State and Reward Design: Towards Reinforcement Teaching

Alex Lewandowski[1,2], **Calarina Muslimani**[1,2], **Matthew E. Taylor**[2], **Jun Luo**[1]
[1]Huawei Noah's Ark Lab, jun.luo1@huawei.com
[2]University of Alberta, {alex3, musliman, matthew.e.taylor}@ualberta.ca

Abstract

A reinforcement learning agent that learns *tabula rasa* will make many missteps on its way to maximize its return. To accelerate learning, we can introduce a teacher agent that learns by observing the student, and acts by tuning the environment. In this paper, we provide a framework for learning to teach reinforcement learning agents by encoding the student's trajectory. We investigate different state representation and reward functions for the teacher. In tabular environments, we conjecture that the greedy policy induced by the learned action-values, not the action-values themselves, is an ideal state representation. We then propose three architectures that encode the student's trajectory to approximate the state representation provided by the greedy policy. Learning the teacher's policy offline, we find that the greedy policy state representation is superior, but that the trajectory based state representation is a close competitor. In addition, we design a new reward function for the teacher that enables the student to convey information about its learning progress. We show that the resultant teacher curriculum increases student learning efficiency compared to training the teacher with a minimal encoded reward function. These findings suggest that a general framework for reinforcement teaching can increase the sample efficiency of reinforcement learning.

1 Introduction

Reinforcement learning agents navigating the world are required to complete tasks of varying difficulty. More difficult tasks take longer to learn, by definition, but it is often the case that learning easier tasks first can help with learning harder tasks [3, 2]. This is the basis for curriculum learning, in which a mechanism assigns tasks to a student agent to help it learn easier tasks before progressing in difficulty [17]. Curricula are often hand-crafted, however, and rely on expert domain knowledge.

If prior knowledge is insufficient to specify a curriculum for an agent, can we learn a mechanism that provides a sequence of tasks to the student? This is the focus of automatic curriculum learning. Instead of relying on expert domain knowledge, the mechanism learns an optimal curriculum over tasks to maximize an agent's competence on the target task or over a set of tasks. Curricula have been learned through a variety of mechanisms, including self-play [25], active domain randomization [15], and reverse goal generation [7]. Other work extends Generative Adversarial Networks [9], to the auto-curricula setting, which allows for one agent to generate tasks/curricula via a generative model and another agent to evaluate whether the proposed task is appropriate using a discriminator [23, 6].

Another promising line of research is student-teacher curriculum learning. In this setting, a teacher agent learns to provide appropriate tasks to a student agent based on the student's current skill set [14]. There are many ways that a teacher can help a student agent learning through reinforcement. For example, the teacher can provide action-advice [5]. Narvekar et al. (2017) proposed a two-level MDP, with the teacher agent operating in a curriculum MDP that chooses tasks for the student, and the student agent operating in an MDP associated with the given task [18]. In the curriculum MDP,

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

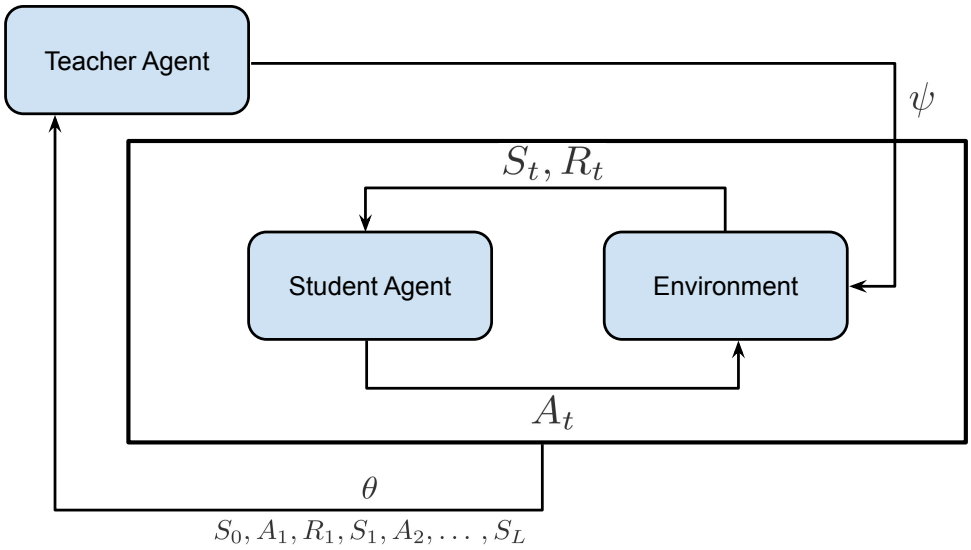

Figure 1: A student and teacher both learn in their own MDP. The student interacts with its environment by sending actions A_t and observing the state and reward S_t, R_t. The teacher agent observes trajectory of the student, and possibly the student's parameters θ, and takes actions ψ that influence the environment.

