# Stochastic Q-learning for Large Discrete Action Spaces

**Fares Fourati**[*]
KAUST
Thuwal, Saudi Arabia

**Vaneet Aggarwal**
Purdue University
West Lafayette, IN, USA

**Mohamed-Slim Alouini**
KAUST
Thuwal, Saudi Arabia

## Abstract

In complex environments with large discrete action spaces, effective decision-making is critical in reinforcement learning (RL). Despite the widespread use of value-based RL approaches like Q-learning, they come with a computational burden, necessitating the maximization of a value function over all actions in each iteration. This burden becomes particularly challenging when addressing large-scale problems and using deep neural networks as function approximators. In this paper, we present stochastic value-based RL approaches which, in each iteration, as opposed to optimizing over the entire set of $n$ actions, only consider a variable stochastic set of a sublinear number of actions, possibly as small as $\mathcal{O}(\log(n))$. The presented stochastic value-based RL methods include, among others, Stochastic Q-learning, StochDQN, and StochDDQN, all of which integrate this stochastic approach for both value-function updates and action selection. The theoretical convergence of Stochastic Q-learning is established, while an analysis of stochastic maximization is provided. Moreover, through empirical validation, we illustrate that the various proposed approaches outperform the baseline methods across diverse environments, including different control problems, achieving near-optimal average returns in significantly reduced time.

## 1 Introduction

Reinforcement learning (RL), a continually evolving field of machine learning, has achieved notable successes, especially when combined with deep learning [46, 57]. While there have been several advances in the field, a significant challenge lies in navigating complex environments with large discrete action spaces [7, 8]. In such scenarios, standard RL algorithms suffer in terms of computational efficiency [1]. Identifying the optimal actions might entail cycling through all of them, in general, multiple times within different states, which is computationally expensive and may become prohibitive with large discrete action spaces [50].

Such challenges apply to various domains, including combinatorial optimization [32, 11, 14, 13], natural language processing [18, 19, 20, 50], communications and networking [30, 12], recommendation systems [7], transportation [2, 16, 28], and robotics [7, 48, 47, 44, 45, 15, 22]. Although tailored solutions leveraging action space structures and dimensions may suffice in specific contexts, their applicability across diverse problems, possibly unstructured, still needs to be expanded. We complement these works by proposing a general method that addresses a broad spectrum of problems, accommodating structured and unstructured single and multi-dimensional large discrete action spaces.

Value-based and actor-based approaches are both prominent approaches in RL. Value-based approaches, which entail the agent implicitly optimizing its policy by maximizing a value function, demonstrate superior generalization capabilities but demand significant computational resources, particularly in complex settings. Conversely, actor-based approaches, which entail the agent directly optimizing its policy, offer computational efficiency but often encounter challenges in generalizing

[*]Corresponding author. Email: fares.fourati@kaust.edu.sa.

17th European Workshop on Reinforcement Learning (EWRL 2024).

across multiple and unexplored actions [7]. While both hold unique advantages and challenges, they represent distinct avenues for addressing the complexities of decision-making in large action spaces. However, comparing them falls outside the scope of this work. While some previous methods have focused on the latter [7], our work concentrates on the former. Specifically, we aim to exploit the natural generalization inherent in value-based RL approaches while reducing their per-step computational complexity.

Q-learning, as introduced by [58], for discrete action and state spaces, stands out as one of the most famous examples of value-based RL methods and remains one of the most widely used ones in the field. As an off-policy learning method, it decouples the learning process from the agent's current policy, allowing it to leverage past experiences from various sources, which becomes advantageous in complex environments. In each step of Q-learning, the agent updates its action value estimates based on the observed reward and the estimated value of the best action in the next state.

Some approaches have been proposed to apply Q-learning to continuous state spaces, leveraging deep neural networks [35, 53]. Moreover, several improvements have also been suggested to address its inherent estimation bias [17, 53, 61, 27, 56]. However, despite the different progress and its numerous advantages, a significant challenge still needs to be solved in Q-learning-like methods when confronted with large discrete action spaces. The computational complexity associated with selecting actions and updating Q-functions increases proportionally with the increasing number of actions, which renders the conventional approach impractical as the number of actions substantially increases. Consequently, we confront a crucial question: *Is it possible to mitigate the complexity of the different Q-learning methods while maintaining a good performance?*

This work proposes a novel, simple, and practical approach for handling general, possibly unstructured, single-dimensional or multi-dimensional, large discrete action spaces. Our approach targets the computational bottleneck in value-based methods caused by the search for a maximum ($\max$ and $\arg\max$) in every learning iteration, which scales as $\mathcal{O}(n)$, i.e., linearly with the number of possible actions $n$. Through randomization, we can reduce this per-step complexity to logarithmic.

We introduce $\mathrm{stoch}\max$ and $\mathrm{stoch}\arg\max$, which, instead of exhaustively searching for the precise maximum across the entire set of actions, rely on at most two random subsets of actions, both of sub-linear sizes, possibly each of size $\lceil \log(n) \rceil$. The first subset is randomly sampled from the complete set of actions, and the second from the previously exploited actions. These stochastic maximization techniques amortize the computational overhead of standard maximization operations in various Q-learning methods [58, 17, 35, 53]. Stochastic maximization methods significantly accelerate the agent's steps, including action selection and value-function updates in value-based RL methods, making them practical for handling challenging, large-scale, real-world problems.

We propose Stochastic Q-learning, Stochastic Double Q-learning, StochDQN, and StochDDQN, which are obtained by changing $\max$ and $\arg\max$ to $\mathrm{stoch}\max$ and $\mathrm{stoch}\arg\max$ in the Q-learning [58], the Double Q-learning [17], the deep Q-network (DQN) [35] and the Double DQN (DDQN) [53], respectively. Furthermore, we observed that our approach works even for Sarsa [41].

We conduct a theoretical analysis of the proposed method, proving the convergence of Stochastic Q-learning, which integrates these techniques for action selection and value updates, and establishing a lower bound on the probability of sampling an optimal action from a random set of size $\lceil \log(n) \rceil$ and analyze the error of stochastic maximization compared to exact maximization. Furthermore, we evaluate the proposed RL algorithms on environments from Gymnasium [3]. For the stochastic deep RL algorithms, the evaluations were performed on control tasks within the MuJoCo environment [51] with discretized actions [7, 48, 47]. These evaluations demonstrate that the stochastic approaches outperform non-stochastic ones regarding wall time speedup and sometimes rewards.

Our key contributions are summarized as follows: We introduce novel stochastic maximization techniques denoted as $\mathrm{stoch}\max$ and $\mathrm{stoch}\arg\max$, offering a compelling alternative to exact maximization operations, particularly beneficial for handling large discrete action spaces, ensuring sub-linear complexity regarding the number of actions. We present a suite of value-based RL algorithms suitable for large discrete actions, including Stochastic Q-learning, Stochastic Sarsa, Stochastic Double Q-learning, StochDQN, and StochDDQN, which integrate stochastic maximization within Q-learning, Sarsa, Double Q-learning, DQN, and DDQN, respectively. We analyze stochastic maximization and demonstrate the convergence of Stochastic Q-learning. Furthermore, we empirically

validate our approach on tasks from the Gymnasium and MuJoCO environments, encompassing various dimensional discretized actions.

## 2 Related Works

While RL has shown promise in diverse domains, practical applications often grapple with real-world complexities. A significant hurdle arises when dealing with large discrete action spaces [7, 8]. Previous research has investigated strategies to address this challenge by leveraging the combinatorial or the dimensional structures in the action space [20, 48, 50, 5, 44, 45, 11, 14, 13, 1, 14, 13, 22]. For example, [20] leveraged the combinatorial structure of their language problem through sub-action embeddings. Compressed sensing was employed in [50] for text-based games with combinatorial actions. [5] formulated the combinatorial action decision of a vehicle routing problem as a mixed-integer program. Moreover, [1] introduced dynamic neighbourhood construction specifically for structured combinatorial large discrete action spaces. Previous works tailored solutions for multi-dimensional spaces such as those in [44, 45, 22], among others, while practical in the multi-dimensional spaces, may not be helpful for single-dimensional large action spaces. While relying on the structure of the action space is practical in some settings, not all problems with large action spaces are multi-dimensional or structured. We complement these works by making no assumptions about the structure of the action space. See Appendix A for more detailed related work overview.

## 3 Problem Description

In the context of a Markov decision process (MDP), we have specific components: a finite set of actions denoted as $\mathcal{A}$, a finite set of states denoted as $\mathcal{S}$, a transition probability distribution $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, a bounded reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1]$. Furthermore, for time step $t$, we denote the chosen action as $\mathbf{a}_t$, the current state as $\mathbf{s}_t$, and the received reward as $r_t \triangleq r(\mathbf{s}_t, \mathbf{a}_t)$. Additionally, for time step $t$, we define a learning rate function $\alpha_t : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The cumulative reward an agent receives during an episode in an MDP with variable length time $T$ is the return $R_t$. It is calculated as the discounted sum of rewards from time step $t$ until the episode terminates: $R_t \triangleq \sum_{i=t}^{T} \gamma^{i-t} r_i$. RL aims to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ mapping states to actions that maximize the expected return across all episodes. The state-action value function, denoted as $Q^\pi(\mathbf{s}, \mathbf{a})$, represents the expected return when starting from a given state $\mathbf{s}$, taking action $\mathbf{a}$, and following a policy $\pi$ afterwards. The function $Q^\pi$ can be expressed using the Bellman equation:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) Q^\pi(\mathbf{s}', \pi(\mathbf{s}')). \tag{1}$$

Two main categories of policies are commonly employed in RL systems: value-based and actor-based policies [46]. This study primarily concentrates on the former type, where the value function directly influences the policy's decisions. An example of a value-based policy in a state $\mathbf{s}$ involves an $\varepsilon_\mathbf{s}$-greedy algorithm, selecting the action with the highest Q-function value with probability $(1 - \varepsilon_\mathbf{s})$, where $\varepsilon_\mathbf{s} \geq 0$, function of the state $\mathbf{s}$, requiring the use of $\arg\max$ operation, as follows:

$$\pi_Q(\mathbf{s}) = \begin{cases} \text{play randomly} & \text{with proba. } \epsilon_\mathbf{s} \\ \arg\max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a}) & \text{otherwise.} \end{cases} \tag{2}$$

Furthermore, during the training, to update the Q-function, Q-learning [58], for example, uses the following update rule, which requires a $\max$ operation:

$$Q_{t+1}(\mathbf{s}_t, \mathbf{a}_t) = (1 - \alpha_t(\mathbf{s}_t, \mathbf{a}_t)) Q_t(\mathbf{s}_t, \mathbf{a}_t) + \alpha_t(\mathbf{s}_t, \mathbf{a}_t) \left[ r_t + \gamma \max_{b \in \mathcal{A}} Q_t(\mathbf{s}_{t+1}, b) \right]. \tag{3}$$

Therefore, the computational complexity of both the action selections in Eq. (2) and the Q-function updates in Eq. (3) scales linearly with the cardinality $n$ of the action set $\mathcal{A}$, making this approach infeasible as the number of actions increases significantly. The same complexity issues remain for other Q-learning variants, such as Double Q-learning [17], DQN [35], and DDQN [53].

When representing the value function as a parameterized function, such as a neural network, taking only the current state $\mathbf{s}$ as input and outputting the values for all actions, as proposed in DQN [35],

the network must accommodate a large number of output nodes, which results in increasing memory overhead and necessitates extensive predictions and maximization over these final outputs in the last layer. A notable point about this approach is that it does not exploit contextual information (representation) of actions, if available, which leads to lower generalization capability across actions with similar features and fails to generalize over new actions.

Previous works have considered generalization over actions by taking the features of an action $\mathbf{a}$ and the current state $\mathbf{s}$ as inputs to the Q-network and predicting its value [59, 33, 52]. However, it leads to further complications when the value function is modeled as a parameterized function with both state $\mathbf{s}$ and action $\mathbf{a}$ as inputs. Although this approach allows for improved generalization across the action space by leveraging contextual information from each action and generalizing across similar ones, it requires evaluating the function for each action within the action set $\mathcal{A}$. This results in a linear increase in the number of function calls as the number of actions grows. This scalability issue becomes particularly problematic when dealing with computationally expensive function approximators, such as deep neural networks [7]. Addressing these challenges forms the motivation behind this work.

# 4 Proposed Approach

To alleviate the computational burden associated with maximizing a Q-function at each time step, especially when dealing with large action spaces, we introduce stochastic maximization methods with sub-linear complexity relative to the size of the action set $\mathcal{A}$. Then, we integrate these methods into different value-based RL algorithms.

## 4.1 Stochastic Maximization

We introduce stochastic maximization as an alternative to maximization when dealing with large discrete action spaces. Instead of conducting an exhaustive search for the precise maximum across the entire set of actions $\mathcal{A}$, stochastic maximization searches for a maximum within a stochastic subset of actions of sub-linear size relative to the total number of actions. In principle, any size can be used, trading off time complexity and approximation. We mainly focus on $\mathcal{O}(\log(n))$ to illustrate the power of the method in recovering Q-learning, even with such a small number of actions, with logarithmic complexity.

