C-LEARNING: HORIZON-AWARE CUMULATIVE ACCESSIBILITY ESTIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

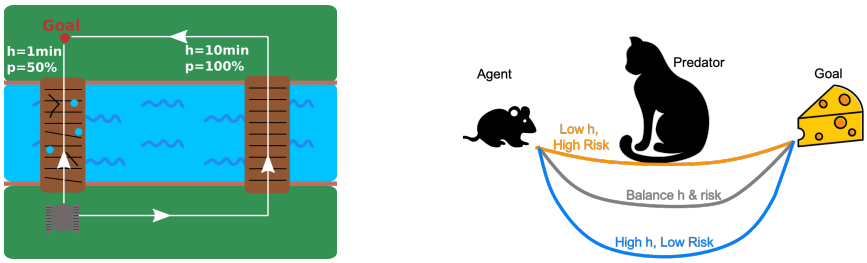
Multi-goal reaching is an important problem in reinforcement learning needed to achieve algorithmic generalization. Existing methods suffer from the fundamental limitation of learning a single way to reach each goal from each state. This does not allow for any trade-off between reaching the goal quickly and reaching it reliably. We introduce the concept of *cumulative accessibility functions* to address this limitation. We show that these functions obey a recurrence relation which enables learning from offline interactions, and that they allow us to balance the speed / reliability trade-off at *test time*. In addition, these functions are monotonic in the planning horizon which decreases the complexity of the learning task. We propose a specially tailored replay buffer that allows efficient learning, validate our approach in a variety of experiments, and demonstrate that it allows the speed / reliability trade-off while outperforming state-of-the-art goal reaching algorithms in planning tasks. Additional visualizations can be found at `https://sites.google.com/view/learning-cae/`

1 INTRODUCTION

In a typical reinforcement learning (RL) problem an agent is trained to maximize the cumulative discounted return of a single reward function, where the discounting factor controls the preference for achieving these rewards earlier in time. Here we consider a more general problem. We wish to train an agent to reach one of several *goals*, of which we select one per episode at test time. Further, the agent must reach this goal within some limited time *horizon*, which is also selected at test time. Instead of treating these problems independently, we train a single agent to generalize over this goal-horizon space.

Multi-goal agents have been studied previously (Kaelbling, 1993; Schaul et al., 2015; Andrychowicz et al., 2017; Ghosh et al., 2019), and are a special case of reinforcement learning (the goal is concatenated onto the state, the reward function is one at the goal and zero elsewhere, and the environment terminates once the goal is reached). In addition to adding test-time flexibility, goal-reaching has been shown to be more sample efficient (Andrychowicz et al., 2017; Ghosh et al., 2019). However, a deficiency in current goal-reaching approaches is that they learn a single policy $\pi(a|s, g)$ over actions a to reach a goal g from state s. There is an unexplored trade-off between reaching the goal reliably and reaching it quickly. Here we will consider 'quickly' to mean within some fixed and finite horizon h. For example, in figure 1a, a robot commanded to reach the other side of a river within 1 minute may have to drive over a dangerous, poorly maintained bridge to have any hope of success. However, given 10 minutes to reach the other side, it may be preferable to drive for 5 minutes along the river bank to a safer bridge, and cross there. The solution to this example is bimodal, but we can envisage problems with a continuous solution space. In figure 1b we sketch an environment in which an agent must reach a goal on the opposite side of some predator. Faster paths pass closer to the predator and incur a greater probability of being eaten and incurring a catastrophic penalty. Current goal-reaching approaches learn only one of the illustrated modes, determined at training time by the reward discounting factor. This does not allow the speed / reliability trade-off to be selected at test-time.

We propose to address this limitation by learning horizon-aware policies $\pi(a|s, g, h)$, which should be followed to reach goal g from state s in at most h steps. The introduction of a time horizon h naturally allows us to tune this trade-off, as an agent wishing to reach the goal faster should select



(a) Two bridges allow the robot to reach its goal on the far side of the river. The left one is in poor repair and offers a faster, but riskier, path.

(b) A continuous spectrum of paths allow the mouse to reach its goal faster, at an increased risk of disturbing the cat and being eaten.

Figure 1: **Variable (finite)-Horizon Bang for Buck problem**. Environments wherein a Speed-Reliability trade-off results in multimodal policies. The paper introduces the framework of Cumulative Accessibility Functions for horizon-aware adaptive behavior to trade-off speed with reliability at test time without retraining.

a policy with a suitably small h value. Conversely, an agent wishing to reach the goal reliably should select a large h value. In order to carry out learning, we introduce the *optimal cumulative accessibility function* $C^*(s, a, g, h)$. This is a generalization of the state-action value function, and corresponds to the probability of reaching goal g from state s after at most h steps if action a is taken, and the agent acts optimally thereafter. Intuitively it is similar to the optimal Q-function, but Q-functions rarely correspond to probabilities. The C^* function does by construction. We derive Bellman backup update rules for C^* which allow it to be learned, a method we call cumulative accessibility estimation, or C-learning. The policies $\pi(a|s, g, h)$ can then be recovered from the C^* function.

We also show, surprisingly, that learning horizon-aware policies which allow the speed / reliability trade-off does not come at the price of otherwise reduced performance. On the contrary, thanks to our proposed replay buffer, we find improved empirical results even in environments where this trade-off is not present.

In addition to providing the speed / reliability trade-off, learning C^* is itself useful, containing information that the policies do not. It tells us whether a goal g is reachable from the current state s within h steps, whereas $\pi(a|s, g, h)$ will always return some action, even for unreachable goals. This naturally lends itself to incorporation into other frameworks, for example, in goal setting for hierarchical RL tasks (Nachum et al., 2018), or safety-critical environments requiring adaptation (Peng et al., 2018; Zhang et al., 2020).

Summary of Contributions: (i) highlighting the importance of the speed vs reliability trade-off in fixed-finite horizon RL, (ii) introducing C-functions and cumulative accessibility estimation, (iii) introducing a novel replay buffer specially tailored for learning C^*, and (iv) empirically showing the effectiveness of our method for goal-reaching as compared to existing alternatives.

2 Background and Related Work

Let us extend the Markov Decision Process (MDP) formalism (Sutton et al., 1998) for goal-reaching. We consider a set of actions $\mathcal{A}$, a state space $\mathcal{S}$ and a goal set $\mathcal{G}$. We assume access to a goal checking function $G : \mathcal{S} \times \mathcal{G} \to \{0, 1\}$ such that $G(s, g) = 1$ if and only if state s achieves goal g. For example, achieving the goal could mean exactly reaching a certain state, in which case $\mathcal{G} = \mathcal{S}$ and $G(s, g) = \mathbb{1}(s = g)$. For many continuous state-spaces hitting a state exactly has zero probability. Here we can still take $\mathcal{G} = \mathcal{S}$, but take $G(s, g) = \mathbb{1}(d(s, g) \leq \epsilon)$ for some radius ϵ and metric d. More general choices are possible. In the Dubins' Car environment the state consists of both the location and orientation of the car: $\mathcal{S} = \mathbb{R}^2 \times S^1$. We take $\mathcal{G} = \mathbb{R}^2$, and $G(s, g)$ checks that the location of the car is within some small radius of g, ignoring the direction entirely. For fixed g, $G(s, g)$ can be thought of as a sparse reward function.

