ABSTRACT

JUNG, GYUHUN. Exploring Batch Deep Reinforcement Leaning and Multi-Task Learning across Intelligent Tutoring Systems: Lessons Learned. (Under the direction of Dr. Min Chi).

Nowadays, Intelligent Tutoring Systems(ITSs) have been vastly utilized as a means of adaptive education and training for many learners and Reinforcement Learning (RL) has been widely used to decide an appropriate teaching strategy in ITS. RL is a promising method for the decision-making process in online games, robotics, the military, and so on. Among the variations of the RL algorithm, Deep Reinforcement Learning (DRL) has shown remarkable performance by combining deep neural networks with reinforcement learning. In this work, we focused on two issues for improving students' performance by deploying DRL-induced policies across two ITSs. First, we explored **1) how effective the DRL-induced policy is** in a probability ITS. and 2) the potency of DRL with Multi-Task Learning (MTL) in a logic tutor. Here MTL is about training several similar tasks simultaneously to enhance the model performance and the generalization ability for each task.

For the first issue, we induced a Batch DRL policy based on different input settings . The policy is trained by a stable DRL algorithm, Double DQN combined with Long Short Term Memory (LSTM). Our empirical results show that the offline DRL policy significantly improves students' learning performance compared to the Expert hierarchical RL policy. For the second issue, we used the pre-developed model-based DRL framework, Dreamer. Also, the bisimulation metrics, the distance measuring how much two states are behaviorally similar, are combined with this framework for making a generalized policy for performing two different tasks in ITSs. Following the empirical result, although we did not get higher student learning performance than the control group, we analyzed the reasons for this and the future works for overcoming the limitations of MTL in real environments.

Exploring Batch Deep Reinforcement Leaning and Multi-Task Learning across Intelligent Tutoring Systems: Lessons Learned

by
GYUHUN JUNG

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina
2023

APPROVED BY:

Dr. Matsuda Noboru	Dr. James Lester

Dr. Min Chi
Chair of Advisory Committee

DEDICATION

I dedicate this thesis to the Republic of Korea Army for allowing me to study abroad and to my family for their support, especially my wife, Shinwon Kim, and my daughter, Yiseo Jung.

BIOGRAPHY

Gyuhun Jung was born in Seoul, South Korea on October 28th, 1991. He graduated from Korea Military Academy in 2015, where he majored in physics. After graduating, he served as an officer in the Republic of Korea Army, where he was stationed in multiple infantry divisions and the Joint Security Area. In the Fall of 2021, he began pursuing his Master's degree in Computer Science in the Department of Engineering at North Carolina State University, located in Raleigh, NC. Soon after, he joined Dr. Min Chi's lab to further his studies.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Min Chi for her invaluable mentorship and dedication in helping me complete my thesis. Her insights, constructive feedback, and guidance have been instrumental in shaping my academic and personal growth, and have helped me overcome many obstacles and achieve my goals.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Today, the Republic of Korea(ROK) military is looking to improve its military capabilities by expanding the application of AI in the education and training of its soldiers. Providing personalized learning for a large number of soldiers is one of the most significant challenges because every male in ROK is required to serve in the military for about two years, so while there are a large number of soldiers, there are relatively few officers and NCOs(Non-Commissioned Officers) available to teach them. Therefore, a system that can teach military knowledge according to a soldier's rank and level is a significant challenge for the South Korean military. The ROK military already uses computer-based training (CBT) to teach basic tactics and weapon operation, but the system is not efficient enough to improve the learning gain. This is because most of the CBT just visualizes the military manual and randomly issues questions from a large

pool of questions to evaluate the learner's learning outcomes.

One of the ways to solve the above is to apply an intelligent tutoring system (ITS), an interactive E-learning system without human intervention. The main advantage of ITS is providing tailored instruction and guidance to each learner by leveraging their ability to adapt to the learning process and individual characteristics of the learner[1]. To maximize this advantage, ITS must provide efficient decision-making processes to adapt to each learner's needs. For that, Reinforcement Learning (RL), a machine learning technique for deciding an effective policy by interacting with the environment, is one of the best approaches because the primary purpose of the RL agent is how to make the best decision maximizing the cumulative future reward provided by the environment. RL has also become a widely used method for decision-making in ITS, allowing the system to make appropriate decisions for enhancing each student's learning gain. Among the variations of RL algorithms, Deep Reinforcement Learning (DRL)[19] has shown outstanding performance by combining deep neural networks with traditional reinforcement learning. Using the advantage of neural networks, DRL can approximate the properties in RL without complex feature engineering. DRL has been successfully applied in various fields, such as online games, robotics, and military applications. In the context of ITS, DRL can help personalize the learning experience by adapting to the individual learning process of each learner.

Following our motivation and the advantages of DRL, this work aims to explore the impact of DRL on ITS and learn lessons about an effective way to apply DRL in ITS. Although it is an appropriate way to use ITS for military tasks, it is currently unavailable because of military security. Thus, we used the other effective ITSs, which are widely applied in previous works. We address **two critical issues** for our purpose. First, we explore the effectiveness of DRL-induced policies in ITS. Second, we study the potency of DRL with Multi-Task Learning(MTL)[8], another critical learning paradigm in RL. We mainly used the same procedures by [3], but our results for both issues show different results and analyses.

1.1 DRL for Pyrenees

We applied our DRL-induced policy in Pyrenees, the ITS teaching probabilistic principles to students. Our DRL algorithm is based on Double Deep Q Network[63], which reduces the over-estimation problem of Deep Q Network[51]. Also, our neural network architecture consists of Long Short Term Memory (LSTM)[30] for handling the long time distance of student learning trajectories data. In addition, we compared the performance of two policies trained by two different input settings per problem. In terms of state s and given time t, the first input setting is to use the most recent three state observations $s_t, s_{t-1}, s_{t-2} (k = 3)$ and the second is to use four state observations $s_t, s_{t-1}, s_{t-2}, s_{t-3} (k = 4)$. We found that two policies with different input settings perform differently depending on the problems, and we deployed our DRL policies in real ITS based on this result. Our empirical result shows that our DRL policies provided better learning gains than the random but reasonable policy.

1.2 Multi-Task DRL for Deep Thought

The main drawback of DRL is that it needs a massive amount of data for training effective policy under the interaction with the environment. Although it is easy to collect lots of experiences in a simple domain like an Atari game, it is not the case in the real world. Here, Multi-Task Learning (MTL) can tackle this problem. MTL is a method of learning similar tasks simultaneously[8] while enhancing the ability to perform each task by sharing knowledge from each domain. This method is promising for alleviating the above problem because MTL uses multiple task-related data for training, so a task with a limited amount of data can also be effectively learned. For this advantage, we explored the performance of Multi-Task DRL in Deep Thought, the ITS teaching propositional logic rules. Here, we used the state-of-the-art model-based DRL algorithm named Dreamer[27], which shows high performance by predicting the RL environment in latent state space. In addition, for generating a shared latent state between two environments, we

applied bisimulation metrics[18], which can measure the behavioral similarity between two states. Unfortunately, our induced Multi-Task DRL cannot get the expected performance in our empirical result. Therefore, we analyzed the potential reasons for this result.

In the following, we will briefly explain RL, MTL, and pedagogical policy induction in ITS in Chapter 2. Then, our main works will be divided into two main parts: Part I) DRL for Pyrenees; Part II) Multi-Task DRL for Deep Thought. Each part has a similar structure as follows. First, we will introduce backgrounds and related work to understanding DRL and MTL. Next, we will describe the ITS used in each part. And then, we will present the model architecture and how we trained our model to induce an effective policy for enhancing students learning outcomes. Finally, we will describe the design of the experiments we carried out and the analysis of the empirical results.

CHAPTER

2

BACKGROUND

This chapter will describe general concepts and information to understand our work. First, we introduce the fundamental notions about Reinforcement Learning (RL), the subarea of Machine Learning (ML) for decision-making. Unlike other ML algorithms, RL is aimed to induce optimal policy for an agent to obtain the highest cumulative reward by interacting with the environment. We will explain the main concepts in RL, the classification of RL, and Deep Reinforcement Learning (DRL), which is directly related to the first methodology in our work. In the second part, we will briefly review the background of Multi-Task Learning (MTL) which is related to our second experiment. MTL is an actively researched branch in the ML world. The objective of multi-task learning is training multiple tasks simultaneously and finding a better model than a single-task trained model. We will cover the concepts of multi-task learning and review the

importance of MTL from the perspective of RL. Lastly, we will describe the pedagogical policy induction using RL and MTL in ITS.

2.1 Reinforcement Learning

With outstanding performance in problem-solving, predicting, and analyzing, Artificial Intelligence (AI) is applied in various areas such as education, E-commerce, military, etc. One rapidly growing area in AI is ML. ML is the process of an agent (machine) learning specific tasks by providing task-related data. Thanks to technological advances providing massive and elaborated data, ML has become the core technology in Robotics, Computer Vision, Natural Language Processing, and so on. ML can be divided into three main branches: supervised, unsupervised, and RL. Here, we mainly focus on RL.

RL is a subfield of ML. It is the most promising approach for solving decision-making problems nowadays. Intuitively, RL is a learning process of what behavior is the most beneficial in a given situation by interacting with the environment. For that, it is required for four main components in RL: *policy*, *state*, *action*, and *reward*. In detail, the RL agent aims to induce the *policy*, which can select the best *action* in a given *state*. And here, the best *action* should achieve the maximum future cumulative *reward*. To capture the action providing the maximum numerical reward, the RL agent repeats trial-and-error throughout the interaction with the environment and learns the best action.

Because of these properties, there exist huge differences between RL and those mentioned two main categories of traditional ML types: supervised learning and unsupervised learning. Supervised learning is the learning process with a labeled training dataset. The supervised learning algorithms target to map the function $\hat{y} = f(x)$. In the function, x is the input data, such as features in the data point, and $\hat{y}$ is the output data which should be trained to be equal to labeled data y. On the other hand, unsupervised learning does not need labeled data. As there is no ground truth in the data, an unsupervised learning algorithm finds out the hidden patterns or

structures of data in its way. The main difference between RL and supervised learning is that there is no designated "Answer" in RL. While supervised learning aims to create a function that can predict the ground truth, there is no true solution in reinforcement learning. Instead, the RL agent tries to learn the best action by comparing reward signals in the action spaces. RL is also not included in the category of unsupervised learning, although the RL agent does not use ground truth. This is because the objective of unsupervised learning is capturing hidden patterns, but the main issue in reinforcement learning is maximizing the future reward.

2.1.1 Model-based and Model-free RL

One of the criteria for splitting RL algorithms is model-based and model-free learning. In model-based RL, the agent performs training with the model of the environment. Here, the model can be defined as the abstraction of the environment dynamics where the agent interacts[47]. Most reinforcement learning environments are formed within Markov Decision Process (MDP). In MDP, the states are stochastically moved into the next state following the transition probability. For model-based RL, the agent needs to simulate the basic components of RL and transition probability. Although it is difficult to simulate the real world accurately, model-based RL currently performs well in the Atari learning environment by combining with deep neural network[27; 35].

In contrast, if the RL agent does not have information about dynamics, we call it model-free RL. Model-free RL agents cannot simulate the environment dynamics themselves. Instead, the agent can sample the experience by interacting with the real environment. By exploring the environment in many steps, the agent tries to converge the policy or value function by recording the intermediate results in a table or approximating them using neural networks. There exist various model-free algorithms achieving excellent performance such as Q-learning[66], Deep Q-Network[51], Proximal Policy Optimization[55] using policy gradient method, Soft Actor-Critic[26] and so on.

2.1.2 Online and Offline RL

Another main category of RL is online and offline RL. In online RL, an agent can gather experience by interacting with the environment directly. With that, the agent re-estimates each step's value function or policy. As shown in Figure 2.1, the new action is selected by the current policy. Observing the training environment, the agent gains new experience with the following state and reward. These are used for updating the policy, and this process is repeated in the online RL. Online RL allows an agent to continually explore the environment as much as they want, which makes it easier to induce effective policy. Most of the prior researched RL algorithms are based on the online environment, such as Atari games, Robotics, or financial trades.