We consider two approaches to stochastic maximization: memoryless and memory-based approaches. The memoryless one samples a random subset of actions $\mathcal{R} \subseteq \mathcal{A}$ with a sublinear size and seeks the maximum within this subset. On the other hand, the memory-based one expands the randomly sampled set to include a few actions $\mathcal{M}$ with a sublinear size from the latest exploited actions $\mathcal{E}$ and uses the combined sets to search for a stochastic maximum. Stochastic maximization, which may miss the exact maximum in both versions, is always upper-bounded by deterministic maximization, which finds the exact maximum. However, by construction, it has sublinear complexity in the number of actions, making it appealing when maximizing over large action spaces becomes impractical.

Formally, given a state $\mathbf{s}$, which may be discrete or continuous, along with a Q-function, a random subset of actions $\mathcal{R} \subseteq \mathcal{A}$, and a memory subset $\mathcal{M} \subseteq \mathcal{E}$ (empty in the memoryless case), each subset being of sublinear size, such as at most $\mathcal{O}(\log(n))$ each, the $\operatorname{stoch}\max$ is the maximum value computed from the union set $\mathcal{C} = \mathcal{R} \cup \mathcal{M}$, defined as:

$$\underset{k \in \mathcal{A}}{\operatorname{stoch}\max}\, Q_t(\mathbf{s}, k) \triangleq \max_{k \in \mathcal{C}} Q_t(\mathbf{s}, k). \tag{4}$$

Besides, the $\operatorname{stoch}\arg\max$ is computed as follows:

$$\underset{k \in \mathcal{A}}{\operatorname{stoch}\arg\max}\, Q_t(\mathbf{s}, k) \triangleq \arg\max_{k \in \mathcal{C}} Q_t(\mathbf{s}, k). \tag{5}$$

In the analysis of stochastic maximization, we explore both memory-based and memoryless maximization. In the analysis and experiments, we consider the random set $\mathcal{R}$ to consist of $\lceil \log(n) \rceil$ actions. When memory-based, in our experiments, within a given discrete state, we consider the two most recently exploited actions in that state. For continuous states, where it is impossible to retain the latest exploited actions for each state, we consider a randomly sampled subset $\mathcal{M} \subseteq \mathcal{E}$, which includes $\lceil \log(n) \rceil$ actions, even though they were played in different states. We demonstrate that this

approach was sufficient to achieve good results in the benchmarks considered; see Section 7.3. Our Stochastic Q-learning convergence analysis considers memoryless stochastic maximization with a random set $\mathcal{R}$ of any size.

*Remark* 4.1. By setting $\mathcal{C}$ equal to $\mathcal{A}$, we essentially revert to standard approaches. Consequently, our method is an extension of non-stochastic maximization. However, in pursuit of our objective to make RL practical for large discrete action spaces, for a given state $\mathbf{s}$, in our analysis and experiments, we keep the union set $\mathcal{C}$ limited to at most $2\lceil \log(n) \rceil$, ensuring sub-linear (logarithmic) complexity.

### 4.2 Stochastic Q-learning

We introduce Stochastic Q-learning, described in Algorithm 1 in Appendix D, and Stochastic Double Q-learning, described in Algorithm 2 in Appendix D, that replace the $\max$ and $\arg\max$ operations in Q-learning and Double Q-learning with $\text{stoch}\max$ and $\text{stoch}\arg\max$, respectively. Furthermore, we introduce Stochastic Sarsa, described in Algorithm 3 in Appendix D, which replaces the maximization in the greedy action selection ($\arg\max$) in Sarsa.

Our proposed solution takes a distinct approach from the conventional method of selecting the action with the highest Q-function value from the complete set of actions $\mathcal{A}$. Instead, it uses stochastic maximization, which finds a maximum within a stochastic subset $\mathcal{C} \subseteq \mathcal{A}$, constructed as explained in Section 4.1. Our stochastic policy $\pi_Q^S(\mathbf{s})$, uses an $\varepsilon_{\mathbf{s}}$-greedy algorithm, in a given state $\mathbf{s}$, with a probability of $(1-\varepsilon_{\mathbf{s}})$, for $\varepsilon_{\mathbf{s}} > 0$, is defined as follows:

$$\pi_Q^S(\mathbf{s}) \triangleq \begin{cases} \text{play randomly} & \text{with proba. } \epsilon_{\mathbf{s}} \\ \text{stoch}\arg\max_{\mathbf{a}\in\mathcal{A}} Q(\mathbf{s},\mathbf{a}) & \text{otherwise.} \end{cases} \tag{6}$$

Furthermore, during the training, to update the Q-function, our proposed Stochastic Q-learning uses the following rule:

$$Q_{t+1}(\mathbf{s}_t,\mathbf{a}_t) = (1-\alpha_t(\mathbf{s}_t,\mathbf{a}_t))\, Q_t(\mathbf{s}_t,\mathbf{a}_t) + \alpha_t(\mathbf{s}_t,\mathbf{a}_t)\left[r_t + \gamma \underset{b\in\mathcal{A}}{\text{stoch}\max}\, Q_t(\mathbf{s}_{t+1},b)\right]. \tag{7}$$

While Stochastic Q-learning, like Q-learning, employs the same values for action selection and action evaluation, Stochastic Double Q-learning, similar to Double Q-learning, learns two separate Q-functions. For each update, one Q-function determines the policy, while the other determines the value of that policy. Both stochastic learning methods remove the maximization bottleneck from exploration and training updates, making these proposed algorithms significantly faster than their deterministic counterparts.

### 4.3 Stochastic Deep Q-network

We introduce Stochastic DQN (StochDQN), described in Algorithm 4 in Appendix D, and Stochastic DDQN (StochDDQN) as efficient variants of deep Q-networks. These variants substitute the maximization steps in the DQN [35] and DDQN [53] algorithms with the stochastic maximization operations. In these modified approaches, we replace the $\varepsilon_{\mathbf{s}}$-greedy exploration strategy with the same exploration policy as in Eq. (6).

For StochDQN, we employ a deep neural network as a function approximator to estimate the action-value function, represented as $Q(\mathbf{s},\mathbf{a};\theta) \approx Q(\mathbf{s},\mathbf{a})$, where $\theta$ denotes the weights of the Q-network. This network is trained by minimizing a series of loss functions denoted as $L_i(\theta_i)$, with these loss functions changing at each iteration $i$ as follows:

$$L_i(\theta_i) \triangleq \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\rho(\cdot)}\left[\left(y_i - Q(\mathbf{s},\mathbf{a};\theta_i)\right)^2\right], \tag{8}$$

where $y_i \triangleq \mathbb{E}\left[r + \gamma\, \text{stoch}\max_{b\in\mathcal{A}} Q(\mathbf{s}',b;\theta_{i-1}) \middle| \mathbf{s},\mathbf{a}\right]$. In this context, $y_i$ represents the target value for an iteration $i$, and $\rho(.)$ is a probability distribution that covers states and actions. Like the DQN approach, we keep the parameters fixed from the previous iteration, denoted as $\theta_{i-1}$ when optimizing the loss function $L_i(\theta_i)$.

These target values depend on the network weights, which differ from the fixed targets typically used in supervised learning. We employ stochastic gradient descent for the training. While StochDQN, like DQN, employs the same values for action selection and evaluation, StochDDQN, like DDQN, trains two separate value functions. It does this by randomly assigning each experience to update one of the two value functions, resulting in two sets of weights, $\theta$ and $\theta'$. For each update, one set of weights determines the policy, while the other set determines the values.

# 5 Stochastic Maximization Analysis

## 5.1 Memoryless Stochastic Maximization

Memoryless stochastic maximization, i.e., $\mathcal{C} = \mathcal{R} \cup \emptyset$, does not always yield an optimal maximizer. To return an optimal action, this action needs to be randomly sampled from the set of actions. Finding an exact maximizer, without relying on memory $\mathcal{M}$, is a random event with a probability $p$, representing the likelihood of sampling such an exact maximizer. In the following lemma, we provide a lower bound on the probability of discovering an optimal action within a uniformly randomly sampled subset $\mathcal{C} = \mathcal{R}$ of $\lceil \log(n) \rceil$ actions, which we prove in Appendix C.1.1.

**Lemma 5.1.** *For any given state* $\mathbf{s}$*, the probability* $p$ *of sampling an optimal action from a uniformly randomly chosen subset* $\mathcal{C}$ *of size* $\lceil \log(n) \rceil$ *actions is at least* $\frac{\lceil \log(n) \rceil}{n}$.

While finding an exact maximizer through sampling may not always occur, the rewards of near-optimal actions can still be similar to those obtained from an optimal action. Therefore, the difference between stochastic maximization and exact maximization might be a more informative metric than just the probability of finding an exact maximizer. Thus, at time step $t$, given state $\mathbf{s}$ and the current estimated Q-function $Q_t$, we define the estimation error as $\beta_t(\mathbf{s})$, as follows:

$$\beta_t(\mathbf{s}) \triangleq \max_{\mathbf{a} \in \mathcal{A}} Q_t(\mathbf{s}, \mathbf{a}) - \operatorname*{stoch}_{\mathbf{a} \in \mathcal{A}} \max Q_t(\mathbf{s}, \mathbf{a}) . \tag{9}$$

Furthermore, we define the similarity ratio $\omega_t(\mathbf{s})$, as follows:

$$\omega_t(\mathbf{s}) \triangleq \operatorname*{stoch}_{\mathbf{a} \in \mathcal{A}} \max Q_t(\mathbf{s}, \mathbf{a}) \,/\, \max_{\mathbf{a} \in \mathcal{A}} Q_t(\mathbf{s}, \mathbf{a}) . \tag{10}$$

It can be seen from the definitions that $\beta_t(\mathbf{s}) \geq 0$ and $\omega_t(\mathbf{s}) \leq 1$. While sampling the exact maximizer is not always possible, near-optimal actions may yield near-optimal values, providing good approximations, i.e., $\beta_t(\mathbf{s}) \approx 0$ and $\omega_t(\mathbf{s}) \approx 1$. In general, this difference depends on the value distribution over the actions.

While we do not make any specific assumptions about the value distribution in our work, we note that with some simplifying assumptions on the value distributions over the actions, one can derive more specialized guarantees. For example, assuming that the rewards are uniformly distributed over the actions, we demonstrate in Appendix C.3 that for a given discrete state $\mathbf{s}$, if the values of the sampled actions independently follow a uniform distribution from the interval $[Q_t(\mathbf{s}, \mathbf{a}_t^\star) - b_t(\mathbf{s}), Q_t(\mathbf{s}, \mathbf{a}_t^\star)]$, where $b_t(\mathbf{s})$ represents the range of the $Q_t(\mathbf{s}, .)$ values over the actions in state $\mathbf{s}$ at time step $t$, then the expected value of $\beta_t(\mathbf{s})$, even without memory, is: $\mathbb{E}\left[\beta_t(\mathbf{s}) \mid \mathbf{s}\right] \leq \frac{b_t(\mathbf{s})}{\lceil \log(n) \rceil + 1}$. Furthermore, we empirically demonstrate that for the considered control problems, the difference $\beta_t(\mathbf{s})$ is not large, and the ratio $\omega_t(\mathbf{s})$ is close to one, as shown in Section 7.4.

## 5.2 Stochastic Maximization with Memory

While memoryless stochastic maximization could approach the maximum value or find it with the probability $p$, lower-bounded in Lemma 5.1, it does not converge to an exact maximization, as it keeps sampling purely at random, as can be seen in Fig. 5 in Appendix F.2.1. However, memory-based stochastic maximization, i.e., $\mathcal{C} = \mathcal{R} \cup \mathcal{M}$ with $\mathcal{M} \neq \emptyset$, can become an exact maximization when the Q-function becomes stable, as we state in the Corollary 5.3, which we prove in Appendix C.2.1, and as confirmed in Fig. 5.

**Definition 5.2.** A Q-function is considered stable for a given time range and state $\mathbf{s}$ when its maximizing action in that state remains unchanged for all subsequent steps within that time, even if the Q-function's values themselves change.

A straightforward example of a stable Q-function occurs during validation periods when no function updates are performed. However, in general, a stable Q-function does not have to be static and might still vary over the rounds; the critical characteristic is that its maximizing action remains the same even when its values are updated. Although the $\operatorname{stoch}\max$ has sub-linear complexity compared to the $\max$, without any assumption of the value distributions, the following corollary shows that, on average, for a stable Q-function, after a certain number of iterations, the output of the $\operatorname{stoch}\max$ matches precisely the output of $\max$.

**Corollary 5.3.** *For a given state* $\mathbf{s}$*, assuming a time range where the Q-function becomes stable in that state,* $\beta_t(\mathbf{s})$ *is expected to converge to zero after* $\frac{n}{\lceil \log(n) \rceil}$ *iterations.*

Recalling the definition of the similarity ratio $\omega_t$, it follows that $\omega_t(\mathbf{s}) = 1 - \beta(s)/\max_{\mathbf{a}\in\mathcal{A}} Q_t(\mathbf{s},\mathbf{a})$. Therefore, for a given state $\mathbf{s}$, where the Q-function becomes stable, given the boundedness of iterates in Q-learning, it is expected that $\omega_t$ converges to one. This observation was confirmed, even with continuous states and using neural networks as function approximators, in Section 7.4.

## 6 Stochastic Q-learning Convergence

In this section, we analyze the convergence of the Stochastic Q-learning, described in Algorithm 1 in Appendix D. This algorithm employs the policy $\pi_Q^S(\mathbf{s})$, as defined in Eq. (6), with $\varepsilon_{\mathbf{s}} > 0$ to guarantee that $\mathbb{P}_\pi[\mathbf{a}_t = \mathbf{a} \mid \mathbf{s}_t = \mathbf{s}] > 0$ for all state-action pairs $(\mathbf{s},\mathbf{a}) \in \mathcal{S}\times\mathcal{A}$. The value update rule, on the other hand, uses the update rule specified in Eq. (7).