In the goal-reaching setting, a policy $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ denotes the set of distributions over $\mathcal{A}$, maps state-goal pairs to an action distribution. The environment dynamics are given by a starting distribution $p(s_0, g)$, usually taken as $p(s_0)p(g)$, and transition probabilities $p(s_{t+1}|s_t, a_t)$.

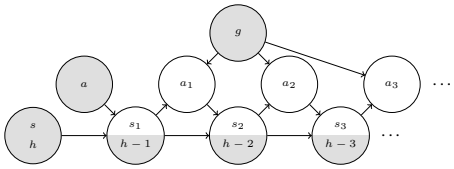


Figure 2: Graphical model depicting trajectories from $\mathbb{P}_{\pi(\cdot|\cdot,g,h)}(\cdot|s_0 = s, a_0 = a)$. Gray nodes denote fixed values, and white nodes stochastic ones. Nodes a, g and s are non-stochastic simply because they are conditioned on, not because they are always fixed within the environment. Note that the values of h decrease deterministically. Nodes corresponding to horizons could be separated from states, but are not for a more concise graph.

Q-Learning: The Q-function (Watkins & Dayan, 1992) $Q^\pi : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ is defined by $Q^\pi(s_t, g, a_t) = \mathbb{E}_\pi[\sum_{i=t}^{\infty} \gamma^{i-t} G(s_t, g)|s_t, a_t]$, where $\gamma \in [0, 1]$ is a discount factor and the expectation is with respect to state-action trajectories obtained by using $\pi(a_i|s_i, g)$. If π^* is an optimal policy in the sense that $Q^{\pi^*}(s, g, a) \geq Q^\pi(s, g, a)$ for every π and $(s, g, a) \in \mathcal{S} \times \mathcal{G} \times \mathcal{A}$, then Q^{π^*} matches the optimal Q-function, Q^*, which obeys the Bellman equation:

$$Q^*(s, g, a) = \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[G(s, g) + \gamma \max_{a' \in \mathcal{A}} Q^*(s', g, a') \right]. \tag{1}$$

In deep Q-learning (Mnih et al., 2015), Q^* is parameterized with a neural network and learning is achieved by enforcing the relationship from equation 1. This is done by minimizing $\mathbb{E}[\mathcal{L}(Q^*(s_i, g_i, a_i), y_i)]$, where y_i corresponds to the expectation in equation 1 and is estimated using a replay buffer of stored tuples (s_i, a_i, g_i, s_{i+1}). Typically $\mathcal{L}$ is chosen as a squared error loss, and the dependency of y_i on Q^* is ignored for backpropagation in order to stabilize training. Once Q^* is learned, the optimal policy is recovered by $\pi^*(a|s, g) = \mathbb{1}(a = \arg\max_{a'} Q^*(s, g, a'))$.

There is ample work extending and improving upon deep Q-learning (Haarnoja et al., 2018). For example Lillicrap et al. (2015) extend it to the continuous action space setting and Fujimoto et al. (2018) further stabilize training. These improvements are fully compatible with goal-reaching (Pong et al., 2019; Bharadhwaj et al., 2020; Ghosh et al., 2019). Andrychowicz et al. (2017) propose a way to store tuples in the replay buffer which is specifically tailored for goal-reaching. Schaul et al. (2015) study value functions for goal-reaching. As previously mentioned, all these methods are limited by the fact that they learn a single way to reach g from s.

3 Cumulative accessibility learning

We now consider horizon-aware policies $\pi : \mathcal{S} \times \mathcal{G} \times \mathbb{N} \to \mathcal{P}(\mathcal{A})$, and define the cumulative accessibility function, or C-function, $C^\pi(s, a, g, h)$ as the probability of reaching goal g from state s in at most h steps by taking action a and following the policy π thereafter. By 'following the policy π thereafter' we mean that after a, the next action a_1 is sampled from $\pi(\cdot|s_1, g, h-1)$, a_2 is sampled from $\pi(\cdot|s_2, g, h-2)$ and so on. See Figure 2 for a graphical model depiction of how these trajectories are obtained. Importantly, an agent need not always act the same way at a particular state in order to reach a particular goal, thanks to horizon-awareness. We use $\mathbb{P}_{\pi(\cdot|\cdot,g,h)}(\cdot|s_0 = s, a_0 = a)$ to denote probabilities in which actions are drawn in this manner and transitions are drawn according to the environment $p(s_{t+1}|s_t, a)$. More formally, C^π is given by:

$$C^\pi(s, a, g, h) = \mathbb{P}_{\pi(\cdot|\cdot,g,h)} \left(\max_{t=0,\ldots,h} G(s_t, g) = 1 \middle| s_0 = s, a_0 = a \right). \tag{2}$$

Proposition 1: C^π can be framed as a Q-function within the MDP formalism, and if π^* is optimal in the sense that $C^{\pi^*}(s, a, g, h) \geq C^\pi(s, a, g, h)$ for every π and $(s, a, g, h) \in \mathcal{S} \times \mathcal{A} \times \mathcal{G} \times \mathbb{N}$, then C^{π^*} matches the optimal C-function, C^*, which obeys the following equation:

$$C^*(s, a, g, h) = \begin{cases} \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} C^*(s', a', g, h-1) \right] & \text{if } G(s, g) = 0 \text{ and } h \geq 1, \\ G(s, g) & \text{otherwise.} \end{cases} \tag{3}$$

Proof: The proof proceeds by first deriving a recurrence relationship that holds for any C^π. In an analogous manner to the Bellman equation in Q-learning, this recurrence involves an expectation over $\pi(\cdot|s', g, h-1)$, which, when replaced by a $\max$ returns the recursion for C^*. See appendix A for a detailed mathematical proof.

Proposition 1 is relevant as it allows us to learn C^*. As mentioned above, this allows goal-reaching policies to be recovered, allows us to test whether a given goal is reachable from a given state, and allows us to make a speed / reliability trade-off for the reachable goals. We now make three key observations. First, we can recover an optimal policy from C^* through:

$$\pi^*(a|s, g, h) = \mathbb{1}\left(a = \arg\max_{a'} C^*(s, a', g, h)\right). \tag{4}$$

Second, an *optimal* C^*-function is non-decreasing in h. We note that this statement is not true in general for non-optimal C-functions. For example, a horizon-aware policy could actively try to avoid the goal for high values of h, and the C^π-function constructed from it would show lower probabilities of success for larger h. See proposition 2 of appendix A for a concrete example of this counter-intuitive behavior.

