Leveraging Deep Reinforcement Learning for Pedagogical Policy Induction in an Intelligent Tutoring System

Markel Sanz Ausin, Hamoon Azizsoltani, Tiffany Barnes and Min Chi
North Carolina State University
Raleigh, NC, 27695
{msanzau,hazizso,tmbarnes,mchi}@ncsu.edu

ABSTRACT

Deep Reinforcement Learning (DRL) has been shown to be a very powerful technique in recent years on a wide range of applications. Much of the prior DRL work took the *online* learning approach. However, given the challenges of building accurate simulations for modeling student learning, we investigated applying DRL to induce a pedagogical policy through an *offline* approach. In this work, we explored the effectiveness of offline DRL for pedagogical policy induction in an Intelligent Tutoring System. Generally speaking, when applying offline DRL, we face two major challenges: one is limited training data and the other is the credit assignment problem caused by delayed rewards. In this work, we used Gaussian Processes to solve the credit assignment problem by estimating the inferred immediate rewards from the final delayed rewards. We then applied the DQN and Double-DQN algorithms to induce adaptive pedagogical strategies tailored to individual students. Our empirical results show that without solving the credit assignment problem, the DQN policy, although better than Double-DQN, was no better than a random policy. However, when combining DQN with the inferred rewards, our best DQN policy can outperform the random yet reasonable policy, especially for students with high pre-test scores.

1. INTRODUCTION

Interactive e-Learning Environments such as Intelligent Tutoring Systems (ITSs) and educational games have become increasingly prevalent in educational settings. In order to design effective interactive learning environments, developers must form the basic core of the system and determine *what* is to be taught and *how*. *Pedagogical strategies* are policies that are used to decide the *how* part, what action to take next in the face of alternatives. Each of these systems' decisions will affect the user's subsequent actions and performance.

Reinforcement Learning (RL) is one of the best machine learning approaches for decision making in interactive environments and RL algorithms are designed to induce effective policies that determine the best action for an agent to take in any given situation to maximize some predefined cumulative reward. In recent years, deep neural networks have enabled significant progress in RL research. For example, Deep Q-Networks (DQNs) [26] have successfully learned to play Atari games at or exceeding human level performance by combining deep convolutional neural networks and Q-learning. Since then, DRL has achieved notable successes in a variety of complex tasks such as robotics control [1] and the game of Go [44]. From DQN, various DRL methods such as Double DQN [51] or Actor-Critic methods [38, 39] were proposed and shown to be more effective than the classic DQN. Despite DRL's great success, there are still many challenges preventing DRL from being applied more broadly in practice, including applying it to educational systems. One major problem is sample inefficiency of current DRL algorithms. For example, it takes DQN hundreds of millions of interactions with the environment to learn a good policy and generalize to unseen states, while we seek to learn policies from datasets with fewer than 800 student-tutor interaction logs.

Generally speaking, there are two major categories of RL: online and offline. Online RL algorithms learn policy while the agent interacts with the environment; offline RL algorithms, by contrast, learn the policy from pre-collected training data. Online RL methods are generally appropriate for domains where the state representation is clear and interacting with simulations and actual environments is relatively computationally cheap and feasible, so most of prior work on DRL mainly took an online learning approach. On the other hand, for domains such as e-learning, building accurate simulations or simulating students is especially challenging because human learning is a rather complex, not fully understood process; moreover, learning policies while interacting with students may not be feasible and more importantly, may not be ethical. Therefore, our DRL approach is offline. This approach was achieved by, first, collecting a training corpus. One common convention, and the one used in our study, is to collect an *exploratory* corpus by training a group of students on an ITS that makes random yet *reasonable* decisions and then apply RL to induce pedagogical policies from that exploratory training corpus. An empirical study was then conducted from a new group of human subjects interacting with different versions of the system. The only difference among the versions was the policy employed by the ITS. Lastly, the students' performance was statistically

Markel Sanz Ausin, Hamoon Azizsoltani, Tiffany Barnes and Min Chi "Leveraging Deep Reinforcement Learning for Pedagogical Policy Induction in an Intelligent Tutoring System" In: *Proceedings of The 12th International Conference on Educational Data Mining (EDM 2019)*, Collin F. Lynch, Agathe Merceron, Michel Desmarais, & Roger Nkambou (eds.) 2019, pp. 168 - 177

compared. Due to cost limitations, typically, only the *best* RL-induced policy was deployed and compared against some baseline policies.

When applying offline DRL to ITSs, we often face one major challenge: our rewards are often not only noisy but also delayed. Given the nature of ITS data collection, the training data including our reward functions is often noisy and our rewards are only the incomplete or imperfect observations of underlying true reward mechanisms. Due to the complex nature of student learning, the most appropriate rewards are (delayed) student learning gains, which are only available after the entire training is complete. For example, hints might improve immediate performance but negatively impact overall learning. On the other hand, when the size of the training data is limited, the availability of "true" immediate rewards is very important for offline RL. Immediate rewards are generally more effective than delayed rewards for offline RL because it is easier to assign appropriate credit or blame when the feedback is tied to a single decision. The more we delay rewards or punishments, the harder it becomes to assign credit or blame properly. Therefore, the challenge is how to distribute the delayed rewards to observable, immediate rewards along each student-system interactive trajectory while taking the noise and uncertainty in the data into account. To tackle this issue, we applied a Gaussian Processes based (GP-based) approach to infer "immediate rewards" from the delayed rewards and then applied DQN to induce two policies: one based on delayed rewards and the other based on the inferred immediate rewards, referred to as DQN-Del and DQN-Inf respectively.

In this work, we used a logic ITS and focused on applying DRL to induce a policy on one type of tutorial decision: whether to present a given problem as a *problem solving* (PS) or a *worked example* (WE). The tutor presents a worked example (WE) by demonstrating the individual steps in an expert solution to a problem. During PS, students are required to complete the problem with tutor support (e.g. hints). The effectiveness of DQN-Del and DQN-Inf are evaluated theoretically using Expected Cumulative Reward (ECR) and empirically through two randomly controlled experiments: one for evaluating the effectiveness of DQN-Del in Spring 2018 and the other for evaluating DQN-Inf in Fall 2018. In each experiment, the effectiveness of the corresponding RL-induced policy was compared against the *Random* policy that flips a coin to decide between WE/PS and the students were randomly assigned into the two conditions while balancing their incoming competence. Overall, the results from both experiments showed no significant difference between the DQN-Del and Random in Spring 2018 and between the DQN-Inf and Random in Fall 2018 on every measure of learning performance.