In the convergence analysis, we focus on memoryless maximization. While the $\operatorname{stoch}\arg\max$ operator for action selection can be employed with or without memory, we assume a memoryless $\operatorname{stoch}\max$ operator for value updates, which means that value updates are performed by maximizing over a randomly sampled subset of actions from $\mathcal{A}$, sampled independently from both the next state $\mathbf{s}'$ and the set used for the $\operatorname{stoch}\arg\max$.

For a stochastic variable subset of actions $\mathcal{C} \subseteq \mathcal{A}$, following some probability distribution $\mathbb{P} : 2^{\mathcal{A}} \to [0,1]$, we consider, without loss of generality $Q(.,\emptyset) = 0$, and define, according to $\mathbb{P}$, a target Q-function, denoted as $Q^*$, as:

$$Q^*(\mathbf{s},\mathbf{a}) \triangleq \mathbb{E}\left[ r(\mathbf{s},\mathbf{a}) + \gamma \max_{b\in\mathcal{C}\sim\mathbb{P}} Q^*(\mathbf{s}',b) \mid \mathbf{s},\mathbf{a} \right]. \tag{11}$$

*Remark* 6.1. The $Q^*$ defined above depends on the sampling distribution $\mathbb{P}$. Therefore, it does not represent the optimal value function of the original MDP problem; instead, it is optimal under the condition where only a random subset of actions following the distribution $\mathbb{P}$ is available to the agent at each time step. However, as the sampling cardinality increases, it increasingly better approximates the optimal value function of the original MDP and fully recovers the optimal Q-function of the original problem when the sampling distribution becomes $\mathbb{P}(\mathcal{A}) = 1$.

The following theorem states the convergence of the iterates $Q_t$ of Stochastic Q-learning with memoryless stochastic maximization to the $Q^*$, defined in Eq. (11), for any sampling distribution $\mathbb{P}$, regardless of the cardinality.

**Theorem 6.2.** *For a finite MDP, as described in Section 3, let* $\mathcal{C}$ *be a randomly independently sampled subset of actions from* $\mathcal{A}$*, of any cardinality, following any distribution* $\mathbb{P}$*, exclusively sampled for the value updates, for the Stochastic Q-learning, given by the following update rule:*

$$Q_{t+1}(\mathbf{s}_t,\mathbf{a}_t) = (1-\alpha_t(\mathbf{s}_t,\mathbf{a}_t))Q_t(\mathbf{s}_t,\mathbf{a}_t) + \alpha_t(\mathbf{s}_t,\mathbf{a}_t)\left[ r_t + \gamma \max_{b\in\mathcal{C}\sim\mathbb{P}} Q_t(\mathbf{s}_{t+1},\mathbf{a}) \right],$$

*given any initial estimate* $Q_0$*,* $Q_t$ *converges with probability 1 to* $Q^*$*, defined in Eq. (11), as long as* $\sum_t \alpha_t(\mathbf{s},\mathbf{a}) = \infty$ *and* $\sum_t \alpha_t^2(\mathbf{s},\mathbf{a}) < \infty$ *for all* $(\mathbf{s},\mathbf{a}) \in \mathcal{S}\times\mathcal{A}$*.*

The theorem's result demonstrates that for any cardinality of actions, Stochastic Q-learning converges to $Q^*$, as defined in Eq. (11), which recovers the convergence guarantees of Q-learning when the sampling distribution is $\mathbb{P}(\mathcal{A}) = 1$.

*Remark* 6.3. In principle, any size can be used, balancing time complexity and approximation. Our empirical experiments focused on $\log(n)$ to illustrate the method's ability to recover Q-learning, even with a few actions. Using $\sqrt{n}$ will approach the value function of Q-learning more closely compared to using $\log(n)$, albeit at the cost of higher complexity than $\log(n)$.

The theorem shows that even with memoryless stochastic maximization, using randomly sampled $\mathcal{O}(\log(n))$ actions, the convergence is still guaranteed. However, relying on memory-based stochastic maximization helps minimize the approximation error in stochastic maximization, as shown in Corollary 5.3, and outperforms Q-learning as shown in the experiments in Section 7.1.

The full proof is provided in Appendix B.

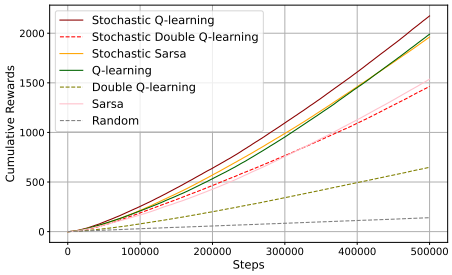



Figure 1: Comparison of stochastic vs. non-stochastic value-based variants on the FrozenLake-v1, with steps on the x-axis and cumulative rewards on the y-axis.

# 7 Experiments

We compare stochastic maximization to exact maximization and evaluate the proposed RL algorithms in Gymnasium [3] and MuJoCo [51] environments. The stochastic tabular Q-learning approaches are tested on CliffWalking-v0, FrozenLake-v1, and a generated MDP environment. Additionally, the stochastic deep Q-network approaches are tested on control tasks and compared against their deterministic counterparts, as well as against DDPG [29], A2C [34], and PPO [43], using Stable-Baselines implementations [21], which can directly handle continuous action spaces. Further details can be found in Appendix E.

## 7.1 Stochastic Q-learning Average Return

We test Stochastic Q-learning, Stochastic Double Q-learning, and Stochastic Sarsa in environments with discrete states and actions. Interestingly, as shown in Fig. 1, our stochastic algorithms outperform their deterministic counterparts. Furthermore, we observe that Stochastic Q-learning outperforms all the methods considered regarding the cumulative rewards in the FrozenLake-v1. Moreover, in the CliffWalking-v0 (as shown in Fig. 9), as well as for the generated MDP environment with 256 actions (as shown in Fig. 11), all the stochastic and non-stochastic methods reach the optimal policy in a similar number of steps.

## 7.2 Exponential Wall Time Speedup

Stochastic maximization methods exhibit logarithmic complexity regarding the number of actions. Therefore, StochDQN and StochDDQN, which apply these techniques for action selection and updates, have exponentially faster execution times than DQN and DDQN, as confirmed in Fig. 2. For the time duration of action selection alone, please refer to Appendix F.1. The time analysis results show that the proposed methods are nearly as fast as a random algorithm that selects actions randomly. Specifically, in the experiments with the InvertedPendulum-v4, the stochastic methods took around 0.003 seconds per step for a set of 1000 actions, while the non-stochastic methods took 0.18 seconds, which indicates that the stochastic versions are 60 times faster than their deterministic counterparts. Furthermore, for the HalfCheetah-v4 experiment, we considered 4096 actions, where one (D)DQN step takes 0.6 seconds, needing around 17 hours to run for 100,000 steps, while the Stoch(D)DQN needs around 2 hours to finish the same 100,000 steps. In other words, we can easily run for 10x more steps in the same period (seconds). This makes the stochastic methods more practical, especially with large action spaces.

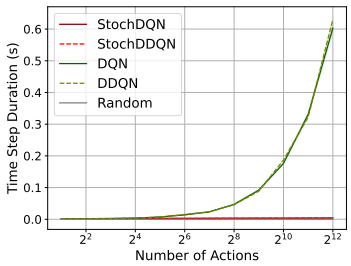



Figure 2: Comparison of wall time in seconds of stochastic and non-stochastic DQN methods on various action set sizes.

## 7.3 Stochastic Deep Q-network Average Return

Fig. 8a shows the performance of various RL algorithms on the InvertedPendulum-v4 task, which has 512 actions. StochDQN achieves the optimal average return in fewer steps than DQN, with a lower

per-step time advantage (as shown in Section 7.2). Interestingly, while DDQN struggles, StochDDQN nearly reaches the optimal average return, demonstrating the effectiveness of stochasticity. StochDQN and StochDDQN significantly outperform DDQN, A2C, and PPO by obtaining higher average returns in fewer steps. Similarly, Fig. 8b in Appendix F.3 shows the results for the HalfCheetah-v4 task, which has 4096 actions. Stochastic methods, particularly StochDDQN, achieve results comparable to the non-stochastic methods. Notably, all DQN methods (stochastic and non-stochastic) outperform PPO and A2C, highlighting their efficiency in such scenarios.

*Remark* 7.1. While comparing them falls outside the scope of our work, we note that DDQN was proposed to mitigate the inherent overestimation in DQN. However, exchanging overestimation for underestimation bias is not always beneficial, as our results demonstrate and as shown in other studies such as [27].

### 7.4 Stochastic Maximization

This section analyzes stochastic maximization by tracking returned values of $\omega_t$ (Eq. (10)) across the steps. As shown in Fig. 3, for StochDQN, for the InvertedPendulum-v4, $\omega_t$ approaches one rapidly, similarly for the HalfCheetah-v4, as shown in Appendix F.2.2. Furthermore, we track the returned values of the difference $\beta_t$ (Eq. (9)) and show that it is bounded by small values in both environments, as illustrated in Appendix F.2.2.

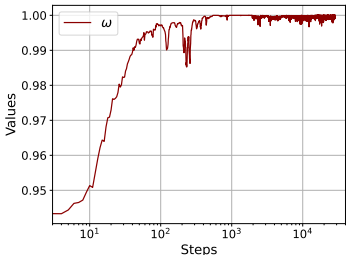



Figure 3: The $\mathrm{stoch}\max$ and $\max$ ratio values tracked over the steps on the InvertedPendulum-v4.

## 8 Discussion

In this work, we focus on adapting value-based methods, which excel in generalization compared to actor-based approaches [7]. However, this advantage comes at the cost of lower computational efficiency due to the maximization operation required for action selection and value function updates. Therefore, our primary motivation is to provide a computationally efficient alternative for situations with general large discrete action spaces.

While the primary goal of this work is to reduce the complexity and wall time of Q-learning-like algorithms, our experiments revealed that stochastic methods not only achieve shorter step times (in seconds) but also, in some cases, yield higher rewards and exhibit faster convergence in terms of the number of steps compared to other methods. These improvements can be attributed to several factors. Firstly, introducing more stochasticity into the greedy choice through $\mathrm{stoch}\arg\max$ enhances exploration. Secondly, Stochastic Q-learning specifically helps to reduce the inherent overestimation in Q-learning-like methods [17, 27, 56]. This reduction is achieved using $\mathrm{stoch}\max$, a lower bound to the $\max$ operation.

Q-learning methods, focused initially on discrete actions, can be adapted to tackle continuous problems with discretization techniques and stochastic maximization. Our control experiments show that Q-network methods with discretization achieve superior performance to algorithms with continuous actions, such as PPO, by obtaining higher rewards in fewer steps, which aligns with observations in previous works that highlight the potential of discretization for solving continuous control problems [7, 48, 47]. Notably, the logarithmic complexity of the proposed stochastic methods concerning the number of considered actions makes them well-suited for scenarios with finer-grained discretization, leading to more practical implementations.

## 9 Conclusion

We propose adapting Q-learning-like methods to mitigate the computational bottleneck associated with the $\max$ and $\arg\max$ operations in these methods. By reducing the maximization complexity from linear to sublinear using $\mathrm{stoch}\max$ and $\mathrm{stoch}\arg\max$, we pave the way for practical and efficient value-based RL for large discrete action spaces. We prove the convergence of Stochastic Q-learning, analyze stochastic maximization, and empirically show that it performs well with significantly low complexity.

## References


[1] Fabian Akkerman, Julius Luy, Wouter van Heeswijk, and Maximilian Schiffer. Handling large discrete action spaces via dynamic neighborhood construction. *arXiv preprint arXiv:2305.19891*, 2023.

[2] Abubakr O Al-Abbasi, Arnob Ghosh, and Vaneet Aggarwal. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4714–4727, 2019.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] Hao Cui and Roni Khardon. Online symbolic gradient-based optimization for factored action mdps. In *IJCAI*, pages 3075–3081, 2016.

[5] Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, 33:609–620, 2020.

[6] Gabriel Dulac-Arnold, Ludovic Denoyer, Philippe Preux, and Patrick Gallinari. Fast reinforcement learning with large action sets using error-correcting output codes for mdp factorization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 180–194. Springer, 2012.

[7] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[8] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

[9] Tobias Enders, James Harrison, Marco Pavone, and Maximilian Schiffer. Hybrid multi-agent deep reinforcement learning for autonomous mobility on demand systems. In *Learning for Dynamics and Control Conference*, pages 1284–1296. PMLR, 2023.

[10] Gregory Farquhar, Laura Gustafson, Zeming Lin, Shimon Whiteson, Nicolas Usunier, and Gabriel Synnaeve. Growing action spaces. In *International Conference on Machine Learning*, pages 3040–3051. PMLR, 2020.

[11] Fares Fourati, Vaneet Aggarwal, Christopher Quinn, and Mohamed-Slim Alouini. Randomized greedy learning for non-monotone stochastic submodular maximization under full-bandit feedback. In *International Conference on Artificial Intelligence and Statistics*, pages 7455–7471. PMLR, 2023.

[12] Fares Fourati and Mohamed-Slim Alouini. Artificial intelligence for satellite communication: A review. *Intelligent and Converged Networks*, 2(3):213–243, 2021.

