to the logistic model type, and DKT belongs to the deep learning-based model type. This way the comparative analysis gives a broader comparison as it evaluates models from all knowledge tracing types.

The following section will discuss and detail the implementation of each of these knowledge tracing models.

Bayesian Knowledge Tracing

For this dissertation, a Python package entitled pyBKT was used to aid in training and testing BKT models (Badrinath et al. (2021)). Similar to scikit-learn, pyBKT provides many helper functions for fitting and customising models. Various combinations or all of the skills in the training dataset can be used when fitting models in pyBKT. For this comparative analysis, all skills in the datasets are used. Models built using pyBKT then use student responses to questions in the training dataset to estimate the BKT parameters for each skill (as discussed in state of the art): prior probability of knowing the skill, probability of transitioning from a "not knowing" state to a "knowing" state, probability of guessing correctly given "not knowing" state, and the probability of answering incorrectly given "knowing" state (slip).

Once the model is trained, pyBKT provides helper functions for evaluating the model. It can take in a testing dataset so that the model can predict whether questions will be answered correctly or incorrectly by students. It also provides many metrics to evaluate the performance of models, such as Area Under the Curve(AUC), and Root Mean Squared Error(RMSE).

Bayesian Knowledge Tracing with Forgetting

An extension of standard BKT, BKT+F incorporates a forget parameter to account for occasions where a student that had previously mastered a specific skill may forget said skill after going a long period without practice. Specifically this forgetting parameter is the probability that the student will transition from a state of knowing to a state of not

knowing a skill, and it increases as the student answers questions that do not require the skill in question. In standard BKT, this parameter F = 0. This forgetting parameter makes the model more sensitive to the time scale at which trials were conducted i.e. time between answering questions that require same skill.

This variation of BKT is also made available in the Python package pyBKT.

Performance Factors Analysis

For the PFA model, the implementation provided by the authors of Knowledge Tracing Machines was used (Vie and Kashima (2019)). Unlike the other models, the PFA model does not just take in the student ID, question ID, skill ID, and correctness of the responses. It must also contain a wins and failures column, where wins corresponds to the number of successful attempts at the corresponding skill, and inversely failures represents the number of failed attempts at the corresponding skill. The code provided includes a script to encode the data into a sparse feature matrix which automatically calculates the wins and failures. Once this sparse feature matrix is created, a Logistic Regression model is trained on it. This model then represents our PFA model and it can be used to predict whether students will answer unseen questions correctly or incorrectly.

Deep Knowledge Tracing

The original authors of the deep knowledge tracing paper published Lua code for their implementation of the model (Piech et al. (2015)). For this dissertation, the model was re-implemented using Python to keep it in line with the other models being analysed. For this specific version of DKT, a Long Short-Term Memory(LSTM) variation of a Recurrent Neural Network(RNN) was used.

3.4 Experiment Setup

To carry out the experiment, one by one the models were trained and evaluated on each of the 4 datasets. The workflow for training and evaluating a model on a dataset is detailed below:

1. An 80/20 split is carried out on the dataset, using 80% for training the model and 20% for testing the model

2. The model is trained using the training data,the input columns used are the student_id, question_id, skill_id, and correctness of the response

3. Once the model has been trained, it is given the first three columns of the testing dataset: student_id, question_id, and skill_id. It then attempts to predict whether the questions were answered correctly or not

4. The predicted outcomes are then compared to the actual correctness of the students' responses and several metrics are taken to evaluate the model's performance

Model parameters and outputs for BKT can be found in the appendix, sections A.1 and A.2

3.5 Evaluation

3.5.1 Evaluation Metrics

After the models made their predictions of whether questions were answered correctly (prediction = 1) or incorrectly (prediction = 0), three metrics were taken to evaluate model performance. These metrics were: Area Under the Curve(AUC), Root Mean Squared Error(RMSE), and the Mean Absolute Error(MAE).

The AUC score is a measure of how well the model can distinguish between instances where the student will be correct or incorrect (target classes). It is therefore essentially a metric of how well our model is predicting. It is especially used in cases where we equally care about the two possible outcomes - answering correctly or answering incorrectly. The closer the AUC score is to 1, the better the model is at making predictions.

The RMSE is then a measure of error in the predictions the model makes, a measure of how far away the prediction is from the true outcome. Although this task may look like classification (predict either correct or incorrect), like many classification tasks, it is really just regression under the hood. This is because the models are actually predicting the probability of the student getting the question correct. Then a threshold of 0.5 is used, this means if the predicted probability of answering correctly is <0.5 the output will be 0 (student will answer incorrectly) and if the probability is ≥ 0.5 the output will be 1 (student will answer correctly). Therefore the measure of error is essentially the distance between the predicted probability and the actual outcome (0 or 1). The RMSE specifically finds the square root of the average squared error across all predictions, the formula for this is as below:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Where $\hat{y}_i$ is the predicted probability of correctness, y_i is the actual target (0 or 1), and n is the total number of predictions.

Finally, the MAE metric is also a measure of error in the predictions, but it takes the average of the absolute error for all predictions made. This metric was taken so that a further statistical analysis can be carried out on the results later. The formula for MAE is as below:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

Where $\hat{y}_i$ is the predicted probability of correctness, y_i is the actual target (0 or 1), and n is the total number of predictions.

3.5.2 Results

In figure 3.2 below we can see the resulting AUC scores achieved by each of the models on each of the datasets.

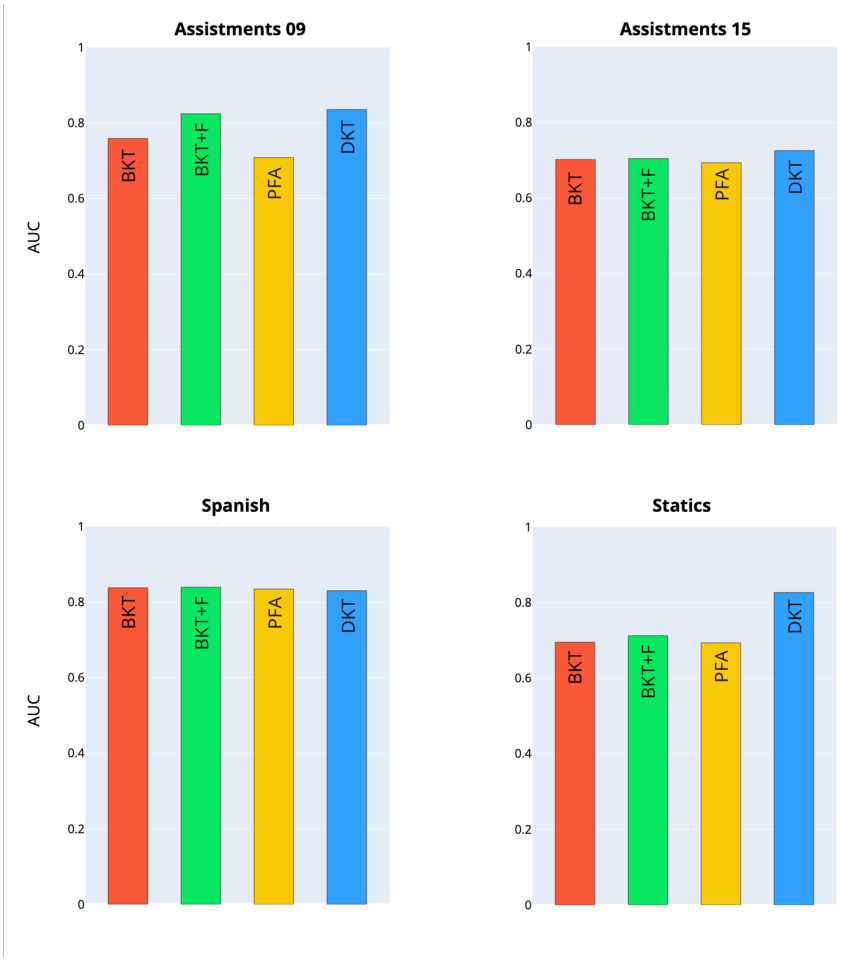

Figure 3.2: AUC results for all KT models

From these results, it appears that all models achieve a decent AUC score with no apparent outliers. In order to get a better look at the differences in results, the graph of AUC scores has been re-scaled between 0.68 and 0.84 on the y-axis, this can be seen below in figure 3.3.

