Inter-agent Transfer Learning in Communication-constrained Settings : A Student Initiated Advising Approach

Argha Boksi
Department of Computer Science and Engineering
Robert Bosch Centre for Data Science and AI
Indian Institute of Technology Madras
Chennai, India
argha.boksi@gmail.com

Balaraman Ravindran
Wadhwani School of Data Science and AI
Department of Computer Science and Engineering
Robert Bosch Centre for Data Science and AI
Indian Institute of Technology Madras
Chennai, India
ravi@cse.iitm.ac.in

ABSTRACT

Deep reinforcement learning algorithms have shown promise in addressing complex decision-making problems, but they often require millions of steps of suboptimal performance to achieve satisfactory results. This limitation restricts the application of Deep RL in many real-world tasks, where agents cannot afford to rely on thousands of learning trials, particularly when each suboptimal trial is costly. The teacher-student framework seeks to enhance the sample efficiency of RL algorithms. In this setup, a teacher agent guides another student agent's exploration by providing advice on the optimal actions to take in specific states. However, in numerous applications, communication is constrained by factors such as available bandwidth or battery power. In this paper we consider a student-initiated advising approach where the student can query the teacher only a predetermined fixed number of times. We introduce a framework, *Ask Important* that - *(a)* ensures effective utilization of the limited advice budget by querying the teacher only in *important* states and *(b)* makes efficient use of the collected demonstration data by introducing an additional demonstration buffer. *Ask Important* framework can be utilised by RL algorithms(which work with discrete action spaces and leverage a replay buffer to store and sample experiences) such as *DQN, Double DQN, Dueling DQN* etc. We explain how *Ask Important* can be integrated within the *DQN* algorithm. We compare *DQN Ask Important* with – *DQN(baseline)* and an ablation of our method. We evaluate these algorithms in three Gymnasium environments – *Acrobot-v1*, *MountainCar-v0* and *LunarLander-v2*. The results show that *DQN Ask Important – (a)* has better initial performance and *(b)* reaches the target average episodic return much faster – than the other two algorithms for all the three environments.

KEYWORDS

Reinforcement Learning, Teacher-student framework, DQN

1 INTRODUCTION

The integration of modern Deep Learning (DL) techniques with traditional Reinforcement Learning (RL) approaches has shown promise [9, 11, 17] in addressing complex decision making problems. However, these algorithms achieve satisfactory performance only after undergoing millions of steps of suboptimal performance. This significantly restricts the use of Deep RL in many real-world tasks. In real-world domains agents - *(a)* cannot rely on thousands of learning trials to attain an effective policy, especially when each sub-optimal trial is costly and *(b)* should have good on-line performance from the start of learning. For instance, an autonomous driving agent must not acquire driving skills through the process of colliding with road barriers and putting the lives of pedestrians at risk.

Utilizing the experience of a more competent agent [5] has emerged as one of the most successful strategies in addressing sample complexity concerns in RL. The Teacher-Student framework [23] is one such paradigm, wherein a teacher agent conveys instruction to a learner agent in real time with the objective of expediting the learning process. The teacher agent in this context is often considered to be an 'expert' or a pre-trained agent that already possesses a good policy for the given task. Using this established policy, it will instruct another RL agent that is in the initial stages of learning the same task.

In theory, it is feasible to receive instructions at each time step of the agent's learning process. However, in numerous applications, communication is constrained by factors such as available bandwidth [18] or battery power, especially when dealing with standalone robots or wireless sensors. In this work we model communication scarcity by introducing a fixed advice budget [1, 23](i.e. a maximum number of inter-agent interactions). This imposes a strict limit on the extent to which agents can communicate.

Previous research has explored two modes of providing advice: student-initiated [3, 4] and teacher-initiated [23]. In teacher-initiated advising the teacher initiates the interaction between the agents. It is assumed that the student's current state is consistently conveyed to the teacher. This can result in substantial communication costs(e.g. high power requirements for operating visual sensors). Hence, in this paper we consider a student-initiated advising approach where in the learner independently determines when to send a query. Once the learner sends its query, the teacher then provides action advice [23] to the learner. This approach necessitates minimal similarity between teachers and students, requiring only a shared action set. The agents can employ distinct learning algorithms. The learner updates its policy based on the provided advice.