There are two potential explanations for such findings. First, our random baseline policy is decently strong. While random policies are usually bad in many RL tasks, in the context of WE vs. PS, our random policies can be strong baselines. Indeed, some learning literature suggests that the best instructional intervention is to alternate WE and PS [35, 41, 36]. Second, there may be an **aptitude-treatment interaction (ATI)** effect [6, 47], where certain students are less sensitive to the induced policies, meaning they achieve a similar learning performance regardless of policies employed; whereas other students are more sensitive, meaning their learning is highly dependent on the effectiveness of the policies. Thus, we divided the students into High vs. Low based on their incoming competence and investigated the ATI effect. While no ATI effect was found between DQN-Del and Random for Spring 2018, a significant ATI effect was found between DQN-Inf and Random in Fall 2018.

In short, we explored applying offline DRL for pedagogical policy induction based on delayed and inferred immediate rewards. Our results showed that no ATI effect was found between DQN-Del and Random in Spring 2018, whereas there was an ATI effect between DQN-Inf and Random in Fall 2018. More specifically, the High incoming competence group benefited significantly more from the DQN-Inf policy than their peers in the Random condition. This result suggests that the availability of inferred immediate rewards was crucial for effectively applying offline DRL for pedagogical policy induction.

2. BACKGROUND

A great deal of research has investigated the differing impacts of worked examples (WE) and problem solving (PS) on student learning [49, 22, 21, 23, 41, 27, 36]. McLaren and colleagues compared WE-PS pairs with PS-only [22]. Every student was given a total of 10 training problems. Students in the PS-only condition were required to solve every problem while students in the WE-PS condition were given 5 example-problem pairs. Each pair consisted of an initial worked example problem followed by tutored problem solving. They found no significant difference in learning performance between the two conditions. However, the WE-PS group spent significantly less time than the PS group.

McLaren and his colleagues found similar results in two subsequent studies [21, 23]. In the former, the authors compared three conditions: WE, PS and WE-PS pairs, in the domain of high school chemistry. All students were given 10 identical problems. As before, the authors found no significant differences among the three groups in terms of learning gains but the WE group spent significantly less time than the other two conditions; and no significant time on task difference was found between the PS and WE-PS conditions.

In a follow-up study, conducted in the domain of high school stoichiometry, McLaren and colleagues compared four conditions: WE, tutored PS, untutored PS, and Erroneous Examples (EE) [23]. Students in the EE condition were given *incorrect* worked examples containing between 1 and 4 errors and were tasked with correcting them. The authors found no significant differences among the conditions in terms of learning gains, and as before the WE students spent significantly less time than the other groups. More specifically, for time on task, they found that: $WE < EE < untutored\ PS < tutored\ PS$. In fact, the WE students spent only 30% of the total time that the tutored PS students spent.

The advantages of WEs were also demonstrated in another study in the domain of electrical circuits [50]. The authors of that study compared four conditions: WE, WE-PS pairs, PS-WE pairs (problem-solving followed by an example problem), and PS only. They found that the WE and WE-PS

students significantly outperformed the other two groups, and no significant differences were found among four conditions in terms of time on task.

In short, prior research has shown that WE can be similar or more effective than PS or alternating PS with WE, and the former can take significantly less time than the latter two [49, 22, 21, 23, 41]. However, there is no widespread consensus on how or when WE vs. PS should be used. This is why we will derive pedagogical strategies for them directly from empirical data.

2.1 ATI Effect

Previous work shows that the ATI effect commonly exists in many real-world studies. More formally, the ATI effect states that instructional treatments are more or less effective to individual learners depending on their abilities [6]. For example, Kalyuga et al. [17] empirically evaluated the effectiveness of worked example (WE) vs. problem solving (PS) on student learning in programmable logic. Their results show that WE is more effective for inexperienced students while PS is more effective for experienced learners.

Moreover, D'Mello et al. [7] compared two versions of ITSs: one is an affect-sensitive tutor which selects the next problem based on students' affective and cognitive states combined, while the other is an original tutor which selects the next problem based on students' cognitive states alone. An empirical study shows that there is no significant difference between the two tutors for students with high prior knowledge. However, there is a significant difference for students with low prior knowledge: those who trained on the affect-sensitive tutor had significantly higher learning gain than their peers using the original tutor.

Chi and VanLehn [4] investigated the ATI effect in the domain of probability and physics, and their results showed that high competence students can learn regardless of instructional interventions, while for students with low competence, those who follow the effective instructional interventions learned significantly more than those who did not. Shen and Chi [43] find that for pedagogical decisions on WE vs. PS, certain learners are always less sensitive in that their learning is not affected, while others are more sensitive to variations in different policies. In their study, they divided students into Fast and Slow groups based on time, and found that the Slow groups are more sensitive to the pedagogical decisions while the Fast groups are less sensitive.

3. RELATED WORK

Deep Reinforcement Learning: In recent years, many DRL algorithms have been developed for various applications such as board games like Go [44, 46], Chess and Shogi [45], robotic hand dexterity [33, 1], physics simulators [19, 29, 30], and so forth. While most DRL algorithms have been mainly applied online, some of them can also be applied offline. More specifically, DRL algorithms such Vanilla Policy Gradient (VPG) [48], Proximal Policy Optimization (PPO) [39], Trust Region Policy Optimization (TRPO) [38], or A3C [24] can only be applied for online learning by interacting with simulations. Some other DRL algorithms can be applied for offline learning using pre-collected training data. These include the Q-learning based approaches such as Deep Q-Network (DQN) [26], Double-DQN [51], prioritized experience replay [37], distributed prioritized experience replay (Ape-X DQN) [14], and the Actor-Critic based methods such as Deep Deterministic Policy Gradient (DDPG) [19], Twin Delayed Deep Deterministic policy gradient (TD3) [9], or Soft Actor-Critic (SAC) [11]. Among them, DQN and its variants have been much more extensively studied, however, it is still not clear whether they can be successfully applied *offline* for pedagogical policy induction for ITSs.

Reinforcement Learning in Education: Prior research using online RL to induce pedagogical policies has often relied on simulations or simulated students, and the success of RL is often heavily dependent on the accuracy of the simulations. Beck et al. [3] applied temporal difference learning, with off-policy ϵ-greedy exploration, to induce pedagogical policies that would minimize student time on task. Iglesias et al. applied another common online approach named Q-learning to induce policies for efficient learning [15, 16]. More recently, Rafferty et al. applied POMDP with tree search to induce policies for faster learning [32]. Wang et al. applied an online Deep-RL approach to induce a policy for adaptive narrative generation in educational game [52]. All of the models described above were evaluated by comparing the induced policy with some baseline policies via simulations or classroom studies.