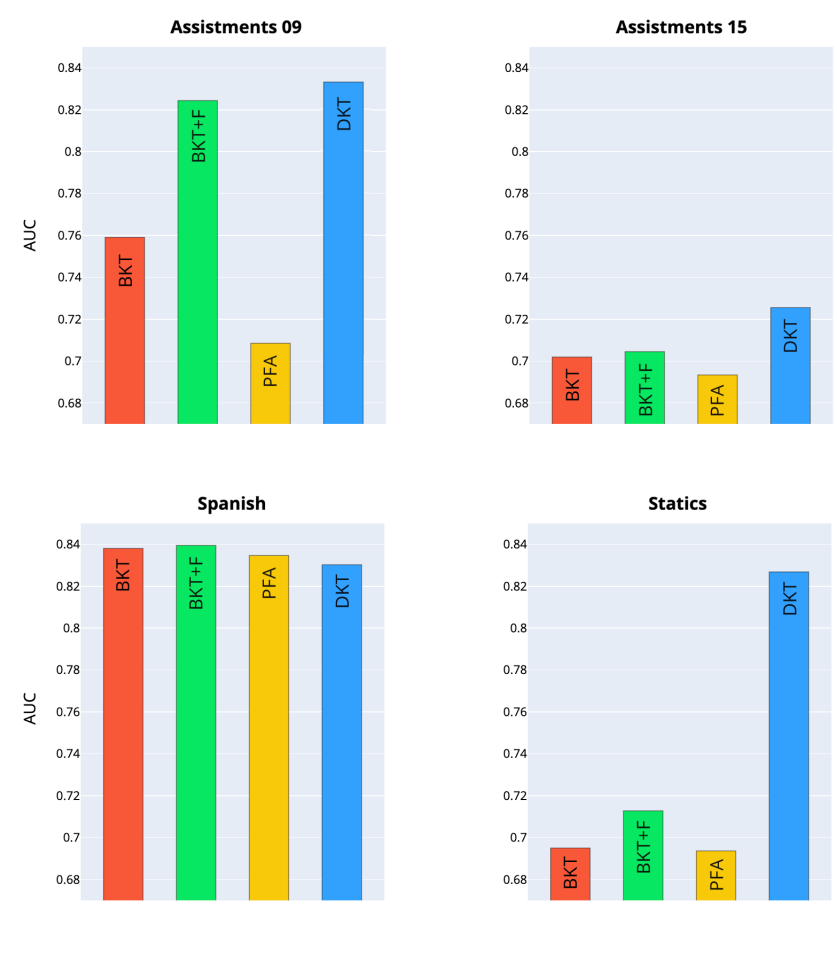

Figure 3.3: AUC results for all KT models scaled up

From this figure, it can be noted how the addition of a forgetting parameter to BKT (BKT+F) improves the performance of the model over standard BKT every time. This of course is due to the ability of the BKT+F model to account for the decay in skill levels when a student has not practiced a particular skill in a long time. This reveals something quite interesting about the datasets in question, and how the courses were conducted. It is likely that in the cases where BKT+F outperformed standard BKT drastically, the practice of skills were much more spaced out, perhaps due to it being a longer course (time taken). This effect is most prominent for the ASSISTments 2009 dataset as the AUC score increased from 0.76 to 0.82 between standard BKT and BKT+F, perhaps indicating the course was conducted over a long period of time causing BKT to not perform as well. In the case of the Spanish dataset, where both the BKT model and BKT+F model achieved an AUC score of 0.70 (and the other models performing similarly), this is especially interesting as the goal of the original paper that constructed the Spanish

dataset was to improve long-term information retention (Lindsey et al. (2014)). These results strengthen the hypothesis of the original paper.

It can also be noted in the results that the DKT model performs the best in almost all scenarios. Like BKT+F, DKT also accounts for decays in knowledge over time and this definitely influences the high AUC scores achieved. But it performs better than the BKT+F model in most cases so it is clear that it has other characteristics that boost its performance. One of the main reasons for this increase in performance could be due to its ability to discover relationships between skills and exercises, which the other models are not capable of doing. If the DKT model considers skill A and skill B to be somewhat related (e.g. addition and counting), then if a student answers skill A correctly, their skill level will increase not only for skill A but also for skill B. This of course is much more reminiscent of human learning and therefore it appears to model the students knowledge much more accurately. Considering DKT is the only model with this ability, it is probable that the skill components assigned by the coordinator of the statics course contained some level of interdependence. This would explain the large difference in AUC scores achieved for this dataset, with DKT receiving an AUC score of 0.83 and the second best model, BKT+F, only receiving an AUC score of 0.71, a 0.12 decrease.

It is also interesting how the models appear to perform worse on the ASSISTments 2015 dataset than they do on the ASSISTments 2009 dataset, especially when the skill components are quite similar and the courses were conduced by the same course coordinator. Even with the ASSISTments 2015 dataset containing far more interactions than the 2009 dataset, it still does not perform as well. Upon further inspection, it was found that although the 2015 dataset contained more interactions overall, it had far more students involved. Working out the average number of interactions per student, it is found that on average each student answers 35 questions (interactions). If we find the same for the 2009 dataset, the students on average answer 86 questions. It is apparent that the low number of interactions per student hinders the performance of the models on the ASSISTments 2015 dataset. This is because the models do not have as much data on the students as the 2009 dataset and therefore cannot become as accustomed to the students' abilities.

A full breakdown of the metrics recorded for each model and dataset, including AUC, RMSE, and MAE, can be found in the appendix A.3.

3.6 Summary of Findings

From the results obtained in this comparative analysis, it is clear that DKT is the better performing model in most scenarios. This is mainly due to its ability to model knowledge

in a much more "human-like" manner, taking into account decays in knowledge over time, interdependence between skills, and the sequences in which questions are given.

It was also found that the addition of a forgetting parameter into standard BKT (BKT+F) improved the performance greatly. This shows how much the timing between studying skills affects human learning and knowledge retention. Perhaps this could be used to study the effects of spacial repetition or other methods of organising educational material in order to improve knowledge retention. This was particularly seen in the scenario with the Spanish dataset, solidifying a hypothesis made in the original paper that aggregated the data.

It was also found that the number of interactions the students have with the system greatly improves model performance. We saw this when the low number of student interactions in the ASSISTments 2015 dataset greatly hindered model performance when compared with the ASSISTments 2009 dataset.

These results also show that the manner in which courses are conducted have a great impact in how these knowledge tracing algorithms perform and this should be considered before choosing which algorithms is best suited to your scenario. For example, in longer courses where the time between studying skills is larger, a model that can account for decays in knowledge over time such as BKT+F or DKT would be far more suitable. How the course coordinator groups skill components is also especially important. For broad skill groupings, a BKT model would suffice, but for extremely granular skill groupings that have a lot of inter-skill relationships, a model that can identify these relationships such as DKT would perform much better.

Chapter 4

Learner Simulators

This part of the dissertation will be approached from the perspective of a researcher/engineer not having access to any suitable historic data or human resources (learners) when designing/testing a system for adaptively recommending educational content. This is quite often the case when building these recommender systems for a new course, in this scenario there would be no historic data on the students or the course (cold start problem). Also, not all researchers or engineers would have access to a huge amount of learners to interact with the system in order to train a sufficient educational recommendation policy.

For this reason, researchers and engineers need a way of building and testing educational recommender systems without the need of historic data or to recruit a huge amount of human learners. This way they can build recommendation systems, experiment with or discover new recommendation strategies, and evaluate their system before deployment, in a manner that is far more accessible and convenient.

So how do we build and evaluate educational-based recommender systems without historic data or human learners? That is where learner simulators come in.

The main objective for this experiment is to prove that with the use of a learner simulator, an effective recommendation policy can be trained without the use of historic data or human resources.

This chapter will discuss the design and development of learner simulators. Section 4.1 will detail what a learner simulator is and what its roles are. Section 4.2 will detail how knowledge is modelled within a learner simulator. And section 4.3 will explain how learner simulators answer questions that are presented to it.

4.1 What are Learner Simulators?

A learner simulator is used to simulate human learner behaviour, answering assignment questions and building up skill levels through practice. Specifically, the role of the simulator is to generate responses to exercises and tests that it has been assigned, but also to maintain and continuously update an internal knowledge state as it practices skills through question answering. The learner simulator models its knowledge state using an internal knowledge tracing algorithm, such as the algorithms investigated previously in this dissertation, this can be thought of as the brain of the learner.

Unlike in the comparative analysis, the learner simulator does not need to be pre-trained on any historic data. Its prior knowledge state before beginning the course can be randomly generated, or it can start off from the beginning, not have any knowledge of the skills that are going to be taught in the course. Then, through carrying out exercises, it can begin to build up its skill levels.

Typically, recommender systems, especially those using reinforcement learning, require a huge amount of interactions (in our case, learners answering questions) for training a sufficient recommendation policy. This is why researchers and engineers building an educational based recommender system would have to recruit a huge amount of human learners to help train their recommendation policy. But now they can instead use learner simulators to interact with and train their recommendation agent, since they essentially mimic the behaviours that human learners would exhibit. This is how learner simulators eliminate the need for large amounts of human resources. This makes it far more convenient for engineers and researchers to design, train, or even discover their own recommendation techniques.