We propose to extend the existing Teacher-Student approaches by introducing a framework, *Ask Important* that - *(a)* ensures effective utilization of the limited advice budget by querying the teacher only in *important* states and *(b)* makes efficient use of the collected demonstration data by introducing an additional demonstration

Proc. of the Adaptive and Learning Agents Workshop (ALA 2024), Avalos, Müller, Wang, Yates (eds.), May 6-7, 2024, Online, https://ala2024.github.io/. 2024.

buffer. *Ask Important* framework can be utilised by RL algorithms which work with discrete action spaces and leverage a replay buffer to store and sample experiences. Examples of such RL algorithms include *DQN, Double DQN, Dueling DQN* etc. We propose an algorithm, *DQN Ask Important* which incorporates the *Ask Important* framework into the DQN algorithm.

We compare *DQN Ask Important* with *DQN* and *DQN(Demonstration in the first K steps)* which is an ablation of our method. We evaluate these algorithms experimentally in three Gymnasium [24] environments – *Acrobot-v1, MountainCar-v0 and LunarLander-v2*. The results show that *DQN Ask Important – (a)* has better initial performance and *(b)* reaches the target average episodic return much faster(i.e. solves the task in much fewer learning trials) – than the other two algorithms for all the three environments.

2 RELATED WORK

Our paper extends and builds upon previous research that explores the student-teacher reinforcement learning framework [1, 4, 8, 23]. Chernova and Veloso [3] introduced a confidence-based approach where a learning agent requests demonstrations when it faces uncertainty about its actions. Unlike the teacher-student framework, in their approach, the agent solely learns from expert demonstrations without receiving feedback from the environment. In active imitation learning [7], an agent has the capability to request an expert for its policy for a specific state. Additionally, the agent can simulate trajectories and does not seek demonstrations during execution. Rosman et al. [15] devised techniques for determining the appropriate moments to provide guidance to an agent, but their approach assumes that teachers have access to a knowledge base containing typical agent trajectories.

In scenarios where the instructor is also an automated agent, the most frequently employed form of instruction is action advice [13, 23]. The field of learning from demonstrations [2] spans various research endeavors focused on agents acquiring the capability to replicate actions demonstrated by an expert. The demonstrations serve as examples of desired behavior, guiding the agent in acquiring a policy that can achieve similar outcomes. This approach is particularly useful in scenarios where exploration is costly or impractical.

There is also interest in the integration of imitation learning and RL [22]. Evidence indicates that demonstration data is beneficial in addressing challenging exploration problems in RL [19]. [16] considers real-world learning with robots, and therefore, it also addresses concerns related to online performance. They initially pre-train the agent using demonstration data before allowing it to engage with the task. One frequently employed technique involves merging samples from demonstrations with samples gathered by an agent into a single experience replay. This ensures the use of demonstrations throughout the learning process alongside the utilization of new experiences. Human Experience Replay [6] is one such algorithm that involves the agent sampling from a replay buffer containing a mixture of both agent and demonstration data. Replay Buffer Spiking [10] is a comparable approach wherein the DQN agent's replay buffer is initiated with demonstration data. However, there is no pre-training of the agent, and the demonstration data is not retained permanently. In [25] ,the demonstrations were stored in a prioritized experience replay alongside the agent's own experiences. The demonstration experiences are given a higher probability of being selected. The idea of storing experiences in two separate replays was introduced in [14]. When the agent samples a mini-batch for learning, it samples a specific amount from each buffer.

3 BACKGROUND

The subsequent sections offer a concise overview of reinforcement learning and the teacher-student framework.

3.1 Reinforcement Learning

RL [21] involves a series of interactions with the environment, where actions are taken and rewards, as well as the subsequent states, are observed. Formally, this entire process can be defined within the framework of a Markov Decision Process (MDP). An MDP is defined by a tuple (S, A, R, T, γ), which consists of a set of states S, a set of actions A, a reward function $R(s, a)$, a transition function $T(s, a, s') = P(s'|s, a)$ and a discount factor γ. In each state $s \in S$, the agent takes an action $a \in A$. Upon taking this action, the agent receives a reward $R(s, a)$ and reaches a new state s', determined from the probability distribution $P(s'|s, a)$. A policy π specifies the action the agent takes for each state. The agent's goal is to find the optimal policy that maximizes the expected discounted total reward over the agent's lifetime.