the state space is the set of all student policies, and the action space is the set of possible tasks for the student agent. The teacher is trained to minimize the time taken for the agent to reach a threshold level of performance. Two issues pertaining to the state representation and the reward structure arise. For the state representation, with function approximation, the state space would be the parameter space of the student agent. The parameter space of a policy is a very large and unstructured state space, which can make learning a policy on these states difficult. Although the parameter-space state representation was simplified using tile coding [19], it is still limited to policies with a fixed and small number of parameters. As for the teacher's reward function, the student only provides binary information on success or failure of a task. The student could convey more meaningful information if it could indicate how it is progressing throughout its training.

These approaches can be contrasted with other formulations of the student-teacher setting that considers the teacher agent as a bandit problem [22, 10]. Work on the bandit formulation emphasizes the use of learning progress signals to enable the teacher to detect how much progress a student is making on a given task. The use of bandits does simplify the problem, because we no longer need to represent the state of the student agent. However, it sacrifices the teacher's ability to differentiate between different students.

In this work, we consider the student-teacher curriculum learning framework, and aim to investigate more general state representations and reward functions to improve the teacher's ability to generate curricula for a student. Our main contributions are as follows (1) We show that the greedy policy is an ideal state representation and then propose an architecture that approximates this state representation from the student's trajectory (2) Develop a novel reward function, based on learning progress signals that enable the student to convey a richer set of information on its learning progress.

2 Background

Before we address the curriculum learning problem, we first briefly describe the Markov Decision Process (MDP) formalism that underpins reinforcement learning [13, 26]. An MDP $\mathcal{M}$ is defined by the tuple $(\mathcal{S}, \mathcal{A}, r, p, \mu)$, where $\mathcal{A}$ denotes the action space, $\mathcal{S}$ is the state space, $r : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function that maps a state and an action to a scalar reward, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, and μ is the initial state distribution.

The curriculum learning problem can also be specified using the MDP formalism. This problem formulation is referred to as a Curriculum MDP $\mathcal{M}^C$ and it is defined by the 6-tuple $\left(\mathcal{S}^C, \mathcal{A}^C, r^C, p^C, S_0^C, S_f^C\right)$ [18]. Each element is defined analogously to the MDP formalism. These elements can be interpreted as follows: the state-space $\mathcal{S}^C$ is the set of all student policies, $\mathcal{A}^C$ denotes the tasks available to the teacher, the reward function $r^C : \mathcal{A}^C \times \mathcal{S}^C \to \mathbb{R}$ incentivizes the teacher to help the student reach competency quickly, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, S_0^C is the distribution over start states and S_f^C is the set of terminal states.

In a Curriculum MDP, the goal of the student is to maximize its return in any environment it encounters, and the teacher's goal is to guide the student to quickly solve the target problem, denoted by $\tilde{\psi}$. For clarity, we will denote the MDP that the student interacts with as the student MDP $\mathcal{M}_\psi^S$ that depends on some hyperparameter settings, or a task encoding, by the subscript ψ. The teacher achieves this by taking actions $\psi \in \mathcal{A}^C$ that change the student's environment $\mathcal{M}_\psi^S$ away from the target problem $\mathcal{M}_{\tilde{\psi}}^S$. The teacher observes the student, selects an action and through the state transition in the teacher's MDP, observes the new student after learning from the proposed action ψ, along with an associated reward for the teacher's action. The role of ψ depends on the problem, but it can control any aspect of the student's MDP, such as the start-state distribution μ^S, reward function r^S or transition function p^S.