On the other hand, in offline RL, the agent cannot interact with the environment. Instead, the complete training data is a priori given and remained as fixed[38]. So, the agent in offline RL aims to induce the optimal policy out of this pre-collected static dataset using sampled batches for training, as it is a so-called batch RL. Following Figure 2.1, the learning process for offline RL is similar to the traditional supervised learning process. Thus, offline RL improves a policy by exploiting collected data as much as possible even if there is no chance to explore the environment[40]. Offline RL can be applied in sensitive areas such as healthcare or education because it is not feasible to induce medical policy while interacting with the patients[3]. In such areas, offline RL has also performed well in combination with deep learning.[32; 36]

2.1.3 Deep Reinforcement Learning

The main problem of the traditional RL algorithm is that it is not available to apply in an environment with a high-dimensional state or action space. Therefore, making more effort to design state representation efficiently is required. The essential advantage of deep learning, which does not need feature engineering, gave clues to solve this problem. Deep Reinforcement Learning (DRL) emphasizes approximating value function or policy through the deep neural network layers. DRL also eliminates the need for manual feature engineering, allowing policies

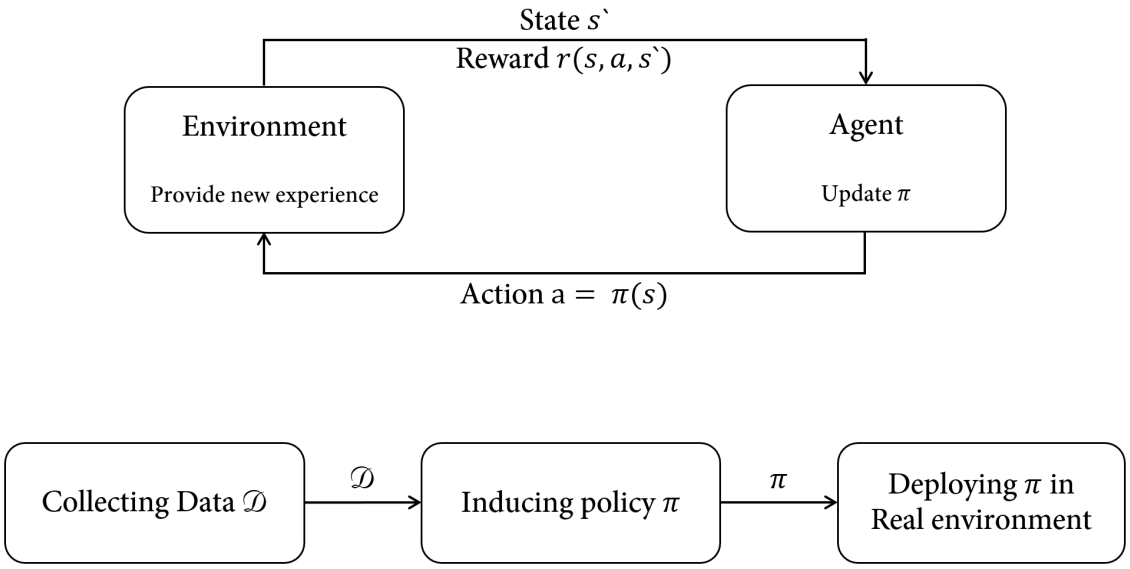

Figure 2.1: Learning procedure of online and offline(Batch) RL. The picture at the top is the process of online RL, and the offline RL drawing is at the bottom.

to be induced from high-dimensional and raw inputs. Starting with Deep Q Networks (DQN)[51], DRL has shown performances that transcend human abilities in a strategic game such as Go [57]. Since then, various versions of DRL have been introduced, such as Double DQN (DDQN)[63] and Dueling DQN[65], Deep Deterministic Policy Gradient (DDPG), which can be applied in continuous action space[43], actor-critic based DRL algorithm[50], Soft Actor-Critic[26] and so on. In addition, DRL has achieved effective results in video games[5; 44], robotics[24], and Natural Language Processes [42].

2.2 Multi-Task Learning

Suppose there are RL policies that are associated but perform different tasks. In this case, one induced RL policy cannot be easily applied to another RL task. This is because each task has different environments, such as state space and reward function. One of the efforts to alleviate this problem is Multi-Task Learning(MTL). MTL is the other learning paradigm in machine

learning. The learning processes of humans influence the task-learning methods of MTL. For example, suppose we learn two similar sports, like Judo and Ju-Jitsu, at the same time. In the process of learning them, the skills learned in judo may be helpful for Ju-Jitsu learning, or vice versa.

As such, MTL is a method of learning several similar tasks simultaneously while enhancing the ability to perform each task [8]. In other words, the purpose of MTL is to create an integrated model with multiple tasks we are interested in. By sharing knowledge from each task, it can be provided an improvement of capabilities over a model with a single task learned. Furthermore, it is expected to have generalized abilities in all tasks learned through the MTL technique. Since MTL uses multiple task-related data for learning, the inefficiency of data that is difficult to be collected can be solved[71]. Due to these advantages, MTL has recently been performed by Computer vision, NLP, education, and so on. [16; 45; 49]

In the view of RL, MTL can also provide RL agents with promising aspects. Basically, the objective of the RL agent is to optimize the decision-making under the collected data. However, the agent needs a lot of interaction with the environment to collect sufficient data for the optimal solution. Although it is easy to gain these data samples in a simple simulated environment, it is not feasible in the real world. To solve this limitation, MTL is used by using the notion of information transfer when we can collect only a limited amount of data to learn each task. [39]

2.3 Pedagogical policy induction in Intelligent Tutoring System (ITS)

Intelligent Tutoring System (ITS), an interactive e-learning environment, is a computer software designed to provide instructional advice to learners without the intervention of human tutors. ITS is mainly designed with pedagogical policy deployment for achieving high learning gain, adaptive learning for each student, one-to-many tutoring, and so on. In the past, pedagogical

policies employed in ITS had small adaptability to learners or stayed at the level for simply implementing existing cognitive or pedagogical theories[13]. But in recent years, the decision-making of pedagogical policies has become more effective because of the employment of data-driven strategies such as RL in ITS.

2.3.1 Reinforcement learning

Leveraging the nature of RL, which makes decisions that maximize future cumulative rewards to be given, many researchers have regarded reinforcement learning as useful means for making pedagogical policies that will maximize students' learning gains. Generally, the main criterion for dividing RL is online and offline. Regarding online RL, Iglesias et al. improved the learning efficiency by using a Q-learning algorithm with an appropriate exploration and exploitation strategy[31]. Fenza et al. utilized the SARSA algorithm to study how intelligent tutor learns the most beneficial next-learning content selection[17]. Georgila et al. used policy iteration-based RL to improve learners' confidence and learning gain in ITS, which learns interpersonal skills [22].

The success of online RL for inducing pedagogical policy highly relies on how accurate the simulation or simulated student is[72]. But building an accurate simulation for a real environment is practically difficult. Because of this hardness, using offline RL for inducing pedagogical policy is also widely researched. Chi et al. showed that policy iteration-based policy trained with pre-collected data bring high learning gains to students in the natural language tutoring system[13]. Shen et al. improved learning gain and reduced students' learning time using offline reinforcement learning based on the least squares policy iteration algorithm[56]. Zhou et al. applied data-driven hierarchical reinforcement learning in ITS[72]. As deep reinforcement learning has recently been in the spotlight, pedagogical policy induction applying offline deep reinforcement learning has also been researched. Ausin et al. applied an offline DRL algorithm to induce tutorial decisions between Worked Example and Problem Solving[2], which inspires

our model in the first case study. Ju et al. identified critical decisions in Deep Q-Network based policy using long and short term rewards [34]. Most recently, Condor et al. applied an offline Double DQN algorithm in ITS for providing automatic instructional feedback in an open-ended question[14]. Although these works can shed light on applying DRL to ITSs, to justify the effectiveness, we still need to investigate whether this way can also positively impact other domains.

2.3.2 Multi-task learning

Meanwhile, previous works for applying multi-task RL to different ITSs are limited so far. [10] proposed a deep learning-based model to predict students' hint-taking and knowledge state. [20] showed that a predictive student modeling using MTL provided a better performance compared to a single-task model in science education ITS. But both of them do not use the RL strategy to formulate their model.

Part I

Deep Reinforcement Learning (DRL) for Pyrenees

CHAPTER

3

DEEP REINFORCEMENT LEARNING (DRL)

3.1 Markov Decision Processes(MDP)

As reinforcement learning is applied in the decision-making process, it needs a model to express the process well. For this reason, the RL algorithms are typically constructed in Markov Decision Process(MDP), a sequential and stochastic model to represent the RL environment. MDP is defined by four main components: state space S, action space A, transition model T, and reward model R. The dynamics of an MDP are as follows: When there is a discrete-time step $t = 0, 1, 2, 3...T$, where T is the terminal time step, there exists a given state in each time

step t which is $s_t \in S$. At that point, the selected action $a_t \in A$ can move the current state to the next state s_{t+1}. This new state is also stochastically decided by transition model T, which is denoted as probability function $p(s_{t+1}|s_t, a_t)$ where $p : S \times A \times S \to [0,1]$. A numerical reward r_{t+1} is given whenever the state changes. It is provided by reward function $r(s_t, a_t, s_{t+1})$ where $r : S \times A \times S \to R$. This sequence is processed until it arrives at the terminal time step. The main objective of reinforcement learning is to find the best policy to maximize the cumulative future rewards in the sequences of the time step. In the framework of MDP, we can express the cumulative reward as a sum of discounted future rewards called return $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + .. + \gamma^{T-t} r_T = r_t + \gamma G_{t+1}$ where $\gamma \in [0,1]$ is a discount factor indicating the current value of future rewards. As the future immediate reward is stochastically obtained due to the transition function, the RL agent should find out the best policy maximizing the expected return $\mathbb{E}[G_t]$. One way to estimate the expected return is a state-value function $V(s)$. It measures how the current state s is valuable for providing future reward. In MDP, the Bellman equation, a recursive form of expected returns, is widely used for expressing the value function as below.

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \big[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi}(s') \big] \tag{3.1}$$

, where π is a fixed policy. A more advanced form for a state-value function is a state-action value function known as Q-value $Q^{\pi}(s,a)$. In a fixed policy π, this function represents the value of action a in a particular state s. Bellman equation for $Q^{\pi}(s,a)$ is expanded like equation (3.2).

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s',a') \tag{3.2}$$

The ultimate goal of reinforcement learning is finding policies that optimize these value functions. This optimal policy is denoted as $\pi^* = \text{argmax}_{\pi} Q^*(s,a)$ where $Q^*(s,a)$ is the optimal action-value function. To solve this problem, we use Bellman optimal equation as follows.

$$Q^{\pi^*}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*(s',a') \tag{3.3}$$

If we know the MDP's entire dynamics and transition function, we can solve the above Bellman equation through dynamic programming. However, these properties are not provided in most situations. One of the famous algorithms applied to these problems is Q-learning[66]. Q-learning aims to optimize the state-action value function above. However, the Q-learning agent proceeds with training without knowing the dynamics of the environment. Instead, the agent experiences new information by exploring the next state based on the Q-value learned in the previous steps. To paraphrase it, the agent has new experiences through data sampling. At the same time, the agent updates the new Q-value by exploiting the current Q-value and the next possible optimal Q-value. This update process is expressed by equation (3.4).

$$Q_{new}(s,a) = (1-\alpha)Q(s,a) + \alpha\big(r(s,a) + \gamma \max_{a'} Q(s',a')\big) \tag{3.4}$$

,where $\alpha \in [0,1]$ is the learning rate. In short, the Q-learning agent interacts with the environment to have a new experience without dynamics and renews the Q-value exploiting the experience and the pre-trained Q-value in the past. It basically has the advantages of simplicity and sample efficiency.

3.2 Q-value based DRL

The main drawback of traditional Q-learning is that it cannot be implemented in a complex and high-dimensional state. New variations of Q-learning by combining with the neural network are developed to alleviate this problem. We are supposed to introduce three representative Q-value based DRL algorithms: Deep Q-Network[51], Double DQN[63] and Dueling DQN[65].

Deep Q-Network

Deep Q-Network(DQN) is an algorithm that combines Q-learning with deep learning[51]. For approximating the state-action value function, DQN uses deep neural networks for handling a large space of states. With that, DQN allows calculating all state-action value functions in action space with particular state input. One characteristic of DQN is that two prime networks are constructed in neural network architecture: Main network θ and Target network θ^{-}. The main network is used to approximate the current Q-value $Q(s,a;\theta)$ and is updated in every step. The Target network initially has the same structure as the main network. Then, it is used to approximate the target value to be trained, as shown in the equation (3.5).

$$y := r + \gamma \max_{a'} Q(s',a';\theta^{-}) \tag{3.5}$$

The main network is trained by minimizing the loss function in equation (3.6).

$$\mathcal{L} = \mathbb{E}\left[(Q(s,a;\theta) - y)^2\right] \tag{3.6}$$

During the parameter updating process, the target network is not updated for a few steps. This is because if the target network is trained per every step, the approximated value by the main network is oscillated due to the rapidly changed target network. To prevent this, the target network is frozen for a certain period of time and updated per multiple constant steps. Another feature is the usage of an experience replay buffer. This buffer contains the last millions of tuples (s,a,r,s'). The agent uses randomly sampled mini-batches of buffer in each training step. By doing this, the state correlation of each data point in batches decreases, and the approximated value converges better.