[13] Fares Fourati, Mohamed-Slim Alouini, and Vaneet Aggarwal. Federated combinatorial multi-agent multi-armed bandits. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 13760–13782. PMLR, 21–27 Jul 2024.

[14] Fares Fourati, Christopher John Quinn, Mohamed-Slim Alouini, and Vaneet Aggarwal. Combinatorial stochastic-greedy bandit. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 12052–12060, 2024.

[15] Glebys Gonzalez, Mythra Balakuntala, Mridul Agarwal, Tomas Low, Bruce Knoth, Andrew W Kirkpatrick, Jessica McKee, Gregory Hager, Vaneet Aggarwal, Yexiang Xue, et al. Asap: A semi-autonomous precise system for telesurgery during communication delays. *IEEE Transactions on Medical Robotics and Bionics*, 5(1):66–78, 2023.

[16] Marina Haliem, Ganapathy Mani, Vaneet Aggarwal, and Bharat Bhargava. A distributed model-free ride-sharing approach for joint matching, pricing, and dispatching using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 22(12):7931–7942, 2021.

[17] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[18] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with an unbounded action space. *arXiv preprint arXiv:1511.04636*, 5, 2015.

[19] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1621–1630, Berlin, Germany, August 2016. Association for Computational Linguistics.

[20] Ji He, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng. Deep reinforcement learning with a combinatorial action space for predicting popular Reddit threads. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1838–1848, Austin, Texas, November 2016. Association for Computational Linguistics.

[21] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[22] David Ireland and Giovanni Montana. Revalued: Regularised ensemble value-decomposition for factorisable markov decision processes. *arXiv preprint arXiv:2401.08850*, 2024.

[23] Tommi Jaakkola, Michael Jordan, and Satinder Singh. Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems*, 6, 1993.

[24] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[25] Minsu Kim, Jinkyoo Park, et al. Learning collaborative policies to solve np-hard routing problems. *Advances in Neural Information Processing Systems*, 34:10418–10430, 2021.

[26] Michail G Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 424–431, 2003.

[27] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*, 2020.

[28] Shurong Li, Chong Wei, and Ying Wang. Combining decision making and trajectory planning for lane changing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):16110–16136, 2022.

[29] 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.

[30] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(4):3133–3174, 2019.

[31] Anuj Mahajan, Mikayel Samvelyan, Lei Mao, Viktor Makoviychuk, Animesh Garg, Jean Kossaifi, Shimon Whiteson, Yuke Zhu, and Animashree Anandkumar. Reinforcement learning in factored action spaces using tensor decompositions. *arXiv preprint arXiv:2110.14538*, 2021.

[32] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

[33] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

[34] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

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

[36] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[37] Jason Pazis and Ron Parr. Generalized value functions for large action sets. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1185–1192, 2011.

[38] Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34:12208–12221, 2021.

[39] Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291. IEEE, 2018.

[40] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1:127–190, 1999.

[41] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[42] Brian Sallans and Geoffrey E Hinton. Reinforcement learning with factored states and actions. *The Journal of Machine Learning Research*, 5:1063–1088, 2004.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] Tim Seyde, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin Riedmiller, Markus Wulfmeier, and Daniela Rus. Is bang-bang control all you need? solving continuous control with bernoulli policies. *Advances in Neural Information Processing Systems*, 34:27209–27221, 2021.

[45] Tim Seyde, Peter Werner, Wilko Schwarting, Igor Gilitschenski, Martin Riedmiller, Daniela Rus, and Markus Wulfmeier. Solving continuous control via q-learning. *arXiv preprint arXiv:2210.12566*, 2022.

[46] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[47] Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pages 5981–5988, 2020.

[48] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 32, 2018.

[49] Guy Tennenholtz and Shie Mannor. The natural language of actions. In *International Conference on Machine Learning*, pages 6196–6205. PMLR, 2019.

[50] Chen Tessler, Tom Zahavy, Deborah Cohen, Daniel J Mankowitz, and Shie Mannor. Action assembly: Sparse imitation learning for text based games with combinatorial action spaces. *arXiv preprint arXiv:1905.09700*, 2019.

[51] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[52] Tom Van de Wiele, David Warde-Farley, Andriy Mnih, and Volodymyr Mnih. Q-learning in enormous action spaces via amortized approximate maximization. *arXiv preprint arXiv:2001.08116*, 2020.

[53] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[54] Hado Van Hasselt and Marco A Wiering. Using continuous action spaces to solve discrete problems. In *2009 International Joint Conference on Neural Networks*, pages 1149–1156. IEEE, 2009.

[55] Gongju Wang, Dianxi Shi, Chao Xue, Hao Jiang, and Yajie Wang. Bic-ddpg: Bidirectionally-coordinated nets for deep multi-agent reinforcement learning. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 337–354. Springer, 2020.

[56] Hang Wang, Sen Lin, and Junshan Zhang. Adaptive ensemble q-learning: Minimizing estimation bias via error feedback. *Advances in Neural Information Processing Systems*, 34:24778–24790, 2021.

[57] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

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

[59] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[60] Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Generating adjacency-constrained subgoals in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21579–21590, 2020.

[61] Zongzhang Zhang, Zhiyuan Pan, and Mykel J Kochenderfer. Weighted double q-learning. In *IJCAI*, pages 3455–3461, 2017.

# A Related Works

While RL has shown promise in diverse domains, practical applications often grapple with real-world complexities. A significant hurdle arises when dealing with large discrete action spaces [7, 8]. Previous research has investigated strategies to address this challenge by leveraging the combinatorial or the dimensional structures in the action space [20, 48, 50, 5, 44, 45, 11, 14, 13, 1, 14, 13, 22]. For example, [20] leveraged the combinatorial structure of their language problem through sub-action embeddings. Compressed sensing was employed in [50] for text-based games with combinatorial actions. [5] formulated the combinatorial action decision of a vehicle routing problem as a mixed-integer program. Moreover, [1] introduced dynamic neighbourhood construction specifically for structured combinatorial large discrete action spaces. Previous works tailored solutions for multi-dimensional spaces such as those in [44, 45, 22], among others, while practical in the multi-dimensional spaces, may not be helpful for single-dimensional large action spaces. While relying on the structure of the action space is practical in some settings, not all problems with large action spaces are multi-dimensional or structured. We complement these works by making no assumptions about the structure of the action space.

Some approaches have proposed factorizing the action spaces to reduce their size. For example, these include factorizing into binary subspaces [26, 42, 37, 6], expert demonstration [49], tensor factorization [31], and symbolic representations [4]. Additionally, some hierarchical and multi-agent RL approaches employed factorization as well [60, 25, 38, 9]. While some of these methods effectively handle large action spaces for certain problems, they necessitate the design of a representation for each discrete action. Even then, for some problems, the resulting space may still be large.

Methods presented in [54, 7, 55] combine continuous-action policy gradients with nearest neighbour search to generate continuous actions and identify the nearest discrete actions. Although these methods have shown good performance on different tasks, they require continuous-to-discrete mapping and are mainly policy-based rather than value-based approaches. In the works of [24] and [39], the cross-entropy method [40] was utilized to approximate action maximization. This approach requires multiple iterations ($r$) for a single action selection. During each iteration, it samples $n'$ values, where $n' < n$, fits a Gaussian distribution to $m < n'$ of these samples, and subsequently draws a new batch of $n'$ samples from this Gaussian distribution. As a result, this approximation remains costly, with a complexity of $\mathcal{O}(rn')$. Additionally, in the work of [52], a neural network was trained to predict the optimal action in combination with a uniform search. This approach involves the use of an expensive autoregressive proposal distribution to generate $n'$ actions and samples a large number of actions ($m$), thus remaining computationally expensive, with $\mathcal{O}(n' + m)$. In [33], sequential DQN allows the agent to choose sub-actions one by one, which increases the number of steps needed to solve a problem and requires $d$ steps with a linear complexity of $\mathcal{O}(i)$ for a discretization granularity $i$. Additionally, [48] employs a branching technique with duelling DQN for combinatorial control problems. Their approach has a complexity of $\mathcal{O}(di)$ for actions with discretization granularity $i$ and $d$ dimensions, whereas our method, in a similar setting, achieves $\mathcal{O}(d \log(i))$. Another line of work introduces action elimination techniques, such as the action elimination DQN [59], which employs an action elimination network guided by an external elimination signal from the environment. However, it requires this domain-specific signal and can be computationally expensive ($\mathcal{O}(n')$ where $n' \leq n$ are the number of remaining actions). In contrast, curriculum learning, as proposed by [10], initially limits an agent's action space, gradually expanding it during training for efficient exploration. However, its effectiveness relies on having an informative restricted action space, and as the action space size grows, its complexity scales linearly with its size, eventually reaching $\mathcal{O}(n)$.

In the context of combinatorial bandits with a single state but large discrete action spaces, previous works have exploited the combinatorial structure of actions, where each action is a subset of main arms. For instance, for submodular reward functions, which imply diminishing returns when adding arms, in [11] and [14], stochastic greedy algorithms are used to avoid exact search. The former evaluates the marginal gains of adding and removing sub-actions (arms), while the latter assumes monotonic rewards and considers adding the best arm until a cardinality constraint is met. For general reward functions, [13] propose using approximation algorithms to evaluate and add sub-actions. While these methods are practical for bandits, they exploit the combinatorial structure of their problems and consider a single-state scenario, which is different from general RL problems.

While some approaches above are practical for handling specific problems with large discrete action spaces, they often exploit the dimensional or combinatorial structures inherent in their considered

problems. In contrast, we complement these approaches by proposing a solution to tackle any general, potentially unstructured, single-dimensional or multi-dimensional, large discrete action space without relying on structure assumptions. Our proposed solution is general, simple, and efficient.

# B Stochastic Q-learning Convergence Proofs

In this section, we prove Theorem 6.2, which states the convergence of Stochastic Q-learning. This algorithm uses a stochastic policy for action selection, employing a $\mathrm{stoch}\arg\max$ with or without memory, possibly dependent on the current state $\mathbf{s}$. For value updates, it utilizes a $\mathrm{stoch}\max$ without memory, independent of the following state $\mathbf{s}'$.

## B.1 Lemma B.1

For the transition probability distribution $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, the set probability distribution $\mathbb{P} : 2^{\mathcal{A}} \to [0, 1]$, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and the discount factor, $\gamma \in [0, 1]$, we define the following contraction operator $\Phi$, defined for a function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as

$$(\Phi q)(\mathbf{s}, \mathbf{a}) = \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left[ r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} q(\mathbf{s}', b) \right]. \tag{12}$$

**Lemma B.1.** *The operator $\Phi$, defined in Eq.* (12)*, is a contraction in the sup-norm, with a contraction factor $\gamma$, i.e., $\|\Phi q_1 - \Phi q_2\|_\infty \leq \gamma \|q_1 - q_2\|_\infty$.*

*Proof.* For the transition probability distribution $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, the set probability distribution $\mathbb{P}$ defined over the combinatorial space of actions, i.e., $\mathbb{P} : 2^{\mathcal{A}} \to [0, 1]$, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and the discount factor $\gamma \in [0, 1]$, for a function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the operator $\Phi$ is defined as follows:

$$(\Phi q)(\mathbf{s}, \mathbf{a}) = \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left[ r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} q(\mathbf{s}', b) \right]. \tag{13}$$

Therefore,

$$\begin{aligned}
\|\Phi q_1 - \Phi q_2\|_\infty &= \max_{\mathbf{s},\mathbf{a}} \left| \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left[ r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} q_1(\mathbf{s}', b) - r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} q_2(\mathbf{s}', b) \right] \right| \\
&= \max_{\mathbf{s},\mathbf{a}} \gamma \left| \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left[ \max_{b \in \mathcal{C}} q_1(\mathbf{s}', b) - \max_{b \in \mathcal{C}} q_2(\mathbf{s}', b) \right] \right| \\
&\leq \max_{\mathbf{s},\mathbf{a}} \gamma \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left| \max_{b \in \mathcal{C}} q_1(\mathbf{s}', b) - \max_{b \in \mathcal{C}} q_2(\mathbf{s}', b) \right| \\
&\leq \max_{\mathbf{s},\mathbf{a}} \gamma \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \max_{z,b} |q_1(z, b) - q_2(z, b)| \\
&\leq \max_{\mathbf{s},\mathbf{a}} \gamma \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \|q_1 - q_2\|_\infty \\
&= \gamma \|q_1 - q_2\|_\infty .
\end{aligned}$$

□

## B.2 Proof of Theorem 6.2

*Proof.* Stochastic Q-learning employs a stochastic policy in a given state $\mathbf{s}$, which use $\mathrm{stoch}\arg\max$ operation, with or without memory $\mathcal{M}$, with probability $(1-\varepsilon_{\mathbf{s}})$, for $\varepsilon_{\mathbf{s}} > 0$, which can be summarized by the following equation:

$$\pi_Q^S(\mathbf{s}) = \begin{cases} \text{play randomly} & \text{with probability } \epsilon_{\mathbf{s}} \\ \mathrm{stoch}\arg\max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a}) & \text{otherwise .} \end{cases} \tag{14}$$

This policy, with $\varepsilon_{\mathbf{s}} > 0$, ensures that $\mathbb{P}_\pi[\mathbf{a}_t = \mathbf{a} \mid \mathbf{s}_t = \mathbf{s}] > 0$ for all $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$.

