Data-Efficient Hierarchical Reinforcement Learning Using Importance Sampling

Seyed Alireza Bakhtiari
s.alireza.bakhtiari.9@gmail.com
Matin Moezzi
matin.moezzi@mail.utoronto.ca

Abstract

Learning goal-directed behavior in environments with sparse feedback is a major challenge for reinforcement learning algorithms. The primary difficulty arises due to insufficient exploration, resulting in an agent being unable to learn robust value functions. Intrinsically motivated agents can explore new behavior for its own sake rather than to directly solve problems. Such intrinsic behaviors could eventually help the agent solve tasks posed by the environment. We propose an approach to improve data-efficiency of hierarchical-DQN (h-DQN) using the importance sampling method. A top-level value function learns a policy over intrinsic goals, and a lower-level function learns a policy over atomic actions to satisfy the given goals. h-DQN allows for flexible goal specifications, such as functions over entities and relations. This provides an efficient space for exploration in complicated environments. We demonstrate the strength of our approach on one problem with very sparse, delayed feedback: a complex discrete stochastic decision process.

I. Introduction

Hierarchical reinforcement learning (HRL) is a promising approach to extend traditional reinforcement learning (RL) methods to solve more complex tasks. Hierarchical reinforcement learning (HRL), in which multiple layers of policies are trained to perform decision-making and control at successively higher levels of temporal and behavioral abstraction, has long held the promise to learn such difficult tasks. By having a hierarchy of policies, of which only the lowest applies actions to the environment, one is able to train the higher levels to plan over a longer time scale. Moreover, if the high-level actions correspond to semantically different low-level behavior, standard exploration techniques may be applied to more appropriately explore a complex environment. Still, there is a large gap between the basic definition of HRL and the promise it holds to successfully solve complex environments.

Reinforcement learning (RL) formalizes control problems as finding a policy π that maximizes expected future rewards [4]. Value functions $V(s)$ are central to RL, and they cache the utility of any state s in achieving the agent's overall objective. Recently, value functions have also been generalized as $V(s, g)$ in order to represent the utility of state s for achieving a given goal $g \in G$ [5], [3]. When the environment provides delayed rewards, we adopt a strategy to first learn ways to achieve intrinsically generated goals, and subsequently learn an optimal policy to chain them together. Each of the value functions $V(s, g)$ can be used to generate a policy that terminates when the agent reaches the goal state g. A collection of these policies can be hierarchically arranged with temporal dynamics for learning or planning within the framework of semi-Markov decision processes [6], [7]. In high-dimensional problems, these value functions can be approximated by neural networks as $V(s, g; \theta)$.

We propose a framework with hierarchically organized deep reinforcement learning modules working at different time-scales. The model takes decisions over two levels of hierarchy – (a) the top level module (meta-controller) takes in the state and picks a new goal, (b) the lower-level module (controller) uses both the state and the chosen goal to select actions either until the goal is reached or the episode is terminated. The meta-controller then chooses another goal and steps (a-b) repeat. We train our model using stochastic gradient descent at different temporal scales to optimize expected future intrinsic (controller) and extrinsic rewards (meta-controller). We improve the classical H-DQN framework sample efficiency by using the controller experiences to train and enhance the meta-controller's policy. We demonstrate the strength of our approach on problems with long-range delayed feedback with a discrete stochastic decision process with a long chain of states before receiving optimal extrinsic rewards.

II. Problem Statement

A. Discrete stochastic decision process

We consider a stochastic decision process where the extrinsic reward depends on the history of visited states in addition to the current state. We selected this task in order to demonstrate the importance of intrinsic motivation for exploration in such environments.