Double DQN

Double DQN is an advanced form of DQN. The main change of Double DQN is based on the observation that DQN can overestimate the Q-value. As shown in equation (3.5), the target network in DQN estimates the target value as the maximum Q-value. This can substantially overestimate Q-value in several Atari games[63]. To alleviate this, the main network θ is used to select the best action for the following action.(equation (3.7)) Target network θ^- is only used for estimating Q-value of the selected best action. (equation (3.8)). Double DQN reduces the overoptimism and shows more stability than DQN during the training phase[63].

$$a' = \text{argmax}_{a'} Q(s', a'; \theta) \tag{3.7}$$

$$y := r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta); \theta^-) \tag{3.8}$$

Dueling DQN

Dueling DQN is another Q-value based algorithm with dueling architecture[65]. In the dueling architecture, the state-action value is separated as a value function and an advantage function, as shown in (3.9).

$$Q^\pi(s,a) = V^\pi(s) + A^\pi(s,a) \tag{3.9}$$

,where $A^\pi(s,a)$ is an advantage function. It can be intuitively understood as $V^\pi(s)$ is the value of the state regardless of the action. This separation can be beneficial because dueling architecture can learn more valuable states than others, and it is unnecessary to approximate all value functions for each action[65]. In Dueling DQN, the last neural network layer is separated into two parts (NN layer α and β). After then, each of the parts estimates the value function(β) and advantage function(α), respectively. The proposed equation for this process is in (3.10).

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} A(s,a',\theta,\alpha)\right) \tag{3.10}$$

CHAPTER

4

PROBABILITY TUTOR: PYRENEES

Pyrenees is an ITS teaching fundamental probabilistic theory such as addition, complement, Bayes rule, etc. It explicitly taught students how to employ a general principle to calculate the required probability of events. It can also provide immediate feedback for incorrect actions by students[12]. We will briefly describe the interface and learning phase of Pyrenees next.

4.1 Tutor Interface

Figure 2.2 shows the interface by the Pyrenees. The current problem is on *the Problem statement* window. The probabilities of events on the problems can be explicit or implicit(applying the problem in real life). In the *Variables window*, students can define variables by directly extracting them from problems or using probabilistic equations. Through the *Dialog window*, the tutor

provides students feedback, hints, or instructions for the next step. And from the *Response window*, students can interact with the tutors by selecting their next actions. Throughout the interaction, the equations required for solving the problem are displayed on *the Equation window*.

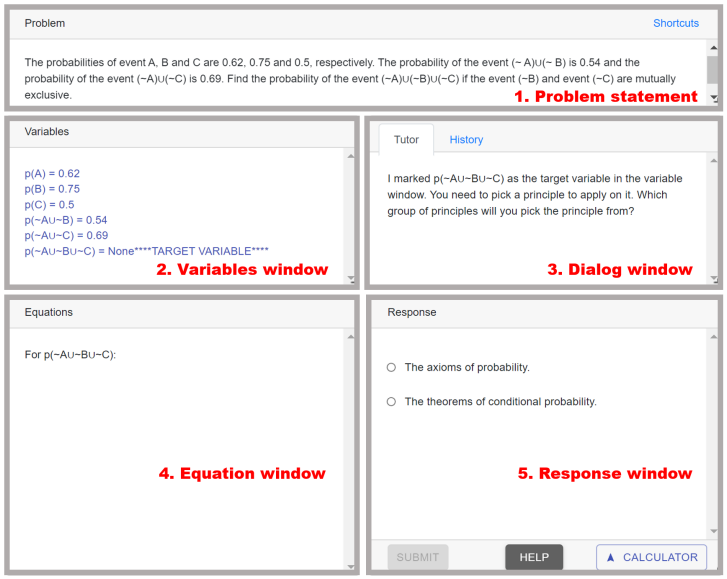

Figure 4.1: Tutor Interface of Pyrenees

4.2 Procedure

The procedure can be divided into four phases: Textbook, Pre-test, Training, and Post-test. In the textbook phase, the tutor gives students a general description of the probabilistic theorem. From that, students can remind the required principles before solving the problems. Then, students take a pre-test containing 14 problems with single or multiple principles. Here, the tutor gives no feedback or hints while students are in the test phase. The pre-test result is graded based on the accuracy of the final result and rule application. In the training phase, students solve 12 problems following the tutor's instruction. Pyrenees explicitly teaches problem-solving

Table 4.1: Problem-solving process example in Pyrenees

Problem : Given $P(A) = 0.4$, $P(B) = 0.5$, $P(\neg A \vee \neg B) = 0.2$, Determine $P(A \wedge B)$.		
	Proposition	**Justification**
Phase 1: *Translating the problem statement*		
1	$P(A)$	Given
2	$P(B)$	Given
3	$P(\neg A \wedge \neg B)$	Given
4	$P(A \wedge B)$	Sought
Phase 2: *Applying principles and generating equations.*		
5	$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$	For $P(A \wedge B)$: Apply Addition Theorem for two events. Delete Sought from $P(A \wedge B)$. Add "Sought": $P(A \vee B)$
6	$P(A \vee B) = P(\neg(\neg A \wedge \neg B))$	For $P(A \vee B)$: Apply The De Morgans Law on $P(A \vee B)$. Delete Sought from $P(A \vee B)$. Add "Sought": $P\neg(\neg A \vee \neg B)$
7	$P(\neg(\neg A \wedge \neg B)) = 1 - P(\neg A \wedge \neg B)$	For $P(\neg(\neg A \wedge \neg B))$: Apply the complement theorem on events $P(\neg(\neg A \wedge \neg B))$ and $P(\neg A \wedge \neg B)$
Phase 3: *Solving equation*		
8	$P(\neg(\neg A \wedge \neg B)) = 0.8$	Solve Equation 7
9	$P(A \vee B) = 0.8$	Solve Equation 8
10	$P(A \wedge B) = 0.1$	Solve Equation 9

strategies to students named Target Variable Strategy(TVS)[11]. This strategy mainly involves three phases: (a) translating the problem statement, (b) applying the principles and generating equations, and (c) solving all equations. In the first phase, students define variables for the given or sought quantities following the problem statement. In the second phase, students apply an appropriate rule for solving the currently sought variables. Here, the tutor provides some of the principles options for applying. If the generated principle contains the unknown variable, the sought variable changes to the unknown variable. Lastly, the tutor helps to solve generated equations if they are enough to solve the problem. Table 2.1 shows a simple example of problem solving process in the Pyrenees using TVS. During the training, students solve the same problem in the same order. Thus, Pyrenees' tutor can only make pedagogical decisions concerning problem type, not contents. There are three problem types while making problem-level decisions: Problem Solving (PS), Worked Example (WE), and FWE (Faded Worked Example). In PS,

students only solve the problem by themselves. The tutor gives some immediate feedback and hints but does not fully show how the problem can be solved. On the other hand, the tutor teaches how to solve the problem step by step without students' effort in WE. FWE is a mixed type of PS and WE. Here, students and tutors solve the problem together by differentiating each step-level problem type. For example, in Figure 2.2, the tutor makes step-level decisions as PS by giving questions for the next action to the student. After then, students take a post-test with 20 problems. They are composed of 14 isomorphic problems to the pre-test problem, and the other 6 non-isomorphic problems. The grading rubrics are the same as those of the pre-test.

CHAPTER

5

DRL POLICY INDUCTION

5.1 Model Architecture

In our first experiment in Pyrenees, we implemented a pedagogical policy induced by one of the representative model-free DRL algorithms: Double DQN. Double DQN, an advanced type of DQN, approximates state-action value function throughout passing Neural Network(NN). More specifically, the model can evaluate the value function $Q(s,a;\theta)$ where a specific state is s and action a applying neural network parameters θ. Although the basic structure of DQN and Double DQN is similar, we select our baseline model as Double DQN because it prevents the model from overestimating value function and also shows better performance on several Atari games[63]. Regarding NN architecture, we combined naive Double DQN with both Fully

Connected and Long Short Term Memory (LSTM) layers inspired by prior work of [2].

5.1.1 Fully Connected and LSTM Layer

A Fully Connected (FC) layer is the simplest form in the types of neural networks. Literally, in the FC layer, all of the units (perceptrons) in the layer are connected to the units of the next layer. Thus, all input units are passed through the next layers, and then, activation functions finally produce a simple weighted sum of n output values where n is the number of output units.

LSTM is a type of Recurrent Neural Network (RNN) architecture combined with a gradient-based method[30]. RNN is an effective method for handling a problem with sequential or time-series data. But simple RNN struggles to solve sequential data problems with long-time distance dependency. To tackle this issue, LSTM has an internal structure for selecting which properties to forget or remember. Figure 3.1 shows the structure and four learning steps(gates) of a single LSTM cell. The first step is to decide which information to remove by passing through forget gate f_t. Next, after determining which states to update by input gate i_t, it generates a candidate memory cell state $\tilde{C}_t$ to update the new cell state. Finally, the output gate o_t filters which information will be transferred to the next layer following the new cell state transformed by the input gate. Equation 5.1 presents how each gate's value is computed, where $[W_f, W_i, W_C, W_o]$ are parameter weights and $[b_f, b_i, b_C, b_o]$ are bias terms. By interacting with these gates, LSTM can maintain the gradient for the states from a long distance of time and resolve the vanishing gradient problem of simple RNN.

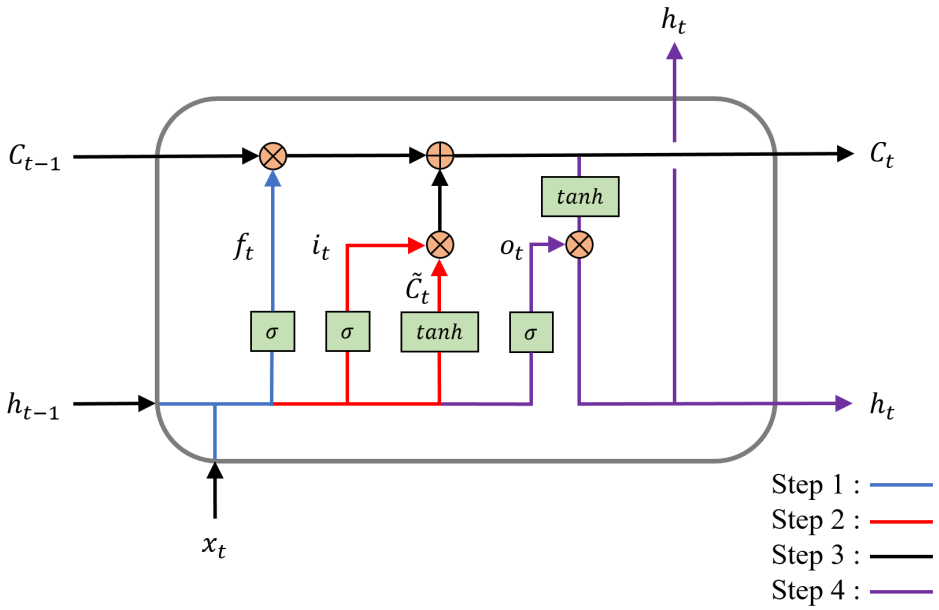

Figure 5.1: LSTM cell structure. In this figure, σ(Sigmoid) and *tanh* are activation functions.

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * tanh(C_t)
\end{aligned}
\tag{5.1}
$$

Our settings of states, actions, and rewards are as follows. In terms of state s, we compared two input settings in a given time t. The first option is to use the most recent three state observations $s_t, s_{t-1}, s_{t-2} (k = 3)$ and the second is to use four state observations $s_t, s_{t-1}, s_{t-2}, s_{t-3} (k = 4)$. Based on the experiment by [2], DRL+LSTM with $k = 3$ shows great performance. Inspired by this result, we decided to use mentioned input settings for exploring the effect of a high number of k. As we use the LSTM layer, These inputs are passed through the layers in a sequential manner. Also, to explore the effectiveness of the pedagogical policy, we construct a model to

approximate the value for the actions: PS or WE in a given state. With regard to the reward function, we used inferred reward in our training process[4], which can distribute the delayed reward for each trajectory by using a neural network. Both ITSs for our experiment have a delayed reward by design. Because we can only know the students learning gain once students finish their learning process, the effectiveness of induced RL policy in ITS is also delayed. For this reason, we used inferred reward, which can generate immediate rewards from delayed rewards using a deep neural network.

Our NN architecture consists of three LSTM layers with 1000 units, respectively, and one FC layer for the output. All layers use Rectified Linear Unit (ReLU) as the activation function. In addition, we use L2 regularization with penalty factor $\lambda = 10^{-4}$ to prevent our model from being overfitting. We trained our model using 500 batches which are randomly selected from a static dataset for 150K steps.