3.2 Q-Learning

Q-learning [26] stands out as one of the most widely used algorithms for addressing sequential decision making problems. The agent learns a Q-value function. The Q-value, denoted as $Q^\pi(s, a)$, for a specific state-action pair (s, a) serves as an estimation of the expected future reward achievable by taking action a in state s and then following policy π. At each time step, the agent operates with a greedy policy π, selecting an action that maximizes the Q-function. The optimal value function, $Q^*(s, a)$, gives maximum values for all states and is derived by solving the Bellman equation -

$$Q^*(s, a) = \mathbb{E}\left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a')\right]$$

We can subsequently employ the Value Iteration algorithm to derive an iterative update formula for learning the Q-values -

$$Q_{i+1}(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q_i(s', a')$$

This approach is effective when dealing with relatively small state and action spaces, allowing the use of a table to manage the Q-values associated with all state-action pairs. However, when the state-action space becomes large, it becomes impractical to precisely compute the optimal Q-value function. Therefore, in Q-Learning, a function approximator is employed instead.

3.3 Deep Q-Network (DQN)

Deep Q-Network (DQN) algorithm, introduced by Mnih et al. [11], is an off-policy, value-based temporal difference (TD) algorithm designed to approximate the Q-function. It is applicable to environments characterized by discrete action spaces. DQN relies on two

essential components to function effectively. Firstly, it incorporates a distinct target network, periodically copied from the primary network every τ steps, ensuring greater stability in target Q-values. Secondly, the agent accumulates all its experiences in a replay buffer D^{replay}, which is subsequently uniformly sampled to execute network updates.

3.4 Teacher-Student Reinforcement Learning

The teacher-student framework [1, 4, 8, 23] involves two agents: a student and a teacher. The teacher has a predetermined policy $\pi_{teacher}$ for interacting with the environment, while the student employs a reinforcement learning algorithm to develop its policy $\pi_{student}$. The teacher can provide guidance to the student at any state s by sharing $\pi_{teacher}(s)$. Both agents must share the same action space. When the student receives advice, it takes the recommended action, treating it like any other action during the learning phase. The student updates its policy based on environmental reward signals, as in typical reinforcement learning, with exploration guided by the teacher's advice.

In the teacher-initiated advising approach [23], the teacher takes the lead in initiating interactions between agents. The learner does not actively seek guidance and relies on the teacher's discretion for when and what advice is provided. Conversely, in student-initiated advising [3, 4], the learner is responsible for instigating interactions. The jointly-initiated advising approach [1] involves the student deciding whether to seek the teacher's attention using student-initiated methods. Subsequently, the teacher independently decides whether to provide advice, using teacher-initiated approaches.

3.5 The Notion of State Importance

In certain tasks, certain states hold more significance than others, and reserving advice for these pivotal states proves to be an effective strategy. Games frequently exhibit both calm and tense moments. In specific scenarios, making the correct move can secure a victory, while an incorrect move can result in defeat. Conversely, in different situations, any move is deemed acceptable. This intuitive concept defines state importance. In the context of students as RL agents with Q-functions, they inherently possess a means to calculate state importance($I(s)$) [4, 23]. $Q(s, a)$ estimates the potential rewards achievable by taking action a in state s. If the Q-values for all actions in s are identical, the choice of action is inconsequential, rendering s unimportant. Conversely, if certain actions in s have higher Q-values, the choice becomes significant, indicating the importance of state s. Formally, $I(s)$ is defined as:

$$I(s) = \max_a Q(s, a) - \min_a Q(s, a) \tag{1}$$

4 APPROACH

We consider a student-initiated advising approach where the student can query the teacher only a predetermined fixed number of times(i.e. limited advice budget). We propose to extend the existing Teacher-Student approaches by introducing a framework *Ask Important*. This framework can be utilised by RL algorithms that – work with discrete action spaces and leverage a replay buffer to store and sample experiences. Examples of such RL algorithms include *DQN, Double DQN, Dueling DQN* etc. In the following subsections we explain how the *Ask Important* framework can be incorporated into the DQN algorithm.

4.1 Data Gathering Policy

In Vanilla DQN, we typically use ϵ-greedy or softmax policy to gather experiences. One problem with these policies is that the exploration strategy is naive. Agents explore randomly and do not use any previously learned knowledge about the environment. Consequently, the agents acquire good policies after undergoing numerous episodes of suboptimal performance. While this scenario is acceptable in simulations, many real-world challenges lack such simulators. In such cases, the agent must learn within the real-world domain, facing genuine consequences for its actions. This necessitates the agent to have good online performance right from the onset of the learning process.