There are 6 possible states and the agent always starts at s_2. The agent moves left deterministically when it chooses a left action; but the action right only succeeds 50% of the time, resulting in a left move otherwise. The terminal state is s_1 and the agent receives the

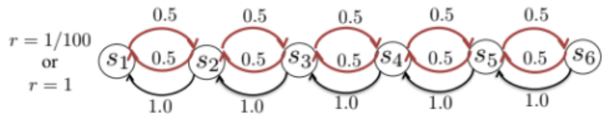


Fig. 1: A stochastic decision process where the reward at the terminal state s1 depends on whether s6 is visited (r = 1) or not (r = 1/100).

reward of 1 when it first visits s_6 and then s_1. The reward for going
to s_1 without visiting s_6 is 0.01. This is a modified version of the MDP in [2], with the reward structure adding complexity to the task. The process is illustrated in Figure 1.

We consider each state as a possible goal for exploration. This encourages the agent to visit state s_6 (whenever it is chosen as a goal) and hence, learn the optimal policy. For each goal, the agent receives a positive intrinsic reward if and only if it reaches the corresponding state.

We have trained classical H-DQN on the mentioned environment. During the investigation of the learning process of the Hierarchical DQN, poor convergence for the meta-controller action value function was apparent. Meta-controller action value functions rarely differ in the training time, resulting in inaccurate estimates for different goals over time. This could be shown according to figure 2. Each cell of each table represents an action value function according to a state and a potential goal for the meta-controller. As we can see, action-value functions for the meta-controller, which are responsible for assigning credits to goals in each state, have failed to assign credits to states on the right side of MDP. However, we expected high value for these goals, as they are the only states near the target state, s_6. Therefore, the performance of the meta-controller during the first 3000 episodes is not satisfying.

On the other hand, the controller managed to achieve a relatively accurate Q function in a shorter time period. This observation could be justified by looking at the number of training samples for each agent. While the controller gets one sample experience for each interaction with the environment, the meta-controller only gets training samples when the controller achieves a goal or the episode is terminated. It is obvious that this way, the meta-controller gets fewer training samples per episode; therefore, it cannot achieve an accurate estimate for the value function in the same time as the controller.

Each row corresponds to a state in the Long-Corridor environment, and each column corresponds to a potential goal for the meta-controller to choose in each state. Red squares imply negative values for the corresponding state and goal and green squares represent positive action values. These figures show that, Q function of each state rarely changes during training and mostly depends on initialization values. This is the case because in the Long-Corridor environment, the value of selecting S6 as the goal state has to be higher than the other states as it is the state with the most extrinsic reward.

To address this problem, we propose a new approach that employs importance sampling for deriving new experiences for the meta-controller. By doing so, the meta-controller will have a richer and larger experience replay buffer to use for training; Therefore, it can learn much faster and also achieve better results.

III. Model

Consider a Markov decision process (MDP) represented by states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, and transition function $\mathcal{T} : (s, a) \to s'$. An agent operating in this framework receives a state s from the external environment and can take an action a, which results in a new state s'. We define the extrinsic reward function as $\mathcal{F} : (s) \to \mathbb{R}$. The objective of the agent is to maximize this function over long periods of time. For example, this function can take the form of the agent's survival time or score in a game.

A. Agents

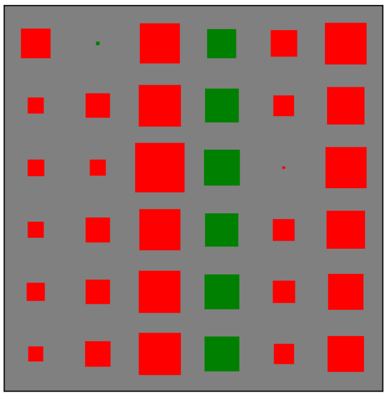
Effective exploration in MDPs is a significant challenge in learning good control policies. Methods such as ϵ-greedy are useful for local exploration but fail to provide impetus for the agent to explore different areas of the state space. In order to tackle this, we utilize a notion of goals$g \in \mathcal{G}$, which provide intrinsic motivation for the agent. The agent focuses on setting and achieving sequences of goals in order to maximize cumulative extrinsic reward.

We use the temporal abstraction of *options* [6] to define policies π_g for each goal g. The agent learns these option policies simultaneously along with learning the optimal sequence of goals to follow. In order to learn each π_g, the agent also has a critic, which provides *intrinsic rewards*, based on whether the agent is able to achieve its goals (see Figure 3).

