Robot Learning A.Y. 24/25 - Lab 2

Mattia Sabato
DAUIN, Dipartimento di Automatica e Informatica
Politecnico di Torino, Italy
mattia.sabato@studenti.polito.it

Abstract: This paper explores various off-policy methods for solving the Cart-Pole environment. Initially, we use a discretized state representation with tabular Q-Learning. We then shift to a continuous state space, employing a Neural Network (NN) paired with experience replay to approximate the action-value function. This approach enables the agent to learn an optimal policy and successfully balance the pole within a few episodes, demonstrating the effectiveness of combining function approximation and replay mechanisms in Reinforcement Learning (RL).

1 Introduction

RL algorithms provide a powerful framework for addressing control problems in unknown and complex environments. By interacting with the environment, these algorithms enable agents to learn optimal behavior through trial and error. Various strategies can be employed to achieve this goal, broadly categorized into value-based and policy-based approaches. In value-based RL, the agent estimates a value function associated with each state or action, with the policy being greedy and implicit. In contrast, policy-based RL involves directly learning an explicit policy without relying on an explicit value function. When estimating action-value functions, which are the main focus in the following, two principal methodologies are employed: on-policy (online) and off-policy (offline). On-policy methods involve updating the policy that is actively used to generate new transitions. On the other hand, off-policy methods leverage multiple policies: one for generating actions and another for learning and improving. This decoupling allows for greater flexibility and the reuse of experience, such as from a stored dataset, but may introduce additional issues related to convergence and stability.

2 On-Policy vs Off-Policy

In unknown environments, where the transition probability matrix and reward function are not available, we cannot rely on model-based methods like Dynamic Programming. Instead, we must use model-free approaches that estimate values through direct interaction with the environment. Despite the absence of a model, the underlying principle remains the same: we alternate between policy evaluation and policy improvement in a cyclic process until convergence to the optimal value function or policy is achieved. This and the following sections focus on Temporal Difference (TD) methods, which are particularly effective in the off-policy setting. Unlike Monte Carlo (MC) methods, which require complete episodes to compute returns, TD methods update estimates incrementally, making them more efficient in many cases. Among the most widely used on-policy TD algorithms is SARSA ("State-Action-Reward-State-Action"), which takes its name from the sequence of states and actions involved in its updates. The SARSA update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\overbrace{R + \gamma Q(s',a')}^{\textbf{TD Target}} - Q(s,a)]}_{\textbf{TD Error}} \tag{1}$$