4.2 Modelling Knowledge

As stated previously, a learner simulator uses an internal knowledge tracing algorithm in order to model its continuously changing knowledge state. Any of the knowledge tracing algorithms that have been explored previously in this dissertation would work for this, but a suitable algorithm should be chosen depending on the characteristics of the course that is to be taught - which was discussed in the results section of the comparative analysis. The internal knowledge tracing algorithm used for the learner simulator operates in the exact same way, at every step it observes the question that the simulator answers, the skill the question requires, and whether the simulator answered the question correctly or not. This way it begins to record and model the simulators individual skill levels. This process is depicted below in figure 4.1.

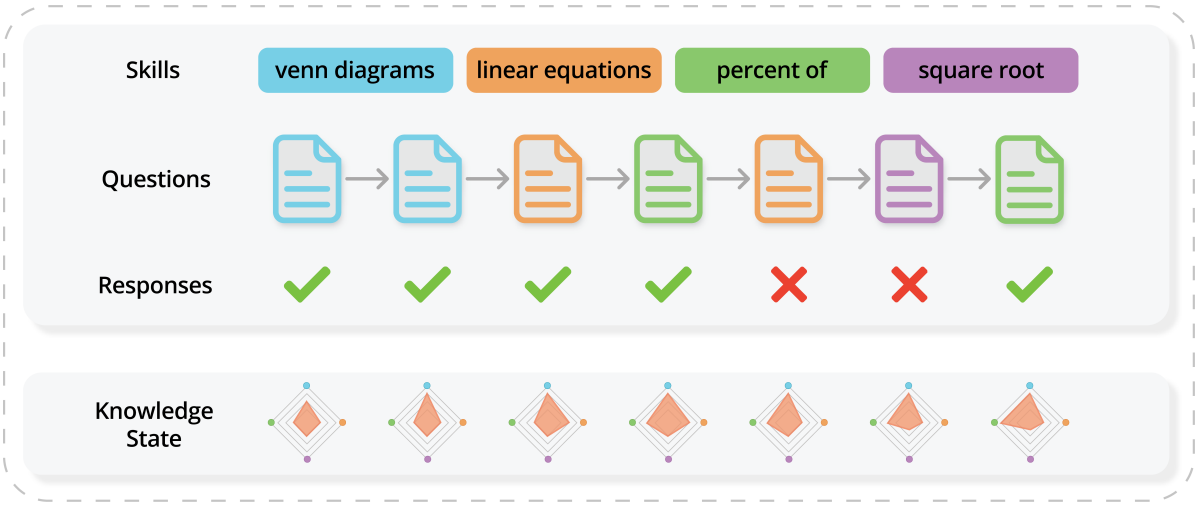

Figure 4.1: KT algorithm modelling knowledge as simulator answers questions. As shown, the knowledge state changes with every correct/incorrect response given by the simulator for each of the 4 skills.

For this dissertation, the knowledge tracing algorithm used to model the learner simulator's knowledge state was standard Bayesian Knowledge Tracing (BKT). The main reasoning for choosing BKT will become apparent later in the report when discussing the recommendation process, but it was mainly chosen as its internal modelling process is much faster and less computationally complex than the other algorithms discussed. This was a design decision made in order to fit the work required for this dissertation within the time limit assigned.

4.2.1 Modelling Simulator Knowledge with BKT

As described in the state-of-the art section of this report, BKT contains 4 parameters for every skill in the course. These parameters are stored within the learner simulator and are updated as it answers questions correctly or incorrectly.

The four parameters (per skill) are:

- P(L) - the probability of the simulator knowing the skill previously.
- P(T) - the probability of transitioning to a "knowing" state given simulator is in "not knowing" state.
- P(G) - the probability of guessing the question correctly given "not knowing" state.
- P(S) - the probability of slipping and answering the question incorrectly given "knowing" state.

For this experiment, the initial value of P(L) is randomly set before the simulator sees any educational material. This is because typically when students start a course, some students may have previous knowledge of the skills required, and the skill levels differentiate between students.

These parameters then get continuously updated as the simulator answers questions correctly or incorrectly. It does this following the BKT formulae previously provided in the report. The formulae have been listed again below:

1. If the learner simulator gets the answer correct, the conditional probability is found using Formula A. Inversely, if the learner simulator gets the answer incorrect, the conditional probability is found using Formula B.

2. The conditional probability is then used to update the probability that the learner simulator now knows the skill p(L) and follows Formula C.

A
$$p(L_t|obs = correct) = \frac{p(L_t) \cdot (1 - p(S))}{p(L_t) \cdot (1 - p(S)) + (1 - p(L_t)) \cdot p(G)}$$

B
$$p(L_t|obs = incorrect) = \frac{p(L_t) \cdot p(S)}{p(L_t) \cdot p(S) + (1 - p(L_t)) \cdot (1 - p(G))}$$

C
$$p(L_{t+1}) = p(L_t|obs) + (1 - p(L_t|obs)) \cdot p(T)$$

We can see that the probability P(L) for each skill will be continuously updated as the simulator answers questions, following the formulae. This probability then encapsulates the overall understanding the student has for the corresponding skill, 0 being the student has absolutely no knowledge of the skill, and 1 meaning the student is a total master of the skill.

4.3 Answering Questions

Every time the learner simulator is presented with a question, we need it to either answer the question correctly or incorrectly. So how is it decided whether the question will be answered correctly or not?

Well, BKT actually contains a final fifth parameter that is stored for every skill. This parameter, P(C), represents the probability that the student will answer the next

question requiring that skill correctly. This probability is found as a combination of the parameters discussed previously, particularly it takes into account the student's level of understanding for that skill P(L), and their probability of guessing P(G) or making a slip P(S). The formula for calculating P(C) is shown below:

$$p(C_{t+1}) = p(L_{t+1}) \cdot (1 - p(S)) + (1 - p(L_{t+1})) \cdot p(G)$$

So when a simulator is presented with a question that requires skill A, it looks for the probability of answering correctly P(C) that is stored for skill A. Then, it randomly samples a binomial distribution with a P(C) probability of answering correctly. The output after sampling (either 1 or 0) then determines whether the question will be answered correctly or incorrectly.

Chapter 5

Adaptive Recommendation

This part of the dissertation will discuss the design and implementation of a reinforcement learning-based recommendation agent for adaptively recommending educational exercises to students.

The recommendation agent in this case will be trained entirely using a learner simulator as discussed in the previous chapter. By recommending educational exercises to a simulator and receiving responses, the agent will be able to adjust and incrementally learn an effective recommendation policy. In our case, we want the recommendation agent to learn a policy for recommending educational exercises that will maximise a student's learning gains whilst minimising information overload (number of exercises assigned). The goal for this experiment is to prove that a learner simulator can be used to train an effective recommendation agent without having access to a large amount of human learners. This makes it far more convenient for engineers and researchers to discover, train, and evaluate educational recommendation systems before deployment.

In this chapter, section 5.1 will give an overview of the recommendation system, section 5.2 will describe the exercises and tests that the system can recommend to the learner simulator, and section 5.3 will discuss the implementation of the recommendation system and the recommendation process.

5.1 Overview of System

As described previously, the objective of the recommendation agent is to recommend educational exercises to students that will best improve their level of knowledge in the course (maximise educational gain), but we want it to do this using the least number of exercises as possible (reduce information overload). In order to train an effective recommendation

policy to achieve this objective, we need the agent to recommend exercises to the learner simulator. The learner simulator will then answer the questions on the exercise, and the agent will analyse the results the simulator achieves. By carrying out this process repeatedly, the agent will incrementally change its recommendation strategy (policy) in order to get the best results out of the the learner simulator using the minimum amount of exercises.

Of course, as this is happening, the learner simulator is continuously updating its knowledge state as it gets questions correct or incorrect. The learner simulator also answers the exercises and tests following the process that was detailed in the last chapter in section 4.3.

The steps taken during the recommendation process have been detailed below:

1. Learner simulator takes a pre-test to gauge skill levels
2. Agent can then assign an exercise to the learner simulator and it analyses the results achieved after the simulator responds
3. Agent repeats step 2 until it thinks the learner simulator is ready to be assigned a post-test
4. Agent assigns post-test to the learner simulator
5. The agent is rewarded for an improvement between the pre-test score and the post-test score, it also gets small rewards for improvements between exercise scores
6. The agent receives a penalty for the number of exercises assigned in between the pre-test and post-test

The goal of the agent here is to receive the largest possible reward. It will achieve this by recommending exercises that will improve the learner simulator's score between pre-test and post-test greatly, whilst keeping the number of exercises to a minimum.

5.2 Exercises & Tests

The role of the recommendation system is to recommend some educational exercises or tests for the learner simulator to carry out. But what exactly are these exercises and tests?