During training student can evaluate the state importance $I(s)$ of a state s by computing equation (1) using its own Q-network ($Q_\theta^{student}$). Student can decide to ask for advice when the value of $I(s)$ exceeds t_{si}, where t_{si} is state importance threshold predefined for the student agent. This idea can be utilized to obtain a data gathering policy $\pi_{student}$. Formally,

$$\pi_{student}(s) = \begin{cases} \underset{a}{\text{argmax}}\, Q^{teacher}(s, a) & \text{if } I(s) \geq t_{si} \\ \epsilon\text{-greedy action w.r.t } Q_\theta^{student} & \text{if } I(s) < t_{si} \end{cases} \tag{2}$$

Agent takes the teacher suggested actions in 'important' states and acts autonomously in other states. This approach has two key advantages. Firstly, it ensures better online performance than ϵ-greedy or softmax policies because the student's exploration is guided by the teacher. Secondly, querying the teacher only in 'important' states reduces the communication overhead and ensures that the demonstration buffer gets filled with more informative demo experiences.

Algorithm 1 Data Gathering Policy

```
Get an initial observation s                          ▷ start of training
t_si = 0                                              ▷ threshold initialized to 0
list = {}                                             ▷ for storing I(s) values
for step = 1, ..., MAX_STEPS do
    I(s) = max_a Q_θ^student(s, a) − min_a Q_θ^student(s, a)
    if I(s) ≥ t_si then
        a = argmax_a Q^teacher(s, a)                  ▷ teacher demonstration
    else
        a = ε-greedy w.r.t Q_θ^student                ▷ autonomous execution
    list.append(I(s))
    t_si = average(list)                              ▷ update threshold
    take action a and get the next state s
    if step > learning_start and step % train_freq == 0 then
        sample a batch and update θ                   ▷ standard DQN update
        list = {}                                     ▷ discard previous I(s) values
        t_si = 0                                      ▷ reset the threshold to 0
```

$\pi_{student}$ defined in (2) ,assumes the availability of t_{si} even before the student can begin its training. Also, identifying such a fixed

threshold can be very challenging and often requires extensive experimentation for many environments. In order to address this, we propose a modified version of (2) in which we initialize $t_{si} = 0$ at the start of training and then we gradually update t_{si} as the student encounters more states. **Algorithm 1** explains our method and presents how it fits within the DQN algorithm.

Every time we update the parameters of $Q_\theta^{student}$ we — *(a)* discard the stored $I(s)$ values and *(b)* reset the threshold t_{si} to 0. At a particular step, the threshold is obtained by averaging the stored $I(s)$ values. This process ensures that for a given state both $I(s)$ and t_{si} are derived using the same Q-network(i.e. current $Q_\theta^{student}$).

4.2 Efficient Use of Demonstrations

Student uses two replay buffers to store the gathered experiences. If the agent takes an ϵ-greedy action, then the corresponding experience gets added into the *ϵ-greedy buffer* ($D^{\epsilon-greedy}$) and if the agent takes a teacher-suggested action, then that particular experience gets added into the *demonstration buffer* (D^{demo}). In our approach we consider the size of $D^{\epsilon-greedy}$ to be a hyperparameter and we do not impose any restriction on it. $D^{\epsilon-greedy}$ is exactly similar to the replay buffer used in DQN. If $D^{\epsilon-greedy}$ is full, the oldest experience is discarded to make space for the latest one. However we fix the size of D^{demo} to be equal to the maximum number of student-teacher interactions allowed (i.e. limited advice budget). Every time the student queries the teacher a demo experience gets generated and it gets stored into D^{demo}. When D^{demo} is full it implies that the student has exhausted the advice budget and the agent cannot ask for more demonstrations. To model the communication constraint we fix the size of D^{demo} to be significantly smaller than the total number of training steps.

The student agent samples batches of data to update the parameters of $Q_\theta^{student}$. A batch B contains experiences sampled from both $D^{\epsilon-greedy}$ and D^{demo}. A hyperparameter ρ , the demo ratio, controls the proportion of data coming from D^{demo} versus from $D^{\epsilon-greedy}$. For example, $\rho = 0.9$ would mean that 90% of the data in that particular batch comes from D^{demo} and the remaining 10% comes from $D^{\epsilon-greedy}$. Figure 1 explains how a training batch is prepared in *DQN Ask Important*. $Q_\theta^{student}$ gets updated with a mix of demonstration and self-generated data using temporal difference(TD) loss. The demo ratio(ρ) is a crucial hyperparameter which must be carefully tuned to achieve good performance.