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

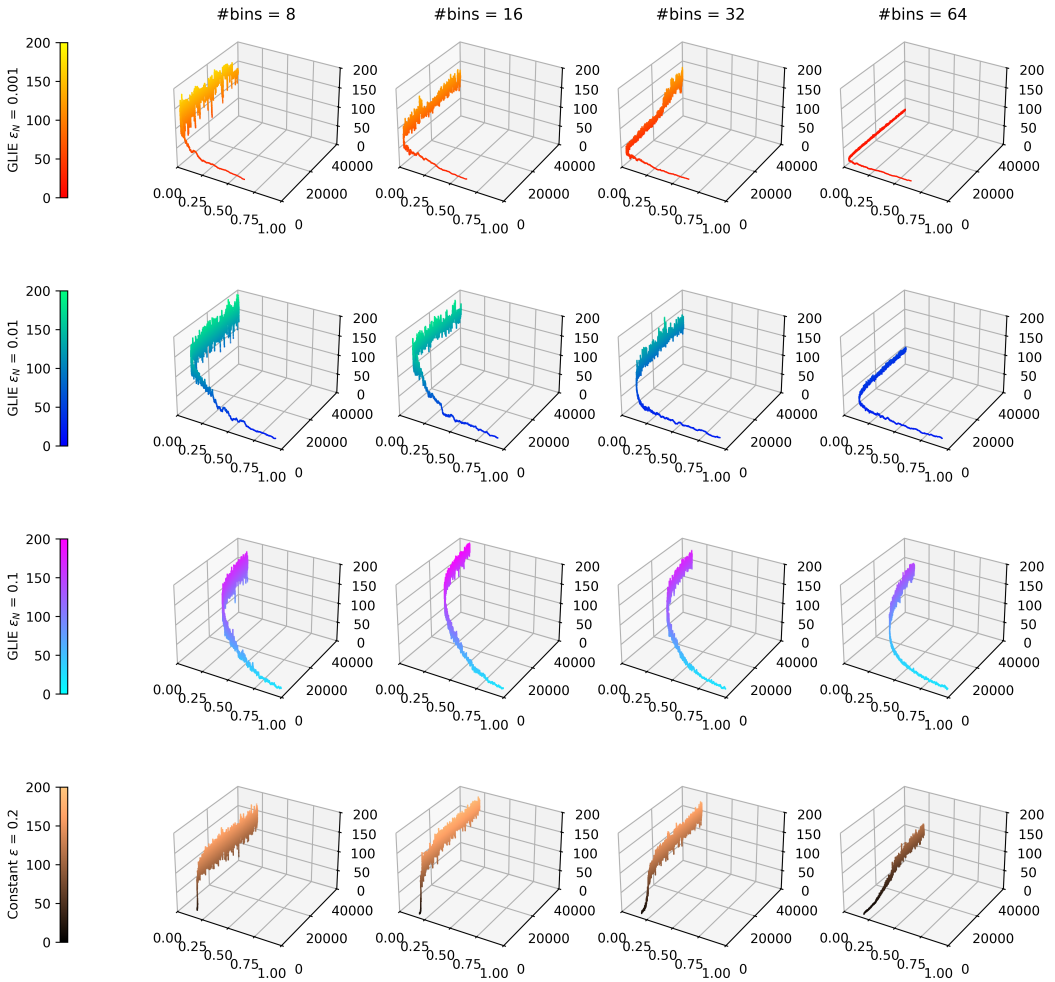

Figure 1: Average total reward computed using a sliding window of 50 episodes, over a total of 40,000, with respect to ϵ. A steep decrease of the last does not allow a proper exploration, whereas keeping it constant prevents the agent from reaching an optimal behavior. The colors intensity represents the magnitude of the reward, bounded by a maximum of 200 corresponding to the timesteps limit of each episode. By making the discretization grid finer the performances notably decrease, reaching a satisfactory reward only when increasing the number of episodes and using a schedule with a limiting value for ϵ high enough.

where $Q(s, a)$ is the action-value function, α is the learning rate, r is the observed reward, γ is the discount factor, and (s', a') is the subsequent state-action pair generated by following the current policy. This incremental update ensures that SARSA directly learns and improves the policy, making it an on-policy method. In TD learning we estimate the action-value function by looking one step ahead, incorporating new information obtained from the immediate reward. By iterating this process over multiple transitions, we expect the estimate Q to converge to the true value function q_π, provided that the policy is improved at each step. This improvement typically involves adopting an ϵ-greedy behavior, which balances exploration and exploitation. Unlike MC methods, TD methods allow updates to the value function during the episode itself, allowing computational efficiency and faster learning. Extensions of TD methods, such as SARSA(λ), introduce multi-step updates by employing eligibility traces. These traces assign credit to prior states and actions based on their relevance to the current learning step, exploiting historical information rather than only considering future rewards. However, TD methods introduce a trade-off: they are subject to bias because updates rely on estimates rather than true values. This bootstrap nature can lead to accumulating errors, especially when combined with off-policy learning and nonlinear function approximation.

Moreover, a notable drawback of on-policy methods is inefficiency in utilizing data. Each transition is used to update the action-value function and then discarded, making it impossible to revisit past experiences. Off-policy methods address this limitation by decoupling the target and behavior policies. The target policy represents the optimal behavior being learned, while the behavior policy is responsible for exploring the environment. This decoupling allows for the reuse of past transitions, improving sample efficiency and enabling learning from stored experience (e.g., through the use of experience replay). One of the most known off-policy algorithms is Q-Learning, where the action-value function is updated using the rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)] \tag{2}$$

where now the TD Target is computed considering the maximum action-value function.