3 The Teacher's MDP

Using reinforcement learning to teach a student can also be formulated solely in terms of the interaction between the student, the student's environment and the teacher. We define the teacher's MDP $\mathcal{M}^T$, as the tuple $\mathcal{M}^T = \{\mathcal{S}^T, \mathcal{A}^T, p^T, r^T, \mu^T\}$. Each element is defined similarly to a Curriculum MDP, except for the state-space $\mathcal{S}^T$ which is no longer the set of policies. The action set for the teacher $\mathcal{A}^T$ corresponds to putting the student in a particular task, the reward function $r^T : \mathcal{A}^T \times \mathcal{S}^T \to \mathbb{R}$ should incentivize the teacher to help the student reach competency quickly, $p^T : \mathcal{S}^T \times \mathcal{A}^T \times \mathcal{S}^T \to [0, 1]$ is the state transition function, and μ^T is the initial state distribution.

In this paper, we investigate different state representations and reward functions for the teacher. The student's environment and its available scenarios ψ defines the action set $\mathcal{A}^T$, while the student's learning algorithm determines the transition function p^T. It remains an open question how to best choose a state representation $\mathcal{S}^T$ (which will induce an initial state distribution μ^T) and a reward function r^T to facilitate efficient learning.

Before a teacher can take actions that improve the student, the teacher must first represent the state of the student. Consider the following analogy: a teacher gives a student an assignment and may provide other assignments based on the student and the result of their assignment. While largely impractical, the brain state of the student represents exactly the student's state, and carries more than sufficient information to inform the teacher. On the other hand, looking at the student's score on the assignment does not provide enough information for effective remediation (which we will show empirically in Section 5.2). The teacher may, instead, process the set of questions and answers that the student provides and, using this information, identify areas of strengths and weaknesses for the student. This information can then be applied by the teacher to assign effective remedial treatment, in the form of a more appropriate assignment. In reinforcement learning terms, the first approach would use the parameters of the student policy [18]. The bandit approach [14], which looks at only the reward, represents the second approach in the analogy and does not provide enough acuity to identify proper remediation. The last approach, which encodes the student trajectory, is the approach we take and describe in Section 3.1

For a teacher to accurately assess a student's performance, it needs to monitor the student's progress in the target task. The student is represented by the state in the teacher's MDP, and rewards should correlate with the student successfully completing the target task. A minimal approach to encoding this goal is to provide a reward of 1 for reaching a threshold level of performance [18]. While this encodes the intended behavior, credit assignment becomes very challenging. In this type of reward structure, the student takes on a passive role within its own learning process. The student receives tasks to learn and can only communicate with the teacher once its performance on the target task has reached a specified value. However, if the student-teacher relationship were bi-directional, this would allow for the student to convey more meaningful information. This can include when the student is

not making progress on a difficult task or when it's ready to receive a more challenging task. As an alternate approach, we can approximate when the student's learning has plateaued on a given task, and use this to bias the teacher on what task to provide next. In this case, we consider how the student is progressing on all tasks, not just the target task. To form this approximation, we take inspiration from methods based on Learning Progress (LP) which consider the change in either loss or a evaluation metric before and after training on a given task [22, 21, 14, 10]. In this setting, as this difference converges to 0, an agent is said to be making less progress on a given task. We describe this in more detail in Section 3.2.

Next, we describe the reward design and state representations. These are two design choices that have many possibilities and hence we designate their own subsection for extended discussion. The design choices discussed here will be later ablated in the experiments in Section 5.

3.1 State Representation

When the student's environment is tabular, one possible state representation uses the action-value table of the student [20, 18], which we denote as the **Student-Q** state representation. When the student uses a function approximator, neither the action-value table nor the policy are available. Instead, the parameters of the function approximator can be concatenated into a vector [19], which we denote as the **Student-Parameter** state representation. Unlike the parameter state representations with function approximation, the Student-Q state representation provides a semantically meaningful representation of the student's current state. Each entry of the action-value table is bounded (if the rewards are bounded) and is a function of the learning rate, as well as the state, actions and rewards encountered by the student in its trajectory. Many action-value tables can represent the same policy, because only the relative difference of action-values determines the optimal action. Ultimately, the behaviour of the student determines it's performance, not the accuracy of its action-value estimates. Our first proposal is to directly encode this by transforming the action-value table into a greedified policy. This policy parameter representation, which we denote **Student-π**, is our first modification and provides intuition for our following generalization.

We generalize the problem of learning a state representation by allowing the teacher to directly encode the student's episode trajectory. The student's episode trajectory is a local representation of the "state" of the student agent, and it represents local information about the policy. Global information of the student policy, on the other hand, would require looking at the parameters and architecture of the policy. We posit that local information from the trajectory, when properly encoded, is enough to approximate the global policy. In Section 4.2, we specify an architecture that can learn to approximate the global policy from the student's trajectory alone.