Our goal is to make efficient use of the gathered demonstration data to accelerate student's learning process so that the agent - *(a)* has good initial performance and also *(b)* ends up solving the task after as few episodes as possible. We propose the following approach. We start with $\rho = \rho_{start}$ and then we linearly decay ρ as $Q_\theta^{student}$ receives more and more parameter updates until ρ becomes equal to ρ_{end}. Afterwards we keep $\rho = \rho_{end}$ for the subsequent parameter updates (see figure 2). We introduce another hyperparameter *demo fraction(f)* which determines the point at which ρ becomes equal to ρ_{end} for the first time. For $f = 0.5$ the corresponding point is $(N \times 0.5) = \frac{N}{2}$ where N is the total number of parameter updates of $Q_\theta^{student}$ network during the entire training period. We obtained best results for the following configuration - $\rho_{start} = 1$, $\rho_{end} = 0.1$ and $f = 0.5$.

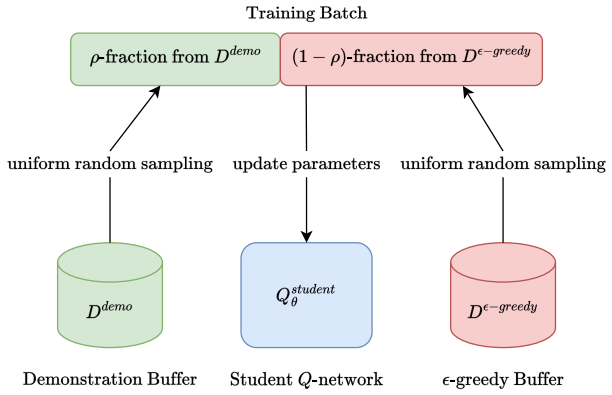


Figure 1: Batch creation process of DQN Ask Important

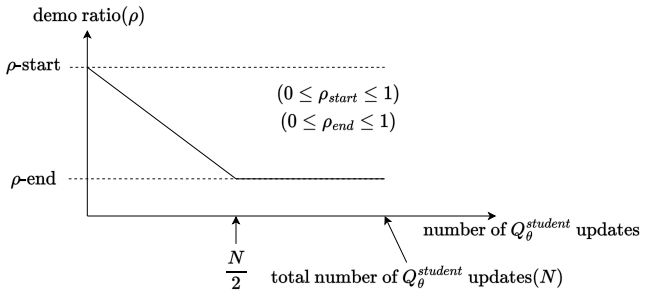


Figure 2: Linear decay of demo ratio(ρ) when $f = 0.5$

The main idea is that the *demo* experiences are more informative than the *ϵ-greedy* experiences and hence the agent can rapidly accelerate its initial performance by sampling more frequently from D^{demo} than from $D^{\epsilon-greedy}$. A high value of ρ_{start} ensures that the initial training batches contain demo experiences predominantly. However, the demonstration data necessarily covers a narrow part of the state space and does not contain any information about many state-action pairs which are relevant for learning a robust policy. Hence the agent must also use its self-generated data effectively. This problem can be addressed by using a low value of ρ_{end} as depicted in figure 2. A slow linear decay of ρ translates to gradual increase of *ϵ-greedy* experiences in the training batches. Once the student has exhausted its advice budget the quality of demonstration data does not improve further. However the quality of self-generated data keeps on improving as training progresses. Hence, the agent heavily relies on *ϵ-greedy* experiences during the later stages of training (after $\frac{N}{2}$ point in figure 2). **Algorithm 2** presents the pseudocode for *DQN Ask Important* algorithm.

5 EXPERIMENTS

In this section we describe the environments used for the evaluation, baselines, experimental setup and results.