3 Tabular Q-Learning

To evaluate the performance of Q-Learning, we used the Cart-Pole environment from Gym. The goal in this environment is to balance a pole on a moving cart for as long as possible without the pole falling over or the cart moving out of bounds. For the following experiments, we constrained each episode to 200 timesteps, used the default reward structure provided by the framework, and implemented a tabular version of Q-Learning. In this setup, the state space was discretized and represented as a multidimensional array, with $Q(s,a)$ values computed in an off-policy fashion. The behavior policy was tested using two strategies: a constant and a decaying ϵ, the last following a Greedy in the Limit with Infinite Exploration (GLIE) schedule, defined as:

$$\epsilon_k \leftarrow \frac{b}{b+k} \tag{3}$$

where b is a hyperparameter controlling the episode k at which ϵ_k approaches the desired limit. A qualitative comparison of different hyperparameter settings is presented in Figure 1. The choice of the limiting ϵ significantly impacts the agent performance. If it decreases too quickly (e.g., a very small limit value), the agent prematurely converges to a suboptimal policy due to insufficient exploration. This issue is particularly impacting when the state space is finely discretized, as the agent tends to getting stuck in early stages of training. Conversely, setting the limiting value to 0.1 yields a favorable trade-off between exploration and exploitation. In contrast, using a constant ϵ enables the agent to achieve high rewards more quickly but with greater variance, as shown in Figure 2. The fixed exploration rate also limited the agent's ability to explore adequately, often leading to worse performances compared to an agent following a decaying schedule.

After training the agent for 500 timesteps, and having reached a satisfactory policy, we visualized the learned value function as a function of x (cart position) and $\dot{x}$ (cart velocity), averaging over θ (pole angle) and $\dot{\theta}$ (pole angular velocity), as shown in Figure 3. The highest values of the value function correspond to states with limited magnitudes of the two variables considered. This outcome is expected, as extreme values often lead to early episode termination, thereby reducing the cumulative reward. We here recall that an episode ends when either the cart moves out of bounds — i.e., $|x|$ exceeds the screen limits — or the pole falls — typically associated with high magnitudes of $\dot{x}$. Additionally, as the grid resolution becomes finer, many combinations exhibit zero values. This can be also motivated by the fact that states associated with extreme conditions are rarely visited during training; hence, their values remain at the initialized default due to the lack of sufficient updates. Initially, the value grid is empty, i.e. all values are the same, as the agent has yet to explore much of the state space. Over the course of training, the grid gradually fills with more accurate estimates for frequently visited states. In the Cart-Pole environment, where the agent is initialized at the start of every episode in a state with variables close to zero, the central region of the grid becomes particularly dense. This observation aligns with the experimental results, reflecting the

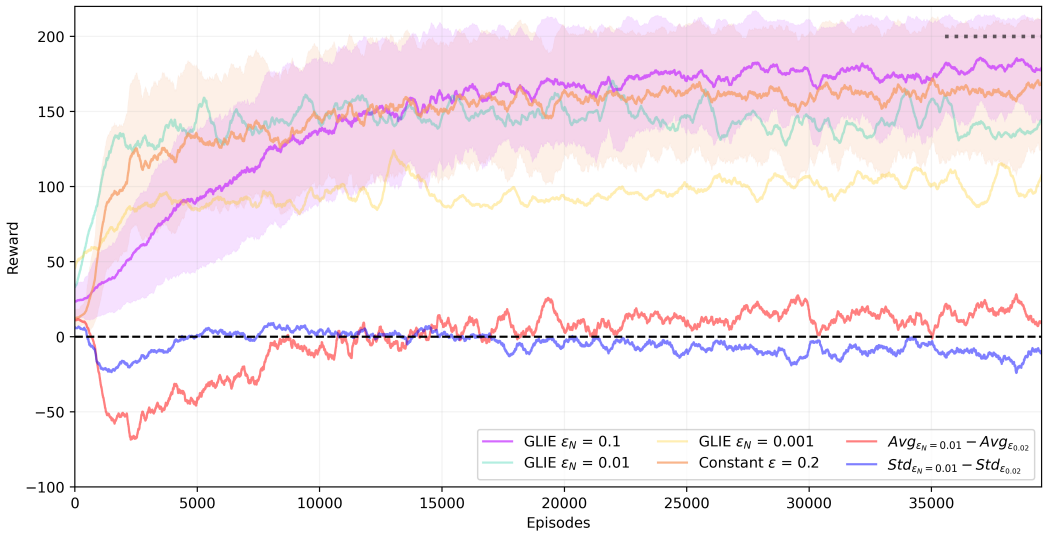

Figure 2: Average total reward computed using a sliding window of 500 episodes, over a total of 40,000, and fixing the number of bins to 16. The agent with the limiting value for ϵ_k equal to 0.1 is able to achieve better and stabler performances with respect to one using a fixed value. This difference is expected to increase with the number of episodes.

agent's tendency to remain in states near the starting position while successfully balancing the pole for longer durations.