There are several advantages of using local information from the trajectory, as opposed to global information about the policy. First, a trajectory-based state representation is agnostic to the parameterized family of the policy. This allows us to use our approach with hand-designed controllers and any neural architecture. Another issue is that the parameter space is unstructured. Many different parameterizations can lead to the same neural network output (e.g., if the rows or columns of all weight matrices are permuted). It is also not straightforward to encode non-linearities and overall network architecture, a process known as fingerprinting [11, 4]. Lastly, we can also scale to policies with larger numbers of parameters, without encountering difficulties with storing large weight matrices and requiring the teacher's neural network to learn from these semantically meaningless states.

3.2 Designing the Reward Function

Within the student-teacher framework, the teacher should not only be enforcing which tasks the student will learn, but ideally the student should communicate with the teacher when it's ready to learn a different task. This bi-directional component can allow for the student to convey when it's not making progress on a difficult task or when it's already learned an easier task. To that end, we design a novel reward function which begins to enable this bi-directionality. The essence of the reward function is based on two signals (1) Stagnation Factor (as defined in Algorithm 1) and (2) Learning Progress (LP). We develop the Stagnation Factor (SF) as a means to inform the teacher on how long the student has made no changes in learning on a task. If SF is high for a given task, we say the student's learning on this task has plateaued. Learning Progress provides information on the

magnitude and direction of the change in student learning. For example, a large positive LP signal indicates the student is making significant progress on a task.

To determine the teacher's reward, we first calculate the LP and SF for a given task, a_T the student encountered. Let SF_{a_T} and LP_{a_T} denote the Stagnation Factor and Learning Progress for the specific task a_T. If SF_{a_T} exceeds a specific threshold, the teacher receives a negative reward. This informs the teacher that with the student's current skill set (1) the task is too difficult and the student is not making any progress on it or (2) the task is too easy and the student needs a different one. If the student's learning hasn't plateaued, we use LP_{a_T} as the teacher's reward. See 1 for pseudo-code of the reward function. In addition, we also reward the teacher with a binary 0/1 reward based on whether or not the student has succeeded on the target task. This avoids the case where the student has converged on the optimal policy for the target task, but the teacher is still penalized for convergence. This design not only enables the student to convey more information about it's learning progress to the teacher but also removes the need for hand-coded success thresholds.

Algorithm 1 LP-SF Teacher Reward Function

Initialize Stagnation Factor (SF) for all N teacher actions to 0
Let $\epsilon > 0$ and Threshold > 0

Student receives task a_T from Teacher
Train student using task a_T and observe $LP_{a_T} = G_t^{a_T} - G_{t-1}^{a_T}$
if $LP_{a_T} \in [0 - \epsilon, 0 + \epsilon]$ **then**
    Increment SF_{a_T} by 1
else
    $SF_{a_T} = 0$
end if

Calculate Teacher Reward:
if $SF_{a_T} >$ Threshold **then**
    $R^T = -1$
else
    $R^T = LP_{a_T}$
end if

4 Learning Offline in a Teacher's MDP

With the teacher's problem formulation from the previous section, we now turn to the problem of learning in the teacher's MDP. Learning in the teacher's MDP is more difficult than in the student's environment, even for simple tabular student environments. The reason for this is that, even if the student's state and action space is discrete, the trajectory generated by the student can be combinatorially large. Even for tabular student environments, the teacher's problem requires function approximation.

4.1 Learning a curriculum offline

One criterion for a good teacher learning algorithm is low sample complexity. Interacting with the teacher's MDP and evaluating a teacher is expensive. A teacher's episode corresponds to an entire training trajectory for the student. Hence, generating numerous teacher episodes involves training numerous student agents. The teacher agent cannot afford an inordinate amount of interaction with the student. One way to meet the sample complexity needs of the teacher is to use off-policy learning, such as Q-learning. Offline learning can also circumvent the costly interaction protocol, and so we investigate state representations and reward functions that allow us to learn from data generated by uniformly random teacher policy. While there is a large and growing literature on offline RL algorithms [28, 27, 8, 12], we found that offline Q-learning (Offline DQN [1], *i.e.* Neural Fitted Q Iteration [24], with target networks and a replay buffer) works in this regime and leave investigation of other offline-specific algorithms for future work.