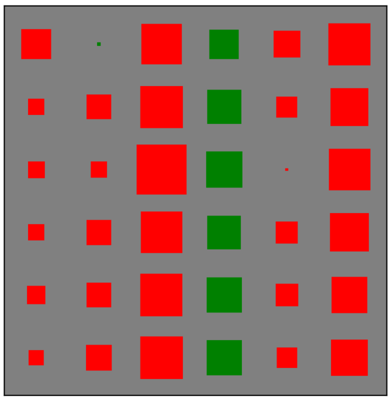
B. Temporal Abstractions

As shown in Figure 3, the agent uses a two-stage hierarchy consisting of a controller and a meta-controller. The meta-controller receives state st and chooses a goal $g_t \in \mathcal{G}$, where $\mathcal{G}$ denotes the set of all possible current goals. The controller then selects an action at using s_t and g_t. The goal g_t remains in place for the next few time steps either until it is achieved or a terminal state is reached. The internal critic is responsible for evaluating whether a goal has been reached and providing an appropriate reward $r_t(g)$ to the controller. The objective function for the controller is to maximize cumulative intrinsic reward: $R_t(g) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(g)$. Similarly, the objective of the meta-controller is to optimize the cumulative extrinsic reward $F_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} f_{t'}$, where f_t are reward signals received from the environment.

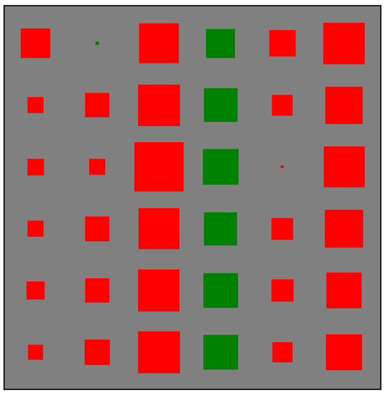
One can also view this setup as similar to optimizing over the space of optimal reward functions to maximize fitness [39]. In our case, the reward functions are dynamic and temporally dependent on the sequential history of goals. Figure 3 provides an illustration of the agent's use of the hierarchy over subsequent time steps.



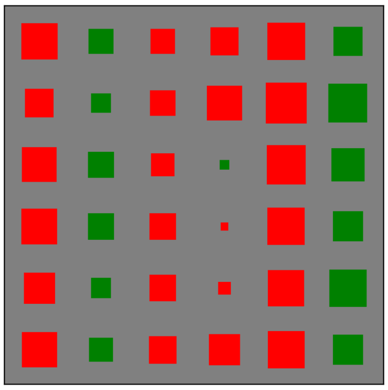
(a)



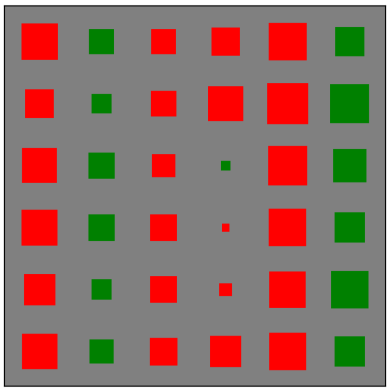
(b)



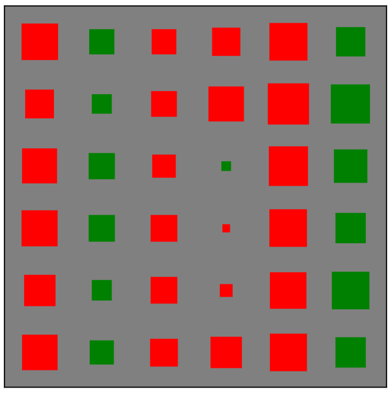
(c)