Proposition 2: C^* is non-decreasing in h.
Proof: Intuitively, the proof consists of showing that an optimal policy can not exhibit the pathology mentioned above. Given an optimal policy $\pi^*(a|s, g, h)$ for a fixed horizon h we construct a policy $\tilde{\pi}$ for $h + 1$ which always performs better, and lower bounds the performance of $\pi^*(a|s, g, h + 1)$. See appendix A for a detailed mathematical proof.

In addition to being an elegant theoretical property, proposition 2 suggests that there is additional structure in a C^* function, structure that mitigates the added complexity from using horizon-aware policies.

Third, just like in Q-learning, the C^* function can modelled as a neural network with parameters θ, denoted C^*_θ. This can be learned by minimizing:

$$\sum_i \left[y_i \log C^*_\theta(s_i, a_i, g_i, h_i) + (1 - y_i) \log\left(1 - C^*_\theta(s_i, a_i, g_i, h_i)\right)\right], \tag{5}$$

where y_i corresponds to the right hand side of equation 3. The sum is over tuples $(s_i, a_i, s'_i, g_i, h_i)$ drawn from a replay buffer, which we specially tailor to successfully learn C^*, and detail in the next section. Note that s'_i is the state the environment transitioned to after taking action a_i from state s_i, and determines the value of y_i. As in Double Q-learning (Van Hasselt et al., 2015), we use a second network $C^*_{\theta'}$ to compute the y_i targets, and do not minimize equation 5 with respect to θ'. We periodically update θ' to match θ. Since C^* corresponds to a probability, we change the usual squared loss to be the binary cross entropy loss. Note that even if the targets are not necessarily binary, the use of binary cross entropy is still justified as it is equivalent to minimizing the $\mathbb{KL}$ divergence between Bernoulli distributions with parameters y_i and $C^*_\theta(s_i, a_i, g_i, h_i)$.

Monotonicity in h could be encoded in the architecture of C^*_θ (Sill, 1998; Wehenkel & Louppe, 2019). However, we found that actively doing so hurt empirical performance. We hypothesize that the network more easily became stuck in local optima when monotonicity was enforced.

3.1 Cumulative accessibility estimation

Our training algorithm, which we call cumulative accessibility estimation (CAE), or C-learning, is detailed in algorithm 1. Below is a description of important subroutines:

1. **Curriculum Learning**. We employ a form of curriculum learning (Elman, 1993). Rollout goals are sampled in an episode-dependent manner (line 9), favoring easier-to-reach goals at the beginning of training. For example, if $\mathcal{G} = \mathcal{S}$ and the agent moves at a limited speed, we sample g from a ball around s_0. We periodically increase the radius of this ball as training progresses.

2. **Reachability-guided Sampling**. When sampling a batch (line 15), for each i we first sample an episode from $\mathcal{D}$, and then a transition (s_i, a_i, s'_i) from that episode. We sample h_i favoring smaller h's at the beginning of training. We achieve this by selecting $h_i = h$ with probability proportional to $h^{-\kappa n_{GD}/N_{GD}}$, where n_{GD} is the current episode number, N_{GD} the total number of episodes, and κ is a hyperparameter. Crucially, we take g_i to be neither the episode goal g, nor the observed state after h_i steps. Instead, we randomly sample g_i to be a potentially reachable goal from s_i in h_i steps. For example, in the same scenario as above, we sample g_i from the set of g's such that $d(s_i, g) \leq h_i$.

3. **C-function Clipping**. When constructing y_i (line 16), if we know for a fact that g_i cannot be reached from s'_i in $h_i - 1$ steps, we output 0 instead of $\max_{a'} C^*_{\theta'}(s'_i, a', g_i, h_i - 1)$. For many

Algorithm 1: Training C-learning Network

Parameter: N_{explore}: Number of random exploration episodes
Parameter: N_{GD}: Number of goal-directed episodes
Parameter: N_{train}: Number of batches to train on per goal-directed episode
Parameter: N_{copy}: Number of batches between target network updates
Parameter: α: Learning rate

1 $\theta \leftarrow$ Initial weights for C^*_θ
2 $\theta' \leftarrow \theta$ `// Copy weights to target network`
3 $\mathcal{D} \leftarrow$ [] `// Initialize experience replay buffer`
4 $n_{\text{b}} \leftarrow 0$ `// Counter for training batches`
5 **repeat** N_{explore} **times**
6 $\quad \mathcal{E} \leftarrow \texttt{get_rollout}(\pi_{\text{random}})$ `// Do a random rollout`
7 $\quad \mathcal{D}.\texttt{append}(\mathcal{E})$ `// Save the rollout to the buffer`
8 **repeat** N_{GD} **times**
9 $\quad g \leftarrow \texttt{goal_sample}(n_{\text{b}})$ `// Sample a goal`
10 $\quad \mathcal{E} \leftarrow \texttt{get_rollout}(\pi_{\text{behavior}})$ `// Try to reach the goal`
11 $\quad \mathcal{D}.\texttt{append}(\mathcal{E})$ `// Save the rollout`
12 $\quad$ **repeat** N_{train} **times**
13 $\quad\quad$ **if** $n_{\text{b}} \mod N_{\text{copy}} = 0$ **then**
14 $\quad\quad\quad \theta' \leftarrow \theta$ `// Copy weights periodically`
15 $\quad\quad \mathcal{B} := \{s_i, a_i, s'_i, g_i, h_i\}_{i=1}^{|\mathcal{B}|} \leftarrow \texttt{sample_batch}(\mathcal{D})$ `// Sample a batch`
16 $\quad\quad \{y_i\}_{i=1}^{|\mathcal{B}|} \leftarrow \texttt{get_targets}(\mathcal{B}, \theta')$
17 $\quad\quad \hat{\mathcal{L}} \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} y_i \log C^*_\theta(s_i, a_i, g_i, h_i) + (1 - y_i) \log\left(1 - C^*_\theta(s_i, a_i, g_i, h_i)\right)$
18 $\quad\quad \theta \leftarrow \theta - \alpha \nabla_\theta \hat{\mathcal{L}}$ `// Update weights`
19 $\quad\quad n_{\text{b}} \leftarrow n_{\text{b}} + 1$ `// Trained one batch`

problems, $d(s'_i, g_i) > h_i - 1$ allows us to set $y_i = 0$. We found this practice, combined with the sampling of g_i described above, to significantly improve performance.