Furthermore, during the training, to update the Q-function, given any initial estimate $Q_0$, we consider a Stochastic Q-learning which uses $\mathrm{stoch}\max$ operation as in the following stochastic update rule:

$$Q_{t+1}(\mathbf{s}_t, \mathbf{a}_t) = (1 - \alpha_t(\mathbf{s}_t, \mathbf{a}_t)) Q_t(\mathbf{s}_t, \mathbf{a}_t) + \alpha_t(\mathbf{s}_t, \mathbf{a}_t) \left[ r_t + \gamma \operatorname*{stoch\,max}_{b \in \mathcal{A}} Q_t(\mathbf{s}_{t+1}, b) \right]. \quad (15)$$

For the function updates, we consider a $\mathrm{stoch}\max$ without memory, which involves a $\max$ over a random subset of action $\mathcal{C}$ sampled from a set probability distribution $\mathbb{P}$ defined over the combinatorial space of actions, i.e., $\mathbb{P} : 2^{\mathcal{A}} \to [0, 1]$, which can be a uniform distribution over the action sets of size $\lceil \log(n) \rceil$.

Hence, for a random subset of actions $\mathcal{C}$, the update rule of Stochastic Q-learning can be written as:

$$Q_{t+1}(\mathbf{s}_t, \mathbf{a}_t) = (1 - \alpha_t(\mathbf{s}_t, \mathbf{a}_t)) Q_t(\mathbf{s}_t, \mathbf{a}_t) + \alpha_t(\mathbf{s}_t, \mathbf{a}_t) \left[ r_t + \gamma \max_{b \in \mathcal{C}} Q_t(\mathbf{s}_{t+1}, b) \right]. \quad (16)$$

We define an optimal Q-function, denoted as $Q^*$, as follows:

$$Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}\left[ r(\mathbf{s}, \mathbf{a}) + \gamma \operatorname*{stoch\,max}_{b \in \mathcal{A}} Q^*(\mathbf{s}', b) \mid \mathbf{s}, \mathbf{a} \right] \quad (17)$$

$$= \mathbb{E}\left[ r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} Q^*(\mathbf{s}', b) \mid \mathbf{s}, \mathbf{a} \right]. \quad (18)$$

Subtracting from both sides $Q^*(\mathbf{s}_t, \mathbf{a}_t)$ and letting

$$\Delta_t(\mathbf{s}, \mathbf{a}) = Q_t(\mathbf{s}, \mathbf{a}) - Q^*(\mathbf{s}, \mathbf{a}), \quad (19)$$

yields

$$\Delta_{t+1}(\mathbf{s}_t, \mathbf{a}_t) = (1 - \alpha_t(\mathbf{s}_t, \mathbf{a}_t)) \Delta_t(\mathbf{s}_t, \mathbf{a}_t) + \alpha_t(\mathbf{s}_t, \mathbf{a}_t) F_t(\mathbf{s}_t, \mathbf{a}_t), \quad (20)$$

with

$$F_t(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} Q_t(\mathbf{s}', b) - Q^*(\mathbf{s}, \mathbf{a}). \quad (21)$$

For the transition probability distribution $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, the set probability distribution $\mathbb{P} : 2^{\mathcal{A}} \to [0, 1]$, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and the discount factor, $\gamma \in [0, 1]$, we define the following contraction operator $\Phi$, defined for a function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as

$$(\Phi q)(\mathbf{s}, \mathbf{a}) = \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left[ r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} q(\mathbf{s}', b) \right]. \quad (22)$$

Therefore, with $\mathcal{F}_t$ representing the past at time step $t$,

$$\begin{aligned} \mathbb{E}[F_t(\mathbf{s}, \mathbf{a}) \mid \mathcal{F}_t] &= \sum_{\mathcal{C} \in 2^{\mathcal{A}}} \mathbb{P}(\mathcal{C}) \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left[ r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} Q_t(\mathbf{s}', b) - Q^*(\mathbf{s}, \mathbf{a}) \right] \\ &= (\Phi(Q_t))(\mathbf{s}, \mathbf{a}) - Q^*(\mathbf{s}, \mathbf{a}). \end{aligned}$$

Using the fact that $Q^* = \Phi Q^*$,

$$\mathbb{E}[F_t(\mathbf{s}, \mathbf{a}) \mid \mathcal{F}_t] = (\Phi Q_t)(\mathbf{s}, \mathbf{a}) - (\Phi Q^*)(\mathbf{s}, \mathbf{a}).$$

It is now immediate from Lemma B.1, which we prove in Appendix B.1, that

$$\|\mathbb{E}[F_t(\mathbf{s}, \mathbf{a}) \mid \mathcal{F}_t]\|_\infty \leq \gamma \|Q_t - Q^*\|_\infty = \gamma \|\Delta_t\|_\infty. \quad (23)$$

Moreover,

$$\begin{aligned} \mathrm{var}[F_t(\mathbf{s}, \mathbf{a}) \mid \mathcal{F}_t] &= \mathbb{E}\left[ \left( r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} Q_t(\mathbf{s}', b) - Q^*(\mathbf{s}, \mathbf{a}) - (\Phi Q_t)(\mathbf{s}, \mathbf{a}) + Q^*(\mathbf{s}, \mathbf{a}) \right)^2 \mid \mathcal{F}_t \right] \\ &= \mathbb{E}\left[ \left( r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} Q_t(\mathbf{s}', b) - (\Phi Q_t)(\mathbf{s}, \mathbf{a}) \right)^2 \mid \mathcal{F}_t \right] \\ &= \mathrm{var}\left[ r(\mathbf{s}, \mathbf{a}) + \gamma \max_{b \in \mathcal{C}} Q_t(\mathbf{s}', b) \mid \mathcal{F}_t \right] \\ &= \mathrm{var}[r(\mathbf{s}, \mathbf{a}) \mid \mathcal{F}_t] + \gamma^2 \mathrm{var}\left[ \max_{b \in \mathcal{C}} Q_t(\mathbf{s}', b) \mid \mathcal{F}_t \right] + 2\gamma \,\mathrm{cov}(r(\mathbf{s}, \mathbf{a}), \max_{b \in \mathcal{C}} Q_t(\mathbf{s}', b) \mid \mathcal{F}_t) \\ &= \mathrm{var}[r(\mathbf{s}, \mathbf{a}) \mid \mathcal{F}_t] + \gamma^2 \mathrm{var}\left[ \max_{b \in \mathcal{C}} Q_t(\mathbf{s}', b) \mid \mathcal{F}_t \right]. \end{aligned}$$

The last line follows from the fact that the randomness of $\max_{b\in\mathcal{C}} Q_t(\mathbf{s}', b) \mid \mathcal{F}_t$ only depends on the random set $\mathcal{C}$ and the next state $\mathbf{s}'$. Moreover, we consider the reward $r(\mathbf{s}, \mathbf{a})$ independent of the set $\mathcal{C}$ and the next state $\mathbf{s}'$, by not using the same set $\mathcal{C}$ for both the action selection and the value update.

Given that $r$ is bounded, its variance is bounded by some constant $B$. Therefore,

$$\begin{aligned}
\text{var}\left[F_t(\mathbf{s}, \mathbf{a}) \mid \mathcal{F}_t\right] &\leq B + \gamma^2 \text{ var}\left[\max_{b\in\mathcal{C}} Q_t(\mathbf{s}', b) \mid \mathcal{F}_t\right] \\
&= B + \gamma^2 \mathbb{E}\left[(\max_{b\in\mathcal{C}} Q_t(\mathbf{s}', b))^2 \mid \mathcal{F}_t\right] - \gamma^2 \mathbb{E}\left[\max_{b\in\mathcal{C}} Q_t(\mathbf{s}', b) \mid \mathcal{F}_t\right]^2 \\
&\leq B + \gamma^2 \mathbb{E}\left[(\max_{b\in\mathcal{C}} Q_t(\mathbf{s}', b))^2 \mid \mathcal{F}_t\right] \\
&\leq B + \gamma^2 \mathbb{E}\left[(\max_{\mathbf{s}'\in\mathcal{S}} \max_{b\in\mathcal{A}} Q_t(\mathbf{s}', b))^2 \mid \mathcal{F}_t\right] \\
&\leq B + \gamma^2 (\max_{\mathbf{s}'\in\mathcal{S}} \max_{b\in\mathcal{A}} Q_t(\mathbf{s}', b))^2 \\
&= B + \gamma^2 \|Q_t\|_\infty^2 \\
&= B + \gamma^2 \|\Delta_t + Q^*\|_\infty^2 \\
&\leq B + \gamma^2 \|Q^*\|_\infty^2 + \gamma^2 \|\Delta_t\|_\infty^2 \\
&\leq (B + \gamma^2 \|Q^*\|_\infty^2)(1 + \|\Delta_t\|_\infty^2) + \gamma^2 (1 + \|\Delta_t\|_\infty^2) \\
&\leq \max\{B + \gamma^2 \|Q^*\|_\infty^2, \gamma^2\}(1 + \|\Delta_t\|_\infty^2) \\
&\leq \max\{B + \gamma^2 \|Q^*\|_\infty^2, \gamma^2\}(1 + \|\Delta_t\|_\infty)^2.
\end{aligned}$$

Therefore, for constant $C = \max\{B + \gamma^2 \|Q^*\|_\infty^2, \gamma^2\}$,

$$\text{var}\left[F_t(\mathbf{s}, \mathbf{a}) \mid \mathcal{F}_t\right] \leq C(1 + \|\Delta_t\|_\infty)^2. \tag{24}$$

Then, by Eq. (23), Eq. (24), and Theorem 1 in [23], $\Delta_t$ converges to zero with probability 1, i.e., $Q_t$ converges to $Q^*$ with probability 1. □

# C Stochastic Maximization

We analyze the proposed stochastic maximization method by comparing its error to that of exact maximization. First, we consider the case without memory, where $\mathcal{C} = \mathcal{R}$, and then the case with memory, where $\mathcal{M} \neq \emptyset$. Finally, we provide a specialized bound for the case where the action values follow a uniform distribution.

## C.1 Memoryless Stochastic Maximization

In the following lemma, we give a lower bound on the probability of finding an optimal action within a uniformly sampled subset $\mathcal{R}$ of $\lceil \log(n) \rceil$ actions. We prove that for a given state $s$, the probability $p$ of sampling an optimal action within the uniformly randomly sampled subset $\mathcal{R}$ of size $\lceil \log(n) \rceil$ actions is lower bounded with $p \geq \frac{\lceil \log(n) \rceil}{n}$.

### C.1.1 Proof of Lemma 5.1

*Proof.* In the presence of multiple maximizers, we focus on one of them, denoted as $\mathbf{a}_0^*$, and then the probability $p$ of sampling at least one maximizer is lower-bounded by the probability $p_{\mathbf{a}_0^*}$ of finding $\mathbf{a}_0^*$, i.e.,

$$p \geq p_{\mathbf{a}_0^*}.$$

The probability $p_{\mathbf{a}_0^*}$ of finding $\mathbf{a}_0^*$ is the probability of sampling $\mathbf{a}_0^*$ within the random set $\mathcal{R}$ of size $\lceil \log(n) \rceil$, which is the fraction of all possible combinations of size $\lceil \log(n) \rceil$ that include $\mathbf{a}_0^*$.

This fraction can be calculated as $\binom{n-1}{\lceil \log(n) \rceil - 1}$ divided by all possible combinations of size $\lceil \log(n) \rceil$, which is $\binom{n}{\lceil \log(n) \rceil}$.

Therefore, $p_{\mathbf{a}_0^*} = \frac{\binom{n-1}{\lceil \log(n) \rceil - 1}}{\binom{n}{\lceil \log(n) \rceil}}$.

Consequently,

$$p \geq \frac{\lceil \log(n) \rceil}{n}. \tag{25}$$

□

## C.2 Stochastic Maximization with Memory

While stochastic maximization without memory could approach the maximum value or find it with the probability $p$, lower-bounded in Lemma 5.1, it never converges to an exact maximization, as it keeps sampling purely at random, as can be seen in Fig. 5. However, stochastic maximization with memory can become an exact maximization when the Q-function becomes stable, which we prove in the following Corollary. Although the $\mathrm{stoch}\,\max$ has sub-linear complexity compared to the max, the following Corollary shows that, on average, for a stable Q-function, after a certain number of iterations, the output of the $\mathrm{stoch}\,\max$ matches the output of max.

**Definition C.1.** A Q-function is considered stable for a given state $\mathbf{s}$ if its best action in that state remains unchanged for all subsequent steps, even if the Q-function's values themselves change.

A straightforward example of a stable Q-function occurs during validation periods when no function updates are performed. However, in general, a stable Q-function does not have to be static and might still vary over the rounds; the key characteristic is that its maximizing action remains the same even when its values are updated. Although the $\mathrm{stoch}\,\max$ has sub-linear complexity compared to the $\max$, without any assumption of the value distributions, the following Corollary shows that, on average, for a stable Q-function, after a certain number of iterations, the output of the $\mathrm{stoch}\,\max$ matches exactly the output of $\max$.

### C.2.1 Proof of Corollary 5.3

*Proof.* We formalize the problem as a geometric distribution where the success event is the event of sampling a subset of size $\lceil \log(n) \rceil$ that includes at least one maximizer. The geometric distribution

gives the probability that the first time to sample a subset that includes an optimal action requires $k$ independent calls, each with success probability $p$. From Lemma 5.1, we have $p \geq \frac{\lceil \log(n) \rceil}{n}$. Therefore, on an average, success requires: $\frac{1}{p} \leq \frac{n}{\lceil \log(n) \rceil}$ calls.