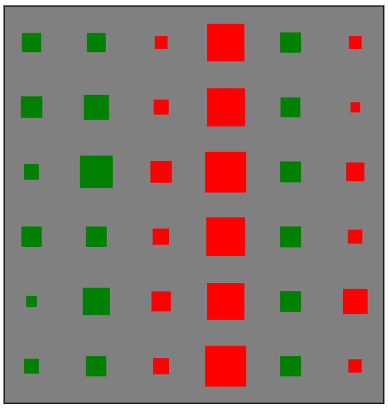
(d)



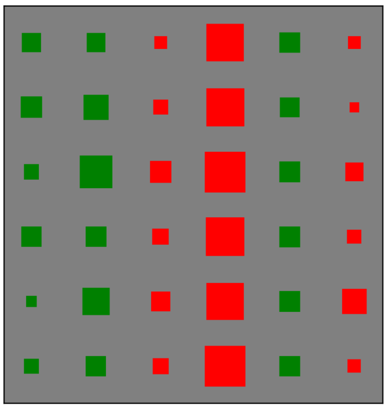
(e)



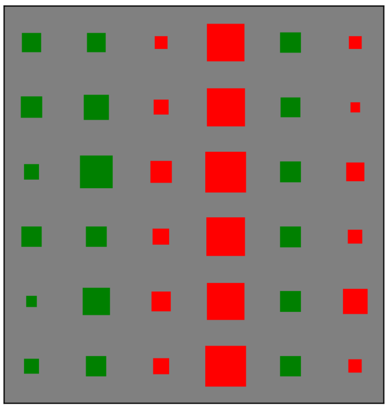
(f)



(g)

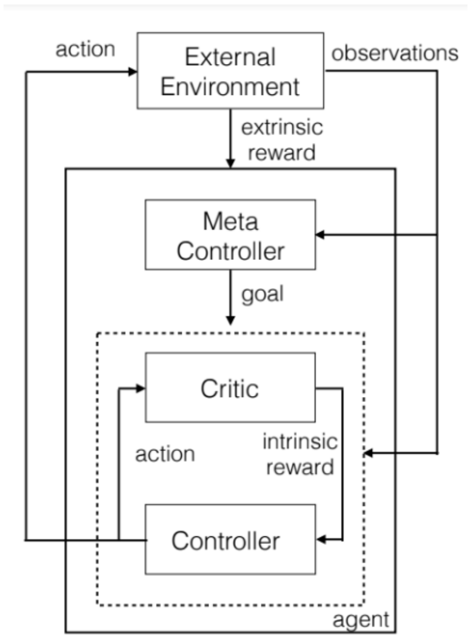


(h)



(i)

Fig. 2: The hinton diagram of the discrete stochastic decision process. Each row represents a state and each column represents a goal. Each cell represents action-value function of meta-controller regarding to states and goals. The green color depicts positive values and the red color depicts negative values. Furthermore, each row illustrates the training procedure after 1000 episodes of training