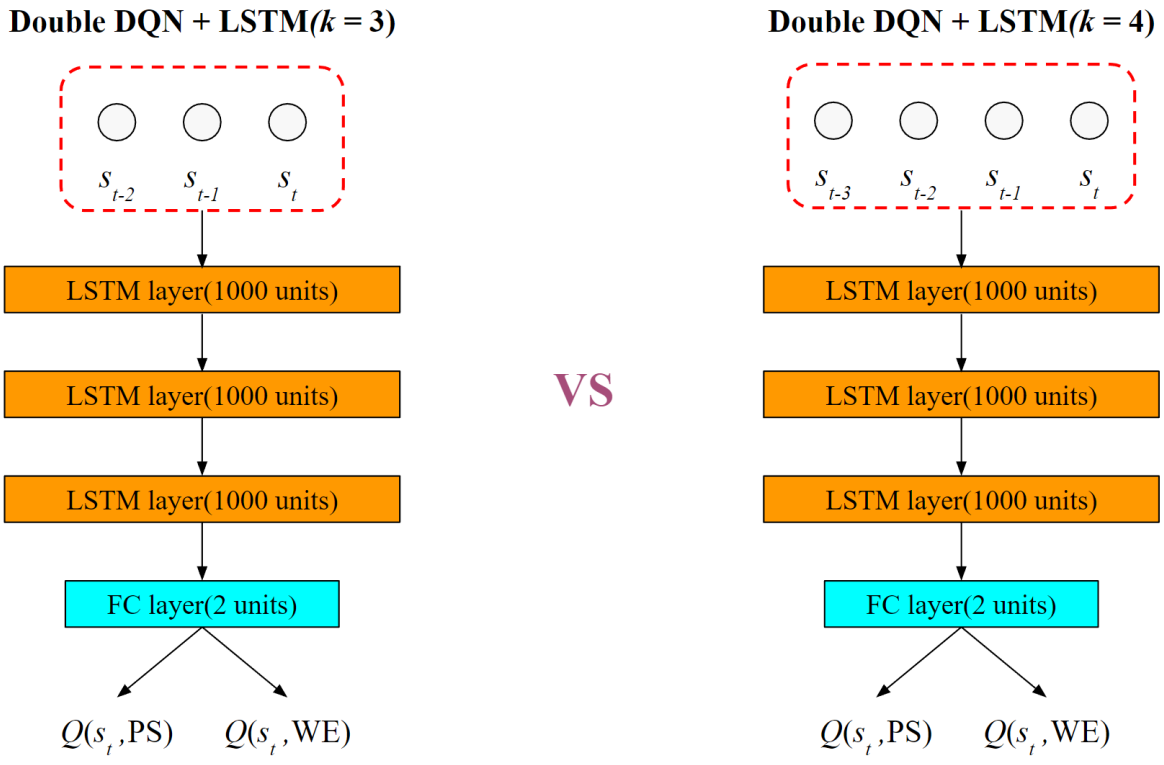

Figure 5.2: Model architecture for our DRL model.

5.2 Model Training

5.2.1 Training Corpus

Our training corpora contain 635 students' problem-solving trajectories in the training phase for several semesters, from Fall 2016 to Spring 2021. We used training corpora for 10 problems in the training phase to induce policies that are adaptive to each problem. For that, we fixed our policy to make problem-level decisions as FWE and concentrated on generating the step-level decision policies for those 10 problems in a training phase. The tutor makes 36 decisions on average per student. For training our model, We extracted 144 state features following the criteria below:

- **Autonomy(16)**: These features are related to the amount of work students have done in a problem or certain time, such as the number of skipped problems *SkipCount* and the number of hints required *hintCount*.
- **Temporal(18)**: These features are related to the time taken by students, such as the average time taken per problem *avgTime*. or deviation of step time per session, level, or problem.
- **Problem Solving(36)**: These features contain information on problems that students encountered in the session, such as the difficulty of the current solved problem, the number of PS and WE problems shown in the current level, or historical information about the current problem.
- **Performance(62)**: These features describe the student's current competence, such as the number of right and wrong applications of rules and the ratio of right actions to total actions.
- **Hints(10)**: These features give more information about hints requested by students, such as the ratio between hint counts and action counts.

We also used normalization because each feature in the data has different scales. We split 90% of students for training data, and the remainder was used for validation data.

During the training, we compared the model effectiveness of two input settings by using Expected Cumulative Rewards (ECR) and Per-Decision Importance Sampling(PDIS). First, ECR is a traditional way to evaluate policy by computing the highest state-action value for all of the starting states[13]. The formula is shown in equation (5.2), where n is the number of initial states, N_i is the number of times that s_i appears on the training data, and $N = \sum_{i=1}^{N} N_i$.

$$ECR_\pi = \frac{1}{N} \sum_{i=1}^{n} N_i \max_a Q(s_i, a) \tag{5.2}$$

Figure 5.3 describes the growth of ECR during training steps. Following the graph, we can check that most of the trained models are getting stable as the training steps are increased.

Although we can evaluate the convergence of state-action value by ECR, the quantitative value of ECR is not appropriate to evaluate the policy because it is not statistically consistent and not appropriate for high-stakes domain[48].

We also apply Off-Policy Evaluation (OPE) to evaluate our induced policies [54]. The evaluation of RL policies is typically done through on-policy methods, where the data used to evaluate the policy is collected by interacting with the environment using the policy being evaluated. However, in certain scenarios, such as offline RL, this is not feasible or desirable. In these cases, we rely on off-policy evaluation metrics to estimate the performance of our policies using data collected by another policy. Thus, we utilized a Per-Decision Importance Sampling(PDIS)[54] as a representative method for off-policy evaluation of our induced policy. PDIS is a type of Importance Sampling(IS) method that approximates the expected reward of the induced policy by computing a ratio of the probability of the target policy $\pi_t(a|s)$ to behavior policy $\pi_b(a|s)$. We used PDIS because it is a commonly used metric in the literature as it is unbiased due to lower variance while estimating the expected reward than simple IS. We compute PDIS value following equation (5.3), where t is a time step and T is the terminal

time of one trajectory. Also, we applied soft-max transformation like equation 5.4 to estimate target policy $\pi_t(a|s)$ because it achieves a great alignment between the theoretical and empirical evaluation results for ITS[33]. For our validation data, we computed an average of PDIS values of each students learning trajectory.

Table 5.1 describes computed PDIS values for two input settings. In the table, the question label only means the order of the problem. This table shows that input settings' impact could differ depending on the problem. In addition, there were no significant patterns about which features of problems make the model with specific input settings better outperform the other. Based on this result, we decided to deploy policies with a mixture of two settings for our first experiment.

$$PDIS_{\pi_t} = \sum_{t=1}^{T} \gamma^{t-1} \left(\prod_{i=1}^{t} \frac{\pi_t(a_i|s_i)}{\pi_b(a_i|s_i)}\right) r_t \tag{5.3}$$

$$\pi_t(a|s) = \frac{e^{Q(s,a)}}{\sum_{a' \in A} e^{Q(s,a')}} \tag{5.4}$$

Table 5.1: PDIS values for $k = 3$ and 4. Electric blue color cells indicate the input settings with a bigger PDIS value.

Question label	**PDIS**($k = 3$)	**PDIS**($k = 4$)
ex132	**0.3810**	0.2093
ex132a	**0.0546**	0.0378
ex152	**11.9814**	0.6722
ex152a	0.0979	**0.1810**
ex152b	**6.9842**	0.3826
ex212	-6.2527	**-6.1222**
ex242	29.7481	**92.3439**
ex252	**9.7131**	1.1763
ex252a	**0.7554**	0.0561
exc137	**0.1765**	0.1599

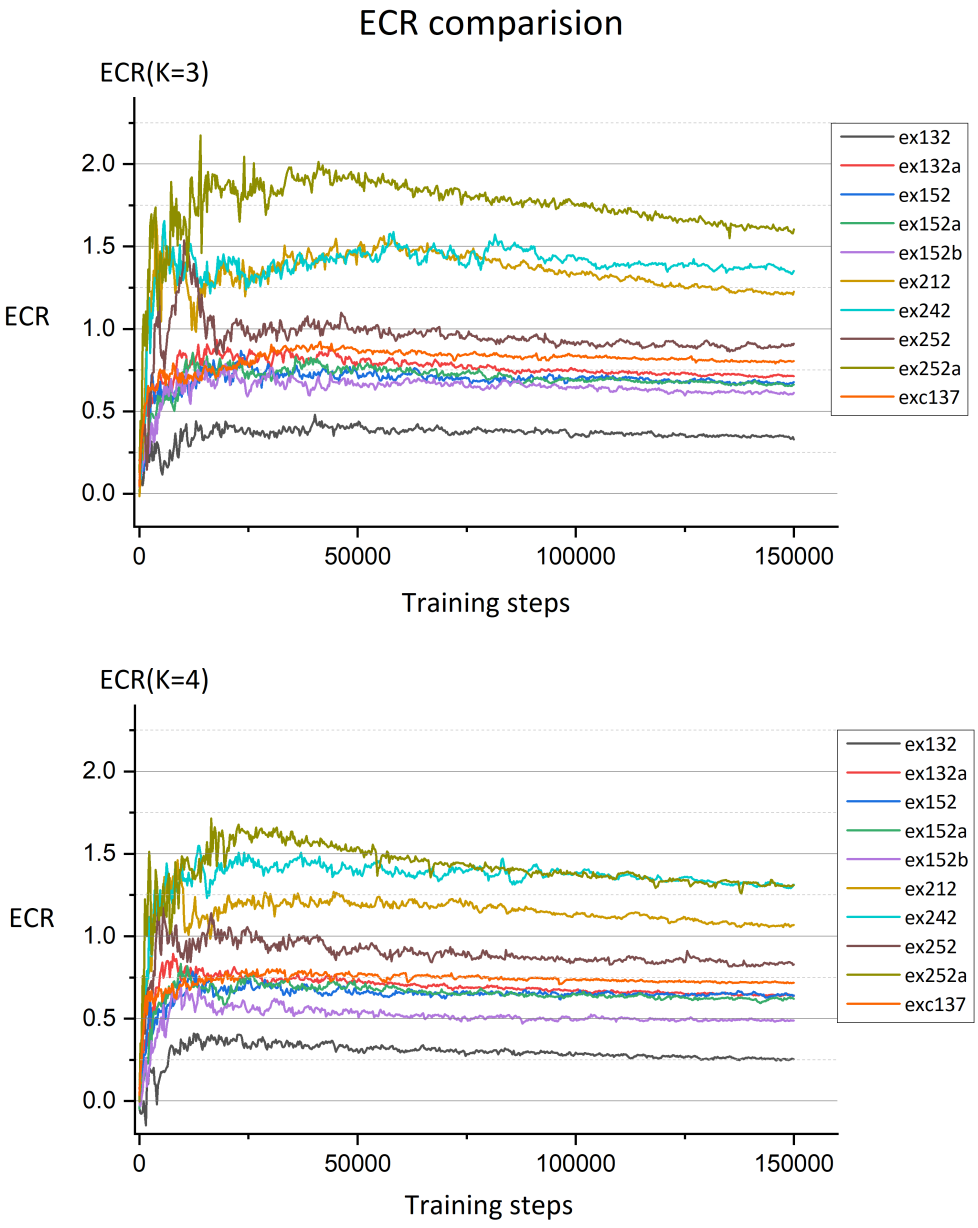

Figure 5.3: ECR evolution of Double DQN+LSTM with $k = 3$ or 4

5.3 Experiment 1: Pyrenees Fall 2022

5.3.1 Experiment Setup

This study was conducted under the Pyrenees tutor. The subjects are undergraduate students who are enrolled in the class in Fall 2022 semester. The study was conducted from November 16th to December 5th. During this period, students could access the system at all times and proceed with their own progress. We divided the experimental group into Control (*Expert*) and Experimental(*DRL*) groups. In the control group, the random but reasonable policy was applied while students were in the training phase. In the experimental group, our trained DRL policies were applied to the tutor. 79 students completed their progress (55 students for control group, 24 students for experimental group). A one-way ANOVA test showed that there is no significant difference on average of pre-test score between the two groups($F(1,77) = 2.654, p = 0.107$); with $Random(\mu = 0.796, \sigma = 0.135)$ and $DRL(\mu = 0.737, \sigma = 0.171)$. This implies that the incoming competencies of students in the two groups are statistically similar.

5.3.2 Experiment Results

In Table 5.2, we summarized the result of our Fall 22 study. The table shows the average score and standard deviation result for five columns. Of the 20 post-test problems, 14 problems were isomorphic types of pre-test, and the remainder consisted of more difficult problems. Thus, We analyzed the post-test score of both general(**Post-test**) and isomorphic(**Post-ISO**) problems. In addition, we analyzed the Normalized Learning Gains(NLG) for the result. NLG is applied for comparing pre-test and post-test performance by taking concepts of normalization(NLG = $posttest - pretest/(1 - pretest)$).

Following the table, our *DRL* policy numerically outperforms the *Expert* policy on both the post-test score and NLG. We conducted ANCOVA and ANOVA tests to verify the result statistically. For the post-test score, an ANCOVA test with pre-test scores as covariates showed

Table 5.2: F22 Pyrenees experiment result.

	Pre-test	**Post-test**	**Post-ISO**	**NLG**	**NLG-ISO**
Expert	**0.792(0.13)**	0.725(0.15)	0.809(0.14)	-0.576(0.77)	-0.001(0.63)
DRL	0.737(0.17)	**0.765(0.25)**	**0.814(0.23)**	**0.140(0.69)**	**0.351(065)**

that the difference in post-test mean scores between the *Expert* and *DRL* policies was statistically significant ($F(1,77) = 8.2965, p = 0.0052$). For the post-ISO score, an ANCOVA test with pre-test scores as covariates showed that there was no significant difference in the difference in post-ISO mean scores between the *Expert* and *DRL* policies ($F(1,77) = 2.3992, p = 0.1256$). On the other hand, the NLG values when using the *DRL* policy were positive, unlike the *Expert* policy. More specifically, a one-way ANOVA test showed that there was a significant difference between the *Expert* and *DRL* policies in both NLG ($F(1,77) = 14.9, p = 0.0002$) and NLG-ISO ($F(1,77) = 5.477, p = 0.0219$) score.