Offline RL approaches, on the other hand, "take advantage of previously collected samples, and generally provide robust convergence guarantees" [40]. Shen et al. applied value iteration and least square policy iteration on a pre-collected training corpus to induce pedagogical policies for improving students' learning performance [43, 42]. Chi et al. applied policy iteration to induce a pedagogical policy aimed at improving students' learning gains [5]. Mandel et al. [20] applied an offline POMDP approach to induce a policy which aims to improve student performance in an educational game. In classroom studies, most models above were found to yield certain improved student learning relative to a baseline policy.

DRL in Education is a subject of growing interest. DRL adds deep neural networks to RL frameworks such as POMDP for function approximation or state approximation [25, 26]. This enhancement makes the agent capable of achieving complicated tasks. Wang et al. [52] applied a DRL framework for personalizing interactive narratives in an educational game called CRYSTAL ISLAND. They designed the immediate rewards based on normalized learning gain (NLG) and found that the students with the DRL policy achieved a higher NLG score than those following the linear RL model in *simulation* studies. Furthermore, Narasimhan et al. [28] implemented a Deep Q-Network (DQN) approach in text-based strategy games, constructed based on Evennia, which is an open-source library and toolkit for building multi-users online text-based games. Using simulations, they found that the DRL policy significantly outperformed the random policy in terms of quest completion.

In summary, compared with MDP and POMDP, relatively little research has been done on successfully applying DRL to the field of ITS. None of the prior research has successfully applied DRL to ITSs without simulated environments,

in order to learn an effective pedagogical strategy that makes students learn in a more efficient manner. Furthermore, no prior work has empirically evaluated any DRL-induced policy to confirm its benefits on real students.

4. METHODS

In RL, the agent interacts with an environment $\mathcal{E}$, and the goal of the agent is to learn a policy that will maximize the sum of future discounted rewards (also known as the return) along the trajectories, where each trajectory is one run through the environment, starting in an initial state and ending in a final state. This is done by learning which action to take for each possible state. In our case, $\mathcal{E}$ is the learning context, and the agent must learn to take the actions that lead to the optimal student learning, by maximizing the return $\mathrm{R} = \sum_{t=0}^{T} \gamma^t r_t$, where r_t is the reward at time step t, T is the time step that indicates the end of the trajectory, and $\gamma \in (0, 1]$ is the discount factor.

4.1 DQN and Double-DQN

Deep Q-Network (DQN) is, fundamentally, a version of Q-learning. In Q-learning, the goal is to learn the optimal action-value function, $Q^*(s, a)$, which is defined as the expected reward obtained when taking the optimal action a in state s, and following the optimal policy π^* until the end of the trajectory. For any state-action pair, the optimal action-value function must follow the Bellman optimality equation in that:

$$Q^*(s, a) = r + \gamma \max_{a'} Q^*(s', a') \tag{1}$$

Here r is the expected immediate reward for taking action a at state s; γ is the discount factor; and $Q^*(s', a')$ is the optimal action-value function for taking action a' at the subsequent state s' and following policy π^* thereafter.

Compared with the original Q-leaning, DQNs use neural networks (NNs) to approximate action-value functions. This is because NNs are great universal function approximators and they are able to handle continuous values in both their inputs and outputs. In order to train the DQN algorithm, two neural networks with equal architectures are employed. One is the main network and its weights are denoted $\boldsymbol{\theta}$ and the other is the target network, and its weights are denoted $\boldsymbol{\theta}^-$. The target value used to train the network is $y \coloneqq r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^-)$. Thus, the loss function that is minimized in order to train the main network is:

$$Loss(\boldsymbol{\theta}) = \mathbb{E}[(y - Q(s, a; \boldsymbol{\theta}))^2] \tag{2}$$

The main network is trained on every training iteration, while the target network is frozen for a number of training iterations. Every m training iterations, the weights of the main neural network are copied into the target network. This is one of the techniques used in order to avoid divergence during the training process. Another one of these techniques was the use of an experience replay buffer. This buffer contains the p most recent (s, a, r) tuples, and the algorithm randomly samples from the buffer when creating the batch on each training iteration. We followed the same procedure, but as our training was performed offline, the experience replay buffer consists of all the samples on our training corpus, and it does not get refreshed over time.

Double-DQN or DDQN was proposed by Van Hasselt et al. [12] who combined it with neural networks in the Double-DQN algorithm [51]. The intuition behind it is to decouple the action selection from the action evaluation. To achieve this, the Double-DQN algorithm uses the main neural network to first select the action that has the highest Q-value for the next state ($\mathrm{argmax}_{a'} Q(s', a', \boldsymbol{\theta})$) and then evaluates the Q-value of the selected action using the target network ($Q(s', \mathrm{argmax}_{a'} Q(s', a'; \boldsymbol{\theta}); \boldsymbol{\theta}^-)$). This simple trick has been proven to significantly reduce overestimations in Q-value calculations, resulting in better final policies. With this technique, the target value used to optimize the main network becomes:

$$y \coloneqq r + \gamma Q(s', \operatorname*{argmax}_{a'} Q(s', a', \boldsymbol{\theta}); \boldsymbol{\theta}^-) \tag{3}$$

The loss function is still the same as in equation 2, but the target value y used in the formula is now updated to be the one in equation 3.

4.2 Fully Connected vs. LSTM

For our NN architectures, we explored two options: Fully connected NNs and Long Short Term Memory (LSTM).

Fully Connected or multi-layer perceptrons are the simplest form of neural network units. They calculate a simple weighted sum of all the input units, and each unit produces an output value that is often passed to an activation function. We used these units to parametrize our neural networks. All the input units are connected to all the units in the first hidden layer, and all those units are connected to every unit in the next hidden layer. This process continues until the final output layer.