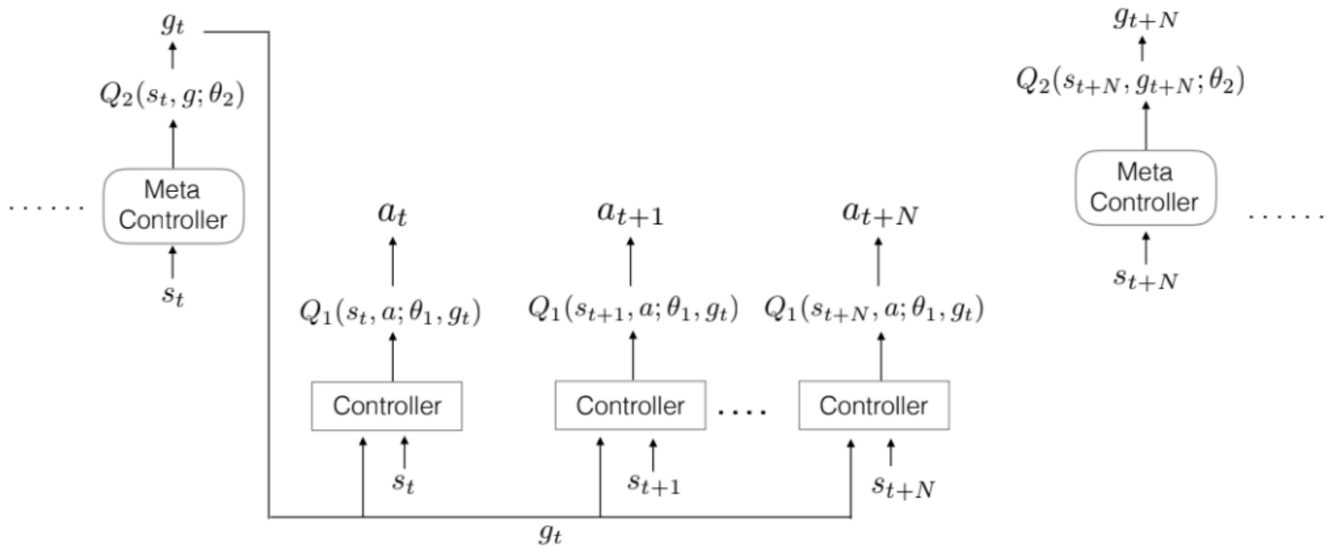


Fig. 3: **Overview:** The agent produces actions and receives sensory observations. Separate deep-Q networks are used inside the meta-controller and controller. The meta-controller that looks at the raw states and produces a policy over goals by estimating the value function $Q_2(s_t, g_t; \theta_2)$ (by maximizing expected future extrinsic reward). The controller takes in states and the current goal, and produces a policy over actions by estimating the value function $Q_2(s_t, a_t; \theta_1, g_t)$ to solve the predicted goal (by maximizing expected future intrinsic reward). The internal critic checks if goal is reached and provides an appropriate intrinsic reward to the controller. The controller terminates either when the episode ends or when g is accomplished. The meta-controller then chooses a new g and the process repeats.

C. Deep Reinforcement Learning with Temporal Abstractions

We use the Deep Q-Learning framework [1] to learn policies for both the controller and the meta-controller. Specifically, the controller estimates the following Q-value function:

$$\begin{aligned} Q_1^*(s,a;g) &= \max_{\pi_{ag}} \mathrm{E}\left[\sum_{t'=t} \gamma^{t'-t} r_{t'} \mid s_t = s, a_t = a, g_t = g, \pi_{ag}\right] \\ &= \max_{\pi_{ag}} \mathrm{E}\left[r_t + \gamma \max_{a_{t+1}} Q_1^*\left(s_{t+1}, a_{t+1}; g\right) \mid s_t = s, a_t = a, g_t = g, \pi_{ag}\right] \end{aligned} \tag{1}$$

where g is the agent's goal in state s and $\pi_{ag} = P(a|s,g)$ is the action policy. Similarly, for the meta-controller, we have:

$$Q_2^*(s,g) = \max_{\pi_g} \mathrm{E}\left[\sum_{t'=t}^{t+N} f_{t'} + \gamma \max_{g'} Q_2^*\left(s_{t+N}, g'\right) \mid s_t = s, g_t = g, \pi_g\right] \tag{2}$$

where N denotes the number of time steps until the controller halts given the current goal, g' is the agent's goal in state s_{t+N}, and $\pi_g = P(g|s)$ is the policy over goals. It is important to note that the transitions (s_t, g_t, f_t, s_{t+N}) generated by Q_2 run at a slower time-scale than the transitions $(s_t, a_t, g_t, r_t, s_{t+1})$ generated by Q_1.