3.1 Greedy Initialization

In our previous experiments, we tested various behaviors and hyperparameter combinations, such as the granularity of the state-space discretization and the limiting value for ϵ_k, including keeping a constant value. Notably, in all cases, we initialized the array storing the action-value function estimates with zero values. In this subsection, we study an alternative approach, exploring the agent's performance when initialized with different initial values and acting greedily. The results, shown in Figure 4, reveal a performance shift correlated with the magnitude of the initial values. Specifically, higher ones lead to improved outcomes, demonstrating a positive correlation. This result aligns with the expectation that, when aiming to maximize a positive reward, such as in the Cart-Pole environment, the action-value function estimates should converge to positive values, bounded above by a constant determined by the environment. By initializing with optimistic values (e.g., assigning high initial estimates across all state-action pairs), the agent, acting greedily, quickly updates its estimates when observing more realistic lower outcomes. This process encourages exploration of alternative actions, leading the agent to converge more efficiently to an optimal policy. This is motivated by the fact that, being disappointed by the choice it made, it will eventually select alternatives when faced with the same state in subsequent timesteps. Finally, once the agent identifies the best action for a given state, it will consistently select it, resulting in stable and high performances. In contrast, initializing with lower values, such as zeros, limits the agent's exploration since it is less likely to deviate from initial poor choices, leading to slower convergence and suboptimal performance. In this case, once received a positive reward, it will permanently select the action leading to it, without ever exploring the other one.

4 Q-Learning with Function Approximation

So far, we have explored a tabular implementation of Q-Learning, which proves to be effective for simple environments with moderately complex underlying relationships. However, this approach fails when generalizing to continuous state spaces, as it is infeasible to build an infinitely large table

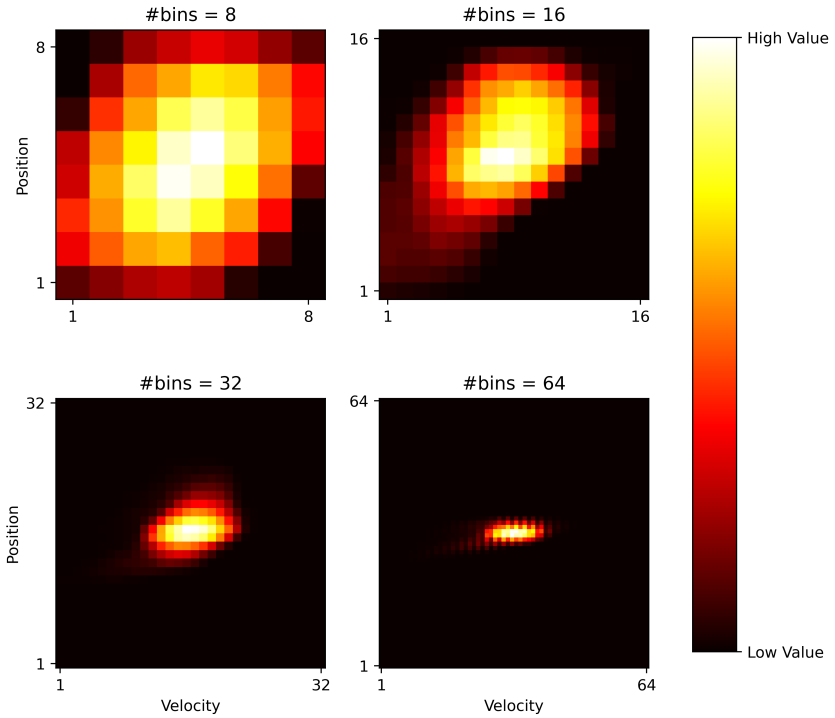

Figure 3: Grid representing the value function obtained through an agent trained over 40,000 episodes with a limit of 500 timesteps and a limiting value for ϵ_k of 0.1. The value is expressed with respect to the position x and the velocity $\dot{x}$, by averaging over the angle θ and the angular velocity $\dot{\theta}$. Higher values correspond to the smallest magnitudes for both of the state's variables, since extreme ones may lead to an early ending of the episode. As the grid gets finer, less combinations are explored due to the limited capacities of the agent.