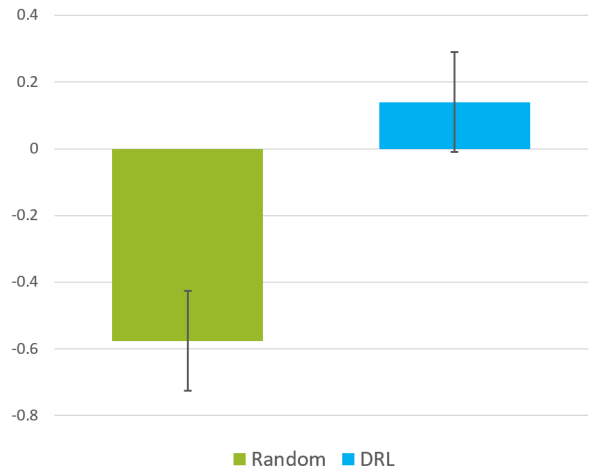

Figure 5.4: Comparison for Normalized Learning Gains in F22 study

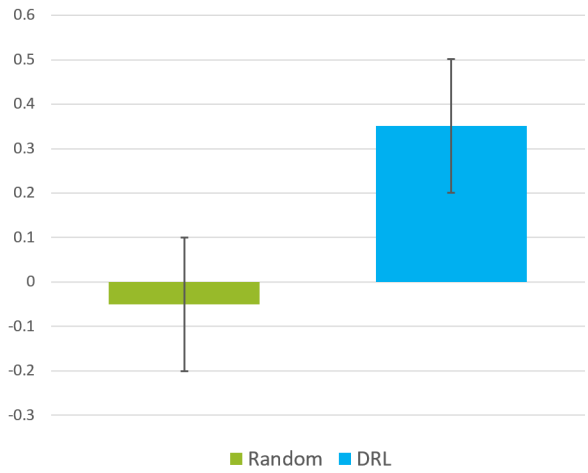

Figure 5.5: Comparison for Normalized Learning Gains(Isomorphic) in F22 study

5.4 Conclusion & Discussion

In this part, we explored the performance of deep reinforcement learning in real ITS. We used the Double DQN algorithm in conjunction with Long Short Term Memory in order to make a robust policy in time-sequential data. We trained our model in an offline approach which uses static data to take into account our environment. By evaluating two different input settings(3 or 4 consecutive states) using Per Decision Important Sampling, we deployed policies with a mixture of two settings for the experiment. Following our Pyrenees result, our induced *DRL* policy can outperform the *Expert* policy, which is a strong baseline model. Thus, we concluded that the offline DRL algorithm could be applied to the real world to improve students' learning performance.

But we need further research to boost the impact of DRL in students' learning phase. As mentioned earlier, we showed that the bigger k did not improve the performance of the model. Also, we cannot capture any patterns of problem features which can decide the better input settings. We need to have further work to identify the reason for this. Also, one of the complements to our model is the interpretability or explainability of DRL policies. Our DRL models can be challenging to interpret, limiting adoption in other real-world settings. Developing

methods to interpret and explain DRL policies could increase the trust and usability of DRL models in ITS environments. We believe it can be a promising future research direction because it can help to increase trust in the model and enable human users to provide feedback or correct errors.

Part II

Multi Task RL (MTRL) for Deep Thought

CHAPTER

6

RELATED WORK

6.1 Multi-Task Reinforcement Learning

The problem of Multi-Task Reinforcement Learning (MTRL) can be defined as obtaining an optimal policy from T tasks under the Markov Decision Processes $\mathcal{M}_t = (\mathcal{S}, \mathcal{A}, T_t, R_t)$ where $t \in \{1, 2, ..., T\}$ with the same state-action space but different transition T and reward model R[7]. As mentioned above, an effective induced policy by MTRL can provide both a reduction of training time for multiple tasks and achievements of improved and generalized performance. Due to these advantages, many approaches exist to achieve an effective MTRL policy. For example, [61] applies model-based RL with the statistics of value function to extract the relationships between multiple tasks. [41] proposed batch MTRL framework, which performs better in a

partially observable MDP environment. [69] focused on resolving data efficiency problems in offline MTRL by applying novel data-sharing skills between each task. In recent years, due to the remarkable achievements of deep reinforcement learning, there have been many attempts to apply it to multi-task learning. [53] combined Deep Q-Network and their novel multi-task learning technique, which can mimic expert decisions in multiple source tasks. [62] represents a novel approach to distill shared DRL policy which can capture common behavior across different tasks. [29]: use the techniques of clipping rewards in the Actor-Critic algorithm to balance the impact of each task learning. [68] proposed a modularization technique to alleviate optimization problems concerning gradient interference of two different Soft Actor-Critic based policies.

The main challenge for MTRL is overcoming the difference in MDPs for each task. If the environments for each task are different, it would not be easy to learn general skills that can be shared across multiple related tasks[64] or filter the wasteful knowledge. To resolve this problem, [67] inferred generalized MDPs over the tasks using a Bayesian framework distribution. [58] leverages context-based representation to overcome negative inferences that affect some aspects only related to a single task. One of the actively researched areas in this area is about learning shared latent representation amongst tasks. Intuitively, shared latent representation can be general knowledge for each task to be learned. If we extract useful generalized knowledge from the domains directly, finding an optimal policy for performing tasks would be easier. [6] focused on learning shared latent representation in the state-action space by using a linear combination of a set of tasks' features. [15] applied a shared deep neural network to extract shared representations among the tasks and showed the advantages of using these abstractions in RL. In our work, we will construct our MTRL model for two ITSs by using shared latent representation learning.

6.2 Latent representation learning

Latent representation learning, a subfield of machine learning, is a technique to map raw observation into latent space. Latent space, also known as latent feature space, is simply an expression of compressed data extracted by original data. Here, the compression implies that it should not only decrease the size of the state but also include features that can express the characteristics of the state well.

In the views of RL, latent representation learning can reduce the limitation of model-based RL algorithm: Difficulty in accurately predicting environment dynamics such as transition or reward function. This limitation is especially evident for multi-dimensional states, which can make 1) unstable environmental predictions and 2)increase time complexity during model training. These problems can be resolved by predicting the environment not on the raw observation states but on the latent space, which encodes only essential information related to the task domain. This technique reduces the time required for learning and facilitates model training by focusing only on representative feature information. In previous studies, RL agents learned through latent presentation learning have shown an improved model performance in simulated games[21; 25; 28].

Although there are various ways to achieve latent presentation, we will focus on the representative method named the reconstruction-based approach. This approach is divided into two phases. In the first step, a raw observation is mapped in latent space(encoding). After that, the latent state is reconstructed to output as similar as possible to the original input(decoding). In these processes, the model is trained to minimize the differences between the original input and reconstructed output. Here, we will introduce two sub-methods among the above approach: Variational Auto Encoder (VAE)[37] and Recurrent State Space Model (RSSM)[28].

VAE is a type of generative neural network in order to learn latent presentation. Unlike the usual autoencoder, which creates a latent state directly for a given input, VAE employs a probabilistic manner for observing latent space. First, the encoder in the VAE produces a

distribution of a mean μ and a standard deviation σ for inputs. Then, The latent state is sampled from the Gaussian distribution $N(\mu, \sigma)$. Finally, this generated latent state undergoes a process of reconstructing the original input by passing through the decoder.

RSSM is another type of reconstruction-based approach introduced by [28] for improving model-based RL performance. The learning process by which RSSM learns the latent state is similar to VAE in that it has the characteristics of stochastic encoding. However, one of the distinguishing features from other models is the simultaneous usage of both the deterministic state model ($h_t = f(h_{t-1}, s_{t-1}, a_{t-1})$) and the stochastic state model ($s_t \sim p(s_t|h_t)$ to generate the latent state. Here, a recurrent neural network is used to generate the deterministic state, and the stochastic state is sampled based on the deterministic state.

CHAPTER

7

LOGIC TUTOR: DEEP THOUGHT

Deep Thought is an intelligent tutoring system teaching basic theories about propositional logic. Basically, Deep Thought shows the premises and allows students to derive a required conclusion by applying logical axioms and theorems like Modus Ponens(MP), Addition, Conjunction, and so on. [60]. Many researchers have applied data-driven hints[52], problem selection[46; 52], or reinforcement learning[2]. This section will describe the Tutor Interface of Deep Thought and how students' learning steps are processed.

7.1 Tutor Interface

Figure 2.3 shows the first round of the Problem Solving(PS) step in Deep Thought. If the current problem type is PS, students should solve the problem by themselves without any intervention

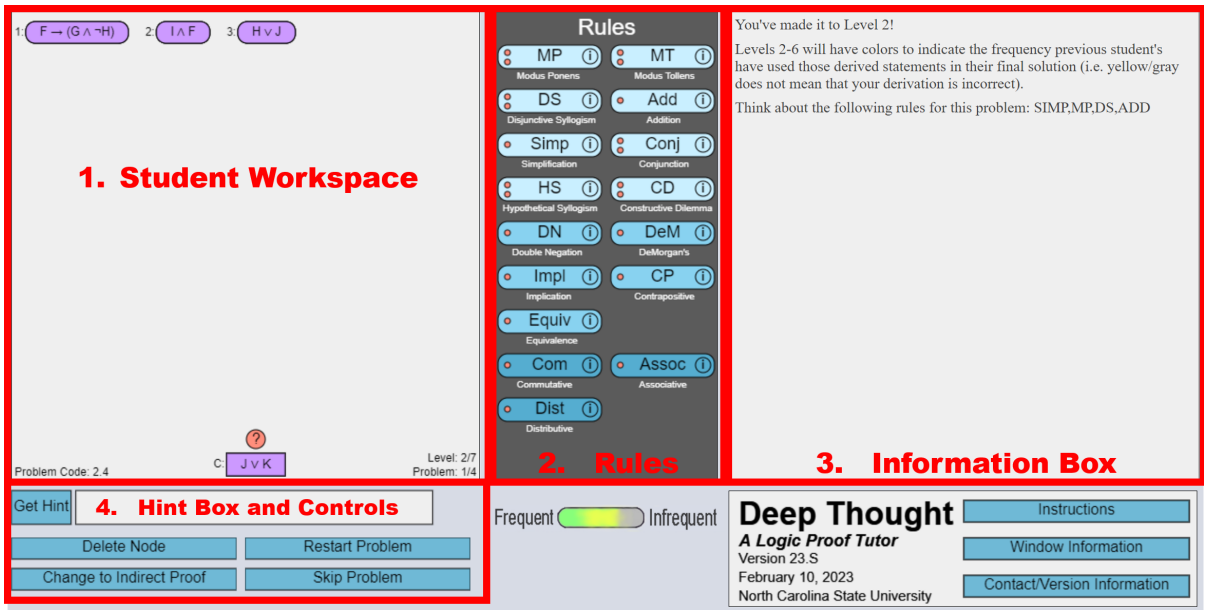

Figure 7.1: Tutor Interface of Deep Thought(Problem Solving)

from the tutor. As we can see, there are four main categories in the Deep Thought interface. First, in the student workspace, a procedure of student for deriving a goal statement is visualized. On the top of the workspace, there are three nodes of premises that can be used for extracting conclusions. By applying propositional logic rules to the premises, students can finally reach the goal node at the bottom of the workspace. The question mark above the conclusion indicates that the problem is not solved yet. Second, there are fundamental logic theorems on the central window. Under the rules box is a frequency bar. This bar shows the frequency of how often the statements derived by students were used for making a conclusion in the past. Third, students can get a hint about which rules are applicable to the question in the information box. Lastly, on the hint and controls box, students can receive a direct hint for deriving rules by clicking the "Get Hint" box. Also, the other options in this part allow students to refresh, skip or solve the problem in indirect proof ways.