Algorithm 2 DQN Ask Important

Inputs : Advice budget(K), Batch size(B), D^{demo} of size K, $D^{\epsilon\text{-greedy}}$ of size M, ρ_{start}, ρ_{end}, f
Initialize student's Q-network parameters θ
Initialize the target network parameters $\phi = \theta$
$t_{si} = 0$ ▷ state importance threshold initialized to 0
$list = \{\}$ ▷ for storing $I(s)$ values
Get an initial observation s ▷ start of training
for $step = 1, \ldots, MAX\_STEPS$ **do**
  $I(s) = \max_a Q_\theta^{student}(s,a) - \min_a Q_\theta^{student}(s,a)$
  if $I(s) \geq t_{si}$ **and** $D^{demo}.full == False$ **then**
    $a = \text{argmax}_a\, Q^{teacher}(s,a)$ ▷ teacher demonstration
    execute a and observe s' and r
    store (s, a, s', r) in D^{demo}
  else
    $a = \epsilon$-greedy w.r.t $Q_\theta^{student}$ ▷ autonomous execution
    execute a and observe s' and r
    store (s, a, s', r) in $D^{\epsilon\text{-greedy}}$
  $list.append(I(s))$
  $t_{si} = average(list)$ ▷ update threshold
  $s = s'$
  if $step > learning\_start$ **then**
    if $step\%train\_frequency == 0$ **then**
      calculate ρ by following the method shown in 2
      construct a batch of size B as shown in 1
      calculate loss using target network
      perform a gradient descent step to update θ
      $list = \{\}$ ▷ discard previous $I(s)$ values
      $t_{si} = 0$ ▷ reset the threshold to 0
    if $step\%target\_frequency == 0$ **then**
      $\phi = \theta$ ▷ update the target network

5.1 Environment Details

We use three different environments for our evaluation: *Acrobot-v1*, *MountainCar-v0* and *LunarLander-v2*. All of them are included in Gymnasium [24].

***Acrobot-v1* :** The acrobot [20] is a two-link robotic system with an actuator at the joint between the links. The system is characterized by 6 variables describing the sine and cosine of the two rotational joint angles and the joint angular velocities. The goal is to swing the end of the lower link up to a specified height. Actions involve applying +1, 0, or -1 torque on the actuator. The episode lasts for 500 steps or until the goal is achieved. A reward of -1 is given at each time step and the reward threshold is -100.

***MountainCar-v0* :** The Mountain Car MDP [12] features a car randomly positioned at the bottom of a sinusoidal valley. The objective is to strategically accelerate the car to generate sufficient momentum for reaching the flag positioned atop the right hill. The state is a two-dimensional vector, containing the car's position and velocity. The available actions are - accelerate to the left, don't accelerate, accelerate to the right. The agent receives a reward of -1 for each time step. An episode ends either when the car reaches the flag or the episode length is 200.

***LunarLander-v2* :** The agent controls a lunar lander that must perform a controlled descent and land on a landing pad. The state is represented by a 8-dimensional continuous vector containing the lander's position, velocity, angle, angular velocity, and two booleans that represent whether each leg is in contact with the ground or not. The agent has four discrete actions - do nothing, fire left orientation engine, fire main engine, fire right orientation engine. The agent receives positive rewards for successfully landing on the landing pad and negative rewards for various actions that deviate from the goal. An episode ends when the lander successfully lands or when the lander crashes or goes out of bounds. An episode is considered a solution if it scores at least 200 points.

5.2 Baselines

In this section we discuss the baseline and the ablation we use to compare against our *DQN Ask Important* approach in the experiments.

***DQN(No demonstrations)* :** DQN [11] is a simple and common baseline method for learning policies when we have environments with discrete action spaces. A DQN agent typically uses ϵ-greedy or softmax policy to gather experiences and stores the experiences into a replay buffer. The Q-network gets updated by using batches sampled from the buffer. Batches contain only ϵ-greedy experiences. This corresponds to $\rho_{start} = 0$ and $\rho_{end} = 0$ in figure 2.

***DQN(Demonstration in the first K steps)* :** In this ablation of our method, we consider that the student queries the teacher in all the first K states it encounters during training, where K is the advice budget. Thereafter the student uses ϵ-greedy policy to gather data. This ablation is very similar to the *early advising* approach proposed in [23]. Agent follows the method described in figure 1 to construct a training batch. Since the first K experiences are all demo experiences, $\rho = 1$ for all the training batches sampled before K training steps. Figure 3 explains how the value of ρ changes as training progresses.