While the last two points require some knowledge of the environment, for example a metric over states, this knowledge is often available. Note also that constructing the targets y_i requires taking a maximum over the action space. While this is straightforward in the tabular case, it is not so for continuous action spaces. Lillicrap et al. (2015) proposed DDPG, a method enabling deep Q-learning in continuous action spaces, and we note that C-learning is compatible with the ideas underpinning DDPG. We present the corresponding algorithmic modifications in appendix C.

Horizon-Independent Policies: To connect with prior work, it can be of interest to recover a horizon-independent policy $\pi^*_\gamma(a|s, g)$ from C^*, where $\gamma \in (0, 1]$ is a hyperparameter controlling the speed vs reliability trade-off. Intuitively, we want to select the smallest value of h for which the probability of success given by C^* exceeds a threshold deemed safe. More formally, we consider a range of viable horizons, $\mathcal{H}$, and find the maximal C^* value, $M(s, g) = \max_{h \in \mathcal{H}, a \in \mathcal{A}} C^*(s, a, g, h)$. We then compute:

$$h_\gamma(s, g) = \arg\min_{h \in \mathcal{H}} \{\max_{a \in \mathcal{A}} C^*(s, a, g, h) : \max_{a \in \mathcal{A}} C^*(s, a, g, h) > \gamma M(s, g)\}, \tag{6}$$

and take $\pi^*_\gamma(a|s, g) = \mathbb{1}\left(a = \arg\max_{a'} C^*(s, a', g, h_\gamma(s, g))\right)$. We use this policy, with added noise for exploration, as the behavior policy in line 10 from Algorithm 1.

Non-Cumulative Accessibility Functions: One could consider defining a non-cumulative version of the C-function, which we call an A-function for "accessibility" (*not* for "advantage"), yielding the probability of reaching a goal in exactly h steps. However, this version is more susceptible to pathological behaviors that hinder learning for certain environments. For illustration, consider an agent that must move one step at a time in the cardinal directions on a checkerboard. Starting on a dark square, the probability of reaching a light square in an even number of steps is always zero, but may be non-zero for odd numbers. An optimal A-function would fluctuate wildly as the step horizon h is increased, resulting in a harder target to learn. Nonetheless, A-functions admit a similar recursion to equation 3, which we include in appendix B for completeness. In any case, the C-function provides a notion of reachability which is more closely tied to related work (Savinov et al., 2018; Venkattaramanujam et al., 2019; Ghosh et al., 2018; Bharadhwaj et al., 2020).

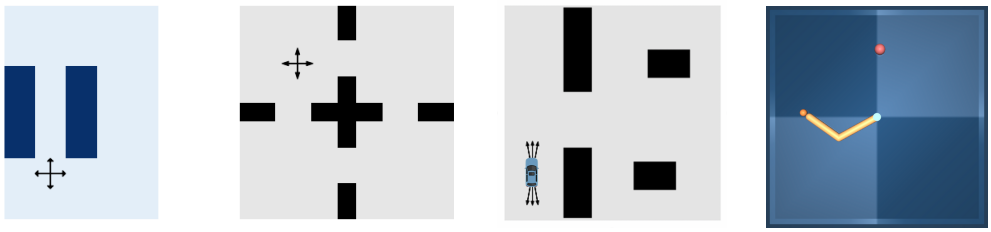


Figure 3: Experimental environments: frozen lake (first), mini maze (second), Dubins' car (third), and reacher (fourth). Arrows represent actions (only their direction should be considered, their magnitude is not representative of the distance travelled by an agent taking an action). See text for description.

At a glance, one might be tempted to think that our only contribution is applying Q-learning for horizon-aware policies. However, the inability of current methods to handle the speed / reliability trade-off at test time strongly motivates the use of horizon-aware policies. We have not only derived Bellman updates for our C^* function, but also shown its monotonicity, a crucial component which facilitates neural network learning in the higher-dimensional goal-horizon space. Since C^* is a probability we are minimizing a binary cross-entropy loss, rather than the MSE loss used in Q-learning, and finally we are also proposing a modified replay buffer which empirically enables us to learn C^*.

We finish this section by noting that Ghosh et al. (2019) recently proposed a method for goal-reaching which is inspired by supervised learning. In their derivations, they also include a horizon h which their policies can depend on, but they drop this dependence in their experiments as they did not see a practical benefit by including h. We find the opposite for C-learning.

4 Experiments

We compare C-learning against GCSL (Ghosh et al., 2019) and deep Q-learning with HER (Andrychowicz et al., 2017) across several environments, which are depicted in Figure 3. All experimental details and hyperparameters are given in appendix D, and we will provide accompanying code by publication time. We evaluate C-learning on the following domains:

1. **Frozen lake** is a 2D navigation environment where the agent must navigate without falling in the holes (dark blue). Both the space state (7×5 grid, with two 3×1 holes) and the action state are discrete. Falling in a hole terminates an episode. The agent's actions correspond to intended directions. The agent moves in the intended direction with probability 0.8, and in perpendicular directions with probabilities 0.1 each. Moving against the boundaries of the environment has no effect. We take $\mathcal{G} = \mathcal{S}$ and $G(s, g) = \mathbb{1}(g = s)$.
2. **Mini maze** is a deterministic 2D navigation environment, where moving against the boundaries or the walls has no effect. Actions and states (24×24 grid) are discrete, and goals also match states.
3. **Dubins' car** is a more challenging deterministic 2D navigation environment, where the agent drives a car which cannot turn by more than $10°$. The states, with spatial coordinates in $[0, 15]^2$, are continuous and include the direction of the car. There are 7 actions: the 6 combinations of $\{\text{left } 10°, \text{ straight}, \text{ right } 10°\} \times \{\text{forward } 1, \text{ reverse } 1\}$, and the option to not move. As such, the set of reachable states is not simply a ball around the current state. The environment also has walls through which the car cannot drive. We take $\mathcal{G} = [0, 15]^2$ and the goal is considered to be reached when the car is within an L_2 distance of 0.5 from the goal, regardless of its direction.
4. **Reacher** is a robotic arm control environment where the agent needs to maneuver the tip of the arm to the target location. We use the hard version of reacher. Actions are continuous and control the joints of the arm, while states correspond to angles of the joints and their momentum. Reaching a goal is defined as being within an L_2 distance of 0.025 of the goal. Since there is no natural metric d for reachability-guided sampling or C-function clipping, we simply sample goals uniformly at random, thus using a simplified version of our replay buffer. We found that C-learning still performed excellently even in this setting, probably due to the fact that a goals occupy a relatively large volume of the space (which is 0.4×0.4) as compared to other environments.