The pre-test and post-test carried out by the simulator is a set of multiple questions that require various different skills. As described previously, BKT assumes all skills are

independent from each other and one question should require only one skill. For this reason, the tests and exercises used for the recommendation in this dissertation only require one skill per question. Of course this can be extended in the future if a learner simulator is implemented that uses a knowledge tracing algorithm such as DKT that can allow for multi-skill questions and inter-skill relationships. As an example, a pre-test or a post-test in our case could consist of six questions where each question requires one of three skills. An example of such a test set up is shown below in table 5.1.

Example test

ID	Skill Required
Q1	Linear Equations
Q2	Square Roots
Q3	Linear Equations
Q4	Percent Of
Q5	Percent Of
Q6	Square Roots

Table 5.1: Example of a pre-test / post-test

When the learner simulator receives a pre-test or a post-test such as this example, it will answer the questions using the process described previously in section 4.3, and its individual skill levels will increase/decrease if it answers the questions correctly/incorrectly. For example, if the learner simulator answers both linear equation questions correct, its skill level for linear equations will increase. Inversely, if the learner simulator gets both square root questions incorrect, its skill level for square roots will decrease.

5.3 Implementation of Recommender System

5.3.1 Recommendation System Components

Actions

For this system, there are two possible actions the agent (specifically the actor) can take. These two actions are:

1. Assign an exercise
2. Assign the post-test

States

As described previously, the state represents current information about the environment that the agent uses to make decisions about what action to take. The state is updated every time an action is performed. The state in our case is a list of three vectors:

1. Pre-test score
2. IDs of exercises answered
3. Scores for each exercise taken

Rewards

After an action is carried out (exercise or post-test), the environment calculates a reward or penalty to give to the agent depending on the results achieved by the simulator. Of course the goal of this recommendation system is to maximise learning gains whilst minimising information overload (number of exercises to carry out). Therefore, the reward function used needs to apply a positive reward for improvements in test results and apply a negative reward for the amount of exercises carried out between the pre-test and post-test.

Specifically, there are two reward functions used during the recommendation process. One reward function is used when the final post-test is carried out to reward the agent for improvements from the initial pre-test to the post-test, and to take away from this reward for the number of exercises there were recommended. This formula can be seen below:

$$R = 1 + (S_{post} - S_{pre}) - (1 + \delta) * N$$

Where S_{post} is the post-test score, S_{pre} is the pre-test score, δ is the penalty, and N is the total number of exercises recommended. The penalty for the final system was set at 1.5 as it struck a balance between allowing the agent to explore the state space while preventing it from recommending too many exercises.

The second reward function is used after every exercise is carried out in order to give the agent a small reward for improvements between exercises. This reward function is shown below:

$$R_t = \gamma * (E_t - E_{t-1})$$

Where γ is a discount factor, E_t is the score received on the current exercise, and E_{t-1} is the score received on the previous exercise. After trial and error, the discount for this particular system was set at 0.8. This was because if the reward for exercise improvement was too large, it would end up recommending too many exercises.

5.3.2 Recommendation Process

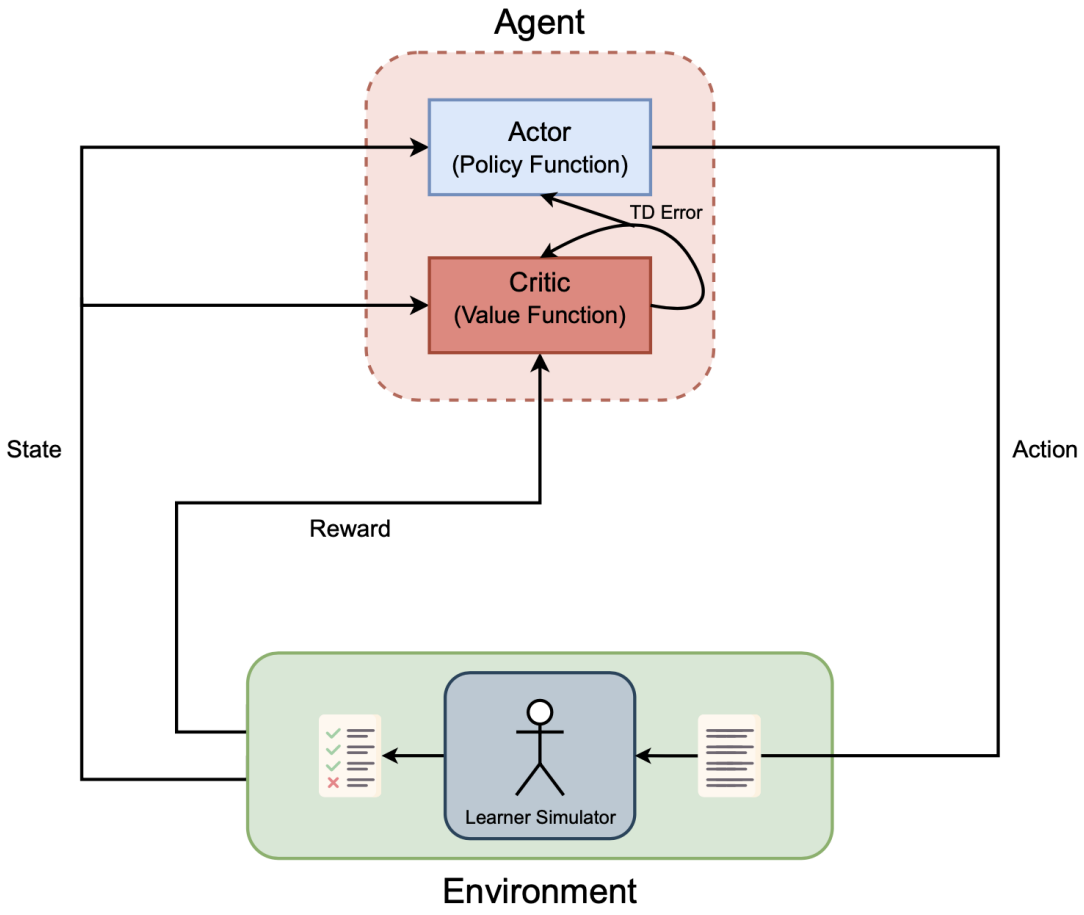

Figure 5.1: Workflow of RL-Based Recommender System

Shown above in figure 5.1 is the entire recommendation workflow for this particular system. As seen, there is an agent that consists of an actor and a critic, and an environment that houses the learner simulator. After the learner simulator carried out the pre-test, the system works as follows:

1. The environment gets the scores of the pre-test and sends this state to the actor and the critic

2. The actor chooses which action to take next (either recommend an exercise or the post-test)

3. The environment receives the exercise or post-test and gives it to the learner simulator

4. The learner simulator answers the questions following the answering process detailed in section 4.3, and it updates its internal knowledge state depending on which skills were answered correctly and incorrectly

5. The environment updates the state to include the scores of the exercise that was just carried out and sends this updated state back to the actor and critic

6. The environment also calculates the reward for the action taken according to the formulae given in section 5.3.1 and sends this to the critic

7. The critic calculates the TD error (which will be discussed below) and informs the actor so it can determine whether to change the current recommendation policy

8. This process repeats until the post-test is carried out, terminating the episode

5.3.3 Agent

As this recommender system is based on actor-critic reinforcement learning methods, the Agent is made up of both a policy function (Actor) and value function (Critic). The role of the Actor in this case is to look at the state of the environment (exercises answered and scores) and to make a decision on what action to take, which in our case is to either recommend an exercise for the learner simulator to carry out or to assign the post-test.

The role of the Critic is to look at the state of the environment and the latest reward received in order to criticise the current recommendation policy (method of choosing actions) being taken by the actor. As seen in the state-of-art chapter 2.2.1, the critic calculates the total reward the agent can expect to receive if the actor continues to follow the recommendation policy it is currently following. Then, on the next action, the Critic reevaluates the new state and determines whether the resulting reward is better or worse than it had initially expected. This difference in reward expectation is called Temporal Difference and is calculated with the formula below.

$$\delta_t = r_{t+1} + \gamma V_{(S_{t+1})} - V_{(S_t)}$$

Where r_{t+1} represents the observed reward for the action taken, $V_{(S_{t+1})}$ represents the estimate of total reward that will be received from the new state until the end of the cycle

following the current policy, γ represents a discount rate, and $V_{(S_t)}$ was the estimate of the total reward the Critic calculated in the previous state. From this you can see that the TD error at each time is the error in the value estimate made by the Critic.

By continuously recommending exercises or post-tests for the learner simulator to carry out, and by criticising the current recommendation policy, the Agent will learn an optimal recommendation policy gradually over time. This is called On-Policy Learning.

5.3.4 Environment

In this system, the environment is where the learner simulator operates. This is where the exercises and post-tests are sent to when the agent chooses an action. Here, the learner simulator will answer the exercise or post-test. The scores they achieve and the number of exercises they have carried out will be used by the environment to calculate the reward. Once the reward is calculated, it will be sent back to the agent (both the actor and the critic), Also, the role of the environment is to check the current state after the exercise or post-test has been answered by the learner simulator and to send that back to the agent also. This state is a vector containing the learner simulator's pre-test score, the ID of the exercises answered so far, and the corresponding scores for these exercises.