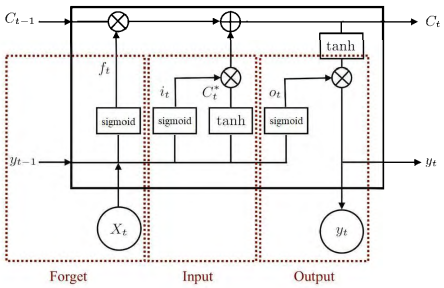

Figure 1: A single LSTM unit containing a forget, input and ouput gate

Long Short-Term Memory (LSTM) is a type of recurrent neural network specifically designed to avoid the vanishing and exploding gradient problems [13]. LSTMs are particularly suitable for tasks where long-term temporal dependencies must be remembered. They achieve this by maintaining the previous information of hidden states as internal memory. Figure 1 shows the architecture of a single LSTM unit. It consists of a memory cell state denoted by C_t and three gates: the forget gate $f_t \in [0, 1]$, the input gate $i_t \in [0, 1]$, and the output gate $o_t \in [0, 1]$. These three gates interact with each other to control the flow of information. During training, the network learns what to memorize and when to allow writing to the cell in order to minimize the

training error. More specifically, the forget gate determines what information from the previous memory cell state is expired and should be removed; the input gate selects information from the *candidate* memory cell state C_t^* to update the cell state; and the output gate filters the information from the memory cell so that the model only considers information relevant to the prediction task. The value of each gate is computed as follows, where $W_{[i,f,C,o]}$ are the weight matrices and $b_{[i,f,C,o]}$ are the bias vectors:

$$\begin{aligned} i_t &= \text{sigmoid}(W_i \cdot [y_{t-1}, X_t] + b_i) \\ f_t &= \text{sigmoid}(W_f \cdot [y_{t-1}, X_t] + b_f) \\ C_t^* &= \tanh(W_C \cdot [y_{t-1}, X_t] + b_c) \\ o_t &= \text{sigmoid}(W_o \cdot [y_{t-1}, X_t] + b_o) \end{aligned} \tag{4}$$

The memory cell value C_t and output value y_t from the LSTM unit are computed using the following formulas:

$$\begin{aligned} C_t &= C_{t-1} \cdot f_t + C_t^* \cdot i_t \\ y_t &= o_t * \tanh(C_t) \end{aligned} \tag{5}$$

4.3 Inferring Immediate Rewards

A historical dataset $\mathcal{H}$ consists of m trajectories, h_1 to h_m and n unknown immediate rewards. We would like to infer the immediate rewards given delayed rewards. In order to infer the immediate rewards, we used a minimum mean square error (MMSE) estimator in the Bayesian setting [18, 8, 10]. Assume $\mathbf{R} = \mathbf{D}\mathbf{r} + \varepsilon$ is a linear process where $\mathbf{D}$ is a known matrix, $\mathbf{r}$ is a $n \times 1$ random vector of unknown immediate rewards, $\mathbf{R}$ is a $m \times 1$ vector of observed delayed rewards and ε is a vector of independent and identically distributed noise with mean of zero and standard deviation of $\sigma_{\mathbf{R}}$. Assuming the discounted sum of the immediate rewards is equal to the delayed rewards, a linear model matrix $\mathbf{D}$ is proposed as:

$$\mathbf{D} = \begin{bmatrix} \overbrace{1 \quad \gamma \quad \gamma^2 \quad \dots}^{h_1} & \overbrace{0 \quad \dots}^{h_2} & \dots \\ 0 \quad \dots \quad 0 & 1 \quad \gamma \quad \gamma^2 \quad \dots \quad 0 & \dots \\ 0 \quad \dots & \dots & \ddots \end{bmatrix} \tag{6}$$

where γ is the discount factor. Following the linear MMSE estimator, we assume that the immediate rewards follow a Gaussian Process defined as $\mathbf{r} \sim \mathcal{N}(\mu_{\mathbf{r}}, \mathbf{C_{rr}})$ where $\mu_{\mathbf{r}}$ is the a priori mean and $\mathbf{C_{rr}}$ is the a priori covariance defined by an appropriate kernel [2]. Using the theorem of conditional distribution of multivariate Gaussian distributions [34], conditional expectation of immediate rewards given delayed rewards $\mathbb{E}[\mathbf{r}|\mathbf{R}]$ or the posterior mean of immediate rewards is:

$$\mathbb{E}[\mathbf{r}|\mathbf{R}] = \mu_{\mathbf{r}} + \mathbf{C_{rr}}\mathbf{D}^T{\mathbf{C_{RR}}}^{-1}(\mathbf{R} - \mathbf{D}\,\mu_{\mathbf{r}}) \tag{7}$$

and the posterior covariance $\mathbf{C}[r|R]$ of inferred immediate rewards given delayed rewards can be calculated as:

$$\mathbb{C}[\mathbf{r}|\mathbf{R}] = \mathbf{C_{rr}} - \mathbf{C_{rr}}\mathbf{D}^T{\mathbf{C_{RR}}}^{-1}\mathbf{D}\mathbf{C}_{\mathbf{rr}}^T \tag{8}$$

where $\mathbf{C_{RR}} = \mathbf{D}\mathbf{C_{rr}}\mathbf{D}^T + \sigma_{\mathbf{R}}^2\mathbf{I}$ and $\mathbf{I}$ is the identity matrix.

Algorithm 1 shows the process used to infer the immediate rewards. Estimation of the mean and covariance of the random column vector $\mathbf{r}$ in Eqs. 7 and 8 requires the inverse of the matrix $\mathbf{C_{RR}}$. By introducing several intermediary variables, this algorithm provides an efficient solution to matrix inversion using the Cholesky decomposition similar to the Gaussian Processes algorithm implementation [34].

Algorithm 1 Immediate reward approximation algorithm.

Inputs: $\mathbf{R}$, $\mu_{\mathbf{r}}$, $\mathbf{C_{rr}}$, $\mathbf{D}$, $\sigma_{\mathbf{R}}^2$
$\mathcal{L} = \text{Cholesky}\left(\mathbf{D}\mathbf{C_{rr}}\mathbf{D}^{\mathrm{T}} + \sigma_{\mathbf{R}}^2\mathbf{I}\right)$
$\boldsymbol{\beta} = \mathcal{L}\backslash(\mathbf{R} - \mathbf{D}\,\mu_{\mathbf{r}})$ forward-substitution algorithm
$\boldsymbol{\alpha} = \mathcal{L}^{\mathrm{T}}\backslash\boldsymbol{\beta}$ back-substitution algorithm
$\overline{\boldsymbol{k}} = \mathbf{D}\mathbf{C}_{\mathbf{rr}}^{\mathrm{T}}$
$\mathbf{v} = \mathcal{L}\backslash\overline{\boldsymbol{k}}$
$\mathbb{E}[\mathbf{r}|\mathbf{R}] = \mu_{\mathbf{r}} + \overline{\boldsymbol{k}}^{\mathrm{T}}\boldsymbol{\alpha}$
$\mathbb{C}[\mathbf{r}|\mathbf{R}] = \mathbf{C_{rr}} - \mathbf{v}^{\mathrm{T}}\mathbf{v}$
return: $\mathbb{E}[\mathbf{r}|\mathbf{R}]$ and $\mathbb{C}[\mathbf{r}|\mathbf{R}]$

5. POLICY INDUCTION

In this section, we will describe our ITS, the training corpus, our policy induction procedure, and theoretical evaluation results.