For a given discrete state $\mathbf{s}$, $\mathcal{M}$ keeps track of the most recent best action found. For $\mathcal{C} = \mathcal{R} \cup \mathcal{M}$,

$$\operatorname*{stoch}_{\mathbf{a} \in \mathcal{A}} \max Q(\mathbf{s}, \mathbf{a}) = \max_{\mathbf{a} \in \mathcal{C}} Q(\mathbf{s}, \mathbf{a}) \geq \max_{\mathbf{a} \in \mathcal{M}} Q(\mathbf{s}, \mathbf{a}). \tag{26}$$

Therefore, for a given state $\mathbf{s}$, on average, if the Q-function is stable, then within $\frac{n}{\lceil \log(n) \rceil}$, $\mathcal{M}$ will contain the optimal action $\mathbf{a}^*$. Therefore, on an average, after $\frac{n}{\lceil \log(n) \rceil}$ time steps,

$$\operatorname*{stoch}_{\mathbf{a} \in \mathcal{A}} \max Q(\mathbf{s}, \mathbf{a}) \geq \max_{\mathbf{a} \in \mathcal{M}} Q(\mathbf{s}, \mathbf{a}) = \max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a}).$$

We know that, $\operatorname{stoch} \max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a}) \leq \max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a})$. Therefore, for a stable Q-function, on an average, after $\frac{n}{\lceil \log(n) \rceil}$ time steps, $\operatorname{stoch} \max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a})$ becomes $\max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}, \mathbf{a})$. □

### C.3 Stochastic Maximization with Uniformly Distributed Rewards

While the above corollary outlines an upper-bound on the average number of calls needed to determine the exact optimal action eventually, the following lemma offers insights into the expected maximum value of a randomly sampled subset of actions, comprising $\lceil \log(n) \rceil$ elements when their values are uniformly distributed.

**Lemma C.2.** *For a given state* $\mathbf{s}$ *and a uniformly randomly sampled subset* $\mathcal{R}$ *of size* $\lceil \log(n) \rceil$ *actions, if the values of the sampled actions follow independently a uniform distribution in the interval* $[Q_t(\mathbf{s}, \mathbf{a}_t^\star) - b_t(\mathbf{s}), Q_t(\mathbf{s}, \mathbf{a}_t^\star)]$*, then the expected value of the maximum Q-function within this random subset is:*

$$\mathbb{E}\left[\max_{k \in \mathcal{R}} Q_t(\mathbf{s}, k) \mid \mathbf{s}, \mathbf{a}_t^\star\right] = Q_t(\mathbf{s}, \mathbf{a}_t^\star) - \frac{b_t(\mathbf{s})}{\lceil \log(n) \rceil + 1}. \tag{27}$$

*Proof.* For a given state $\mathbf{s}$ we assume a uniformly randomly sampled subset $\mathcal{R}$ of size $\lceil \log(n) \rceil$ actions, and the values of the sampled actions are independent and follow a uniform distribution in the interval $[Q_t(\mathbf{s}, \mathbf{a}_t^\star) - b_t(\mathbf{s}), Q_t(\mathbf{s}, \mathbf{a}_t^\star)]$. Therefore, the cumulative distribution function (CDF) for the value of an action $\mathbf{a}$ given the state $\mathbf{s}$ and the optimal action $\mathbf{a}_t^*$ is:

$$G(y; \mathbf{s}, \mathbf{a}) = \begin{cases} 0 & \text{for } y < Q_t(\mathbf{s}, \mathbf{a}_t^*) - b_t(\mathbf{s}) \\ y & \text{for } y \in [Q_t(\mathbf{s}, \mathbf{a}_t^*) - b_t, Q_t(\mathbf{s}, \mathbf{a}^*)] \\ 1 & \text{for } y > Q_t(\mathbf{s}, \mathbf{a}_t^*)\,. \end{cases}$$

We define the variable $x = (y - (Q_t(\mathbf{s}, \mathbf{a}_t^*) - b_t(\mathbf{s})))/b_t(\mathbf{s})$.

$$F(x; \mathbf{s}, \mathbf{a}) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \in [0, 1] \\ 1 & \text{for } x > 1\,. \end{cases}$$

If we select $\lceil \log(n) \rceil$ such actions, the CDF of the maximum of these actions, denoted as $F_{\max}$ is the following:

$$\begin{aligned} F_{\max}(x; \mathbf{s}, \mathbf{a}) &= \mathbb{P}\left(\max_{a \in \mathcal{R}} Q_t(\mathbf{s}, \mathbf{a}) \leq x\right) \\ &= \prod_{a \in \mathcal{R}} \mathbb{P}\left(Q_t(\mathbf{s}, \mathbf{a}) \leq x\right) \\ &= \prod_{a \in \mathcal{R}} F(x; \mathbf{s}, \mathbf{a}) \\ &= F(x; \mathbf{s}, \mathbf{a})^{\lceil \log(n) \rceil}. \end{aligned}$$

The second line follows from the independence of the values, and the last line follows from the assumption that all actions follow the same uniform distribution.

The CDF of the maximum is therefore given by:

$$F_{\max}(x; \mathbf{s}, \mathbf{a}) = \begin{cases} 0 & \text{for } x < 0 \\ x^{\lceil \log(n) \rceil} & \text{for } x \in [0, 1] \\ 1 & \text{for } x > 1\,. \end{cases}$$

Now, we can determine the desired expected value as

$$\begin{aligned} \mathbb{E}\left[\max_{\mathbf{a} \in \mathcal{R}} \frac{Q_t(\mathbf{s}, \mathbf{a}) - (Q_t(\mathbf{s}, \mathbf{a}_t^*) - b_t(\mathbf{s}))}{b_t(\mathbf{s})}\right] &= \int_{-\infty}^{\infty} x \, \mathrm{d}F_{\max}(x; \mathbf{s}, \mathbf{a}) \\ &= \int_0^1 x \, \mathrm{d}F_{\max}(x; \mathbf{s}, \mathbf{a}) \\ &= [x F_{\max}(x; \mathbf{s}, \mathbf{a})]_0^1 - \int_0^1 F_{\max}(x; \mathbf{s}, \mathbf{a}) \, \mathrm{d}x \\ &= 1 - \int_0^1 x^{\lceil \log(n) \rceil} \, \mathrm{d}x \\ &= 1 - \frac{1}{\lceil \log(n) \rceil + 1}. \end{aligned}$$

We employed the identity $\int_0^1 x \, \mathrm{d}\mu(x) = \int_0^1 1 - \mu(x) \, \mathrm{d}x$, which can be demonstrated through integration by parts. To return to the original scale, we can first multiply by $b_t$ and then add $Q_t(\mathbf{s}, \mathbf{a}_t^*) - b_t(\mathbf{s})$, resulting in:

$$\mathbb{E}\left[\max_{\mathbf{a} \in \mathcal{R}} Q_t(\mathbf{s}, \mathbf{a}) \mid \mathbf{s}, \mathbf{a}_t^*\right] = Q_t(\mathbf{s}, \mathbf{a}_t^*) - \frac{b_t(\mathbf{s})}{\lceil \log(n) \rceil + 1}.$$

□

As an example of this setting, for $Q_t(\mathbf{s}, \mathbf{a}_t^\star) = 100$, $b_t = 100$, for a setting with $n = 1000$ actions, $\lceil \log(n) \rceil + 1 = 11$. Hence the $\mathbb{E}\left[\max_{k \in \mathcal{R}} Q_t(\mathbf{s}, k) \mid \mathbf{s}, \mathbf{a}_t^\star\right] \approx 91$. This shows that even with a randomly sampled set of actions $\mathcal{R}$, the $\mathrm{stoch}\max$ can be close to the max. We simulate this setting in the experiments in Fig. 5.

Our proposed stochastic maximization does not solely rely on the randomly sampled subset of actions $\mathcal{R}$ but also considers actions from previous experiences through $\mathcal{M}$. Therefore, the expected $\mathrm{stoch}\max$ should be higher than the above result, providing an upper bound on the expected $\beta_t$ as described in the following corollary of Lemma C.2.

**Corollary C.3.** *For a given discrete state* $\mathbf{s}$*, if the values of the sampled actions follow independently a uniform distribution from the interval* $[Q_t(\mathbf{s}, \mathbf{a}_t^\star) - b_t(\mathbf{s}), Q_t(\mathbf{s}, \mathbf{a}_t^\star)]$*, then the expected value of* $\beta_t(\mathbf{s})$ *is:*

$$\mathbb{E}\left[\beta_t(\mathbf{s}) \mid \mathbf{s}\right] \leq \frac{b_t(\mathbf{s})}{\lceil \log(n) \rceil + 1}. \tag{28}$$

*Proof.* At time step $t$, given a state $\mathbf{s}$, and the current estimated Q-function $Q_t$, $\beta_t(\mathbf{s})$ is defined as follows:

$$\beta_t(\mathbf{s}) = \max_{\mathbf{a} \in \mathcal{A}} Q_t\left(\mathbf{s}, \mathbf{a}\right) - \operatorname*{stoch\,max}_{\mathbf{a} \in \mathcal{A}} Q_t\left(\mathbf{s}, \mathbf{a}\right). \tag{29}$$

For a given state $\mathbf{s}$ and a uniformly randomly sampled subset $\mathcal{R}$ of size $\lceil \log(n) \rceil$ actions and a subset of some previous played actions $\mathcal{M} \subset \mathcal{E}$, using the law of total expectation,

$$\begin{aligned}
\mathbb{E}\left[\beta_t(\mathbf{s}) \mid \mathbf{s}\right] &= \mathbb{E}\left[\mathbb{E}\left[\beta_t(\mathbf{s}) \mid \mathbf{s}, \mathbf{a}_t^\star\right] \mid \mathbf{s}\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\max_{k \in \mathcal{A}} Q_t(\mathbf{s}, k) - \operatorname*{stoch\,max}_{k \in \mathcal{A}} Q_t(\mathbf{s}, k) \mid \mathbf{s}, \mathbf{a}_t^\star\right] \mid \mathbf{s}\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\max_{k \in \mathcal{A}} Q_t(\mathbf{s}, k) - \max_{k \in \mathcal{R} \cup \mathcal{M}} Q_t(\mathbf{s}, k) \mid \mathbf{s}, \mathbf{a}_t^\star\right] \mid \mathbf{s}\right] \\
&\leq \mathbb{E}\left[\mathbb{E}\left[\max_{k \in \mathcal{A}} Q_t(\mathbf{s}, k) - \max_{k \in \mathcal{R}} Q_t(\mathbf{s}, k) \mid \mathbf{s}, \mathbf{a}_t^\star\right] \mid \mathbf{s}\right] \\
&= \mathbb{E}\left[Q_t(\mathbf{s}, \mathbf{a}_t^*) - \mathbb{E}\left[\max_{k \in \mathcal{R}} Q_t(\mathbf{s}, k) \mid \mathbf{s}, \mathbf{a}_t^\star\right] \mid \mathbf{s}\right].
\end{aligned}$$

Therefore by Lemma C.2:

$$\begin{aligned}
\mathbb{E}\left[\beta_t(\mathbf{s}) \mid \mathbf{s}\right] &\leq \mathbb{E}\left[Q_t(\mathbf{s}, \mathbf{a}_t^*) - (Q_t(\mathbf{s}, \mathbf{a}_t^*) - \frac{b_t(\mathbf{s})}{\lceil \log(n) \rceil + 1}) \mid \mathbf{s}\right] \\
&= \mathbb{E}\left[\frac{b_t(\mathbf{s})}{\lceil \log(n) \rceil + 1} \mid \mathbf{s}\right] \\
&= \frac{b_t(\mathbf{s})}{\lceil \log(n) \rceil + 1}.
\end{aligned}$$

□

# D Pseudocodes

**Algorithm 1** Stochastic Q-learning

Initialize $Q(\mathbf{s}, \mathbf{a})$ for all $\mathbf{s} \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$
**for** each episode **do**
  Observe state $\mathbf{s}$.
  **for** each step of episode **do**
    Choose $\mathbf{a}$ from $\mathbf{s}$ with policy $\pi_Q^S(\mathbf{s})$.
    Take action $\mathbf{a}$, observe $r$, $\mathbf{s}'$.
    $b^* \leftarrow \text{stoch} \arg\max_{b \in \mathcal{A}} Q(\mathbf{s}', b)$.
    $\Delta \leftarrow r + \gamma Q(\mathbf{s}', b^*) - Q(\mathbf{s}, \mathbf{a})$.
    $Q(\mathbf{s}, \mathbf{a}) \leftarrow Q(\mathbf{s}, \mathbf{a}) + \alpha(\mathbf{s}, \mathbf{a})\Delta$.
    $\mathbf{s} \leftarrow \mathbf{s}'$.
  **end for**
**end for**