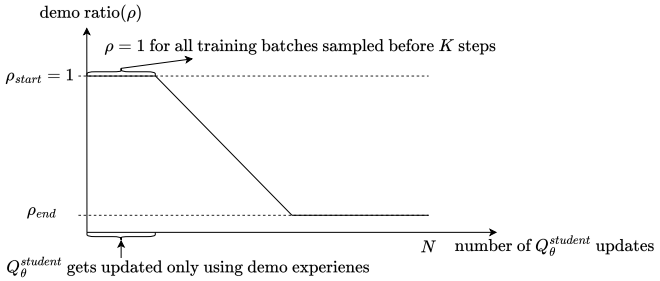
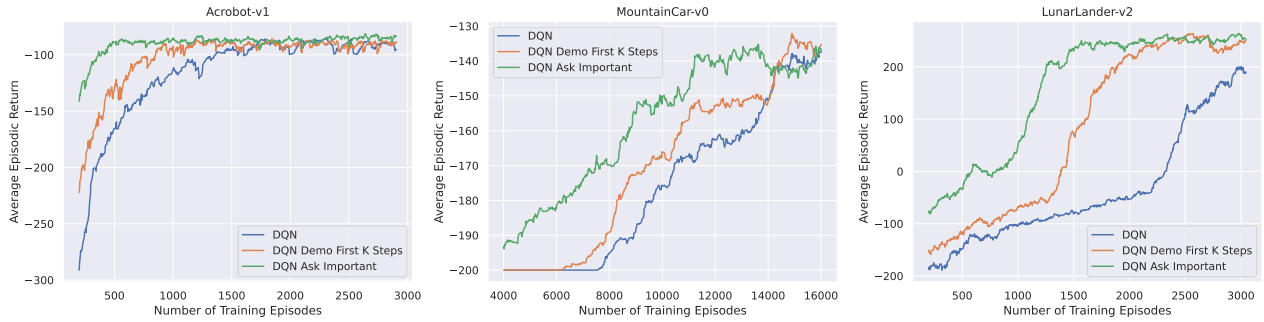


Figure 3: DQN(Demonstration in the first K steps)

5.3 Results

For a particular environment we obtained $Q^{teacher}$ using vanilla DQN algorithm. Acting greedily w.r.t $Q^{teacher}$ solves the task. During training of the student agent we utilized $Q^{teacher}$ to get teacher demonstrations. The networks $Q^{teacher}$, $Q_\theta^{student}$ and Q^{target} have the same architecture. For both *DQN Ask Important* and *DQN(demo first K steps)* we fixed the advice budget(K) to be equal to 10% of



(a) Acrobot-v1 **(b) MountainCar-v0** **(c) LunarLander-v2**

Figure 4: Average episodic return plots for the three environments (averaged over five runs)

Environment	Algorithm	Number of training steps(T)	Advice budget(K) = 10% of T	Target average episodic return(R)	Number of episodes needed to reach R
Acrobot-v1	DQN	350k	-	-100	≈1400
	DQN(demo first K steps)		35k		≈860
	DQN Ask Important		35k		**≈400**
MountainCar-v0	DQN	3 million	-	-140	≈15k
	DQN(demo first K steps)		300k		≈14.3k
	DQN Ask Important		300k		**≈11.2k**
LunarLander-v2	DQN	1 million	-	200	≈3000
	DQN(demo first K steps)		100k		≈1900
	DQN Ask Important		100k		**≈1300**

Table 1: Results for the three environments (averaged over five runs)

the total number of training steps. The size of D^{demo} is also K for both the algorithms. We have used $\rho_{start} = 1$, $\rho_{end} = 0.1$ and $f = 0.5$ to run our experiments. A small value of ρ_{end} ensures the use of demonstrations throughout the learning process alongside the utilization of new experiences.

Now we explain the process we have followed to obtain the average episodic return plots shown in figure 4. During a run of a particular algorithm, we evaluated the resulting policies after every training episode completion. For example, after 100 training episodes of DQN we have a policy π_{100}^{DQN}(i.e. greedy w.r.t current $Q_{\theta}^{student}$). Then we evaluate π_{100}^{DQN} by computing the average episodic return of 10 evaluation episodes. Thereafter we follow the same process to obtain the average episodic returns of π_{101}^{DQN}, π_{102}^{DQN} and so on till the last training episode. Thus we obtain an average episodic return plot for one seed. We average across five different runs to obtain average episodic return plot for DQN for a particular environment. We repeat the same process to obtain the average episodic return plots of *DQN Ask Important* and *DQN(demo first K steps)* for the same environment. Average episodic return plots of the three algorithms for the three environments are shown in figure 4.

It is evident from figure 4 and table 1 that *DQN Ask Important* has the best initial performance among the three methods. Also *DQN Ask Important* reaches the target average episodic return much faster than the two other methods for all the three environments. Better performance of *DQN Ask Important* and *DQN(demo first K steps)* compared to vanilla DQN indicate that demo experiences are more useful than ϵ-greedy experiences during the early stages of training. Effective utilization of demo experiences in the early phase results in 'good' initial performance of the agent.