5.1 Logic ITS

The logic tutor used in this study is named Deep Thought (DT), and it uses a graph-based environment to solve logic proofs. It is used in the undergraduate level Discrete Mathematics class at North Carolina State University. To complete a problem, students iteratively apply rules to logic statement nodes in order to derive the conclusion node. DT automatically checks the correctness of each step and provides immediate feedback on any rule that is applied incorrectly. The tutor consists of 6 levels, with 3 to 4 problems per level. Each problem can be represented as Problem Solving (PS) or as Worked Example (WE). Figure 2 (left) shows the user interface for PS, and Figure 2 (right) shows the interface for WE.

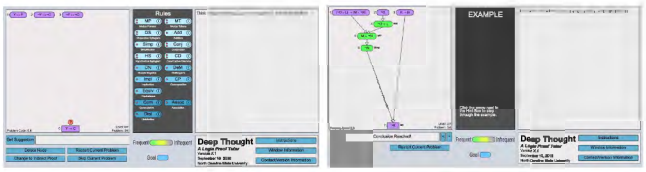

Figure 2: User Interface for DT. **Left**: PS. **Right**: WE.

5.2 Training Corpus

Our training corpus contains 786 complete student trajectories collected over five semesters. On average, each student spent two hours to complete the tutor. For each student, the tutor makes about 19 decisions. From our student-system interaction logs, we extracted a total of 142 state features:

- **Autonomy:** 10 features describing the amount of work done by the student.
- **Temporal:** 29 features, including average time per step, the total time spent on the current level, the time spent on PS, the time spent on WE, and so on.

- **Problem Solving:** 35 features such as the difficulty of the current problem, the number of easy and difficult problems solved on the current level, the number of PS and WE problems seen in the current level, or the number of nodes the student added in order to reach the final solution.
- **Performance:** 57 features such as the number of incorrect steps, and the ratio of correct to incorrect rule applications for different types of rules.
- **Hints:** 11 features such as the total number of hints requested or the number of hints the tutor provided without the student asking for them.

The features contain non-negative continuous values. As their range varies significantly (time can be a large number while problem difficulty is always between 1 and 9), we normalized each feature to the range [0, 1]. Input feature normalization has been shown to improve the stability of the learning process on neural networks, and often leads to faster convergence.

To induce our pedagogical policy, while previous research mainly used learning gains or time on task as reward function, our reward function here is based on the *improvement of learning efficiency*, which balances both learning gain improvement and time on task improvement. In this way, if two students have the same amount of learning gain, the one who takes shorter time would get higher reward. To calculate their learning efficiency, we used students' scores obtained on each level divided by the training time on the level. Students must solve the last problem on each level without help, and we use this as a level score. The range of the score for each level is $[-100, +100]$, and the learning gain for level L is calculated as $Score_L - Score_{L-1}$, thus having a range of $[-200, +200]$.

5.3 Training Process

For both DQN and Double DQN, we explored using Fully Connected (FC) NNs or using LSTM to estimate the action-value function Q. Our FC has four fully connected layers of 128 units each, uses Rectified Linear Unit (ReLU) as the activation function. Our LSTM architecture consists of two layers of 100 LSTM units each, with a fully connected layer at the end. Additionally, for either FC or LSTM, for a given time t, we explored three input settings: to use only the current state observation s_t ($k = 1$), to use the last two state observations: s_{t-1} and s_t ($k = 2$), and to use the last three: s_{t-2}, s_{t-1} and s_t ($k = 3$).

In the case of the fully connected (FC) model, the observations are concatenated and passed to the input layer as a flat array of values. For LSTM, the input state observations are passed to the network in a sequential manner. These past observations provide extra information about the performance of the student in the previous states. However, including previous states also add complexity to the network, which can slow down the learning process and can increase the risk of converging to a weaker final policy. As the number of parameters increases in the NNs, the chance that our NN would get stuck at a local optima increases, especially when our training data is limited. L2 regularization was used to get a model that generalizes better. We trained our models for 50,000 iterations, using a batch size of 200.

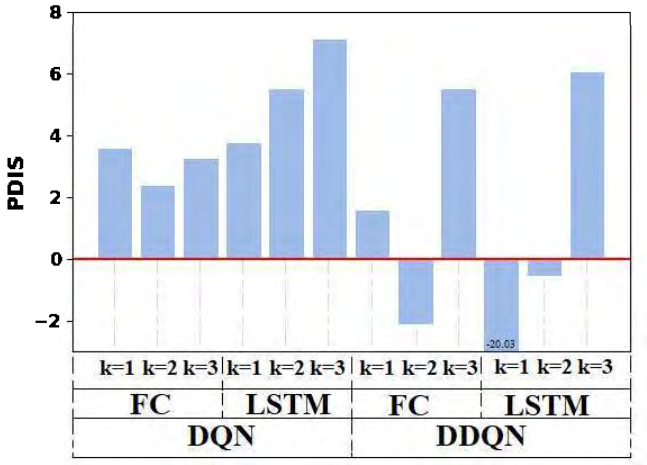

Figure 3: Importance sampling results.

5.4 Induced Policy

First, we induced the DQN-Del policy using delayed rewards only. Our training data was split: 90% of the students for training data and 10% for testing data. We trained all 12 of our models (DQN and Double-DQN with either FC layers or LSTM layers, and with $k = \{1, 2, 3\}$) on the training data and evaluated their performance on testing data. We repeated this process twice with two different test sets and reported their average performance on a series of popular off-policy evaluation metrics. Among them, Expected Cumulative Reward (ECR) is the most widely used. However, Per-Decision Importance Sampling (PDIS) has shown to be more robust [31] .

ECR is simply calculated by averaging over the highest Q-value for all the initial states in the validation set. The formula is described in Equation 9.

$$ECR = \frac{1}{N}\sum_{i=1}^{N}\max_{a} Q(s_i, a) \tag{9}$$

s_i is an initial state, and N denotes the number of trajectories in the validation set.

PDIS [31] is an alternative to regular Importance Sampling, to reduce variance in the estimations. The PDIS results of the 12 models are shown in Figure 3. The PDIS result of the random policy is used to set $y = 0$ (the red line) in Figure 3. Much to our surprise, while double DQN has shown to be much more robust in online DRL applications, its performance is generally worse than DQN here, especially when $k = 1$ and $k = 2$. Figure 3 shows that the best policy is induced using DQN with the LSTM architecture for $k = 3$, and thus is selected as DQN-Del. We also compare the selected policy with the remaining ones using ECR and other evaluation metrics and the results showed using DQN with the LSTM architecture for $k = 3$ is always among the best policies across different evaluation metrics.