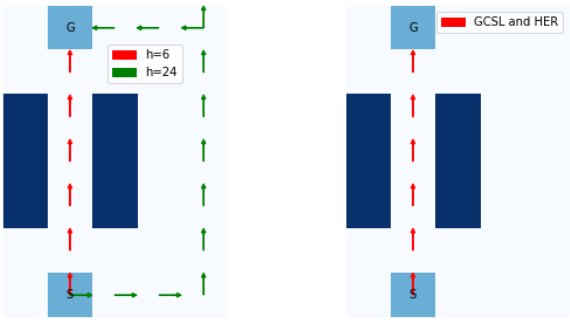
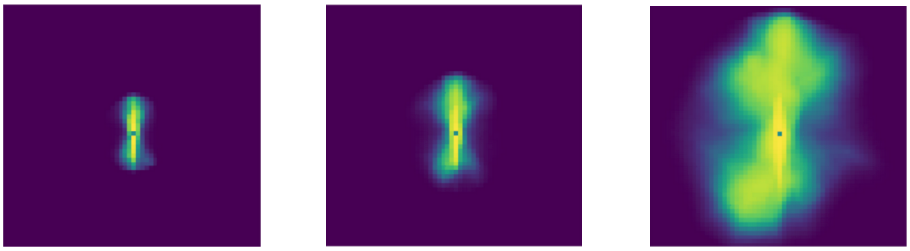


Figure 4: Multimodal policies recovered by C-learning in frozen lake for different values of h for reaching (G) from (S) (left); unimodal policies recovered by GCSL and HER (right).



(a) $h = 10$ (b) $h = 15$ (c) $h = 30$

Figure 5: Heatmaps over the goal space of $\max_a C^*(s, a, g, h)$ with a fixed s and h for Dubins' car.

Exp 1. Speed vs reliability trade-off: We use frozen lake to illustrate the trade-off between speed and reliability. At test time, we set the starting state and goal as shown in Figure 4. Notice that, given enough time, an agent can reach the goal with near certainty by going around the holes on the right side of the lake. However, if the agent is time-constrained the optimal policy must accept the risk of falling in a hole. We see that C-learning does indeed learn both the risky and safe policies. Other methods, as previously explained, can only learn one. To avoid replotting the environment for every horizon, we have plotted arrows in Figure 4 corresponding to $\arg\max_a \pi^*(a|s_0, g, h)$, $\arg\max_a \pi^*(a|s_1, g, h-1)$, $\arg\max_a \pi^*(a|s_2, g, h-2)$ and so on, where s_{t+1} is obtained from the previous state and action while ignoring the randomness in the environment (i.e. $s_{t+1} = \arg\max_s p(s|s_t, a_t)$). In other words, we are plotting the most likely trajectory of the agent. When given the minimal amount of time to reach the goal ($h = 6$), the CAE agent learns correctly to accept the risk of falling in a hole by taking the direct path. When given four times as long ($h = 24$), the agent takes a safer path by going around the hole. Notice that the agent makes a single mistake at the upper right corner, which is quickly corrected when the value of h is decreased. On the other hand, we can see on the right panel that both GCSL and HER recover a single policy, thus not enabling the speed vs reliability trade-off. Surprisingly, GCSL learns to take the high-risk path, despite Ghosh et al.'s intention to incentivize paths that are guaranteed to reach the goal.

Exp 2. Reachability learning: To demonstrate that we are adequately learning reachability, we removed the walls from Dubins' car and further restricted the turning angles to $5°$, thus making the dumbbell shape of the true reachable region more extreme. In Figure 5, we show that our learned C^* function correctly learns which goals are reachable from which states for different time horizons in Dubins' car: not only is the learned C^* function increasing in h, but the shapes are as expected. None of the competing alternatives recover this information in any way, and thus comparisons are not available. As previously mentioned, the optimal C^*-function in this task is not trivial. Reachability is defined by the "geodesics" in $\mathcal{S} = [0, 15]^2 \times S^1$ that are constrained by the finite turning radius, and thus follows a more intricate structure than a ball in $\mathbb{R}^2$.

Exp 3. Goal-reaching: For a qualitative comparison of performance for goal-reaching, Figure 6 shows trajectories for C-learning and competing alternatives for Dubin's car. We can see that C-learning finds a more direct route, while other methods find inefficient paths. This result shows that learning horizon-aware policies does not come at the cost of worsened goal-reaching. For a more quantitative comparison, we show the behavior of the average final distance to the goal throughout training in Figure 7. Other than for Dubins' car, we can see that C-learning is not slower to converge.

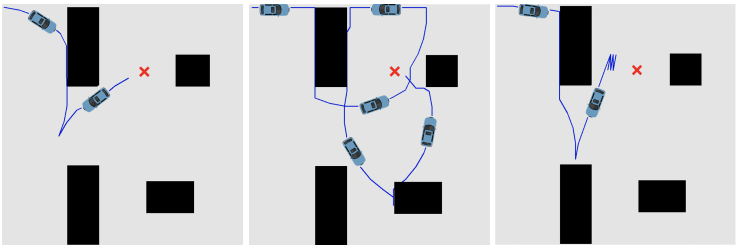


Figure 6: Trajectories learned by C-learning (left), GCSL (middle) and HER (right).

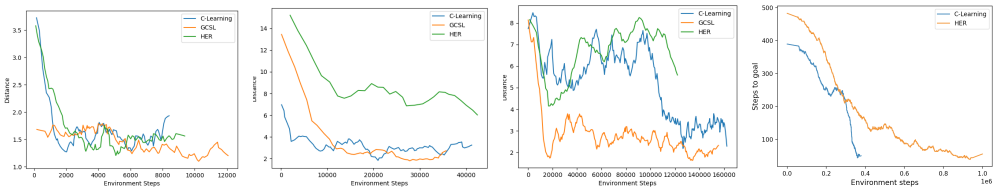


Figure 7: Average over goals of final distance to goal throughout training for frozen lake (first), mini maze (second), and Dubins' car (third). As is commonly done for reacher, we instead plot path length (fourth).

This is also impressive, as we are learning a more complicated objective, which contains information not recoverable by the alternatives. We also point out that, while it appears some training curves are not fully converged, we did did train for longer and observed decreased performance. Finally, for a more granular look into our quantitative results we partition goals into easy, medium, and hard to reach (see appendix D for details). Table 1 shows both overall performance at the end of training, as well as within each goal category. We can see that C-learning either outperforms competing alternatives, or remains competitive. Even more impressively, in Dubins' car, C-learning manages to reach hard goals in a particularly efficient way, even more so than medium ones. We also note that, strangely, while C-learning dominates easy and medium goals in mini maze, it struggles to reach hard goals. We found this particularly puzzling given the good performance in the more challenging environments.

Table 1: Relevant metrics, stratified by difficulty of goal (see figure 8 from appendix D). Runs in **bold** are either the best on the given metric for the given environment, or have overlapping error bars with the best.