5.3.5 Episode Example

An episode in this case is defined as taking a learner simulator from a pre-test to post-test. Shown below in figure 5.2 is an example output from the recommendation system after a single episode terminates.

```
------------------------------------------------
pre-test:        [1 0 0 0 0 0 1 0 1 1 1 0 0 0 1 1 1 0 1 0 0 0 0 1 1]
post-test:       [1 0 0 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0]
episode 183   episode reward 10.6   average reward 7.0
actions taken:  [7, 11, 20]
rewards:  [0.8746896591546225, 0.9146101038546527, 8.8]
------------------------------------------------
```

Figure 5.2: An example output of a single episode taking a learner simulator from pre-test to post-test

In this example, the pre-test vector represents whether each question in the test was answered correctly (represented by 1) or incorrectly (represented by 0). The actions taken are the exercises that were recommended to the learner simulator after taking the pre-test: firstly exercise 7 is recommended, then exercise 11 is recommended, and then

the post-test is recommended. Here the post-test is represented by 20 as there as there are 20 exercises in total with IDs 0-19, and the post-test having an ID of 20. We can also see there are small rewards given for the first two exercises carried out, and then a reward of 8.8 is given for the improvements between the pre-test and post-test. This reward of 8.8 is relatively high because the agent only recommended two exercises previously to recommending the post-test. The post-test vector then represents whether each question in the post-test was answered correctly or incorrectly. We can also see the total episode reward is 10.6 and the average reward for all episodes carried out so far is 7.0.

Chapter 6

Analysis & Evaluation

In this chapter, the performance of the recommendation agent will be analysed and evaluated. The goal here is to show that a recommendation policy trained using learner simulators can be effective in terms of producing large knowledge gains in students using an optimal amount of educational exercises. By achieving this, it can be proven that the use of learner simulators eliminates the need for researchers and engineers to have access to a large amount of historic data or human learners when investigating new recommendation systems. In order to do this, the knowledge gains achieved by learner simulators following an adaptive recommendation policy will be compared with simulators following three different baseline learning paths.

6.1 Experiment Setup

To evaluate the reinforcement learning-based recommendation system, it will be compared with three baseline methods for defining a learning path (selecting exercises). All scenarios will have the same setup, using a BKT-based learner simulator to answer tests and exercises that have been assigned to it, but the method used for selecting these exercises and tests will differ. As normal, the learner simulators following each of the four methods (1 adaptive learning path, 3 baseline learning paths) will start off by answering a pre-test, then exercises will be assigned following the corresponding exercise selection method, and then the learner simulators will answer a post-test. To evaluate all four methods, the number of exercises assigned and the learning gains achieved by the learner simulators as a result will be compared.

For all four exercise selection methods, there will be 20 exercises in total to choose from that consist of 5 possible skill components. Each of the possible exercises can only be assigned once. All of these methods were executed for 5000 episodes (going from pre-test

to post-test 5000 times) so the reason why 20 exercises was chosen as the maximum was to keep training times down.

6.1.1 Learning Path Selection Strategies

RL-Based Recommendation

As discussed in the previous chapter, the reinforcement learning-based recommendation system uses an intelligent actor-critic agent to choose the exercises that the learner simulator carries out. The agent has a reward function built in that favours larger knowledge gains using fewer exercises, so it is expected that the learner simulator following this strategy will carry our far less than the entire 20 possible exercises. The agent in this case decides when it believes the learner simulator is ready to answer the post-test.

Baseline #1

For the first baseline learning strategy, the learner simulator will be required to answer all 20 exercises available between the pre-test and post-test. So for this learning strategy, there is no intelligent agent involved to carefully select the exercises the learner should carry out, and there is no agent to decide when the learner should take the post-test, they have to carry out all available exercises before answering the post-test. The reason this was decided to be a baseline strategy is because if the student answers all of the exercises that are available, then it would be expected that they would achieve the largest possible gain in knowledge. The goal here would be to show that the adaptive recommendation agent can produce similar learning gains in learner simulators using far less exercises, producing a learning gain that is close to or better than this presumed maximum knowledge gain.

Baseline #2

For the second baseline learning strategy, the learner simulator will have a predefined learning path of 15 randomly selected exercises from the possible 20 that are available. Therefore the learner simulator will answer the pre-test, carry out 15 randomly selected exercises, and then carry out the post-test. Since the RL-based recommendation system will undoubtedly recommend less than the entire 20 exercises, it will be insightful to compare it with a baseline that also uses less. This way we can compare the learning gains of 15 randomly selected exercises versus the exercises carefully selected by the recommendation agent.

Baseline #3

Similar to the second baseline, the third baseline learning strategy selects a predefined learning path. But this time it randomly chooses 10 exercises rather than 15 exercises. Again this is to compare the learning gains achieved by the learner simulators following this baseline strategy versus the learner simulators following an adaptive learning path recommended by the agent which should consist of a similar number of exercises.

6.2 Results

As mentioned previously, the experiments were executed for 5000 episodes. An episode in our case is taking a learner simulator from the pre-test to the post-test. Hence, all four exercise selection methods repeated this process 5000 times and the number of exercises carried out along with the learning gains achieved as a result were plotted. The learning gains for this experiment were defined as the improvement in test scores from the pre-test to the post-test i.e. if a learner simulator achieved 65% in the pre-test and 95% in the post test, their learning gain would be 30%.

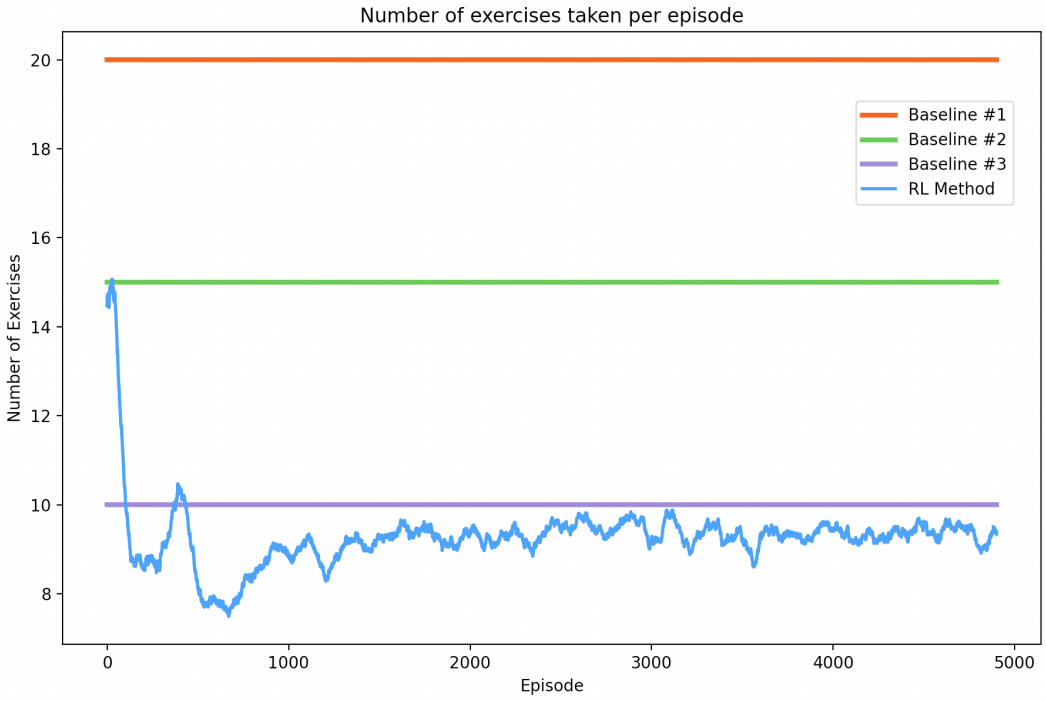

Figure 6.1: Number of exercises per episode

Figure 6.1 above depicts the number of exercises carried out by the learner simulators between the pre-test and post-test, over the 5000 episodes. As expected, the three baseline models are constant as the number of exercises in their learning path is predefined at 20, 15, and 10. As shown by the blue line, the number of exercises carried out by the simulator varies between episodes, this is because it has an intelligent agent that decides which exercises they do and when they are ready to take the post test. As discussed previously, the RL agent receives a penalty for the number of exercises recommended to the learner simulator, for this reason the agent quickly realises that it risks getting a large penalty for recommending too many exercises. This is why we can see in the graph that the agent initially recommends around 15 exercises between the pre-test and post-test but it quickly drops off when it receives a penalty and it begins ease out at around 9 exercises per episode.