To evaluate the impact of Inferred rewards on the DQN in-

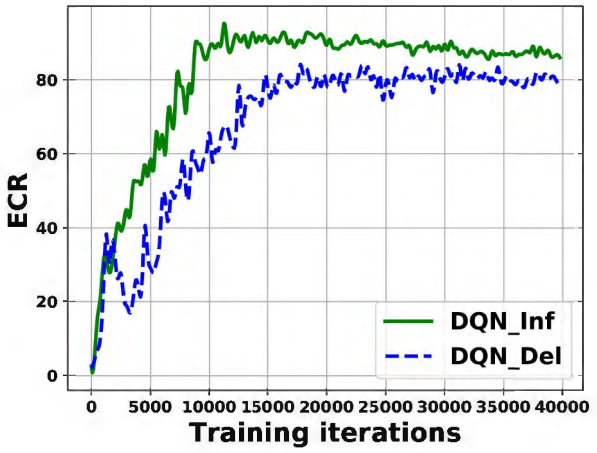

Figure 4: ECR evolution of DQN-Del and DQN-Inf.

duced policies, we used the same approach to induce the DQN-Inf policy and the only major difference is that we used the inferred immediate rewards in the training dataset, calculated through Algorithm 1. During the training process, we calculated the ECRs of DQN with the LSTM architecture for $k = 3$ using the original delayed rewards (DQN-Del) vs. using the inferred immediate rewards (DQN-Inf). The evolution of the ECR values for each policy during the training process is shown in Figure 4, showing that using the inferred rewards we can theoretically converge faster and to a better policy.

6. EMPIRICAL EXPERIMENT SETUP

Two empirical experiments were conducted, one in the Spring 2018 semester and one in the Fall 2018 semester. They were both conducted in the undergraduate Discrete Mathematics class at North Carolina State University.

6.1 Experiment 1: Spring 2018

84 students from the Spring 2018 class were randomly assigned to the *Random* (control) group and the *DQN-Del* group. Because both *WE* and *PS* are considered to be *reasonable* educational interventions in the context of learning, we refer to our control random policy as a *random yet reasonable* policy or Random in the following. The assignment was done in a balanced random manner, using the pre-test score to ensure that the two groups had similar prior knowledge. $N = 45$ and $N = 39$ were assigned to Random and DQN-Del respectively. Among them, $N = 41$ Random students and $N = 33$ DQN-Del students completed the training. A χ^2 test showed no significant differences between the completion rates of the two different groups: $\chi^2(1, N = 84) = 0.053, p = 0.817$.

6.2 Experiment 2: Fall 2018

98 students from the Fall 2018 Discrete Mathematics class were distributed into two conditions. The two conditions are the *Random* (control) group and the *DQN-Inf* group. The group sizes were as follows: $N = 49$ for Random, and $N = 49$ for DQN-Inf. A total of 84 students completed the experiment and their distribution was as follows: $N = 43$ for Random, and $N = 41$ for DQN-Inf. A χ^2 test of independence showed no significant differences between the completion rates of the two different groups: $\chi^2(1, N = 98) = 0.025, p = 0.872$.

6.3 Performance Measure

Our tutor is consisting of 6 strictly ordered levels of proof problems. All of the students received the same set of problems in level 1. Their initial proficiency is calculated based upon the number of mistakes made on the final problem of level 1 and the total training time on level 1. The proficiency reflects how well they understand the knowledge and can apply the logic rules in the proof process before the tutor follows different pedagogical policies. In each sequential level, DT will follow the corresponding policies to determine the next problem to be WE or PS. The last problem on each level is used as a mini-posttest to measure students' performance on that level.

When inducing both the DQN-Del and DQN-Inf, we calculated our reward function based upon the improvement of students' learning efficiency which is defined as level scores divided by the training time on that level. So to measure student performance, we first calculate the learning efficiency on each level as: the score obtained by the student in the last problem of that level, divided by the total time (in minutes). In this study, we use student learning efficiency in level 1 as their pretest efficiency score and their learning efficiency in level2-level6 as the post-test efficiency scores. Since our DQNs used learning efficiency improvement as their rewards, we expect that the DRL-induced policy would cause students to have higher post-test efficiencies.

7. RESULTS

7.1 Experiment 1 Results

No significant difference was found on the pre-test efficiency between the Random and DQN-Del: $t(72) = 1.086, p = 0.281$. We divided the students into high pre-test efficiency ($n = 37$) and low pre-test efficiency ($n = 37$) groups, based upon their learning efficiency on the pre-test. As expected, there was a significant difference between the high and low efficiency students on their pre-test efficiency: $t(72) = 9.570$, $p < 0.001$. The partition mentioned above resulted in four groups, based upon their incoming efficiency and condition: DQN-Del-High ($n = 16$), DQN-Del-Low ($n = 17$), Random-High ($n = 21$), and Random-Low ($n = 20$). A t-test showed no significant difference in the pre-test efficiencies either between the two low groups, Random-Low and DQN-Del-Low, or between the two high efficiency groups. These results show that there is no significant difference in the pre-test efficiency across conditions.

A two-way ANCOVA test on the post-test efficiency, using Condition {*Random, DQN-Del*} and Incoming Competency {*Low, High*} as factors and pre-test efficiency as a covariate, showed that there is no significant main effect of Condition $F(1, 69) = 2.633$, $p = 0.109$, and no significant main effect of Incoming Efficiency $F(1, 69) = 0.036$, $p = 0.849$. No interaction (ATI) effect was found either $F(1, 69) = 1.285$, $p = 0.261$. Thus, we conclude there was no difference between the two conditions in the Spring 2018 study.

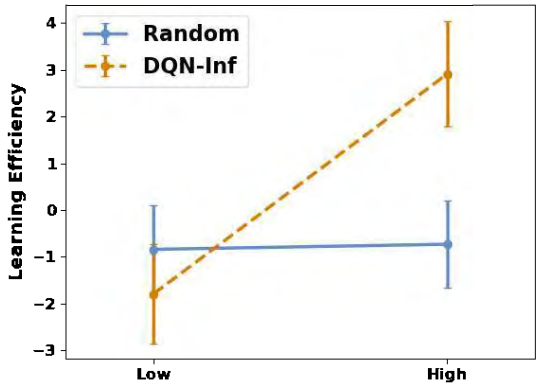

Figure 5: Post-Test Learning Efficiency across different groups for the Fall 2018 study.