We can represent $Q^\star(s,g) \approx Q(s,g;\theta)$ using a non-linear function approximator with parameters θ, called a deep Q-network (DQN). Each $Q \in Q_1, Q_2$ can be trained by minimizing corresponding loss functions – $L_1(\theta_1)$ and $L_2(\theta_2)$. We store experiences (s_t, g_t, f_t, s_{t+N}) for Q_2 and $(s_t, a_t, g_t, r_t, s_{t+1})$ for Q_1 in disjoint memory spaces D_1 and D_2 respectively. The loss function for Q_1 can then be stated as:

$$L_1\left(\theta_{1,i}\right) = \mathrm{E}_{(s,a,g,r,s')\sim D_1}\left[\left(y_{1,i} - Q_1\left(s,a;\theta_{1,i},g\right)\right)^2\right] \tag{3}$$

where i denotes the training iteration number and $y_{1,i} = r + \gamma max_{a'} Q_1(s', a'; \theta_{1,i-1}, g)$.
Following [28], the parameters $\theta_{1,i-1}$ from the previous iteration are held fixed when optimising the loss function. The parameters θ_1 can be optimized using the gradient:

$$\begin{aligned} &\nabla_{\theta_{1,i}} L_1\left(\theta_{1,i}\right) \\ &\quad = \mathrm{E}_{(s,a,r,s'\sim D_1)}\left[\left(r + \gamma \max_{a'} Q_1\left(s', a'; \theta_{1,i-1}, g\right) - Q_1\left(s,a;\theta_{1,i},g\right)\right) \nabla_{\theta_{1,i}} Q_1\left(s,a;\theta_{1,i},g\right)\right)\right] \end{aligned} \tag{4}$$

The loss function L_2 and its gradients can be derived using a similar procedure.

D. Learning Algorithm

We learn the parameters of h-DQN using stochastic gradient descent at different time scales – experiences (or transitions) from the controller are collected at every time step but experiences from meta-controller are only collected when the controller terminates (i.e. when a goal is re-picked or the episode ends). Each new goal g is drawn in an ϵ-greedy fashion with the exploration probability ϵ_2 annealed as learning proceeds (from a starting value of 1).

In the controller, at every time step, an action is drawn with a goal using the exploration probability $\epsilon_{1,g}$ which is dependent on the current empirical success rate of reaching g. The model parameters (θ_1, θ_2) are periodically updated by drawing experiences from replay memories D_1 and D_2), respectively.

E. Enriching the meta-controller's experience buffer using Importance Sampling

However, in most cases, the controller meets many other goals during its way to the given goal. For instance, in the Long Corridor environment (Fig. 2), the controller will pass all of the states in its way from s_1 to s_6. But only the tuple $(s_1, s_6, \text{Cumulative\_Reward}, g_6)$ will be saved in the meta-controller experience buffer.

Suppose that goal g^m has been met during the controller trajectory from s_T to g_T, which represents the meta-controller state at time step T and the meta-controller goal at time step T, respectively.
The trajectory of the controller can be represented like this:

$$\tau(T) = (s_t = s_T, a_t, s_{t+1}, a_{t+1}, \ldots, s_{t+i} = g^m, a_{t+i}, \ldots, s_{t+N} = g_T) \tag{5}$$

The experience tuple (state: S_t, next_state: s_{t+N}, reward: Cumulative_reward, action: g_t) will be stored in the meta-controller experience buffer based on the above controller trajectory. But, it is possible to use the trajectory from s_t to s_{t+i}, which is g^m, to create a new experience for the meta-controller (Figure 4).

The mentioned trajectory (Eq. 5) has been acquired by following the policy $\pi_{controller}(.|s_t; g_T)$ by the controller agent, which takes g_T given by the meta-controller as the parameter to the policy function. As a result, we cannot directly use the reward of this trajectory in a new experience tuple with the g^m as the goal state, because choosing g^m as the goal state