To get a better look at this, a bar chart of the average number of episodes has been provided in figure 6.2 below.

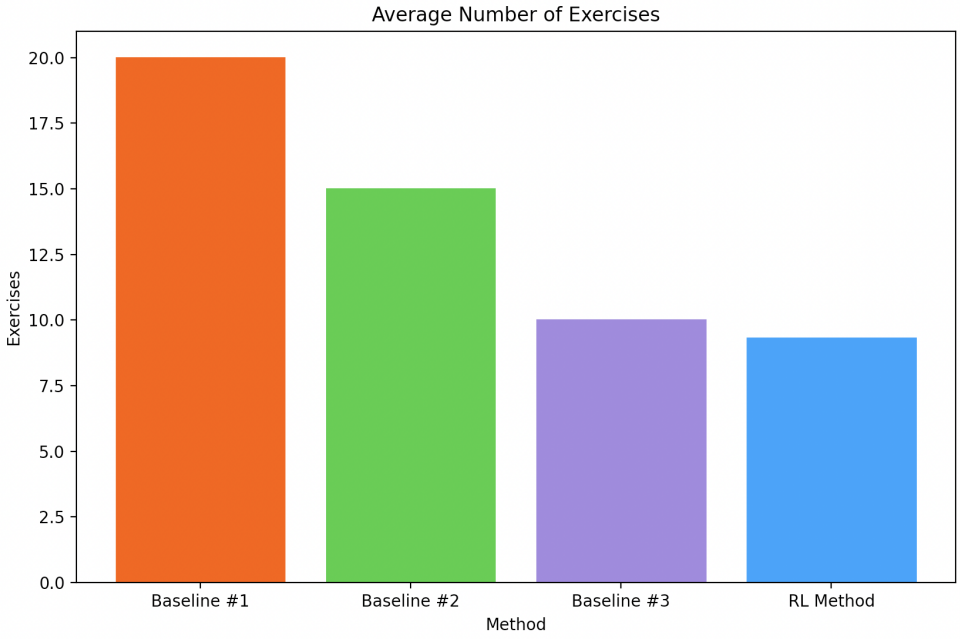

Figure 6.2: Average number of exercises per episode

Again, here we can see the baseline learning paths are constant at 20, 15, and 10 exercises per episode. The learner simulator following the adaptive path of the RL recommendation agent carries out on average 9.3 exercises per episode, the least of all learning path selection strategies.

As the RL agent recommends the least number of exercises, it would be assumed that the learner simulator following its leaning path would produce the least learning gains. To examine this, the learning gains for each episode were calculated and the average found. As mentioned previously, this learning gain is the percentage increase in scores between the pre-test and post-test. This information has been depicted below in figure 6.3.

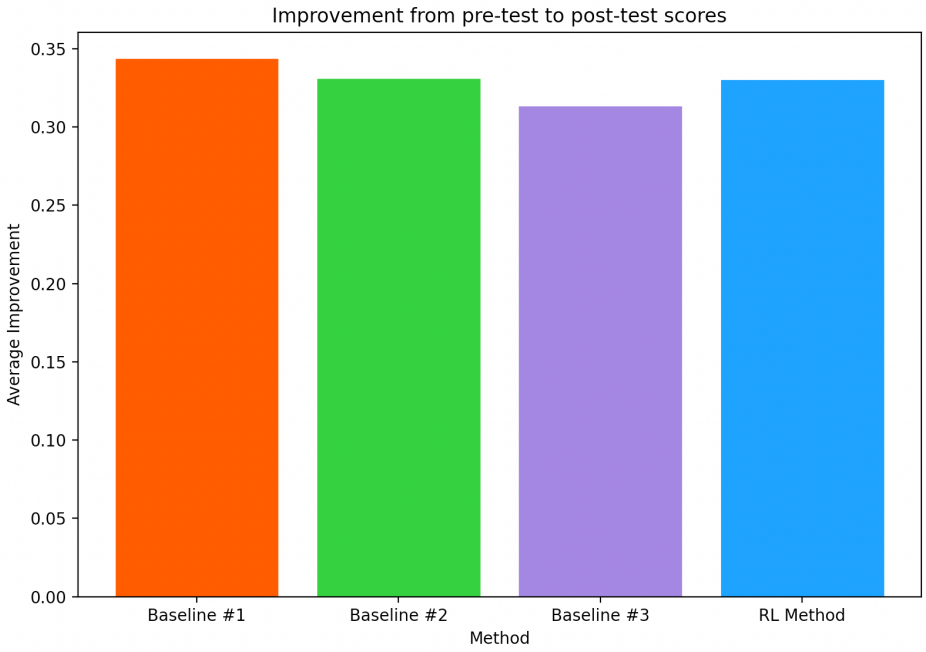

Figure 6.3: Improvement between pre-test and post-test

As shown, the learner simulators appear to exhibit a similar gain in knowledge. Baseline #1 (20 exercises per episode) appears to produce slightly better knowledge gains with Baseline #3 (10 exercises per episode) appearing to produce the least. However, in order to examine this further, the bar chart has been scaled between 0.31 (31% improvement) and 0.36 (36% improvement) in figure 6.4 below.

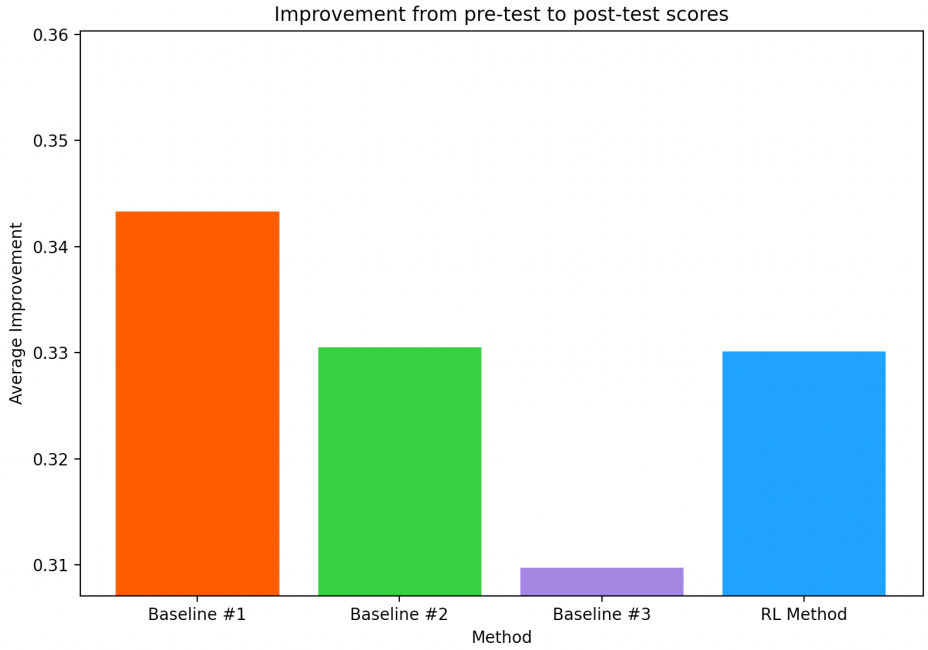

Figure 6.4: Improvement between pre-test and post-test scaled

Here we can see clearly that learner simulators following Baseline #1 produce the best learning gains. This was to be expected as these leaner simulators answer all 20 exercises that are available and therefore get more opportunities to study and increase their skill levels (following BKT). As they carry out all available exercises, this is deemed to be the maximum knowledge gain possible. On closer inspection, it can be seen that the learner simulator follow the adaptive path of the RL agent produces on average a learning gain of 33.05%. So compared with the average learning gain of 34.2% produced by Baseline #1, this is actually a good result. This is because learner simulators following Baseline #1 carry out all 20 exercises whereas the learner simulator following the adaptive RL agent only carries out 9 exercises on average. To produce such a similar learning gain whilst carrying out <50% of the exercises reduces information overload greatly. This means a student would only have to carry out half the exercises in order to exhibit the same level of learning.

It can also be seen that the learner simulator following the RL agent produces an average learning gain extremely similar to that of Baseline #2. In Baseline #2 the learner simulator carries out 15 randomly selected exercises from the pool of 20. Both methods show about a 33% increase in test scores between the pre-test and post-test. Again, learner simulators following Baseline #2 carry out on average 6 more exercises than those following the RL agent, so to exhibit the same gain in knowledge is a good result here.

Similar to the last comparison, the RL agent is reducing information overload on the student greatly by recommending less educational exercises to complete whilst achieving the same knowledge gains.

As shown, Baseline #3 achieves on average a learning gain of 31%. The learner simulators following this strategy had to carry out 10 exercises per episode. Comparing this to the RL method that recommends on average 9 exercises, the expectation is that they would produce similar learning gains. But in actuality, the learner simulators following an adaptive learning path assigned by the RL agent produces a 2% greater learning gain on average.