to represent an uncountable set of real-valued states. In such scenarios, we turn to function approximators, such as NNs, which enable generalization to unseen states. To evaluate the effectiveness of this approach, we implemented a Deep Q-Network (DQN), leveraging a NN combined with experience replay. This method allows the agent to learn in an off-policy manner and balance the pole without requiring a discrete state-space approximation. The algorithm and hyperparameters choices are detailed in Algorithm 1. The use of experience replay enables sampling from stored transitions, allowing updates based on a stronger policy from prior experiences. By decorrelating updates and breaking temporal dependencies in the data, we can significantly improve learning stability and performances. A crucial modification is the introduction of a custom reward function to replace the default Cart-Pole reward, the former being defined as:

$$R_{t+1}(s_{t+1}, s_t) = 1 + (1 - 2 * \mathbb{1}_{|\theta_{t+1}| > |\theta_t|}) * 0.5 + (1 - 2 * \mathbb{1}_{|\dot{\theta}_{t+1}| > |\dot{\theta}_t|}) * 0.5 + (1 - 2 * \mathbb{1}_{|\dot{x}_{t+1}| > |\dot{x}_t|}) \quad (4)$$

with $R_{t+1}(\cdot) = -2$ if the episode terminates prematurely (i.e., before reaching the maximum timestep limit) and $R_{t+1}(\cdot) = +2$ otherwise.

This reward function addresses a key limitation of the default constant reward of $+1$, which fails to provide the agent with meaningful feedback regarding the long-term consequences of its actions. Under the default reward scheme, the agent receives valuable feedback only upon failure, since it receives the same reward in all other steps, making it difficult to learn an effective policy. The new reward function incorporates feedback on the local impact of each action by penalizing increases in the pole angle $|\theta|$, the angular velocity $|\dot{\theta}|$ and the velocity of the cart $\dot{x}$, as these are strong proxies

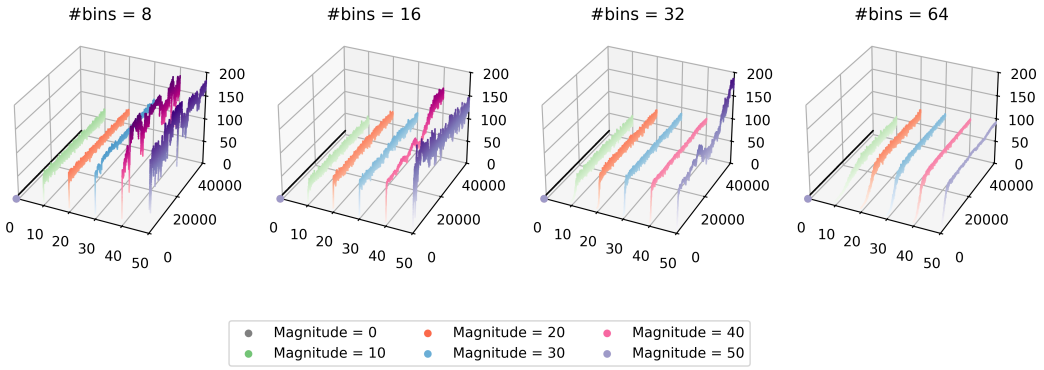

Figure 4: Average total reward computed using a sliding window of 50 episodes, over a total of 40,000, with a limit of 200 timesteps. Varying the magnitude of the values with which we initialize the action-value function leads to remarkable differences in the total reward achieved. Starting with zeros always leads to no improvement at all, whereas using optimistic estimates allows to reach good performances.

for the pole's balance. This design encourages the agent to take actions that minimize deviations and maintain stability, improving its ability to learn an optimal balancing strategy.