Both *DQN Ask Important* and *DQN(demo first K steps)* use the gathered demonstrations efficiently by following the methods described in 2 and 3 respectively. However *DQN Ask Important* has much better performance than *DQN(demo first K steps)*. This empirically establishes the effectiveness of the data gathering policy used by *DQN Ask Important*. Even though both the algorithms end up collecting the same number of demo experiences(due to the fixed advice budget) the demo experiences gathered by *DQN Ask Important* are more informative for learning.

6 CONCLUSION

In this paper, we addressed the problem of inter-agent transfer learning in communication-constrained settings. Our proposed framework, *Ask Important – (a)* allows to model communication scarcity by imposing a strict limit on the maximum number of inter-agent interactions, *(b)* can be utilized by RL algorithms(which work with discrete action spaces and uses a replay buffer to store and sample experiences) such as DQN, Double DQN, Dueling DQN etc.

We discussed in detail how *Ask Important* can be integrated into an existing RL algorithm(by taking DQN as an example). We empirically show that the incorporation of *Ask Important – (a)* ensures much better utilization of the limited advice budget, *(b)* rapidly accelerates initial performance and allows students to solve the tasks in very few learning trials – for three Gymnasium environments.

REFERENCES

[1] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara Grosz. 2016. Interactive teaching strategies for agent training. In *In Proceedings of IJCAI 2016*.

[2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57, 5 (2009), 469–483.

[3] Sonia Chernova and Manuela Veloso. 2007. Confidence-based policy learning from demonstration using gaussian mixture models. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. 1–8.

[4] Jeffery Allen Clouse. 1996. *On integrating apprentice learning and reinforcement learning*. University of Massachusetts Amherst.

[5] Felipe Leno Da Silva, Matthew E Taylor, and Anna Helena Reali Costa. 2018. Autonomously Reusing Knowledge in Multiagent Reinforcement Learning.. In *IJCAI*. 5487–5493.

[6] Ionel-Alexandru Hosu and Traian Rebedea. 2016. Playing atari games with deep reinforcement learning and human checkpoint replay. *arXiv preprint arXiv:1607.05077* (2016).

[7] Kshitij Judah, Alan Paul Fern, Thomas G Dietterich, and Prasad Tadepalli. 2014. Active lmitation learning: formal and practical reductions to IID learning. *J. Mach. Learn. Res.* 15, 1 (2014), 3925–3963.

[8] Felipe Leno Da Silva, Garrett Warnell, Anna Helena Reali Costa, and Peter Stone. 2020. Agents Teaching Agents: A Survey on Inter-agent Transfer Learning. *Good Systems-Published Research* (2020).

[9] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17, 1 (2016), 1334–1373.

[10] Zachary C Lipton, Jianfeng Gao, Lihong Li, Xiujun Li, Faisal Ahmed, and Li Deng. 2016. Efficient exploration for dialog policy learning with deep BBQ networks\& replay buffer spiking. *CoRR abs/1608.05081* (2016).

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[12] Andrew William Moore. 1990. *Efficient memory-based learning for robot control*. Technical Report. University of Cambridge, Computer Laboratory.

[13] Shayegan Omidshafiei, Dong-Ki Kim, Miao Liu, Gerald Tesauro, Matthew Riemer, Christopher Amato, Murray Campbell, and Jonathan P How. 2019. Learning to teach in cooperative multiagent reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 6128–6136.

[14] Tom Le Paine, Caglar Gulcehre, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, et al. 2019. Making efficient use of demonstrations to solve hard exploration problems. *arXiv preprint arXiv:1909.01387* (2019).

[15] Benjamin Rosman and Subramanian Ramamoorthy. 2014. Giving advice to agents with hidden goals. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1959–1964.

[16] Stefan Schaal. 1996. Learning from demonstration. *Advances in neural information processing systems* 9 (1996).

[17] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[18] Peter Stone and Manuela Veloso. 1999. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence* 110, 2 (1999), 241–273.

[19] Kaushik Subramanian, Charles L Isbell Jr, and Andrea L Thomaz. 2016. Exploration from demonstration for interactive reinforcement learning. In *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*. 447–456.

[20] Richard S Sutton. 1995. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems* 8 (1995).

[21] Richard S Sutton, Andrew G Barto, et al. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

[22] Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. 2011. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. 617–624.

[23] Lisa Torrey and Matthew Taylor. 2013. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. 1053–1060.

[24] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. 2023. Gymnasium. https://doi.org/10.5281/zenodo.8127026

[25] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. 2017. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817* (2017).

[26] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8 (1992), 279–292.