7.2 Experiment 2 Results

In fall 2018, again no significant difference was found on the pre-test efficiency between the Random and DQN-Inf groups: $t(82) = -0.333, p = 0.739$. The students were also divided into high pre-test efficiency ($n = 42$) and low pre-test efficiency ($n = 42$) groups. A t-test showed a significant difference between the high and low efficiency students on the pre-test efficiency: $t(82) = 6.38, p < 0.001$. The same four groups were formed, based upon their incoming efficiency and condition: DQN-Inf-High ($n = 20$), DQN-Inf-Low ($n = 21$), Random-High ($n = 22$), and Random-Low ($n = 21$). A t-test showed no significant difference on the pre-test efficiencies when comparing the Random-Low and DQN-Inf-Low groups: $t(40) = 0.027, p = 0.978$. No significant difference was found either, when performing a t-test on the two high efficiency groups: $t(40) = -0.698, p = 0.489$. This shows that there is no significant difference on the pre-test efficiency across conditions during the Fall 2018 study.

A two-way ANOVA test using Condition {*Random, DQN-Inf*} and Incoming Competency {*Low, High*} as two factors showed a significant interaction effect on students' post-test efficiency: $F(1, 80) = 5.038, p = 0.027$ (as shown in Figure 5). To be more strict, we ran a two-way ANCOVA test using Condition and Incoming Competency as two factors and pre-test efficiency as a covariate. This analysis also showed a significant interaction effect on students' post-test efficiency: $F(1, 79) = 4.687, p = 0.033$. Thus, by taking the pre-test efficiency into consideration, there is still a significant interaction effect. No significant main effect was found from either Condition or Incoming Competency. A one-way ANCOVA test on the post-test efficiency for the Low competency groups, using Condition {*Random-Low, DQN-Inf-Low*} as a factor and pre-test competency as a covariate showed no significant difference on the post-test efficiency $F(1, 39) = 0.429$, $p = 0.516$. However, a significant difference was found for the High groups $F(1, 39) = 5.513$, $p = 0.024$, with means -0.719 for Random-High and 2.916 for DQN-Inf-High (as shown in Figure 5).

7.3 Log Analysis

This section will show more details on the different types of tutorial decisions made across the different conditions and studies. The features that were analyzed include the total number of problems each student encountered (TotalCount), the number of problems solved (PSCount), the number of difficult problems solved (diffPSCount), the number of WEs seen (WECount), and the number of difficult WEs seen (diffWECount). Table 1 shows the summary of these five features for each condition and study. Columns 3 and 4 show the mean and standard deviation of each condition for these categories. Column 5 shows the statistical results of different t-tests comparing the two conditions.

No significant difference is found for the total number of problems seen by each group. However, we observed that for the features diffPSCount, WECount and diffWECount, a significant difference was found only during the Spring 2018 study. Looking at the mean values, we notice that the DQN-Del policy assigned fewer WE and more PS problems. However, this did not improve the performance of the students in the DQN-Del group during this study. During the Fall 2018 study, we only observe a significant difference in the number of PS problems assigned. No significant difference was found in the remaining categories.

When we analyze the logs for the High competency students, table 2 shows the values of those same features, but only for the High competency students in each study. During the Spring 2018 semester, we find a statistically significant difference for TotalCount, PSCount, and diffWECount, and we find a marginal difference for WECount. This shows that the DQN-Del policy gave more PS problems, fewer WE, and fewer difficult WE problems, but no significant difference was found in students' post-test performance. The Fall 2018 study results show no significant or marginal difference in any of the five categories. Despite this fact, the DQN-Inf policy implemented in the Fall 2018 study outperformed the Random policy for the High competency students. We can also observe how, in Table 2, the standard deviation for the DQN groups is often larger than the standard deviation for the Random groups. This makes sense because we expect all the students in the Random group to have a similar values in each category. However, it looks like the DQN policy is assigning more PS to certain students, and more WE to other students, resulting in a larger standard deviation.

In short, our log analysis results show that it is not about the total amount of PSs and WEs that students received that matters, but rather how or when they receive which.

8. CONCLUSIONS

We used offline Deep Reinforcement Learning algorithms in conjunction with inferred immediate rewards to induce a pedagogical policy to improve the students' learning efficiency for a logic tutor. Our results showed that our DRL-induced pedagogical policy can outperform the Random policy, which is a strong baseline here. More specifically, there was an ATI effect in the Fall 2018 study in that the high incoming competency students were benefited more from our DRL-induced policy, by achieving better post-test learning efficiency than other groups. Our results showed that our proposed Gaussian Processes based approach to infer "im-

Table 1: Log analysis results for per semester and condition.

Feature	Semester	Random	DQN	Significance	
TotalCount	Spring	22.68(5.05)	24.02(5.29)	$t(72) = -1.118$,	$p = 0.267$
	Fall	23.81(3.32)	25.26(5.37)	$t(82) = -1.489$,	$p = 0.141$
PSCount	Spring	14.82(5.29)	17.08(6.11)	$t(72) = -1.691$,	$p = 0.095$•
	Fall	**14.38(2.30)**	**15.73(3.68)**	$\mathbf{t(82) = -2.029}$,	$\mathbf{p = 0.046}$*
diffPSCount	Spring	5.19(1.74)	4.85(2.06)	$t(72) = 0.765$,	$p = 0.446$
	Fall	7.54(1.57)	8.19(2.31)	$t(82) = -1.501$,	$p = 0.137$
WECount	**Spring**	**7.85(1.17)**	**6.94(1.87)**	$\mathbf{t(72) = 2.466}$,	$\mathbf{p = 0.016}$*
	Fall	9.43(1.57)	9.52(2.37)	$t(82) = -0.210$,	$p = 0.833$
diffWECount	**Spring**	**3.85(1.33)**	**2.61(1.87)**	$\mathbf{t(72) = 3.226}$,	$\mathbf{p = 0.002}$*
	Fall	2.15(1.42)	2.02(1.23)	$t(82) = 0.469$,	$p = 0.639$

Table 2: Log analysis results for the high competency groups per semester.

Feature	Semester	Random	DQN	Significance	
TotalCount	**Spring**	**21.52(2.18)**	**24.31(4.07)**	$\mathbf{t(35) = -2.471}$,	$\mathbf{p = 0.021}$*
	Fall	24.27(0.76)	25.95(7.58)	$t(40) = -0.984$,	$p = 0.337$
PSCount	**Spring**	**13.61(2.49)**	**17.50(4.67)**	$\mathbf{t(35) = -3.008}$,	$\mathbf{p = 0.006}$*
	Fall	14.59(0.66)	15.95(4.98)	$t(40) = -1.208$,	$p = 0.241$
diffPSCount	Spring	5.57(1.43)	5.12(2.30)	$t(35) = 0.680$,	$p = 0.502$
	Fall	7.68(0.83)	8.75(2.93)	$t(40) = -1.570$,	$p = 0.130$
WECount	Spring	7.90(1.33)	6.81(1.86)	$t(35) = 1.981$,	$p = 0.058$•
	Fall	9.68(0.94)	10.00(3.19)	$t(40) = -0.428$,	$p = 0.672$
diffWECount	**Spring**	**4.23(1.41)**	**2.43(1.82)**	$\mathbf{t(35) = 3.271}$,	$\mathbf{p = 0.002}$*
	Fall	2.18(1.46)	2.50(1.27)	$t(40) = -0.750$,	$p = 0.457$

mediate rewards" from the delayed rewards seems reasonable and works pretty well here. Thus, offline DRL can be successfully applied to real-life environments even with a limited training dataset with delayed rewards.