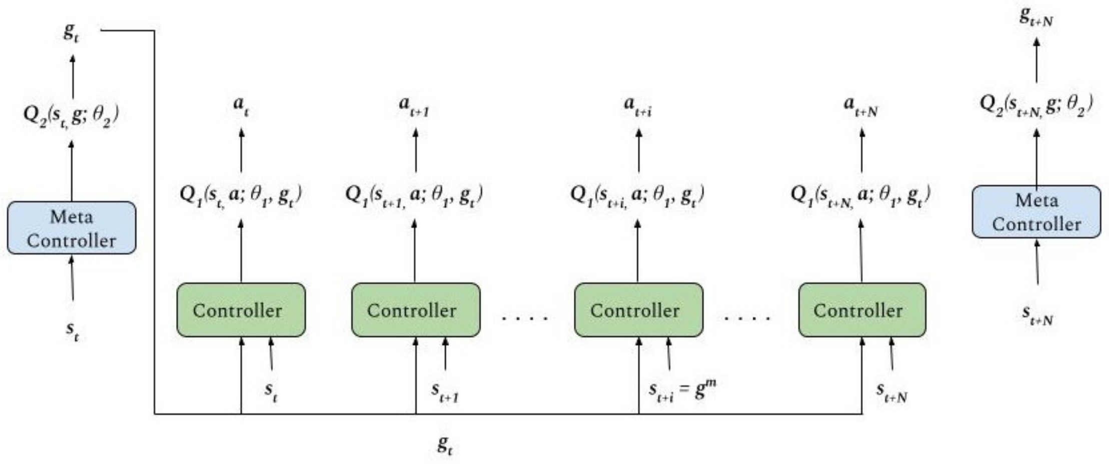


Fig. 4: The process of deriving a new experience based on a goal during the contoller's training (g^m) with the given goal (g_t)

would have changed the controller policy, from $\pi_{controller}(.|s_t; g_T)$ to the new policy $\pi_{controller}(.|s_t; g^m)$, resulting in a new trajectory from s_t to g^m.

In order to address this issue, we can use importance sampling to acquire the estimation of the reward of following the policy $\pi_{controller}(.|s_t; g^m)$ from the sample that has been obtained from following the policy $\pi_{controller}(.|s_t; g_T)$. To be accurate, this method estimates the reward of one policy based on the sample of the another policy, which is the classical usage of importance sampling. To be specific, the probability of confronting the path $s_t, a_t, s_{t+1}, \ldots, s_{t+i}$ by following the policy $\pi_{controller}(.|s_t; g^m)$ is as follows:

$$\begin{aligned} & Pr\left\{a_t, s_{t+1}, a_{t+1}, \ldots, s_{t+i} = g^m \mid s_t, a_{t:t+i} \sim \pi_{controller}(. \mid s_t; g^m)\right\} \\ & = \pi_{controller}(a_t|s_t; g^m) P(s_{t+1}|s_t, a_t) \pi_{controller}(a_{t+1}|s_{t+1}; g^m) \ldots P(s_{t+i}|s_{t+i-1}, a_{t+i-1}) \\ & = \prod_{k=t}^{t+i-1} \pi_{controller}(a_k|s_k; g^m) P(s_{k+1}|s_k, a_k) \end{aligned} \tag{6}$$

where P here is the state-transition probability function.

Thus, the relative probability of the trajectory under the target and behavior policies (the importance-sampling ratio) is:

$$\rho_{t:t+i} = \frac{\prod_{k=t}^{t+i-1} \pi_{controller}\left(a_k \mid s_k; g^m\right)}{\prod_{k=t}^{t+i-1} \pi_{controller}\left(a_k \mid s_k; g_T\right)} \tag{7}$$

Although the trajectory probabilities depend on the MDP's transition probabilities, which are generally unknown, they appear identically in both the numerator and denominator, and thus cancel. The importance sampling ratio ends up depending only on the two policies and the sequence, not on the MDP.

Recall that we wish to estimate the expected returns (values) under the target policy, $\pi_{controller}(.|s_t; g^m)$, but all we have are returns g_T due to the behavior policy $\pi_{controller}(.|s_t; g_T)$. These returns have the wrong expectation $\mathbb{E}[g_t|s_t = s] = V_{controller}(s; g_T)$ and so cannot be averaged to obtain $V_{controller}(s; g^m)$. This is where importance sampling comes in. The ratio $\rho_{t:t+i}$ transforms the returns to have the right expected value:

$$\mathbb{E}[g_t \rho_{t:t+i} \mid s_t = s] = V_{controller}(s, g^m) \tag{8}$$

By doing so, we can now add new experience with corresponding goal state, g^m, and action value as equation (Eq. 8) to the experience replay buffer.