4.2 Learning a state representation

In the tabular setting, the tabular policy is an ideal state representation. When using function approximation however, we no longer have access to the tabular policy. We can still log information about the policy in the states encountered by the student agent. Trajectory may differ in length, and thus this state representation would not have a constant size. In addition, the states encountered along the trajectory vary depending on the policy. These two issues prevent us from naively concatenating the outputs along the trajectory, and force us to consider an encoding of the trajectory instead.

We now propose an architecture for learning a state representation from the student's trajectory. Consider a trajectory $\tau = \{S_0, A_1, R_1, S_1, A_2, \dots, S_L\}$ of length L. Trajectories encode temporal information relating states, actions and rewards. One way to encode a temporal sequence is by using a recurrent architecture. The inputs for the recurrent architecture at each time step is the concatenation of the state s_t with either the: action (a), student's action-value at s_t ($Q(s_t)$) or the agent's policy at s_t ($\pi(s_t)$. Where both elements in the concatenated vector are first fed through separate fully-connected neural networks, hence concatenating their representation. We denote these three possibilities as: **Encode-Action**, **Encode-Q** and **Encode-π**. The goal of these three methods is to encode the trajectory and learn an approximation of the global policy of the student agent. We include the Q variant to compare against the "Student-Q" state representation, and we include the "action" variant because actions are the only information we typically have about a student.

5 Experiments

We conduct experiments using a Four Rooms grid world environment for the student where the teacher has control over the start state of the student agent. The target task is starting in a state that is on the opposite corner from the goal. While this is easily solvable for a tabular learning agent, we limit the number of steps the student agent can take to only 25 steps in a single episode. The low number of steps in an episode makes it unlikely for the student agent to ever reach the goal without a teacher connecting these partial trajectories from the start state to the goal.

In all experiments, we compare against a uniformly random teacher policy and a teacher that selects the action that corresponds to the target task ("no teacher"). In the state representation experiments, the learning algorithm for the teacher is an offline DQN [16, 1] and the teacher learns from 1000 episodes from a uniformly random teacher policy. In the reward design experiments, the teacher is trained with the online DQN algorithm over 1000 episodes. In both cases, the student is a Tabular Q-Learning agent.

5.1 Reward Design Experiments

For the Four Rooms domain, we experimented with two teacher action sets. The regular action set contained the four corridor states and the start state of the target task. The larger action set expanded on the regular action set by including 3 additional states neighboring the start state. In the reward design experiments, our goal is to understand the effectiveness of our reward function. Therefore, we only use one state representation for the teacher which is the Return representation concatenated with the teacher action, teacher reward, indicator (1/0) of student's success on the target task and the Stagnation Factor for each teacher action.

In these experiments, we compared our LP-SF reward function against the standard reward function used in the state representation experiments, in which the teacher is rewarded -1 until the student can solve the target task. To isolate the effects of the reward functions, we used the same state representations in both settings. In addition, we also compared against two other baselines, random teacher and no teacher at all. Figure 2 shows the learning curves from both the regular action set experiment (right) and larger action set experiment (left). We found that in the regular action set experiment, training the teacher with our LP-SF reward function greatly improves the teacher's curriculum over the three baselines. This resultant curriculum allows the student agent to not only learn faster but to achieve a higher final performance than learning from the other three curricula. Moreover, for the larger action set experiment, we found that training the teacher with our LP-SF reward function improves the student's learning efficiency compared to all baselines, and allows the student to reach higher final performance than using the standard reward function and using no teacher at all.

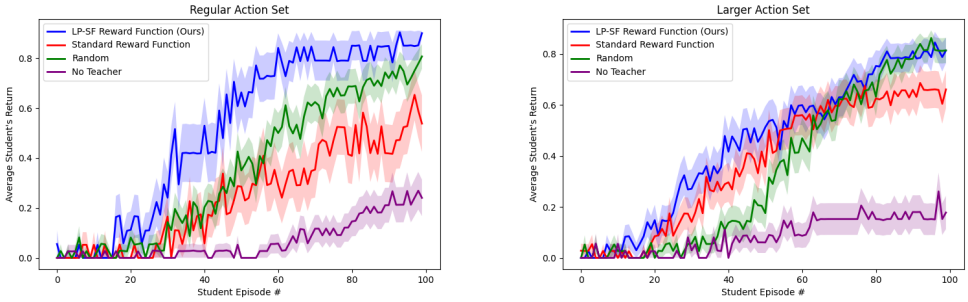

Figure 2: Left: Regular Action Set. Right: Larger Action Set. In both experiments, the teacher is trained with our reward function discussed in 3.2. The resultant curriculum allows for the student to learn much faster compared with the all baselines.