6.3 Summary of Findings

The goal prior to carrying out this experiment was to prove that a learner simulator can be used to train an effective recommendation agent. This was to show that researchers and engineers would not require the use of large amounts of historic data or access to a huge amount of human learners in order to design, train, and test a recommendation system for education. Through the results of this experiment, it has been proven that learner simulators are in fact an appropriate solution when an engineer or researcher is in a position where they do not have access to these resources. This allows them to train their systems far more conveniently whilst investigating novel recommendation strategies or for testing their system before deployment.

In order to prove that a recommendation policy trained using learner simulators can in fact be effective, the objective was to show that a reinforcement learning based learning path can produce better or similar learning gains to various baseline strategies. This was proven by showing that learner simulators following an RL-based recommendation agent produced the same learning gains using 9 exercises as those carrying out 15 exercises following a baseline strategy. This reduces informational overload greatly by assigning far less exercises to students whilst still achieving the same learning gains. It was also shown that learner simulators carrying out all 20 exercises did not achieve much more of a learning gain than those following the adaptive strategy, with just over 1% in the difference. This is a big achievement considering the simulators carried out less than half of the exercises, on average only completing 9. Also, another baseline strategy that assigned 10 exercises to learner simulators actually produced a worse learning gain to the RL strategy. Even though students took one more exercise on average, they showed 2% less of a learning gain. Of course the next step would be to train and deploy one of these RL-based recommendation agents to an online learning platform and to investigate whether similar results are found.

The recommendation system implemented as part of this dissertation may not be the most modern or complex, and other methods may produce larger knowledge gains, but now researchers have the opportunity to do just that. By achieving the main goal of this dissertation, proving that learner simulators can in fact be used to train effective recommendation policies, researchers can now use these to aid them in investigating their own recommendation strategies. Learner simulators make it much more convenient for researchers and engineers to implement their own recommendation systems and now hopefully they can be used to discover the next state-of-the-art educational recommendation methods.

Chapter 7

Conclusions & Future Work

The journey through answering the various research questions and objectives set out at the start of this dissertation lead to some extremely insightful discoveries. In this chapter we will summarise the goals and findings of this dissertation, discuss the possibilities and limitations of the approach taken, and provide some opportunities for future work.

7.1 Summary

Before conducting this dissertation, four research questions were set out to be answered. In this section we will go through the research questions and discuss the findings that came about as a result.

7.1.1 Question 1

How do state-of-the-art knowledge tracing algorithms compare to each other when trained on historic learner sequence data? What are their strengths and weaknesses based on the analysis?

Answer: Through the comparative analysis conducted, we discovered many differences between four state-of-the-art knowledge tracing algorithms.

Firstly, it was discovered that the addition of a forgetting parameter to standard BKT (BKT+F) improved the accuracy of performance predictions greatly. This highlighted the affect of time on human learning. Skill levels degrade if they have not been practiced in some time, and that is why models that incorporate forgetting such as BKT+F and DKT could better model this behaviour. Therefore the ability to model forgetting is a clear weakness of BKT.

It was also discovered that how many times a student interacts with a system (an-

swering questions) is pinnacle for how well the knowledge tracing algorithms can model their knowledge. This was seen when the models that were trained on the ASSISTments 2009 dataset were able to predict performance much more accurately than the models trained on the ASSISTments 2015 dataset, due to its large number of interactions. It was also noted that DKT performed the best by around 3.5% on on the ASSISTments 2015 dataset, highlighting that a strength of DKT is its ability to perform on sparse data.

It was also discovered that a clear weakness of BKT, BKT+F, and PFA was how they model skills as being independent from each other. Of course skills in the real world tend to relate to each other in ways, such as counting and addition. DKT is much better suited to cases where skills have a large number of inter-relationships as it can automatically infer these relationships and capture them in the modelling process. This is an immense strength of DKT.

Through these discoveries it is made clear that the strengths and weaknesses highlighted above should be greatly considered before choosing a knowledge tracing algorithm that is best suited for your application.

7.1.2 Question 2

Can knowledge tracing algorithms be used to create learner simulators for imitating student learning behaviour?

Answer: Knowledge tracing algorithms can in fact be used as internal knowledge modelling algorithms and question answering mechanisms in order to create simulated learners. We saw this in chapter 4 when BKT was used to keep and continuously update the knowledge state of a proposed simulated learner. It did this by keeping parameters for recording the skill levels of all skills that a learner simulator may encounter. This mechanism then represents the brain of human learners as they build up their skills through practice. Then we also saw in section 4.3 that an answering mechanism was implemented whereby a parameter is kept that models the probability the learner simulator will get the next question requiring a certain skill correctly. This was then used to sample a binomial distribution which would determine whether the response would be correct or incorrect. The simulator also used slip and guess parameters from BKT to closer mimic real learner behavior whereby questions could be mistakenly answered incorrectly, or the correct answer could be guessed. Overall the investigation of this research question was successful as fully functional simulated learners were created using BKT.

7.1.3 Question 3

Can these learner simulators be used to adequately train an intelligent recommendation agent for adapting student learning paths?

Answer: The learner simulator created was in fact successful in training an RL-based recommendation agent for adapting a learning path. We saw this in chapter 5 where an Actor-Critic reinforcement learning method was used to create an agent for adaptively recommending exercises for students to carry out. The goal of the specific recommendation agent investigated was to maximise a student's knowledge gain using an optimal amount of exercises, reducing information overload. We saw that the agent was well trained using the learner simulator as it was successful in producing similar learning gains in learner simulators using far less exercises than baseline learning paths that were compared. Therefore, the experiments carried out in order to answer this research question was a success, and an effective recommendation agent can be trained using learner simulators.

7.1.4 Question 4

How does an adaptive learning path compare to a suitable baseline learning path? Does it provide any gains in student knowledge?

Answer: Through the experiments carried out in chapter 6, we noted how learner simulators following the learning path adapted by the intelligent recommendation agent experienced a 33% increase in test scores on average using 9 exercises. On the other hand, learner simulators following a baseline learning path experience a similar gain in knowledge but by completing 15 exercises.

Similarly, we saw how another baseline learning path whereby a learner simulator had to answer 10 exercises, the learning gains produced were less than that of the simulators following the intelligent recommendation agent. Specifically, there was on average a 2% improvement in test scores then learner simulators followed the adaptive agent even though they were answering one less exercise.

From this we can see that an adaptive learning path can provide gains in student knowledge, the gains were not extremely large but other recommendation strategies developed in the future may increase these. However, this experiment would now need to be conducted through an online educational platform to human learners to see if similar results are obtained.

For the reasons highlighted above, this research question was also answered successfully through experimentation.

7.2 Possibilities

In the domain of adaptive education, and specifically reinforcement learning approaches to education, learner simulators have great potential. When building these adaptive systems, historic data has always been a huge pitfall. If an engineer does not have access to previous data on the students in a course, how are they going to design systems for recommending content to the students? Or what about systems for detecting students at risk of failure? This cold start problem has long been an issue in the domain of adaptive education. But now engineers can use learner simulators to mimic human learning behaviour and do not require access to historic data.

Reinforcement learning has always been notorious for requiring a huge amount of interactions in order to train an effective policy, and this made it really difficult in the field of adaptive education. Researchers and engineers would require access to a large amount of human learners and an appropriate online platform just to train their initial model, this of course is not something that everyone has access to. Learner simulators make it far more convenient for designing, training, and testing reinforcement learning agents. The possibilities with this are endless as researchers can now discover new reinforcement learning strategies for education, without having access to these resources. Not even just for exercise recommendation as explored in this dissertation, but any reinforcement learning solution. Such as flagging students at risk of failure or dropping out, or for adapting study plans to tackle students' weaknesses. Learner simulators open the door to many future opportunities.

7.3 Limitations

During the course of this dissertation, various limitations have been discovered. One example of this would be building the learner simulator with standard BKT as its internal knowledge modelling algorithm. As discussed previously, BKT does not have the ability to capture inter relationships between skills. This of course is a common feature in learning, for example counting and addition. The ability of a student to perform addition is heavily related to their ability to count. How can they perform addition without counting? It is important for the knowledge tracing algorithm to understand these relationships to more accurately model a student's knowledge state. Also, standard BKT does not take into account that a student may forget skills if they have not practiced them in a long time and believes that once a skill is mastered it is not forgotten. This is a clear limitation of the learner simulator implemented as part of this dissertation as forgetting skills with time is a common trait in human learning.

In terms of the recommendation strategy itself, a limitation here would be the fact that the agent still takes some time to train. For the experimentation carried out in this dissertation, the recommendation process was cycled for a total of 5000 episodes. This is not a lot of episodes when in a testing environment using learner simulators. However, if this system was to be deployed to an online educational platform, thousands of students would need to work their way through the course before the agent begins to learn the optimal recommendation strategy for each student.