Acknowledgements
This research was supported by the NSF Grants #1432156, #1651909, and #1726550.

9. REFERENCES

[1] M. Andrychowicz, B. Baker, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[2] H. Azizsoltani and E. Sadeghi. Adaptive sequential strategy for risk estimation of engineering systems using gaussian process regression active learning. *Engineering Applications of Artificial Intelligence*, 74:146–165, 2018.

[3] J. Beck, B. P. Woolf, and C. R. Beal. Advisor: A machine learning architecture for intelligent tutor construction. *AAAI/IAAI*, 2000(552-557):1–2, 2000.

[4] 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.

[5] 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. *UMUAI*, 21(1-2):137–180, 2011.

[6] L. Cronbach and R. Snow. *Aptitudes and instructional methods: A handbook for research on interactions.* Oxford, England: Irvington, 1977.

[7] S. D'Mello, B. Lehman, et al. A time for emoting: When affect-sensitivity is and isnâĂŹt effective at promoting deep learning. In *ITS*, pages 245–254. Springer, 2010.

[8] J. T. Flam, S. Chatterjee, et al. On mmse estimation: A linear model under gaussian mixture statistics. *IEEE Transactions on Signal Processing*, 60(7):3840–3845, 2012.

[9] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[10] D. Guo, S. Shamai, and S. Verdú. Mutual information and minimum mean-square error in gaussian channels. *arXiv preprint cs/0412108*, 2004.

[11] T. Haarnoja, A. Zhou, et al. Soft actor-critic algorithms and applications. *arXiv:1812.05905*, 2018.

[12] H. V. Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

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

[14] D. Horgan, J. Quan, et al. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[15] 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(1):89–106, 2009.

[16] A. Iglesias, P. Martínez, R. Aler, and F. Fernández. Reinforcement learning of pedagogical policies in adaptive and intelligent educational systems. *Knowledge-Based Systems*, 22(4):266–270, 2009.

[17] S. Kalyuga, P. Ayres, P. Chandler, and J. Sweller. The

expertise reversal effect. *Educational psychologist*, 38(1):23–31, 2003.

[18] N. Kim, Y. Lee, and H. Park. Performance analysis of mimo system with linear mmse receiver. *IEEE Transactions on Wireless Communications*, 7(11), 2008.

[19] T. P. Lillicrap, J. J. Hunt, et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[20] T. Mandel, Y.-E. Liu, et al. Offline policy evaluation across representations with applications to educational games. In *AAMAS*, pages 1077–1084, 2014.

[21] B. M. McLaren and S. Isotani. When is it best to learn with all worked examples? In *AIED*, pages 222–229. Springer, 2011.

[22] B. M. McLaren, S.-J. Lim, and K. R. Koedinger. When and how often should worked examples be given to students? new results and a summary of the current state of research. In *CogSci*, pages 2176–2181, 2008.

[23] B. M. McLaren, T. van Gog, et al. Exploring the assistance dilemma: Comparing instructional support in examples and problems. In *Intelligent Tutoring Systems*, pages 354–361. Springer, 2014.

[24] V. Mnih, A. P. Badia, et al. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[27] A. S. Najar, A. Mitrovic, and B. M. McLaren. Adaptive support versus alternating worked examples and tutored problems: Which leads to better learning? In *UMAP*, pages 171–182. Springer, 2014.

[28] K. Narasimhan, T. Kulkarni, and R. Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.

[29] X. B. Peng, P. Abbeel, et al. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):143, 2018.

[30] X. B. Peng, G. Berseth, et al. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[31] D. Precup, R. S. Sutton, and S. P. Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, pages 759–766. Citeseer, 2000.

[32] A. N. Rafferty, E. Brunskill, et al. Faster teaching via pomdp planning. *Cognitive science*, 40(6):1290–1332, 2016.

[33] A. Rajeswaran, V. Kumar, et al. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[34] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[35] A. Renkl, R. K. Atkinson, et al. From example study to problem solving: Smooth transitions help learning. *The Journal of Experimental Education*, 70(4):293–315, 2002.

[36] R. J. Salden, V. Aleven, et al. The expertise reversal effect and worked examples in tutored problem solving. *Instructional Science*, 38(3):289–307, 2010.

[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[38] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *Icml*, volume 37, pages 1889–1897, 2015.

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

[40] D. Schwab and S. Ray. Offline reinforcement learning with task hierarchies. *Machine Learning*, 106(9-10):1569–1598, 2017.

[41] R. Schwonke, A. Renkl, et al. The worked-example effect: Not an artefact of lousy control conditions. *Computers in Human Behavior*, 25(2):258–266, 2009.

[42] S. Shen, M. S. Ausin, B. Mostafavi, and M. Chi. Improving learning & reducing time: A constrained action-based reinforcement learning approach. In *UMAP*, pages 43–51. ACM, 2018.

[43] S. Shen and M. Chi. Reinforcement learning: the sooner the better, or the later the better? In *UMAP*, pages 37–44. ACM, 2016.

[44] D. Silver, A. Huang, C. J. Maddison, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[45] D. Silver, T. Hubert, J. Schrittwieser, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[46] D. Silver, J. Schrittwieser, K. Simonyan, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[47] R. E. Snow. Aptitude-treatment interaction as a framework for research on individual differences in psychotherapy. *Journal of Consulting and Clinical Psychology*, 59(2):205–216, 1991.

[48] R. S. Sutton, D. A. McAllester, et al. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[49] J. Sweller and G. A. Cooper. The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction*, 2(1):59–89, 1985.

[50] T. Van Gog, L. Kester, and F. Paas. Effects of worked examples, example-problem, and problem-example pairs on novices' learning. *Contemporary Educational Psychology*, 36(3):212–218, 2011.

[51] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.

[52] P. Wang, J. Rowe, W. Min, B. Mott, and J. Lester. Interactive narrative personalization with deep reinforcement learning. In *IJCAI*, 2017.