Our results suggest that by allowing the student to indirectly communicate with the teacher on when it should receive a new task via the reward function creates for an improved curriculum over the three baselines.

However, we do observe that the differences in the student's learning efficiency is not as significant in the larger action set experiment compared to the regular action set experiment. We hypothesize that the current reward design can be affected by the size of the teacher's action space. More specifically, if the teacher's action set consists of several sub-tasks that may be 'easy' but not necessary for the student to learn in order to improve learning on the target task. In the LP-SF reward function, the teacher is always incentivised for proposing easier tasks first (i.e tasks for which the student can make progress). Therefore, this can possibly delay the teacher from selecting a harder task, which may reduce the overall learning efficiency of the student. In future work, we aim to explore adaptations to the reward design that alleviate this issue.

As for the teacher's curricula that emerged, we observed the general pattern of the teacher selecting easier tasks, followed by harder tasks. Due to the nature of the reward function, the teacher is encouraged to select tasks for which the student is making progress. At the start of learning, the student has limited skills, therefore it makes more progress on easier tasks. However, once the student has successfully learned the easier tasks, its performance plateaus and its learning progress goes to 0, so the teacher is encouraged to select gradually more difficult tasks for which it can make progress.

In addition, we also observed that once the student has successfully learned the target task, the teacher begins to "randomly" sample amongst all tasks. By this point in student training, the student can solve all tasks, therefore the teacher is rewarded similarly for any proposed task. This would result in all teacher actions having similar action-values.

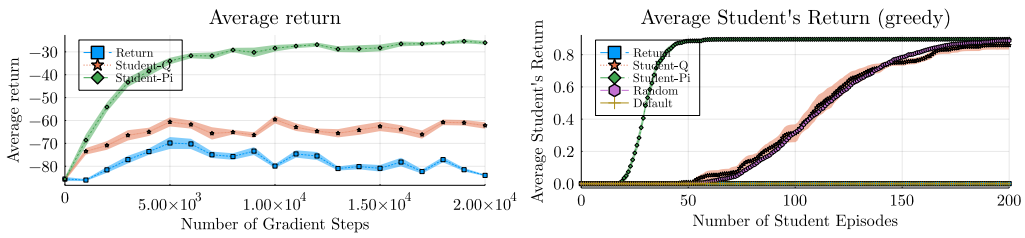

Figure 3: Four Rooms MDP, comparing state representations that do not encode the student trajectory. Left: The returns achieved by the teacher (evaluated in the environment) during offline learning. Right: The returns achieved by the student using a fully trained teacher. Default and Return are tied at the bottom, Random and Student-Q are tied in the middle.

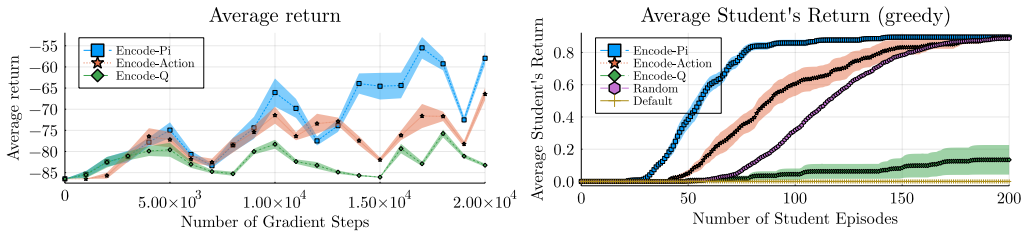

Figure 4: Four Rooms MDP, comparing state representations that learn different encodings of the student's trajectory Left: The returns achieved by the teacher (evaluated in the environment) during offline learning. Right: The returns achieved by the student using a fully trained teacher.

5.2 State Representation Experiments

We investigate two classes of state representations: those that do and do not use a trajectory encoder. For the state representation experiments, we first show the performance amongst state representations that do not encode the trajectory. As discussed in 3.1, we introduce Student-π and compare it against Student-Q. We also compare it to a naive heuristic that only looks at the last teacher's action and the sum of reward along the trajectory that the student accrued in that task (denoted as the "Return" state representation).