ENV	METHOD	FINAL DISTANCE				PATH LENGTH			
		EASY	MED	HARD	**ALL**	EASY	MED	HARD	**ALL**
Frozen lake	CAE	–	–	–	**1.54**	–	–	–	**5.14**
	GCSL	–	–	–	**1.50**	–	–	–	**7.18**
	HER	–	–	–	**2.01**	–	–	–	**6.58**
Mini maze	CAE	**0.02**	**0.34**	11.87	3.15	**5.69**	**14.29**	**11.73**	**11.51**
	GCSL	0.32	1.20	**5.10**	**1.96**	12.37	**13.61**	18.50	14.52
	HER	5.16	3.02	7.91	4.77	81.27	111.62	18.28	80.70
Dubins' car	CAE	2.09	**2.78**	**0.91**	**1.78**	**9.05**	**30.02**	**20.79**	**21.53**
	GCSL	**0.33**	**2.28**	5.63	3.42	**8.06**	61.31	87.77	62.63
	HER	5.05	6.94	7.60	6.86	51.98	57.57	64.87	59.78
Reacher	CAE	–	–	–	**0.022**	–	–	–	**37.32**
	HER	–	–	–	0.025	–	–	–	42.20

5 Conclusions

In this paper we introduced C-learning, a Q-learning-inspired method for goal-reaching. Unlike previous approaches, we propose the use of horizon-aware policies, and show that not only can these policies be tuned to reach the goal faster or more reliably, but they also outperform horizon-unaware approaches for goal-reaching. We hope our method will inspire further research into horizon-aware policies and their benefits.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017.

Homanga Bharadhwaj, Animesh Garg, and Florian Shkurti. Leaf: Latent exploration along the frontier. *arXiv preprint arXiv:2005.10934*, 2020.

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

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*, 2018.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to Reach Goals via Iterated Supervised Learning. *arXiv e-prints*, art. arXiv:1912.06088, December 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

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.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 1–8. IEEE, 2018.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015.

Joseph Sill. Monotonic networks. In *Advances in neural information processing systems*, pp. 661–667, 1998.

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

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

Srinivas Venkattaramanujam, Eric Crawford, Thang Doan, and Doina Precup. Self-supervised learning of distance functions for goal-conditioned reinforcement learning. *arXiv preprint arXiv:1907.02998*, 2019.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. In *Advances in Neural Information Processing Systems*, pp. 1545–1555, 2019.

Jesse Zhang, Brian Cheung, Chelsea Finn, Sergey Levine, and Dinesh Jayaraman. Cautious adaptation for reinforcement learning in safety-critical settings. *arXiv preprint arXiv:2008.06622*, 2020.

A PROOFS OF PROPOSITIONS

Proposition 1: C^π can be framed as a Q-function within the MDP formalism, and if π^* is optimal in the sense that $C^{\pi^*}(s,a,g,h) \geq C^\pi(s,a,g,h)$ for every π and $(s,a,g,h) \in \mathcal{S} \times \mathcal{A} \times \mathcal{G} \times \mathbb{N}$, then C^{π^*} matches the optimal C-function, C^*, which obeys the following equation:

$$C^*(s,a,g,h) = \begin{cases} \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} C^*(s',a',g,h-1) \right] & \text{if } G(s,g) = 0 \text{ and } h \geq 1, \\ G(s,g) & \text{otherwise.} \end{cases}$$

Proof: First, we frame C-functions within the MDP formalism. Consider a space state given by $\mathcal{S}' = \mathcal{S} \times \mathcal{G} \times \mathbb{N}$, where states corresponding to reached goals (i.e. $G(s,g) = 1$) or with last coordinate equal to 0 (i.e. $h = 0$) are considered terminal. The dynamics in this environment are given by:

$$p(s_{t+1}, g_{t+1}, h_{t+1} | s_t, g_t, h_t, a_t) = p(s_{t+1}|s_t, a_t) \mathbb{1}(g_{t+1} = g_t) \mathbb{1}(h_{t+1} = h_t - 1) \,. \tag{7}$$

That is, states evolve according to the original dynamics, while goals remain unchanged and horizons decrease by one at every step. The initial distribution is given by:

$$p(s_0, g_0, h_0) = p(s_0) p(g_0, h_0 | s_0) \,, \tag{8}$$

where $p(s_0)$ corresponds to the original starting distribution and $p(g_0, h_0|s_0)$ needs to be specified beforehand. Together, the initial distribution and the dynamics, along with a policy $\pi(a|s,g,h)$ properly define a distribution over trajectories, and again, we use $\mathbb{P}_{\pi(\cdot|\cdot,g,h)}$ to denote probabilities with respect to this distribution. The reward function $r : \mathcal{S}' \times \mathcal{A} \to \mathbb{R}$ is given by:

$$r(s,a,g,h) = G(s,g) \,. \tag{9}$$

By taking the discount factor to be 1, and since we take states for which $G(s,g) = 1$ to be terminal, the return (sum of rewards) over a trajectory $(s_0, s_1, \dots)$ with $h_0 = h$ is given by:

$$\max_{t=0,\dots,h} G(s_t, g) \,. \tag{10}$$

For notational simplicity we take $G(s_t, g) = 0$ whenever t is greater than the length of the trajectory to properly account for terminal states. Since the return is binary, its expectation matches its probability of being equal to 1, so that indeed, the Q-functions in this MDP correspond to C^π:

$$C^\pi(s,a,g,h) = \mathbb{P}_{\pi(\cdot|\cdot,g,h)} \left(\max_{t=0,\dots,h} G(S_t, g) = 1 \middle| S_0 = s, A_0 = a \right) .$$

Now, we derive the Bellman equation for our C-functions. Trivially:

$$C^\pi(s,a,g,h) = G(s,g), \tag{11}$$

whenever $G(s,g) = 1$ or $h = 0$. For the rest of the derivation, we assume that $G(s,g) = 0$ and $h \geq 1$. We note that the probability of reaching g from s in at most h steps is given by the probability of reaching it in exactly one step, plus the probability of not reaching it in the first step and reaching it in at most $h-1$ steps thereafter. Formally:

$$\begin{aligned} & C^\pi(s,a,g,h) \\ & = C^\pi(s,a,g,1) + \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[(1 - G(s',g)) \mathbb{E}_{a' \sim \pi(\cdot|s',g,h-1)} \left[C^\pi(s',a',g,h-1) \right] \right] \\ & = \mathbb{E}_{s' \sim p(\cdot|s,a)} [G(s',g)] + \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[(1 - G(s',g)) \mathbb{E}_{a' \sim \pi(\cdot|s',g,h-1)} \left[C^\pi(s',a',g,h-1) \right] \right] \\ & = \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\mathbb{E}_{a' \sim \pi(\cdot|s',g,h-1)} \left[C^\pi(s',a',g,h-1) \right] \right] , \end{aligned} \tag{12}$$

where the last equality follows from the fact that $C^\pi(s',a',g,h-1) = 1$ whenever $G(s',g) = 1$. Putting everything together, we obtain the Bellman equation for C^π:

$$C^\pi(s,a,g,h) = \begin{cases} \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\mathbb{E}_{a' \sim \pi(\cdot|s',g,h-1)} \left[C^\pi(s',a',g,h-1) \right] \right] & \text{if } G(s,g) = 0 \text{ and } h \geq 1, \\ G(s,g) & \text{otherwise.} \end{cases} \tag{13}$$