Algorithm 1: Deep Q-Network

Initialize replay memory $\mathcal{D}$ to 100,000;
Initialize action-value function Q with weights θ;
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$;
for *episode = 1, 250* **do**
  Initialize sequence $s_1 = \{(x_1, \dot{x}_1, \theta_1, \dot{\theta}_1)\}$;
  while *episode is not terminated* **do**
    With probability $\epsilon = 0.1$ select a random action a_t
    otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$;
    Execute action a_t and observe reward r_{t+1}, state s_{t+1} and
    check if the episode terminates $\mathbb{1}_{t+1}$;
    Store transition $(s_t, s_{t+1}, \mathbb{1}_{t+1}, r_{t+1}, a_t)$ in $\mathcal{D}$;
    Sample random minibatch of 64
    transitions $(s_{i,t}, s_{i,t+1}, \mathbb{1}_{i,t+1}, r_{i,t+1}, a_{i,t})_{i=1,\dots,64}$;
    Set $y_j = r_{t+1}$ if episode terminates at $t+1$,
    otherwise $y_j = r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a; \theta^-)$;
    Perform 5 gradient descent steps with batch size of 32
    on $(y_j - Q(s_t, a_t; \theta)$ with respect to the network parameters θ;
    Every 50 episodes reset $\hat{Q} = Q$;
  end
end

The results presented in Figure 5 demonstrate that the agent achieves high performance within a limited number of episodes, highlighting the rapid convergence capabilities of the DQNs. However, despite its effectiveness, the simplicity of the standard algorithm, while appealing, results in some instabilities during training.

Even with the inclusion of experience replay the training process exhibits persistent variance, likely due to overoptimistic value estimates. In contrast, more advanced techniques, such as Double Deep Q-Networks (DDQN), address this issue, significantly reducing overestimation bias and improving stability. While DQNs have proven capable of generalizing to continuous state spaces, they struggle in environments with continuous action spaces. This is due to the computation of the maximum Q-value, which becomes infeasible in a continuous action domain due to the infinite number of

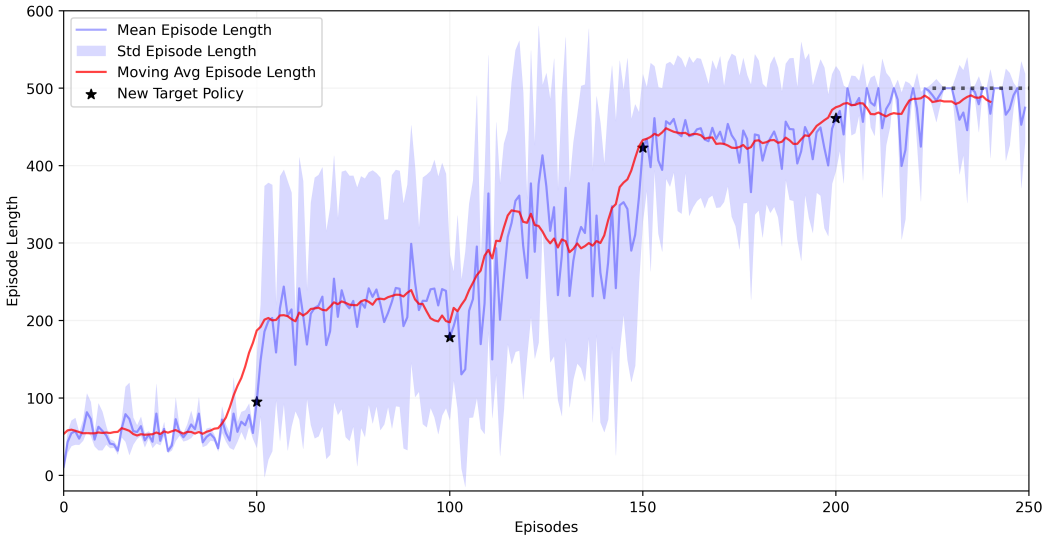

Figure 5: Average total reward computed across 4 different training procedures over a total 250 episodes with a timesteps limit of 500 and changing the target policy every 50 episodes. The agent boosts its performances cyclically at every update of the last, increasing the total reward after an initial transient. The variance also decreases as the training proceeds, proving a stable and effective convergence.

possible actions. Policy-based algorithms provide an effective alternative in such scenarios. Instead of maximizing over actions, these methods learn a probability distribution over the lasts, enabling the agent to select them based on sampling rather than deterministic choices.