For the non-encoding state representations in Figure 3 (left), we find strong evidence that the Student-π state representation is strictly better than the Student-Q representation. This confirms our hypothesis that only the relative magnitude of action-values is pertinent to predicting the student's task. Referring now to 3 (right), we see that Student-Q is able to match the random teacher policy, but not improve over it. We also find that the "Return" state representation is not expressive enough for the teacher to direct the student, and does much worse than random.

Turning to the encoding state representations in Figure 4 (left), we find that the encoding architectures that use the policy (Encode-π) performs significantly better than the other state representations. The state representation that uses actions (Encode-Action) is able to improve over the random teacher. The reason that Encode-π is superior to Encode-Action is because Encode-Action encodes exploratory actions, which the agent will likely not repeat. The poor performance of Encode-Q can be attributed to the same problem of the Student-Q state representation, only exacerbated by the fact that Encode-Q can only use local Q-values for the states in the trajectory. Comparing the results in Figure 3 and Figure 4, we find that both Encode-π and Encode-Action are close to the ideal of Student-π, while only using local information from the trajectory. This demonstrates the importance of incorporating policy information at a local and global level to represent the state of the student agent.

6 Discussion

For future work, we want to consider adaptations to the reward design which allow for better guidance from the student. In the current design, the teacher is discouraged from selecting tasks for which the student's performance has plateaued. However, the teacher does not receive information on why the student's performance has plateaued. Therefore, the teacher is unable to determine whether the previous task was too easy or too hard. In future work, we want to consider adaptations that differentiate between these cases. This can allow the teacher to make more informed decisions on which tasks will result in the most efficient learning for the student. There is also much room for innovation in the encoding architecture. In addition, we would like a better understanding of the learning dynamics for the encoder in conjunction with bootstrapping and the use of target networks.

In this paper, we have investigated reward design and state representation in student-teacher curriculum learning. For reward design, we have demonstrated empirically that by tuning the reward based on learning progress improves curriculum learning by promoting credit assignment. For state representation, we have conjectured that the tabular greedy policy is an optimal representation for the teacher's MDP, and this has been verified in our experiments. Our experiments also show that it is possible to approximate this representation by encoding the trajectory. These findings suggest that a general framework for reinforcement teaching can increase the sample efficiency of reinforcement learning.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 104–114. PMLR, 2020.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[3] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[4] Francesco Faccio, Louis Kirsch, and Jürgen Schmidhuber. Parameter-based value functions. *arXiv:2006.09226*, 2020.

[5] Anestis Fachantidis, Matthew Taylor, and I. Vlahavas. Learning to teach reinforcement learning agents. *Machine Learning and Knowledge Extraction*, 1, March 2019.

[6] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. *arXiv:1705.06366*, 2017.

[7] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv:1707.05300*, 2017.

[8] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv:2106.06860*, 2021.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv:1406.2661*, 2014.

[10] Alex Graves, Marc G. Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. *arXiv:1704.03003*, 2017.

[11] Jean Harb, Tom Schaul, Doina Precup, and Pierre-Luc Bacon. Policy evaluation networks. *arXiv:2002.11833*, 2020.

[12] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[13] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[14] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *arXiv:1707.00183*, 2017.

[15] Bhairav Mehta, Tristan Deleu, Sharath Chandra Raparthy, Chris J. Pal, and Liam Paull. Curriculum in gradient-based meta-reinforcement learning. *arXiv:2002.07956*, 2020.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv:1312.5602*, 2013.

[17] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv:2003.04960*, 2020.

[18] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2536–2542, 2017.

[19] Sanmit Narvekar and Peter Stone. Learning curriculum policies for reinforcement learning. *arXiv:1812.00285*, 2018.

[20] Sanmit Narvekar and Peter Stone. Generalizing curricula for reinforcement learning. 2020.

[21] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

[22] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. *arXiv:1910.07224*, 2019.

[23] Sebastien Racaniere, Andrew K. Lampinen, Adam Santoro, David P. Reichert, Vlad Firoiu, and Timothy P. Lillicrap. Automated curricula through setter-solver interactions. *arXiv:1909.12892*, 2019.

[24] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[25] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv:1712.01815*, 2017.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 2018.

[27] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv:1911.11361*, 2019.

[28] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv:2005.13239*, 2020.