Recall that the optimal policy is defined by the fact that $C^* \geq C^\pi$ for any arguments and for any policy π. We can by maximizing equation 13 that, given the $C^*(\cdot,\cdot,\cdot,h-1)$ values, the optimal policy values at horizon $h-1$ must be $\pi^*(a|s,g,h-1) = \mathbb{1}\left(a = \arg\max_{a'} C^*(s,a,g,h-1)\right)$. We thus we obtain:

$$C^*(s,a,g,h) = \begin{cases} \mathbb{E}_{s'\sim p(\cdot|s,a)}\left[\max_{a'\in\mathcal{A}} C^*(s',a',g,h-1)\right] & \text{if } G(s,g)=0 \text{ and } h\geq 1, \\ G(s,g) & \text{otherwise.} \end{cases}$$

Note that, as in Q-learning, the Bellman equation for the optimal policy has been obtained by replacing the expectation with respect to a' with a $\max$. □

Proposition 2: C^* is non-decreasing in h.

As mentioned in the main manuscript, one might naively think that this monotonicity property should hold for C^π for any π. However this is not quite the case, as π depends on h and a perverse policy may actively avoid the goal for large values of h. Restricting to optimal C^*-functions, such pathologies are avoided. As an example, consider an environment with three states, $\{0,1,2\}$, and two actions $\{-1,+1\}$. $s_{t+1} = \max(0,\min(2,s_t+a_t))$. That is, the agent moves deterministically in the direction of the action unless doing so would move it out of the domain. Goals are defined as states. Let π be such that $\pi(a|s=1,g=2,h=1) = \mathbb{1}(a=1)$ and $\pi(a|s=1,g=2,h=2) = \mathbb{1}(a=-1)$. While clearly a terrible policy, π is such that $C^\pi(s=0,a=1,g=2,h=2) = 1$ and $C^\pi(s=0,a=1,g=2,h=3) = 0$, so that C^π can decrease with h.

Proof: Fix a distribution p on the action space. For any policy $\pi(a|s,g,h)$, we define a new policy $\tilde{\pi}$ as:

$$\tilde{\pi}(a|s,g,h+1) = \begin{cases} \pi(a|s,g,h), & \text{if } h>0 \\ p(a) & \text{, otherwise.} \end{cases} \tag{14}$$

The new policy $\tilde{\pi}$ acts the same as π for the first h steps and on the last step it samples an action from the fixed distribution p. The final step can only increase the cumulative probability of reaching the goal, therefore:

$$C^{\tilde{\pi}}(s,a,g,h+1) \geq C^\pi(s,a,g,h). \tag{15}$$

Since equation 15 holds for all policies π, taking the maximum over policies gives:

$$\begin{aligned} C^*(s,a,g,h+1) &\geq \max_\pi C^{\tilde{\pi}}(s,a,g,h+1) \\ &\geq \max_\pi C^\pi(s,a,g,h) \\ &= C^*(s,a,g,h). \end{aligned} \tag{16}$$

□

B NON-CUMULATIVE CASE

We originally considered a non-cumulative version of the C-function, giving the probability of reaching the goal in exactly h steps. We call this function the accessibility function, A^π (despite our notation, this is unrelated to the commonly used advantage function), defined by:

$$A^\pi(s,a,g,h) = \mathbb{P}_{\pi(\cdot|\cdot,g,h)}\left(G(s_h,g)=1|s_0=s,a_0=a\right). \tag{17}$$

Here the trivial case is only when $h=0$, where we have:

$$A^\pi(s,a,g,0) = G(s,g) \tag{18}$$

If $h\geq 1$, we can obtain a similar recursion to that of the C-functions. Here we no longer assume that states which reach the goal are terminal. The probability of reaching the goal in exactly h steps is equal to the probability of reaching it in $h-1$ steps *after* taking the first step. After the first action a, subsequent actions are sampled from the policy π:

$$\begin{aligned} A^\pi(s,a,g,h) &= \mathbb{P}_{\pi(\cdot|\cdot,g,h)}\left(G(s_h,g)=1|s_0=s,a_0=a\right) \\ &= \mathbb{E}_{s'\sim p(\cdot|s,a)}\left[\mathbb{E}_{a'\sim\pi(\cdot|s',g,h-1)}\left[\mathbb{P}_{\pi(\cdot|\cdot,g,h)}\left(G(s_h,g)=1|s_1=s',a_1=a'\right)\right]\right] \\ &= \mathbb{E}_{s'\sim p(\cdot|s,a)}\left[\mathbb{E}_{a'\sim\pi(\cdot|s',g,h-1)}\left[\mathbb{P}_{\pi(\cdot|\cdot,g,h-1)}\left(G(s_{h-1},g)=1|s_0=s',a_0=a'\right)\right]\right] \\ &= \mathbb{E}_{s'\sim p(\cdot|s,a)}\left[\mathbb{E}_{a'\sim\pi(\cdot|s',g,h-1)}\left[A^\pi(s',a',g,h-1)\right]\right]. \end{aligned} \tag{19}$$

By the same argument we employed in proposition 1, the recursion for the optimal A-function, A^*, is obtained by replacing the expectation with respect to a' with $\max$. Putting this together with the base case, we have:

$$A^*(s,a,g,h) = \begin{cases} \mathbb{E}_{s'\sim p(\cdot|s,a)}\left[\max_{a'\in\mathcal{A}} A^*(s',a,g,h-1)\right] & \text{if } h \geq 1, \\ G(s,g) & \text{otherwise.} \end{cases} \tag{20}$$

Note that this recursion is extremely similar to that of C^*, and differs only in the base cases for the recursion. The difference is empirically relevant however for the two reasons that make C^* easier to learn: C^* is monotonic in h, and the cumulative probability is more well-behaved generally.

C CONTINUOUS ACTION SPACE CASE

As mentioned in the main manuscript, C-learning is compatible with the ideas that underpin DDPG (Lillicrap et al., 2015), allowing it to be used in continuous action spaces. This requires the introduction of another neural network approximating a deterministic policy, $\mu_\phi : \mathcal{S} \times \mathcal{G} \times \mathbb{N} \to \mathcal{A}$, which is intended to return $\mu(s,g,h) = \arg\max_a C^*(s,a,g,h)$. This network is trained alongside C^*_θ as detailed in Algorithm 2.