Figure 2.4 shows the interface when the current problem type is Worked Example. (WE) Here, tutors will display a standard procedure for how to reach a conclusion without problem-solving by students. There are no rules that appear here, and students can move on to the

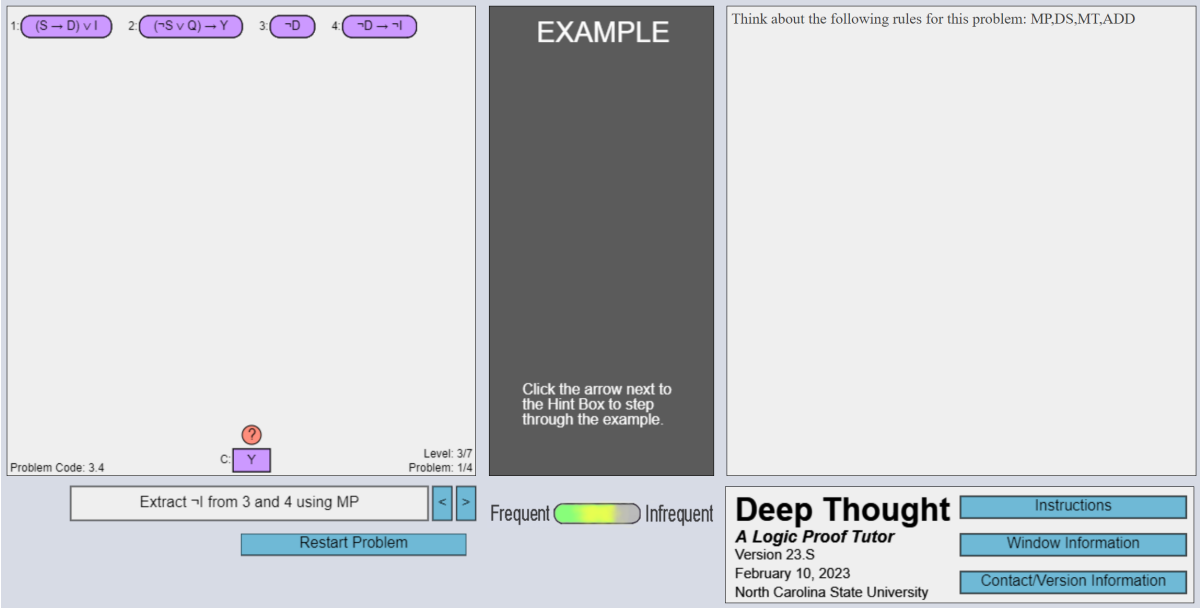

Figure 7.2: Tutor Interface of Deep Thought(Worked Example)

previous or next step by clicking arrows right next to the hint box. Students can explore forward or backward procedures in the WE problem, and each step is also visualized in the workspace.

7.2 Procedure

In the problem in Figure 2.1, let's check problem-solving process step by step. Figure 2.4 shows an example of this process. As mentioned on the interface, the green color nodes have a high frequency for the conclusion. And yellow and gray nodes have medium and low frequencies, respectively. For the first step, node 5 is derived by applying simplification (Simp) in node 2. Here, if the tutor verifies that the rule application is correct, the derived node(5) is connected from its base node(2). Also, the number and applied rule is labeled to the derived node. Similarly, node 5 and node 1 can derive node 6 by applying Modus Ponens(MP). Likewise, node 6 derives node 8 by clicking rule Simp, and nodes 3 and 8 derive node 10 using the rule Disjunctive Syllogism(DS). Finally, the student made a conclusion by applying Addition to node 10. Here, it does not mean that node 4,7,9 is incorrectly derived. It means that these nodes have not been

used much to draw conclusions.

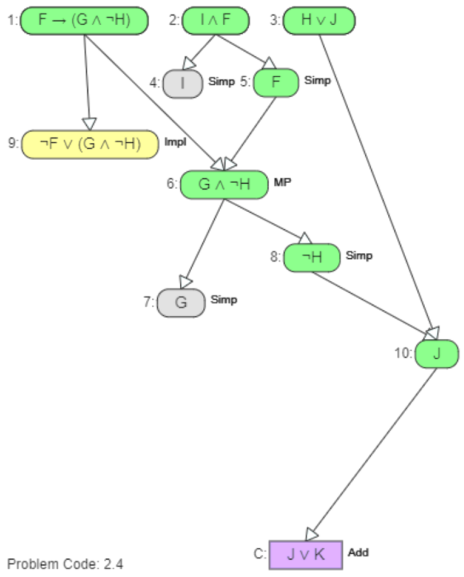

Figure 7.3: Problem solving process example in Deep Thought

Students should proceed to seven levels to complete the learning procedure of Deep Thought. The first level is composed of four questions. In the first two questions, the tutor familiarizes students with the Deep Thought system and interface. Here, students also solve the first two questions with instructions and tutor examples. The following two questions, called pre-test, are solved only by students.

Next, the 2 to 6 levels are the training phase. Each level consists of eight problems. These eight problems have a similarity in applying specific logic rules, but the difficulties are different. In each level, Deep Thought selects four out of eight problems. Three problems are selected for training, and the difficulty is adapted to the student's current performance. The last problem is a level post-test problem, and it is fixed as the most challenging problem out of the eight problems set. One thing to note is that the induced pedagogical policies are employed only in the training phase. As Deep Thought uses the same problem selection algorithm for all students, similar to Pyrenees, our pedagogical decision using DRL is concentrated on selecting the current problem

type: PS or WE [3].

Lastly, students take the post-test after finishing the training phase. These are composed of more difficult problems compared to the problems in the training phase. The score for the pre-test and post-test are calculated by two criteria. One is the number of incorrect actions, and the other is the time for solving the problem. More specifically, a good score can be obtained by making small mistakes and taking less time to finish problem [3].

CHAPTER

8

MTRL POLICY INDUCTION

8.1 Model architecture

For the second experiment in Deep Thought, we used a model-based DRL algorithm named Dreamer[27]. Unlike the first experiment, we tried to generate a multi-task DRL policy, which is expected to improve performance by merging two different ITSs. For unifying the two systems, we combined Dreamer with bisimulation metrics which can quantitatively measure how similar two states are from the views of long-term behavior[18].

As previously described, an effective multi-task RL policy can achieve better performance than an RL policy trained for single-task. To confirm this, we evaluate that multi-task DRL policy theoretically outperforms single-task DRL models and empirically verify its effectiveness

by experimenting in the real environment. Our model is mainly divided into two approaches, as shown below.

1. **Latent representation learning**
2. **Shared latent representation learning**

8.1.1 Latent representation learning

For our approach, we constructed our model architecture on the basis of Dreamer, a model-based DRL with the environment of latent space.[27]. The Dreamer agent learns the latent dynamics model based on the RSSM to predict and estimate the action and value model in that latent space. Similarly, our model architecture for learning latent presentation was constructed as follows. We numbered our latent space model**(1)**, value model**(2)** and action model**(3)**.

$$\textbf{1.1) Encoder} : \hat{o}_t \sim enc(o_t) \tag{8.1}$$

$$\textbf{1.2) Dynamics} : p(z_t|z_{t-1}, a_{t-1}, \hat{o}_t) \tag{8.2}$$

$$\textbf{1.3) Reward} : r_t \sim R(z_t) \tag{8.3}$$

$$\textbf{1.4) Decoder} : o_t \sim dec(z_t) \tag{8.4}$$

$$\textbf{2) Value} : V_t \sim V(z_t) \tag{8.5}$$

$$\textbf{3) Action} : a_t \sim A(z_t) \tag{8.6}$$

To further explain the model, we used fully connected layers for our encoder and decoder. Also, the dynamics model is based on RSSM architecture. Appendix A describes the dynamics model's inner structure for a more detailed explanation. And the rest architecture is similar to Dreamer. One thing to note is that we used two different encoder/decoder sets for two ITSs. It means that we constructed enc_1 and enc_2 for embedding the raw observation into latent state

$\hat{o}_t^{ITS1}$ and $\hat{o}_t^{ITS2}$ respectively and dec_1 and dec_2 reconstructing latent state to original observation state.

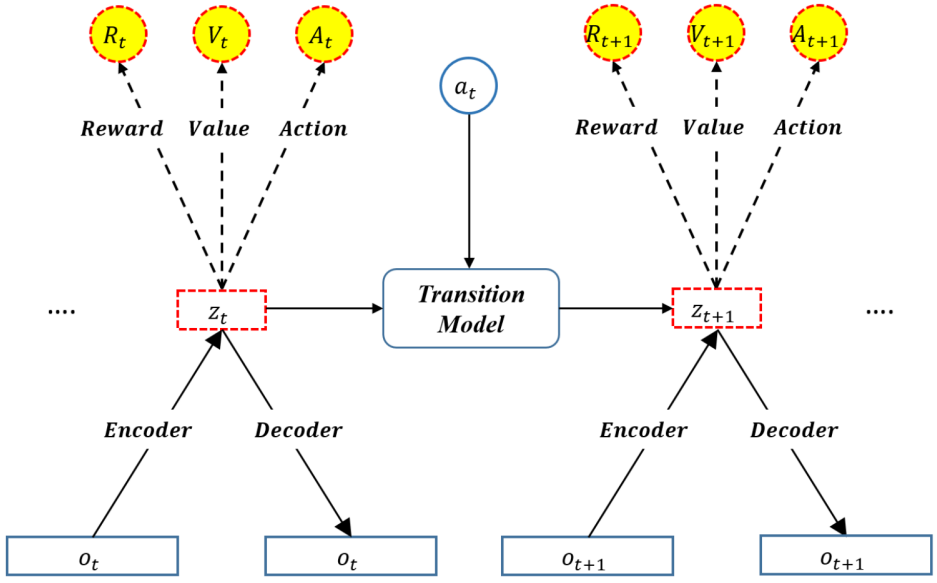

Figure 8.1: Model architecture generating latent space model for one ITS domain

8.1.2 Shared latent representation learning

Aiming to achieve generalization ability between two ITSs: Deep Thought and Pyrenees, we will describe how we apply the bisimulation metrics in this section. Our purpose is to find out the latent space of two different ITSs and the shared latent representation, which means a combination of two different environments(Markov Decision Processes) in ITSs. We suppose that this integration will improve capabilities in all tasks simultaneously learned since we use various related tasks for learning. To accomplish it, we approached generating a shared latent state between two different MDPs by using bisimulation metrics which can quantitatively measure how the two states in MDPs are behaviorally similar[18].

Before we describe the concept of bisimulation metrics, let's begin by assuming that our

two ITS environments can be defined as Markov Decision Process(MDP). MDP has four components: state s, action space a, transition model p, and reward model r. When there is a discrete time step $t = 0,1,2,3...T$, where T is the terminal time step, a given state s_t can be converted to the new state s_{t+1} by the selected action a_t. This new state is stochastically decided by transition model $p(s_{t+1}|s_t,a_t)$. A reward r_{t+1} is given by reward function $r(s_t,a_t,s_{t+1})$ if we arrive new state. In the framework of MDP, we can express the cumulative reward as a sum of discounted future rewards called Return, $G_t = \sum_{t=1}^{T} \gamma^{t-1} r_t$ where γ is a discount factor. Our RL agent aims to find out policy maximizing the expected return, $\mathbb{E}[G_t]$.

Although MDP is a great model for planning or decision-making, it is challenging to apply in practice because the state space of MDP is too large in many real environments, which can potentially increase the time complexity and occur a divergence of value functions. To alleviate it, the notion of bisimulation is widely used. Bisimulation is a concept of a relationship between two different states in MDPs. This relationship can cluster two states if they are behaviorally equivalent. More specifically, if the two states s_i and s_j have the same reward function and transition to the next bisimilar state, we can define the two states are bisimilar. The relationship can be defined as follow [23].

$$\begin{aligned} &\textbf{Condition 1.} \quad \forall a \in A, R(s_i,a) = R(s_j,a) \\ &\textbf{Condition 2.} \quad \forall a, \forall C, \sum_{s' \in C} p(s_i,a,s') = \sum_{s' \in C} p(s_j,a,s') \end{aligned} \tag{8.7}$$

, where C is the set of all clusters of states that are bisimilarly equivalent.

The limitation of the bisimulation relationship is too strict to be satisfied. The two states are no longer considered bisimilar if there are slight differences in reward and transition function. The proposed way to solve it is bisimulation metrics. Bisimulation metrics[18] is a pseudometrics-based distance that can measure how much the two states are behaviorally similar. The equation for bisimulation metrics is followed below.

$$Bisim(s_i, s_j) = \max_{a \in A} \big((1-c)|r^a_{s_i} - r^a_{s_j}| + cW_2(p^a_{s_i}, p^a_{s_j}) \big) \tag{8.8}$$

,where c is a discount factor and W_2 is a 2-Wasserstein distance between two different probability distribution. To make a generalized policy, we enforced our neural network-based latent space model(equations 8.1 to 8.4) to contain the characteristics of bisimulation metrics. For that, by including a loss function in the latent space model, we approximated the L1 distance d between two latent states to be as same as bisimulation metrics as follows. We numbered our bisimulation metrics model as **(4)**.

$$\textbf{4) Bisimulation metrics} : d(z_i, z_j) \sim Bisim(z_i, z_j) \tag{8.9}$$

In detail for the training process, we sampled batches by two ITS datasets B_{DT}, B_{Py}. During training for bisimulation metrics, we made a pair of two states $z_i \in B_{DT}$ and $z_j \in B_{Py}$ for computing bisimulation metrics. This approach is inspired by [70] and [9]. [70] proposed a framework to find out domain invariant characteristics in different states using bisimulation metrics. By reducing irrelevant features, they provide a generalization ability and improved performance in a single task. [9] applies bisimulation metrics to transfer policy between two different MDPs.