**Algorithm 2** Stochastic Double Q-learning

Initialize $Q^A(\mathbf{s}, \mathbf{a})$ and $Q^B(\mathbf{s}, \mathbf{a})$ for all $\mathbf{s} \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$, $n = |\mathcal{A}|$
**for** each episode **do**
  Observe state $\mathbf{s}$.
  **for** each step of episode **do**
    Choose $\mathbf{a}$ from $\mathbf{s}$ via $Q^A + Q^B$ with policy $\pi^S_{(Q^A+Q^B)}(\mathbf{s})$ in Eq. (6).
    Take action $\mathbf{a}$, observe $r$, $\mathbf{s}'$.
    Choose either UPDATE(A) or UPDATE(B), for example randomly.
    **if** UPDATE(A) **then**
      $\Delta^A \leftarrow r + \gamma Q^B(\mathbf{s}', \text{stoch} \arg\max_{b \in \mathcal{A}} Q^A(\mathbf{s}', b)) - Q^A(\mathbf{s}, \mathbf{a})$.
      $Q^A(\mathbf{s}, \mathbf{a}) \leftarrow Q^A(\mathbf{s}, \mathbf{a}) + \alpha(\mathbf{s}, \mathbf{a})\Delta^A$.
    **else if** UPDATE(B) **then**
      $\Delta^B \leftarrow r + \gamma Q^A(\mathbf{s}', \text{stoch} \arg\max_{b \in \mathcal{A}} Q^B(\mathbf{s}', b)) - Q^B(\mathbf{s}, \mathbf{a})$.
      $Q^B(\mathbf{s}, \mathbf{a}) \leftarrow Q^B(\mathbf{s}, \mathbf{a}) + \alpha(\mathbf{s}, \mathbf{a})\Delta^B$.
    **end if**
    $\mathbf{s} \leftarrow \mathbf{s}'$.
  **end for**
**end for**

**Algorithm 3** Stochastic Sarsa

Initialize $Q(\mathbf{s}, \mathbf{a})$ for all $\mathbf{s} \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$, $n = |\mathcal{A}|$
**for** each episode **do**
  Observe state $\mathbf{s}$.
  Choose $\mathbf{a}$ from $\mathbf{s}$ with policy $\pi_Q^S(\mathbf{s})$ in Eq. (6).
  **for** each step of episode **do**
    Take action $\mathbf{a}$, observe $r$, $\mathbf{s}'$.
    Choose $\mathbf{a}'$ from $\mathbf{s}'$ with policy $\pi_Q^S(\mathbf{s}')$ in Eq. (6).
    $Q(\mathbf{s}, \mathbf{a}) \leftarrow Q(\mathbf{s}, \mathbf{a}) + \alpha(\mathbf{s}, \mathbf{a})[r + \gamma Q(\mathbf{s}', \mathbf{a}') - Q(\mathbf{s}, \mathbf{a})]$.
    $\mathbf{s} \leftarrow \mathbf{s}'$; $\mathbf{a} \leftarrow \mathbf{a}'$.
  **end for**
**end for**

**Algorithm 4** Stochastic Deep Q-Network (StochDQN)

**Algorithm parameters:** learning rate $\alpha \in (0, 1]$, replay buffer $\mathcal{E}$, update rate $\tau$.
**Initialize:** neural network $Q(\mathbf{s}, \mathbf{a}; \theta)$ with random weights $\theta$, target network $\hat{Q}(\mathbf{s}, \mathbf{a}; \theta^-)$ with $\theta^- = \theta$, set of actions $\mathcal{A}$ of size $n$.
**for** each episode **do**
  Initialize state $\mathbf{s}$.
  **while** not terminal state is reached **do**
    Choose $\mathbf{a}$ from $\mathbf{s}$ using a stochastic policy as defined in Eq. (14) using $Q(\mathbf{s}, .; \theta)$.
    Take action $\mathbf{a}$, observe reward $r(\mathbf{s}, \mathbf{a})$ and next state $\mathbf{s}'$.
    Store $(\mathbf{s}, \mathbf{a}, r(\mathbf{s}, \mathbf{a}), \mathbf{s}')$ in replay buffer $\mathcal{E}$.
    Compute target values for the mini-batch:

$$y_i = \begin{cases} r_i & \text{if } \mathbf{s}'_i \text{ is terminal} \\ r_i + \gamma \hat{Q}(\mathbf{s}'_i, \text{stoch arg max}_{\mathbf{a}' \in \mathcal{A}} \hat{Q}(\mathbf{s}'_i, \mathbf{a}'; \theta^-); \theta^-) & \text{otherwise.} \end{cases}$$

    Perform a gradient descent step on the loss:

$$\mathcal{L}(\theta) = \frac{1}{\lceil \log(n) \rceil} \sum_{i=1}^{\lceil \log(n) \rceil} (y_i - Q(\mathbf{s}_i, \mathbf{a}_i; \theta))^2.$$

    Update the target network weights:

$$\theta^- \leftarrow \tau \cdot \theta + (1 - \tau) \cdot \theta^-.$$

    Update the Q-network weights using gradient descent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}(\theta).$$

    $\mathbf{s} \leftarrow \mathbf{s}'$.
  **end while**
**end for**

# E Experimental Details

## E.1 Environments

We test our proposed algorithms on a standardized set of environments using open-source libraries. We compare stochastic maximization to exact maximization and evaluate the proposed stochastic RL algorithms on Gymnasium environments [3]. Stochastic Q-learning and Stochastic Double Q-learning are tested on the CliffWalking-v0, the FrozenLake-v1, and a generated MDP environment, while stochastic deep Q-learning approaches are tested on MuJoCo control tasks [51].

### E.1.1 Environments with Discrete States and Actions

We generate an MDP environment with 256 actions, with rewards following a normal distribution of mean -50 and standard deviation of 50, with 3 states. Furthermore, while our approach is designed for large discrete action spaces, we tested it in Gymnasium environments [3] with only four discrete actions, such as CliffWalking-v0 and FrozenLake-v1. CliffWalking-v0 involves navigating a grid world from the starting point to the destination without falling off a cliff. FrozenLake-v1 requires moving from the starting point to the goal without stepping into any holes on the frozen surface, which can be challenging due to the slippery nature of the ice.

### E.1.2 Environments with Continuous States: Discretizing Control Tasks

We test the stochastic deep Q-learning approaches on MuJoCo [51] for continuous states discretized control tasks. We discretize each action dimension into $i$ equally spaced values, creating a discrete action space with $n = i^d$ $d$-dimensional actions. We mainly focused on the inverted pendulum and

the half-cheetah. The inverted pendulum involves a cart that can be moved left or right, intending to balance a pole on top using a 1D force, with $i = 512$ resulting in 512 actions. The half-cheetah is a robot with nine body parts aiming to maximize forward speed. It can apply torque to 6 joints, resulting in 6D actions with $i = 4$, which results in 4096 actions.

### E.2 Algorithms

#### E.2.1 Stochastic Maximization

We have two scenarios, one for discrete and the other for continuous states. For discrete states, $\mathcal{E}$ is a dictionary with the keys as the states in $\mathcal{S}$ with corresponding values of the latest played action in every state. In contrast, $\mathcal{E}$ comprises the actions in the replay buffer for continuous states. Indeed, we do not consider the whole set $\mathcal{E}$ either. Instead, we only consider a subset $\mathcal{M} \subset \mathcal{E}$. For discrete states, for a given state $\mathbf{s}$, $\mathcal{M}$ includes the latest two exploited actions in state $\mathbf{s}$. For continuous states, where it is impossible to retain the last exploited action for each state, we consider randomly sampled subset $\mathcal{M} \subset \mathcal{E}$, which includes $\lceil \log(n) \rceil$ actions, even though they were played in different states. In the experiments involving continuous states, we demonstrate that this was sufficient to achieve good results, see Section 7.3.

#### E.2.2 Tabular Q-learning Methods

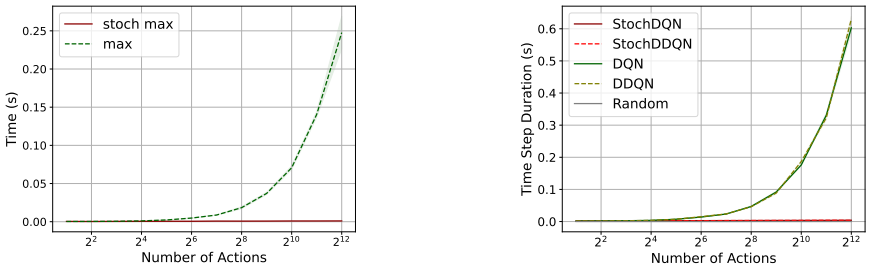
We set the training parameters the same for all the Q-learning variants. We follow similar hyper-parameters as in [17]. We set the discount factor $\gamma$ to 0.95 and apply a dynamical polynomial learning rate $\alpha$ with $\alpha_t(\mathbf{s}, \mathbf{a}) = 1/z_t(\mathbf{s}, \mathbf{a})^{0.8}$, where $z_t(\mathbf{s}, \mathbf{a})$ is the number of times the pair $(\mathbf{s}, \mathbf{a})$ has been visited, initially set to one for all the pairs. For the exploration rate, we use use a decaying $\varepsilon$, defined as $\varepsilon(\mathbf{s}) = 1/\sqrt{(}z(\mathbf{s}))$ where $z(\mathbf{s})$ is the number of times state $\mathbf{s}$ has been visited, initially set to one for all the states. For Double Q-learning $z_t(\mathbf{s}, \mathbf{a}) = z_t^A(\mathbf{s}, \mathbf{a})$ if $Q^A$ is updated and $z_t(\mathbf{s}, \mathbf{a}) = z_t^B(\mathbf{s}, \mathbf{a})$ if $Q^B$ is updated, where $z_t^A$ and $z_t^B$ store the number of updates for each action for the corresponding value function. We averaged the results over ten repetitions. For Stochastic Q-learning, we track a dictionary $\mathcal{D}$ with keys being the states, and values being the latest exploited action. Thus, for a state $\mathbf{s}$, the memory $\mathcal{M} = \mathcal{D}(\mathbf{s})$, thus $\mathcal{M}$ is the latest exploited action in the same state $\mathbf{s}$.

#### E.2.3 Deep Q-network Methods

We set the training parameters the same for all the deep Q-learning variants. We set the discount factor $\gamma$ to 0.99 and the learning rate $\alpha$ to 0.001. Our neural network takes input of a size equal to the sum of the dimensions of states and actions with a single output neuron. The network consists of two hidden linear layers, each with a size of 64, followed by a ReLU activation function [36]. We keep the exploration rate $\varepsilon$ the same for all states, initialize it at 1, and apply a decay factor of 0.995, with a minimum threshold of 0.01. For $n$ total number of actions, during training, to train the network, we use stochastic batches of size $\lceil \log(n) \rceil$ uniformly sampled from a buffer of size $2\lceil \log(n) \rceil$. We averaged the results over five repetitions. For the stochastic methods, we consider the actions in the batch of actions as the memory set $\mathcal{M}$. We choose the batch size in this way to keep the complexity of the Stochastic Q-learning within $\mathcal{O}(\log(n))$.

### E.3 Compute and Implementation

We implement the different Q-learning methods using Python 3.9, Numpy 1.23.4, and Pytorch 2.0.1. For proximal policy optimization (PPO) [43], asynchronous actor-critic (A2C) [34], and deep deterministic policy gradient (DDPG) [29], we use the implementations of Stable-Baselines [21]. We test the training time using a CPU 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 1.69 GHz. with 16.0 GB RAM.




(a) Action Selection Time

(b) Full Step Duration

Figure 4: Comparison results for the stochastic and deterministic methods. The x-axis represents the number of possible actions, and the y-axis represents the time step duration of the agent in seconds.

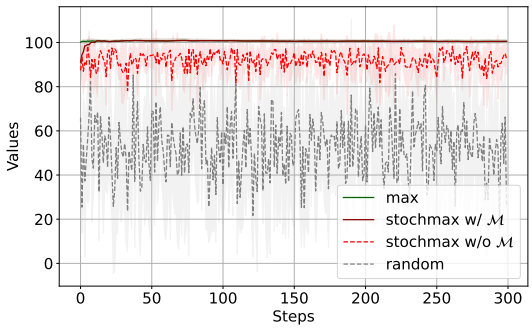



Figure 5: $\mathrm{stoch}\,\max$ with (w/) and without (w/o) memory $\mathcal{M}$ vs. $\max$ on uniformly distributed action values as described in Section C. The x-axis and y-axis represent the steps and the values, respectively.

# F Additional Results

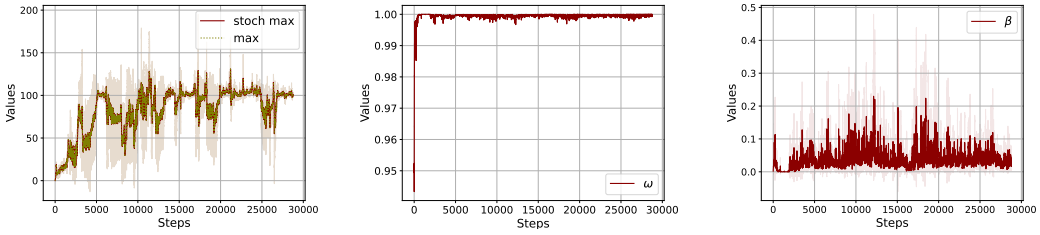
## F.1 Wall Time Speed

Stochastic maximization methods exhibit logarithmic complexity regarding the number of actions, as confirmed in Fig. 4a. Therefore, both StochDQN and StochDDQN, which apply these techniques for action selection and updates, have exponentially faster execution times compared to both DQN and DDQN, which can be seen in Fig 4b which shows the complete step duration for deep Q-learning methods, which include action selection and network update. The proposed methods are nearly as fast as a random algorithm, which samples and selects actions randomly and has no updates.