Although there are some clear limitations, this does not take from the research, instead it gives way to many opportunities for future work and development.

7.4 Future Work

There are many ways in which the work carried out as part of this dissertation can be extended and built upon.

One major opening for future work would be to build more learner simulators using different knowledge tracing algorithms. As discussed, the learner simulator implemented as part of this research used standard BKT and this gave way to some limitations. Building a learner simulator using an algorithm such as DKT would overcome many of the limitations discovered. With DKT as the internal knowledge modelling algorithm, relationships between skills can be discovered and understood, relationships between exercises can be found automatically, explicit skill labels would not be required as DKT can infer these, the order in which exercises are carried out can be used to influence skill modelling, and also DKT takes into consideration that skill levels can decay with time. The use of DKT as an internal modelling algorithm would make for a far more robust and accurate learner simulator. Also, the other algorithms discussed as part of the comparative analysis such as BKT+F and PFA could be used for these learner simulators. It would be interesting to carry this out in the future and compare their workings.

It would also be interesting to try out new recommendation strategies. There are many other policy-based reinforcement learning methods out there such as Asynchronous Advantage Actor Critic (A3C) methods. A3C could prove advantageous over standard Actor-Critic as it runs multiple agents on their own copies of an environment in parallel, allowing it to test out more policies in less time. Also, a method such as Deep Deterministic Policy Gradient (DDPF) would be interesting to investigate for educational recommendation as it uses experience replay to learn from all of its previous experiences and not just its more recent outcomes.

The use of learner simulators can be used to train any educational-based reinforcement learning system, not just for recommending educational content. An example that

was provided previously was to train an intelligent RL agent to detect students at risk of failure or dropping out. Here learner simulators can be used to help train an RL agent to detect clear signs of failure, such as a decline in grades over time, missing assignments and tests, etc. Another suitable application would be to train an intelligent agent to create custom study plans or revision notes based on areas students are struggling with. The possibilities in adaptive education are endless and learner simulators can be used in many of these scenarios.

7.5 Final Thoughts

The main objective for this dissertation was to investigate whether knowledge tracing algorithms could be used to implement learner simulators for mimicking human learner behaviour, and if these learner simulators could be used to train an effective recommendation agent. This objective came about as the availability of historic learner data has long been an issue for researchers in the field of adaptive education, making it much more difficult to implement adaptive educational systems. Also, researchers would require huge amounts of human learners to train their intelligent agents, which is not always feasible. Through this dissertation, we have proven that learner simulators can in fact be implemented based on knowledge tracing algorithms, and that they can be used to train recommendation policies. With the use of learner simulators, researchers and engineers can now investigate and implement their own adaptive educational systems without having to worry about data sources and large amounts of human resources. For this reason, learner simulators have the potential to pave ways for many advancements in the field of adaptive education.

Bibliography

Awais, M., Habiba, U., Khalid, H., Shoaib, M., and Arshad, S. (2019). An adaptive feedback system to improve student performance based on collaborative behavior. *IEEE Access*, PP:1–1.

Badrinath, A., Wang, F., and Pardos, Z. A. (2021). pybkt: An accessible python library of bayesian knowledge tracing models. *CoRR*, abs/2105.00385.

Corbett, A. T. and Anderson, J. R. (1994). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4.

Lindsey, R. V., Shroyer, J. D., Pashler, H., and Mozer, M. (2014). Improving students' long-term knowledge retention through personalized review. *Psychological Science*, 25:639 – 647.

Liu, Q., Shen, S., Huang, Z., Chen, E., and Zheng, Y. (2021). A survey of knowledge tracing. *CoRR*, abs/2105.15106.

Nurjanah, D. (2016). Good and similar learners' recommendation in adaptive learning systems. pages 434–440.

Pavlik, P. I., Cen, H., and Koedinger, K. R. (2009). Performance factors analysis –a new alternative to knowledge tracing. In *Proceedings of the 2009 Conference on Artificial Intelligence in Education: Building Learning Systems That Care: From Knowledge Representation to Affective Modelling*, page 531–538, NLD. IOS Press.

Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L., and Sohl-Dickstein, J. (2015). Deep knowledge tracing. *ArXiv*, abs/1506.05908.

PSLC DataShop (2021). PSLC DataShop. `https://pslcdatashop.web.cmu.edu/`. Accessed: 2021-08-22.

Ramirez, S., El Mawas, N., and Heutte, J. (2021). Machine Learning techniques for Knowledge Tracing: A systematic literature review.

Vie, J.-J. and Kashima, H. (2019). Knowledge Tracing Machines: Factorization Machines for Knowledge Tracing. pages 750–757.

V.R, D. R., Tripathy, B., Singh, T., and Khanna, S. (2014). Reinforcement learning approach towards effective content recommendation in mooc environments. *Proceedings of the 2014 IEEE International Conference on MOOCs, Innovation and Technology in Education, IEEE MITE 2014*, pages 285–289.

Wolff, A., Zdráhal, Z., Herrmannova, D., Kuzilek, J., and Hlosta, M. (2014). Developing predictive models for early detection of at-risk students on distance learning modules. *CEUR Workshop Proceedings*, 1137.

Appendix A

Comparative Analysis Appendix

A.1 BKT Model Parameters

```
------------------------------------------------
BKT Model Parameters
------------------------------------------------



                        value
skill param   class
7     prior   default 0.11072
      learns  default 0.01213
      guesses default 0.57320
      slips   default 0.00160
      forgets default 0.00000
...                       ...
113   prior   default 0.49989
      learns  default 0.02002
      guesses default 0.00015
      slips   default 0.04848
      forgets default 0.00000

[615 rows x 1 columns]
```

Figure A.1: Parameters of a fitted BKT model on ASSISTments 2009 data

A.2 BKT Model Predictions

```
------------------------------------------------
BKT Model Predictions
------------------------------------------------


       user_id skill_name  correct_prediction
33           1         82                   1
34           1         82                   0
797          1         30                   0
4832         1         59                   1
16515        1         32                   1
20655        1         53                   0
21191        1         21                   1
25858        1         14                   1
27119        1         49                   0
30987        1         98                   1


------------------------------------------------
BKT Results
------------------------------------------------


Standard BKT RMSE: 0.4163354725931473
Standard BKT AUC: 0.7592980925923227
Standard BKT MAE: 0.3496961173789976
```

Figure A.2: BKT model output predictions and metrics on ASSISTments 2009 data

A.3 Metrics for all models and datasets

ASSIST 2009 Results

Model	AUC	RMSE	MAE
BKT	0.76	0.42	0.35
BKT+F	0.82	0.39	0.32
PFA	0.71	0.44	0.39
DKT	0.84	0.43	0.35

Table A.1: Results obtained for ASSISTments 2009 dataset

ASSIST 2015 Results

Model	AUC	RMSE	MAE
BKT	0.70	0.42	0.35
BKT+F	0.70	0.42	0.35
PFA	0.69	0.43	0.36
DKT	0.73	0.41	0.34

Table A.2: Results obtained for ASSISTments 2015 dataset

Spanish 2013 Results

Model	AUC	RMSE	MAE
BKT	0.84	0.36	0.25
BKT+F	0.84	0.36	0.25
PFA	0.83	0.36	0.26
DKT	0.83	0.37	0.25

Table A.3: Results obtained for Spanish 2013 dataset

Statics 2011 Results

Model	AUC	RMSE	MAE
BKT	0.69	0.40	0.32
BKT+F	0.71	0.40	0.31
PFA	0.69	0.40	0.33
DKT	0.83	0.36	0.25

Table A.4: Results obtained for Statics 2011 dataset

Appendix B

Adaptive Recommendation Appendix

B.1 RL Agent Output

```
--------------------------------------------------
pre-test:        [0 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1 1]
post-test:       [0 0 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1]
episode 181   episode reward 7.0   average reward 7.0
actions taken:  [11, 9, 19, 20]
rewards:  [0.8365116420730186, 0.669209313658415, 0.8365116420730186, 4.699999999999999]
--------------------------------------------------
pre-test:        [0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 1 1 1 0]
post-test:       [1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1]
episode 182   episode reward 12.2   average reward 7.0
actions taken:  [3, 0, 17, 15, 20]
rewards:  [0.956352499790037, 0.7650819998320296, 0.8746896591546225, 0.956352499790037, 8.6]
--------------------------------------------------
pre-test:        [1 0 0 0 0 0 1 0 1 1 1 0 0 0 1 1 1 0 1 0 0 0 0 1 1]
post-test:       [1 0 0 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0]
episode 183   episode reward 10.6   average reward 7.0
actions taken:  [7, 11, 20]
rewards:  [0.8746896591546225, 0.9146101038546527, 8.8]
--------------------------------------------------
```

Figure B.1: Example output of 3 episodes taking a learner simulator from pre-test to post-test