In summary, our model architecture was initially constructed by following three parts: 1) latent space model(encoder, decoder, transition model, reward model), 2) bisimulation metrics in latent space, and 3)estimated value and action model. Our encoder comprises three FC layers with $256 \times 128 \times 64$ units. For the decoder, we used three FC layers with $128 \times 128 \times N$ units, where N is the number of features in raw observation. We used RSSM for the transition model, which consists of 32 latent units for each deterministic and stochastic state. For training, we sampled 64 batches with a length of 3 and used Adam as an optimizer. We have learning rates of $6 \times 10^{-4}, 8 \times 10^{-5}, 8 \times 10^{-5}, 10^{-5}$ for latent space, value, action, and bisimulation model,

respectively. The whole loss function during our model training can be defined as follows.

$$\mathcal{L} = \mathcal{L}_{latent} + \mathcal{L}_{value} + \mathcal{L}_{action} \tag{8.10}$$

$$\mathcal{L}_{latent} = \mathcal{L}_{enc} + \mathcal{L}_{dec} + \mathcal{L}_{transition} + \mathcal{L}_{reward} + \mathcal{L}_{bisim} \tag{8.11}$$

Figure 8.2 shows how our model works. The flow of model architecture consists of three main steps. First, raw observations of each ITS domain are mapped into a latent state using different encoders. Second, the distance of latent states from each domain is trained to be the same as bisimulation metrics for generating shared latent states between each task. Third, the other properties like transition, reward, value, and policy networks are trained in the latent space.

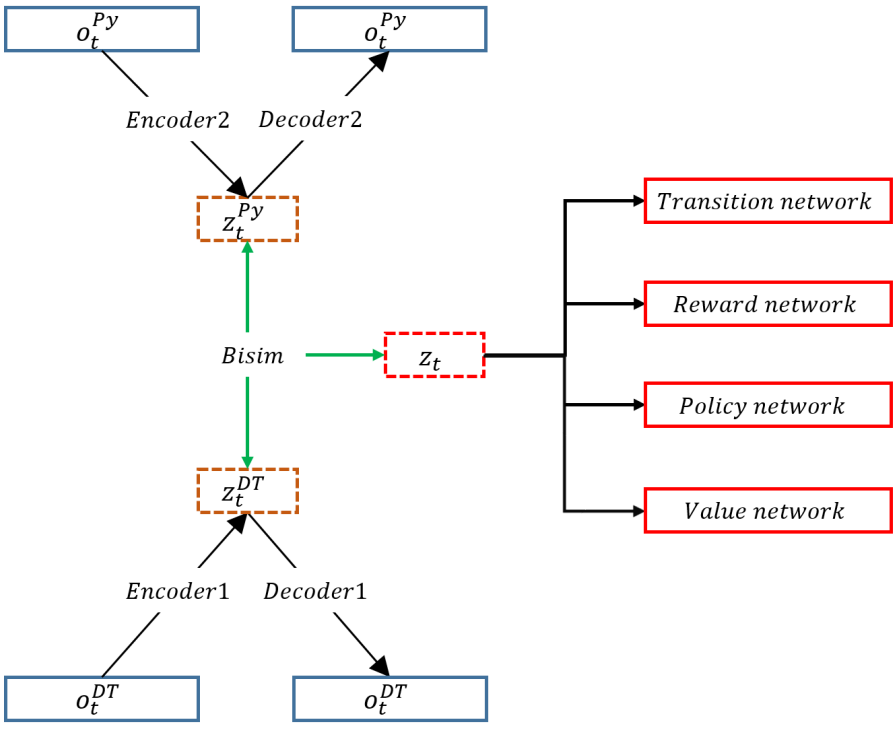

Figure 8.2: Model architecture generating shared latent space model for two ITSs domain.

8.2 Model Training

8.2.1 Training Corpus

We used log data of both Deep Thought and Pyrenees for our training corpora. Here, we used problem-level decision data which is different from our first case study. In Deep Thought data, there are 1330 students' trajectories of problem solving in the training phase from the Fall 17 to Fall 20 semester. The tutor makes 24 decisions on average per student. For training, We extracted 144 state features. In Pyrenees data, there are 1148 students from Fall 16 to Fall 19 semester. The Pyrenees tutor makes 12 decisions for each student. For training, We extracted 133 state features. The categories for feature selection are the same as that of Deep Thought, but the inner elements are somewhat different because of the differences in the system and teaching subjects between the two ITSs. Similar to our first experiment, we applied normalization for each data and split 90% of students' data into training data and the remainder for validation data. Here, we evaluate induced policy's performance only for our experiment's Deep Thought task.

As we mentioned, we followed the architecture of Dreamer in our model. The difference is combining bisimulation metrics training with the Dreamer model to achieve better performance by learning two tasks. However, during training, the combined model suffered from divergence of the loss function. To resolve this instability, we calibrated the learning rate on the training step for bisimulation metrics to be smaller than the other learning rates. Then, we compared the PDIS value of the trained model for the Deep Thought task with that of four Q-value based models: DQN + LSTM($k = 3, 4$) and Double DQN + LSTM($k = 3, 4$).

We computed the average of 50 PDIS values for the multi-task learning model. This is because the model uses both deterministic and stochastic states for getting the latent state. As it means that the same raw observations can be in different latent states per evaluation, we computed several times for the final PDIS value. Table 8.1 shows the final PDIS values for five models. It indicates that the PDIS value for the multi-task learning model is bigger than the

Table 8.1: PDIS values for multi-task reinforcement learning model and DQN-based models. Electric blue color cells indicate the input settings with a bigger PDIS value.

Model	PDIS
DQN+LSTM($k = 3$)	5991.92
DQN+LSTM($k = 4$)	3958.61
DDQN+LSTM($k = 3$)	5619.66
DDQN+LSTM($k = 4$)	2007.60
MTRL model	**6330.35**

other four DRL models. Based on this result, we concluded that the multi-task learning policy theoretically outperforms single-task DRL models and decided to deploy this policy to show effectiveness in real environments.

8.3 Experiment 2: Deep Thought Spring 2023

8.3.1 Experiment Setup

This study was conducted under a Deep Thought tutor. The subjects are undergraduate students enrolled in the course in Spring 2023. The study was in progress from February 10th to February 20th. It is the same as the first study in that students could access the tutor whenever they wanted in the experiment period. We divided the student group into the control($Expert$) and experimental($MTRL$) groups. The random but reasonable policy was applied in the control group while students were in the training phase. In the experimental group, our trained MTRL policy was applied to the tutor. 66 students completed their progress(37 students for control group, 29 students for experimental group). A one-way ANOVA test showed that there is no significant difference on average of pretest score between the two groups($F(1,64) = 0.003, p = 0.954$); with $Expert(\mu = 0.500, \sigma = 0.301)$ and $MTRL(\mu = 0.495, \sigma = 0.273)$. This implies that the competence of students in the two groups is statistically similar.

8.3.2 Experiment Results

In Table 8.2, we summarized the result of our Spring 23 study. The table shows the result of the average and standard deviation for 3 columns. Here, we also analyzed the squared Normalized Learning Gains(NLG)($posttest - pretest/\sqrt{1 - pretest}$). The squared root reduces the variance and the difference between different incoming competence groups.

Table 8.2: F23 Deep Thought experiment result.

	Pre-test	**Post-test**	**NLG**	**Training Time(min)**
Expert	**0.500(0.301)**	**0.667(0.206)**	**0.100(0.557)**	79.86(34.07)
MTRL	0.495(0.273)	0.612(0.167)	0.009(0.579)	**72.62(29.01)**

Following the table, the *Expert* policy numerically outperforms our *MTRL* policy on both the post-test score and NLG. We conducted ANCOVA and ANOVA tests to verify the result statistically. For the post-test score, an ANCOVA test with pre-test scores as covariates showed that the difference in post-test mean scores between the *Expert* and *DRL* policies was not statistically significant ($F(1,64) = 1.381, p = 0.244$). Also, for the NLG values, a one-way ANOVA test showed that there was no significant difference between the *Expert* and *MTRL* policies in NLG score($F(1,64) = 0.425, p = 0.517$). The last column in Table 8.2 shows the average value of total training time (in minutes) students spent on the training phase for each condition. A one-way ANOVA test using the condition as a factor showed there was no significant difference in the average of training time($F(1,64) = 0.833, p = 0.365$).

8.4 Conclusion & Discussion

In this part, we explored the impact of Multi-Task Deep Reinforcement Learning across Intelligent Tutoring Systems. Our purpose is to achieve generalization ability and improved

performance by utilizing two ITS environments. We used one of the state-of-the-art model-based DRL frameworks named Dreamer, which estimates optimal policy in latent space. Also, we combined our model with bisimulation metrics to achieve generalization ability. However, our result showed that our induced policy is not capable of improving student performance compared with random policy, although the result is not statistically meaningful.

We summarized the problems with our policy induction process into two main categories. : 1)inconsistency of our dataset and 2)model training instability. First, the dataset of our ITS used in the study consists of several years of student trajectories. For years, the structure and functionality have been changing from the current ITS. Using such an inconsistent dataset could have been a hindrance to model training. Second, there would be instability in the model

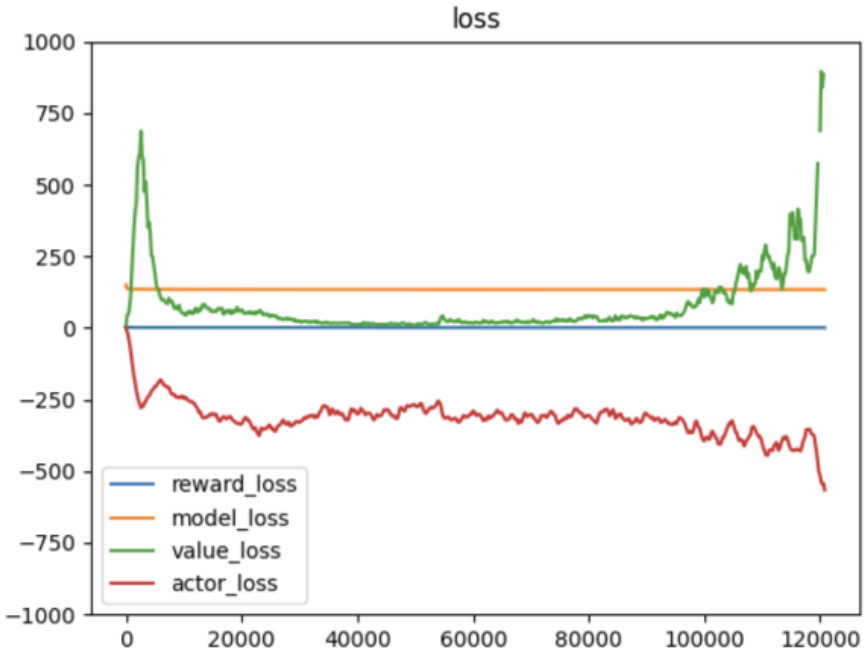

Figure 8.3: Loss function growth for Multi-Task Reinforcement Learning model.

during model training. To suppress the problem, we adjusted the model configuration to achieve convergence of the loss function. But still, when we applied more training steps to achieve better

model performance, the loss function diverged. This is believed to have affected the value of PDIS. Due to the internal characteristics of Dreamer, which is a stochastic model, the PDIS value varies from each evaluation, and the inter-evaluation variance was high. We believed that this instability could be a major reason why the learning gain of students was lower than expected when we applied our induced policy to the real world. Due to the above issues, we concluded that further data improvement and optimization efforts are necessary to achieve generalization ability in different environments. However, we believe that the above research has set another direction for achieving effective multi-task reinforcement learning in the future.

CHAPTER

9

CONCLUSIONS AND FUTURE WORK

In this work, we applied Batch Deep Reinforcement Learning to two real-world ITSs that can provide adaptive training and education to many learners. Our purpose was to explore the effectiveness of DRL for providing learning gains for learners and find usable lessons about the effective way to apply DRL in ITSs. For this purpose, we studied two sub-parts.

In the first part, we applied our DRL-induced policy to the probability tutor in order to verify the improvement of learners' learning gain by determining efficient problem types for each state. We induced a pedagogical policy based on the Double Deep Q Network, which can compensate for the overestimation of the Deep Q network. In addition, we incorporated Long Short Term Memory into the structure of the neural network, which can effectively handle time series data, including long time distance. Inspired by prior work, We compared the state input two settings

on the students' trajectories data of 10 problems in the training phase and deployed the policy with the higher PDIS values for each problem. Our empirical results show that the *DRL*-induced policy provided more learning gain to the learners than the *Expert* policy. Following the result, we confirmed that a DRL-based policy could be a very efficient way to improve learners' abilities. In our future work, technically, we need to improve the interpretability of our policy, which is a weakness of the deep neural network model itself. And from a pedagogical point of view, we need to add other parts that can communicate with the learner, such as a simple explanation of the problem or feedback, not only the current problem type, to boost the learning gain of learners.