## F.2 Stochastic Maxmization

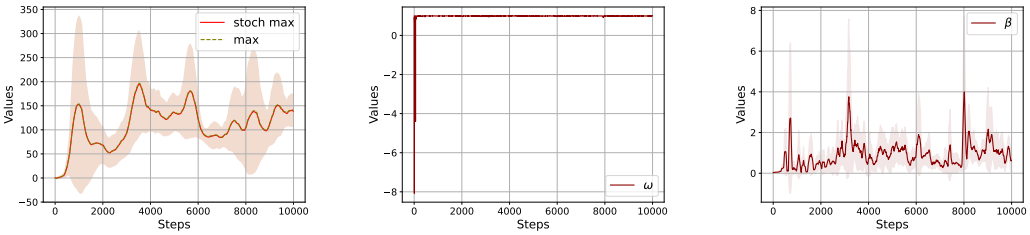
### F.2.1 Stochastic Maxmization vs Maximization with Uniform Rewards

In the setting described in Section C.3 with 5000 uniformly independently distributed action values in the range of [0, 100], as shown in Fig. 5, $\mathrm{stoch}\,\max$ without memory, i.e., $\mathcal{M} = \emptyset$ reaches around 91 in average return, and keeps fluctuating around, while $\mathrm{stoch}\,\max$ with $\mathcal{M}$ quickly achieves the optimal reward.




(a) stoch max vs max (b) Ratio $\omega_t$ (c) Difference $\beta_t$

Figure 6: Comparison results for the stochastic and non-stochastic methods for the Inverted Pendulum with 512 actions.




(a) stoch max vs max (b) Ratio $\omega_t$ (c) Difference $\beta_t$

Figure 7: Comparison results for the stochastic and non-stochastic methods for the Half Cheetah with 4096 actions.

### F.2.2 Stochastic Maximization Analysis

In this section, we analyze stochastic maximization by tracking returned values across rounds, $\omega_t$ (Eq. (10)), and $\beta_t$ (Eq. (9)), which we provide here. At time step $t$, given a state $\mathbf{s}$, and the current estimated Q-function $Q_t$, we define the non-negative underestimation error as $\beta_t(\mathbf{s})$, as follows:

$$\beta_t(\mathbf{s}) = \max_{\mathbf{a}\in\mathcal{A}} Q_t(\mathbf{s},\mathbf{a}) - \operatorname*{stoch\,max}_{\mathbf{a}\in\mathcal{A}} Q_t(\mathbf{s},\mathbf{a}). \tag{30}$$

Furthermore, we define the ratio $\omega_t(\mathbf{s})$, as follows:

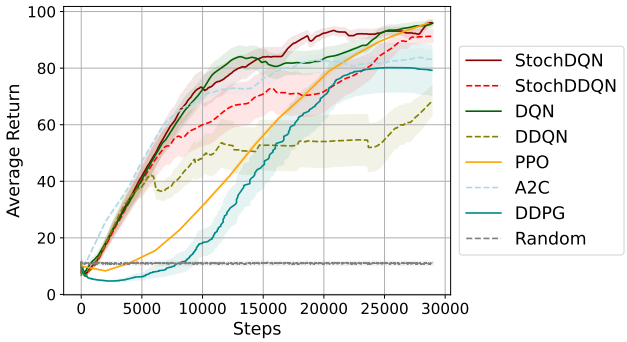
$$\omega_t(\mathbf{s}) = \frac{\operatorname{stoch}\max_{\mathbf{a}\in\mathcal{A}} Q_t(\mathbf{s},\mathbf{a})}{\max_{\mathbf{a}\in\mathcal{A}} Q_t(\mathbf{s},\mathbf{a})}. \tag{31}$$

It follows that:

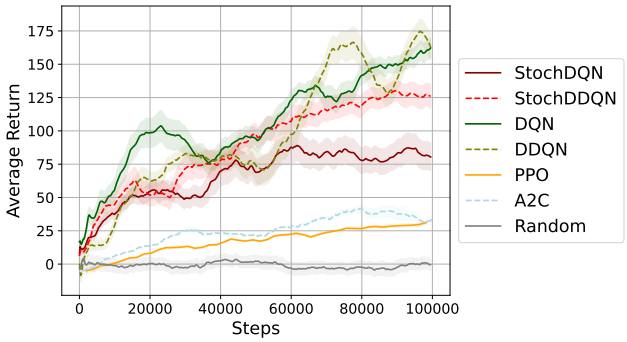
$$\omega_t(\mathbf{s}) = 1 - \frac{\beta_t(\mathbf{s})}{\max_{\mathbf{a}\in\mathcal{A}} Q_t(\mathbf{s},\mathbf{a})}. \tag{32}$$

For Deep Q-Networks, for the InvertedPendulum-v4, both $\operatorname{stoch}\max$ and max return similar values (Fig. 6a), $\omega_t$ approaches one rapidly (Fig. 6b) and $\beta_t$ remains below 0.5 (Fig. 6c). In the case of HalfCheetah-v4, both $\operatorname{stoch}\max$ and max return similar values (Fig. 7a), $\omega_t$ quickly converges to one (Fig. 7b), and $\beta_t$ is upper bounded below eight (Fig. 7c).

While the difference $\beta_t$ remains bounded, the values of both $\operatorname{stoch}\max$ and max increase over the rounds as the agent explores better options. This leads to the ratio $\omega_t$ converging to one as the error becomes negligible over the rounds, as expected according to Eq. (32).




(a) Inverted Pendulum




(b) Half Cheetah

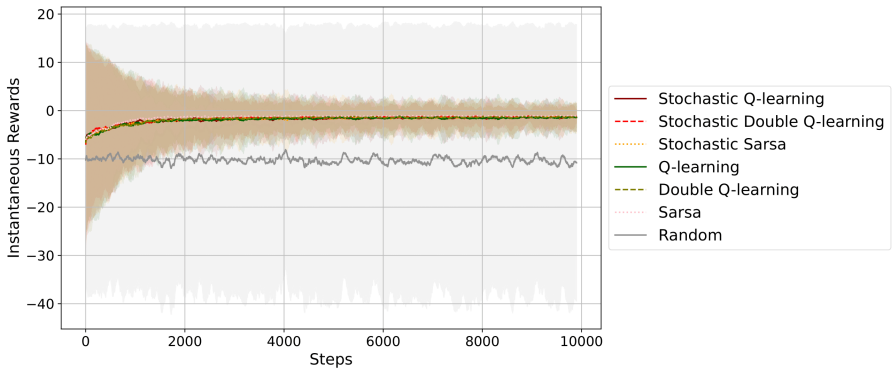
Figure 8: Stochastic vs non-stochastic of deep Q-learning variants on Inverted Pendulum and Half Cheetah, with steps on the x-axis and average returns, smoothed over a size 100 window on the y-axis.

### F.3 Stochastic Q-network Reward Analysis

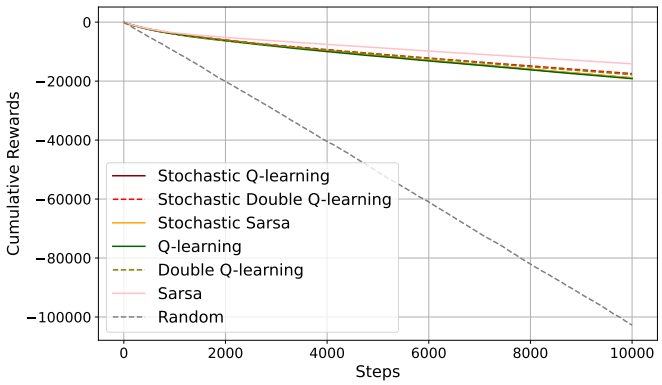
As illustrated in Fig. 8a and Fig. 8b for the inverted pendulum and half cheetah experiments, which involve 512 and 4096 actions, respectively, both StochDQN and StochDDQN attain the optimal average return in a comparable number of rounds to DQN and DDQN. Additionally, StochDQN exhibits the quickest attainment of optimal rewards for the inverted pendulum. Furthermore, while DDQN did not perform well on the inverted pendulum task, its modification, i.e., StochDDQN, reached the optimal rewards.

### F.4 Stochastic Q-learning Reward Analysis

We tested Stochastic Q-learning, Stochastic Double Q-learning, and Stochastic Sarsa in environments with both discrete states and actions. Interestingly, as shown in Fig. 10, our stochastic algorithms outperform their deterministic counterparts in terms of cumulative rewards. Furthermore, we notice that Stochastic Q-learning outperforms all the considered methods regarding the cumulative rewards. Moreover, in the CliffWalking-v0 (as shown in Fig. 9), as well as for the generated MDP environment with 256 possible actions (as shown in Fig. 11), all the stochastic and non-stochastic algorithms reach the optimal policy in a similar number of steps.

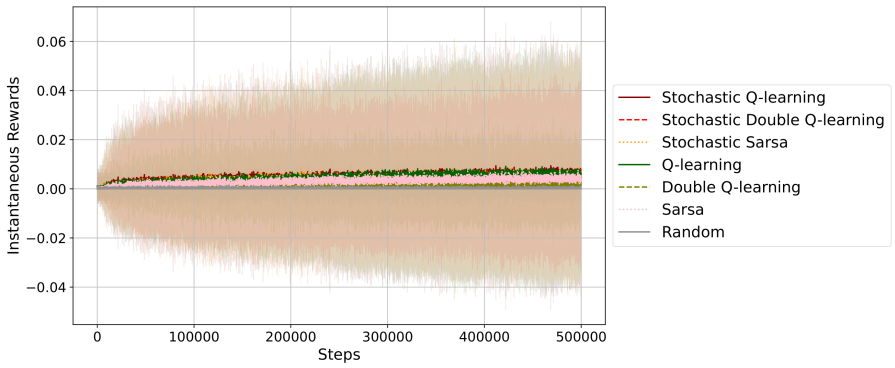



(a) Instantaneous Rewards

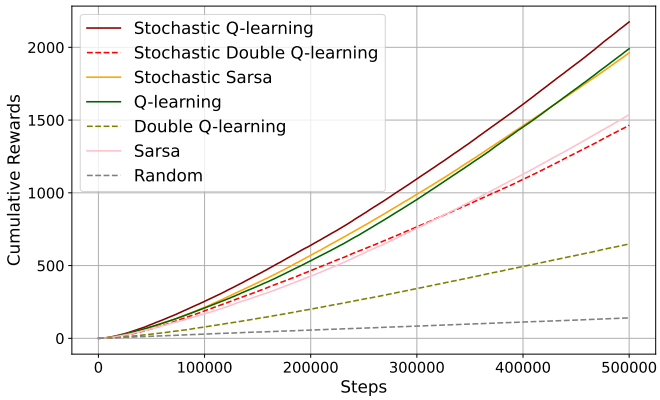



(b) Cumulative Rewards

Figure 9: Comparing stochastic and non-stochastic Q-learning approaches on the Cliff Walking, with steps on the x-axis, instantaneous rewards smoothed over a size 1000 window on the y-axis for plot (a), and cumulative rewards on the y-axis for plot (b).

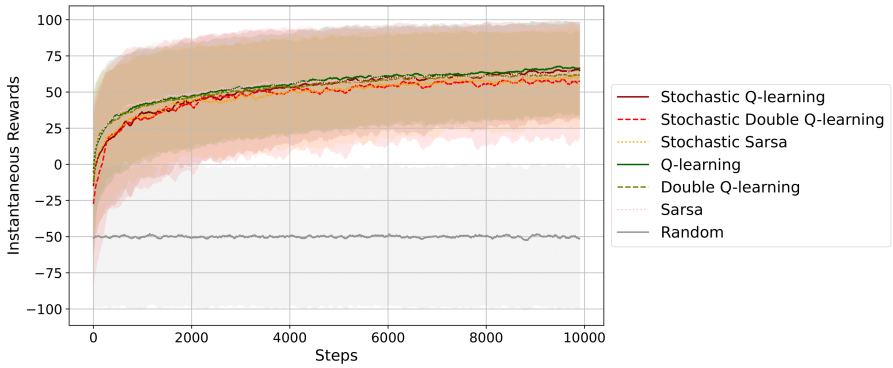



(a) Instantaneous Rewards

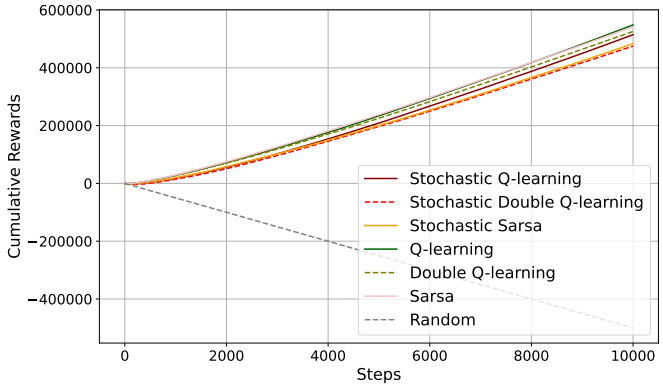



(b) Cumulative Rewards

Figure 10: Comparing stochastic and non-stochastic Q-learning approaches on the Frozen Lake, with steps on the x-axis, instantaneous rewards smoothed over a size 1000 window on the y-axis for plot (a), and cumulative rewards on the y-axis for plot (b).




(a) Instantaneous Rewards




(b) Cumulative Rewards

Figure 11: Comparing stochastic and non-stochastic Q-learning approaches on the generated MDP environment, with steps on the x-axis, instantaneous rewards smoothed over a size 1000 window on the y-axis for plot (a), and cumulative rewards on the y-axis for plot (b).