IV. Experiments

We compare the performance of our approach with two baselines. The first baseline is DQN and the other one is h-DQN. In our experiments, all ϵ parameters are annealed from 1 to 0.1 over 50,000 steps. The learning rate is set to 0.00025. Figure 5 plots the rolling episodes scores for all methods in each episode. We see that our approach, h-DQN using importance sampling, outperforms both baselines.

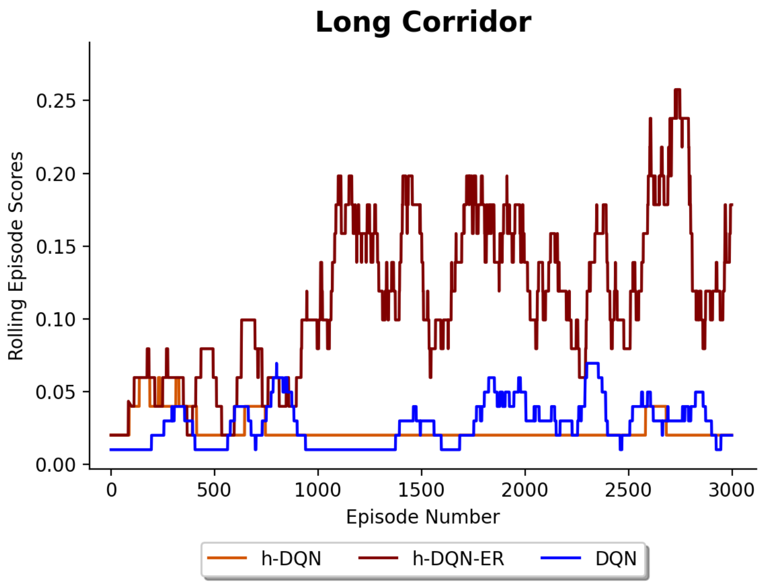


Fig. 5: h-DQN-ER represents the h-DQN approach improved by importance sampling. The data-efficient h-DQN outperforms both DQN and h-DQN.

V. Conclusion

We presented an approach to improve the sample efficiency of h-DQN using importance sampling. Our method generates experiences for the meta-controller during the controller's training using trajectories that the controller has followed and obtained rewards. To be specific, The proposed approach approximates a new experience for meta-controller from the controller trajectories using importance sampling. In fact, importance sampling enables us to derive experiences for the meta-controller based on other goals during training with the given goal. We also observe that our approach outperforms both DQN and h-DQN without improvements in the discrete stochastic decision process environment (Fig. 1).

Suppose that the controller follows the following trajectory from s_t to $s_N = g_T$:

$$s_t, a_t, s_{t+1}, a_{t+1}, \dots, s_{t+i} = g^m, \dots, s_N$$

where g_T is the chosen goal by the meta-controller and $g_m \neq g_T$ is in the set of all possible goals. This trajectory is followed under $\pi_{controller}(.|s_t; g_T)$. The cumulative reward obtained by this trajectory is $R_t(g_T) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(g_T)$.
We wish to estimate the expected returns under $\pi_{controller}(.|s_t; g_m)$. Using the importance sampling ratio (ρ) we can estimate both the expected returns and value function under this policy $\pi_{controller}(.|s_t; g_m)$.

$$\begin{aligned} R_t(g_m) &= \rho . R_t(g_T) \\ v_{\pi_m} &= \mathbb{E}[\rho . R_t(g_m) | s_t] \end{aligned}$$

where $\pi_m = \pi_{controller}(.|s_t; g_m)$

References

[1] 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. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[2] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29:4026–4034, 2016.

[3] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR.

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

[5] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

[6] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[7] Hengshuai Yao, Csaba Szepesvári, Rich Sutton, Joseph Modayil, and Shalabh Bhatnagar. Universal option models. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 990–998, Cambridge, MA, USA, 2014. MIT Press.