In the second part, we decided to induce a policy that applies Multi-Task Learning to the state-of-the-art model-based DRL. This is because MTL is considered an important learning paradigm in RL in that each model can learn two similar tasks simultaneously to achieve better performance and data efficiency. Our Multi-Task DRL model is based on Dreamer, which estimates the value function and policy in the latent state space. Also, in terms of MTL, to learn the shared latent representation of each ITS domain, we combined bisimulation metrics that can measure the behavioral similarity of states. In the model validation process, we confirmed that our Multi-Task DRL policy shows higher PDIS values than other DQN-based policies. However, our empirical results show that the Multi-Task DRL policy does not provide more learning gain than the Expert policy in Deep thought. The main reason for this result is that our model cannot achieve stability. The loss function in the model learning process diverges when additional training is performed for better performance. This instability is considered to be why the model did not produce as much effect as expected when applied to the actual ITS.

In summary, we believe that the two studies show that Batch DRL can be a very promising method for providing adaptive learning to a large number of learners. However, it is necessary to check the performance of DRL on ITS in other domains, and we are confident that the above advantages will only become evident when efforts are made to optimize for an effective model.

REFERENCES

[1] J. R. Anderson, C. F. Boyle, and B. J. Reiser. Intelligent tutoring systems. *Science*, 228(4698):456–462, 1985.

[2] M. S. Ausin. Leveraging deep reinforcement learning for pedagogical policy induction in an intelligent tutoring system. In *In: Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019),*, 2019.

[3] M. S. Ausin. *A Transfer Learning Framework for Human-Centric Deep Reinforcement Learning with Reward Engineering*. North Carolina State University, 2021.

[4] M. S. Ausin, M. Maniktala, T. Barnes, and M. Chi. Tackling the credit assignment problem in reinforcement learning-induced pedagogical policies with neural networks. In *Artificial Intelligence in Education: 22nd International Conference, AIED 2021, Utrecht, The Netherlands, June 14–18, 2021, Proceedings, Part I*, pages 356–368. Springer, 2021.

[5] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[6] D. Borsa, T. Graepel, and J. Shawe-Taylor. Learning shared representations in multi-task reinforcement learning. *arXiv preprint arXiv:1603.02041*, 2016.

[7] D. Calandriello, A. Lazaric, and M. Restelli. Sparse multi-task reinforcement learning. *Advances in neural information processing systems*, 27, 2014.

[8] R. Caruana. *Multitask learning*. Springer, 1998.

[9] P. Castro and D. Precup. Using bisimulation for policy transfer in mdps. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, pages 1065–1070, 2010.

[10] R. Chaudhry, H. Singh, P. Dogga, and S. K. Saini. Modeling hint-taking behavior and knowledge state of students with multi-task learning. *International Educational Data Mining Society*, 2018.

[11] M. Chi and K. VanLehn. The impact of explicit strategy instruction on problem-solving behaviors across intelligent tutoring systems. In *Proceedings of the annual meeting of the cognitive science society*, volume 29, 2007.

[12] M. Chi and K. VanLehn. Meta-cognitive strategy instruction in intelligent tutoring systems: how, when, and why. *Journal of Educational Technology & Society*, 13(1):25–39, 2010.

[13] M. Chi, K. VanLehn, D. Litman, and P. Jordan. Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. *User Modeling and User-Adapted Interaction*, 21:137–180, 2011.

[14] A. Condor and Z. Pardos. A deep reinforcement learning approach to automatic formative feedback. 2022.

[15] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, J. Peters, et al. Sharing knowledge in multi-task deep reinforcement learning. In *8th International Conference on Learning Representations,{ICLR} 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, pages 1–11. OpenReview. net, 2020.

[16] T. Evgeniou and M. Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, 2004.

[17] G. Fenza, F. Orciuoli, and D. G. Sampson. Building adaptive tutoring model using artificial neural networks and reinforcement learning. In *2017 IEEE 17th international conference on advanced learning technologies (ICALT)*, pages 460–462. IEEE, 2017.

[18] N. Ferns, P. Panangaden, and D. Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011.

[19] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[20] M. Geden, A. Emerson, J. Rowe, R. Azevedo, and J. Lester. Predictive student modeling in educational games with multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 654–661, 2020.

[21] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pages 2170–2179. PMLR, 2019.

[22] K. Georgila, M. G. Core, B. D. Nye, S. Karumbaiah, D. Auerbach, and M. Ram. Using reinforcement learning to optimize the policies of an intelligent tutoring system for interpersonal skills training. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 737–745, 2019.

[23] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003.

[24] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[25] D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[27] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[28] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[29] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

[30] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[31] A. Iglesias, P. Martínez, R. Aler, and F. Fernández. Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. *Applied Intelligence*, 31:89–106, 2009.

[32] S. Ju, Y. J. Kim, M. S. Ausin, M. E. Mayorga, and M. Chi. To reduce healthcare workload: Identify critical sepsis progression moments through deep reinforcement learning. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1640–1646. IEEE, 2021.

[33] S. Ju, S. Shen, H. Azizsoltani, T. Barnes, and M. Chi. Importance sampling to identify empirically valid policies and their critical decisions. In *EDM (Workshops)*, pages 69–78, 2019.

[34] S. Ju, G. Zhou, T. Barnes, and M. Chi. Pick the moment: Identifying critical pedagogical decisions using long-short term rewards. *International Educational Data Mining Society*, 2020.

[35] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[36] Y. J. Kim and M. Chi. Time-aware q-networks: Resolving temporal irregularity for deep reinforcement learning. *arXiv preprint arXiv:2105.02580*, 2021.

[37] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[38] S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[39] A. Lazaric and M. Ghavamzadeh. Bayesian multi-task reinforcement learning. In *ICML-27th international conference on machine learning*, pages 599–606. Omnipress, 2010.

[40] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[41] H. Li, X. Liao, and L. Carin. Multi-task reinforcement learning in partially observable stochastic environments. *Journal of Machine Learning Research*, 10(5), 2009.

[42] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[43] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[44] L. T. Liu, U. Dogan, and K. Hofmann. Decoding multitask dqn in the world of minecraft. In *The 13th European Workshop on Reinforcement Learning (EWRL)*, volume 2016, 2016.

[45] P. Liu, X. Qiu, and X. Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.

[46] Z. Liu, B. Mostafavi, and T. Barnes. Combining worked examples and problem solving in a data-driven logic tutor. In *Intelligent Tutoring Systems: 13th International Conference, ITS 2016, Zagreb, Croatia, June 7-10, 2016. Proceedings 13*, pages 347–353. Springer, 2016.

[47] F.-M. Luo, T. Xu, H. Lai, X.-H. Chen, W. Zhang, and Y. Yu. A survey on model-based reinforcement learning. *arXiv preprint arXiv:2206.09328*, 2022.

[48] T. Mandel, Y.-E. Liu, S. Levine, E. Brunskill, and Z. Popovic. Offline policy evaluation across representations with applications to educational games. In *AAMAS*, volume 1077, 2014.

[49] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

[50] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[51] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[52] B. Mostafavi and T. Barnes. Evolution of an intelligent deductive logic tutor using data-driven elements. *International Journal of Artificial Intelligence in Education*, 27:5–36, 2017.

[53] E. Parisotto, J. L. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[54] D. Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[55] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[56] S. Shen, M. S. Ausin, B. Mostafavi, and M. Chi. Improving learning & reducing time: A constrained action-based reinforcement learning approach. In *Proceedings of the 26th conference on user modeling, adaptation and personalization*, pages 43–51, 2018.

[57] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[58] S. Sodhani, A. Zhang, and J. Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pages 9767–9779. PMLR, 2021.

[59] N. Srivastava, W. Talbott, M. B. Lopez, S. Zhai, and J. Susskind. Robust robotic control from pixels using contrastive recurrent state-space models. *arXiv preprint arXiv:2112.01163*, 2021.

[60] J. Stamper, M. Eagle, T. Barnes, and M. Croy. Experimental evaluation of automatic hint generation for a logic tutor. *International Journal of Artificial Intelligence in Education*, 22(1-2):3–17, 2013.

[61] F. Tanaka and M. Yamamura. Multitask reinforcement learning on the distribution of mdps. In *Proceedings 2003 IEEE international symposium on computational intelligence in robotics and automation. Computational intelligence in robotics and automation for the new millennium (Cat. No. 03EX694)*, volume 3, pages 1108–1113. IEEE, 2003.

[62] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[63] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[64] N. Vithayathil Varghese and Q. H. Mahmoud. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9):1363, 2020.

[65] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[66] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[67] A. Wilson, A. Fern, S. Ray, and P. Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022, 2007.

[68] R. Yang, H. Xu, Y. Wu, and X. Wang. Multi-task reinforcement learning with soft modularization. *Advances in Neural Information Processing Systems*, 33:4767–4777, 2020.

[69] T. Yu, A. Kumar, Y. Chebotar, K. Hausman, S. Levine, and C. Finn. Conservative data sharing for multi-task offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34:11501–11516, 2021.

[70] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[71] Y. Zhang and Q. Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021.

[72] G. Zhou, H. Azizsoltani, M. S. Ausin, T. Barnes, and M. Chi. Hierarchical reinforcement learning for pedagogical policy induction. In *Artificial Intelligence in Education: 20th International Conference, AIED 2019, Chicago, IL, USA, June 25-29, 2019, Proceedings, Part I 20*, pages 544–556. Springer, 2019.

APPENDIX

APPENDIX

A

RSSM ARCHITECTURE

Figure C.1 shows the inner structure of RSSM. The steps are divided into two main steps. At first, generate prior latent state z_{prior} using previous latent state z_{t-1} and previous action a_{t-1}, where z_{t-1} is a multi-dimensional vector of concatenating deterministic state h_{t-1} and stochastic state s_{t-1}, In prior latent state, h_t is a state passing through the fully connected layer and GRU layer. On the other hand, s_{prior} is sampled within the Normal distribution $N(\mu_p, \sigma_p)$ where μ_p and σ_p are generated by ***Prior*** layer.

Next, generate latent posterior state z_t with raw observation o_t. With embedded raw observation o_t, post stochastic state s_t is also sampled from $N(\mu, \sigma)$ where μ and σ are generated by ***Post*** layer. Finally, the latent state z_t comes by concatenating h_t and s_t. This latent state is applied to estimate the reward, value, and action model in our model architecture. The progress

above is expressed as follows[59].

$$h_t = GRU(h_{t-1}, TRANS(s_{t-1}, a_{t-1})) \tag{A.1}$$

$$\mu_p, \sigma_p = \textbf{\textit{Prior}}(h_t) \tag{A.2}$$

$$s_{prior} = \mu_p + \sigma_p * N(0,1) \tag{A.3}$$

$$\mu, \sigma = \textbf{\textit{Post}}(h_t, Embed(o_t)) \tag{A.4}$$

$$s_t = \mu + \sigma * N(0,1) \tag{A.5}$$

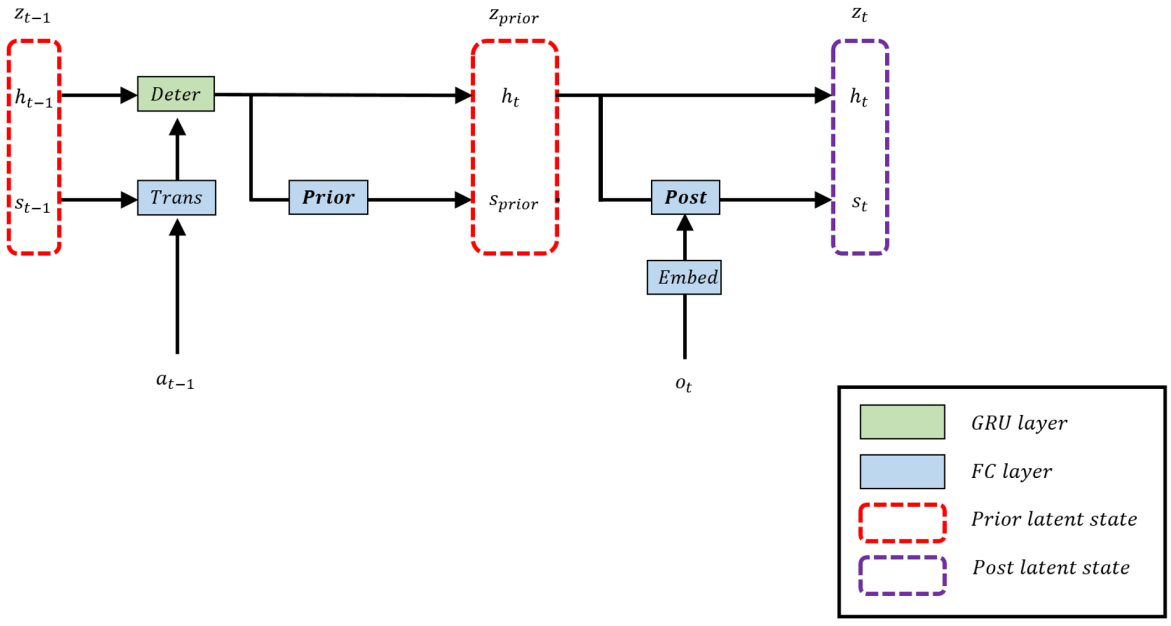

Figure A.1: Inner architecture of RSSM