Algorithm 2: Training C-learning Network in Continuous Action Spaces

Parameter: N_{explore}: Number of random exploration episodes
Parameter: N_{GD}: Number of goal-directed episodes
Parameter: N_{train}: Number of batches to train on per goal-directed episode
Parameter: N_{copy}: Number of batches between target network updates
Parameter: α: Learning rate

$\theta \leftarrow$ Initial weights for C^*_θ
$\theta' \leftarrow \theta$ `// Copy weights to target network`
$\phi \leftarrow$ Initial weights for μ_ϕ
$\mathcal{D} \leftarrow$ `[]` `// Initialize experience replay buffer`
$n_{\text{b}} \leftarrow 0$ `// Counter for training batches`
repeat N_{explore} **times**
  $\mathcal{E} \leftarrow$ `get_rollout`(π_{random}) `// Do a random rollout`
  $\mathcal{D}$.`append`$(\mathcal{E})$ `// Save the rollout to the buffer`
repeat N_{GD} **times**
  $g \leftarrow$ `goal_sample`(n_{b}) `// Sample a goal`
  $\mathcal{E} \leftarrow$ `get_rollout`(π_{behavior}) `// Try to reach the goal`
  $\mathcal{D}$.`append`$(\mathcal{E})$ `// Save the rollout`
  repeat N_{train} **times**
    if $n_{\text{b}} \mod N_{\text{copy}} = 0$ **then**
      $\theta' \leftarrow \theta$ `// Copy weights periodically`
    $\mathcal{B} := \{s_i, a_i, s'_i, g_i, h_i\}_{i=1}^{|\mathcal{B}|} \leftarrow$ `sample_batch`$(\mathcal{D})$ `// Sample a batch`
    $\{y_i\}_{i=1}^{|\mathcal{B}|} \leftarrow$ `get_targets`$(\mathcal{B}, \theta')$
    $\hat{\mathcal{L}}_1 \leftarrow \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|} y_i \log C^*_\theta(s_i,a_i,g_i,h_i) + (1-y_i)\log\left(1 - C^*_\theta(s_i,a_i,g_i,h_i)\right)$
    $\theta \leftarrow \theta - \alpha\nabla_\theta\hat{\mathcal{L}}_1$ `// Update weights`
    $\hat{\mathcal{L}}_2 \leftarrow -\frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|} C^*_\theta(s_i, \mu_\phi(s_i,g_i,h_i), g_i, h_i)$
    $\phi \leftarrow \phi - \alpha\nabla_\phi\hat{\mathcal{L}}_2$ `// Update weights`
    $n_{\text{b}} \leftarrow n_{\text{b}} + 1$ `// Trained one batch`

We do point out that the procedure to obtain a horizon-agnostic policy, and thus π_{behavior}, is also modified from the discrete version. Here, $M(s,g) = \max_{h\in\mathcal{H}} C^*(s,\mu(s,g,h),g,h)$, and:

$$h_\gamma(s,g) = \arg\min_{h\in\mathcal{H}}\{C^*(s,\mu(s,g,h),g,h) : C^*(s,\mu(s,g,h),g,h) > \gamma M(s,g)\}, \tag{21}$$

where we now take $\pi^*_\gamma(a|s,g)$ as a point mass at $\mu(s,g,h_\gamma(s,g))$.

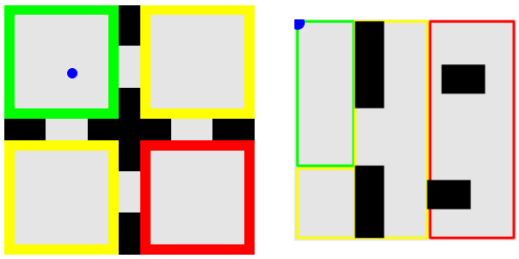


Figure 8: Partitioning of environments into easy (green), medium (yellow) and hard (red) goals to reach from a given starting state (blue). The left panel shows mini maze, and the right shows Dubins' car.

D EXPERIMENTAL DETAILS

For all experiments including baselines, we run tests 3 times and report the best performing results, except for frozen lake, where we use 5 seeds to account for the randomness of the environment. The error bars reported in Table 1 are over goals, and again, to account for the randomness of frozen lake, we evaluate each goal 20 times.

D.1 FROZEN LAKE

For all methods, we train for 300 episodes, each one of maximal length 50 steps, we use a learning rate 10^{-3}, a batch size of size 256, and train for 64 gradient steps per episode. We use a 0.1-greedy for the behavior policy. We use a neural network with two hidden layers of respective sizes 60 and 40 with ReLU activations. We use 15 fully random exploration episodes before we start training. We take $p(s_0)$ as uniform among non-hole states during training, and set it as a point mass at $(1, 0)$ for testing. We set $p(g)$ as uniform among states during training, and we evaluate at every goal during testing. For C-learning, we use $\kappa = 3$, and copy the target network every 10 steps. We take the metric d to be the L_1 norm, completely ignoring the holes so as to not use too much environment information. We do point out that, while C-learning did manage to recover the safe path for large h, it did not do always do so: we suspect that the policy of going directly up is more likely to be explored. However, we never observed GCSL taking the safe path.

D.2 MINI MAZE

We again train for 300 episodes, now of maximal length 150. We train 32 gradient steps per episode, and additionally decay the exploration noise of the behavior policy throughout training according to $\epsilon = 0.5/(1 + n_{GD}/150)$. The network we used has two hidden layers of size 200 and 100 with ReLU activations, respectively. We take the metric d to be the L_1 norm, completely ignoring walls. We take $p(s_0)$ as a point mass both for training and testing. We use $p(g)$ during training as specified in Algorithm 1, and evaluate all the methods on a fixed set of goals. All the other hyperparameters are as in frozen lake. Figure 8 shows the split between easy, medium, and hard goals.

D.3 DUBINS' CAR

We train for 2500 episodes, each one of maximal length 150 steps and train for 32 gradient steps per episode. We use a neural network with two hidden layers of respective sizes 400 and 300 with ReLU activations. We take the metric d to be the L_2 norm, completely ignoring walls. We take $p(s_0)$ as a point mass both for training and testing. We use $p(g)$ during training as specified in Algorithm 1, and evaluate all the methods on a fixed set of goals. All other hyperparameters are as in frozen lake. The partition of goals into easy, medium and hard is specified in Figure 8, where the agent always starts at the upper left corner.

D.4 REACHER

We train for 2000 episodes, with a maximal length of 500 steps. We take 64 gradient steps per episode, and only update ϕ with half the frequency of θ. All the other hyperparameters are as in Dubin's car.