Abstract—Stochastic exploration is the key to the success of the deep $Q$-network (DQN) algorithm. However, most existing stochastic exploration approaches either explore actions heuristically regardless of their $Q$ values or couple the sampling with $Q$ values, which inevitably introduce bias into the learning process. In this article, we propose a novel preference-guided $\epsilon$-greedy exploration algorithm that can efficiently facilitate exploration for DQN without introducing additional bias. Specifically, we design a dual architecture consisting of two branches, one of which is a copy of DQN, namely, the $Q$ branch. The other branch, which we call the preference branch, learns the action preference that the DQN implicitly follows. We theoretically prove that the policy improvement theorem holds for the preference-guided $\epsilon$-greedy policy and experimentally show that the inferred action preference distribution aligns with the landscape of corresponding $Q$ values. Intuitively, the preference-guided $\epsilon$-greedy exploration motivates the DQN agent to take diverse actions, so that actions with larger $Q$ values can be sampled more frequently, and those with smaller $Q$ values still have a chance to be explored, thus encouraging the exploration. We comprehensively evaluate the proposed method by benchmarking it with well-known DQN variants in nine different environments. Extensive results confirm the superiority of our proposed method in terms of performance and convergence speed.

Index Terms—Data efficiency, deep $Q$ network (DQN), deep reinforcement learning (DRL), preference-guided exploration, stochastic policy.

Manuscript received 14 April 2022; revised 30 January 2023 and 26 May 2023; accepted 10 September 2023. Date of publication 3 October 2023; date of current version 3 December 2024. This work was supported in part by the A*STAR AME Young Individual Research Grant under Project A2084c0156; in part by the MTC Individual Research Grants under Project M22K2c0079; in part by the ANR-NRF Joint Grant under Project NRF2021-NRF-ANR003 HM Science; in part by the Nanyang Technological University, Singapore, through the SUG-NAP Grant; and in part by the MOE AcRF Tier 1 Funding under Project RG132/23. (Corresponding author: Chen Lv.)

Wenhui Huang, Jingda Wu, Xiangkun He, and Chen Lv are with the School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore 639798 (e-mail: wenhui001@e.ntu.edu.sg; jingda001@e.ntu.edu.sg; xiangkun.he@ntu.edu.sg; lyuchen@ntu.edu.sg).

Cong Zhang and Jie Zhang are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: cong030@e.ntu.edu.sg; ZhangJ@ntu.edu.sg).

Data is available on-line at https://github.com/OscarHuangWind/Preference-Guided-DQN-Atari.

Digital Object Identifier 10.1109/TNNLS.2023.3317628

I. INTRODUCTION

Deep reinforcement learning (DRL) algorithms have been applied to address complex control tasks from a wide range of domains with a series of remarkable successes, including but not limited to games [1], [2], vision problems [3], [4], robotics locomotion [5], [6], and autonomous driving [7], [8]. With the powerful representation capabilities of data-driven approaches [9], [10], [11], DRL can make appropriate decisions by automatically reasoning and extracting structural knowledge from various raw features, such as images [12] and electronic signals [13].

Deep $Q$ network (DQN) is perhaps the most popular among various RL algorithms [14], [15] for solving discrete decision-making problems. The rationale of DQN is to select the action that maximizes the expected state–action return, i.e., the $Q$ value, which is represented by a deep neural network. DQN has been successfully applied to the Atari games and presented human-level performance [12]. However, the original DQN algorithm possesses many defects. In particular, it suffers the overestimation issue where the value of the action maximizing $Q$ value is consistently overestimated. To address this issue, double DQN [16] and dueling DQN [17] have been proposed. Other technical amelioration includes combining the dueling

IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 35, NO. 12, DECEMBER 2024 18553

NOMENCLATURE

$\alpha$ Proportion.
$\mathbb{E}$ Expectation.
* Optimal symbol.
$\nabla$ Derivative operation.
$\eta$ Action preference.
$D_{KL}$ Kullback–Leibler (KL) divergence.
$\pi_{PG}$ Preference-guided $\epsilon$-greedy policy.
$Q^\pi$ $Q$ values for arbitrary policy $\pi$.
$Q^\alpha$ $Q$ values follow action preference distribution.
$Q_{PG}^\pi$ $Q$ values for preference-guided $\epsilon$-greedy policy.
$A^\pi$ Advantage function for arbitrary policy $\pi$.
$A^\alpha$ Advantage function follows action preference distribution.
$H^\pi$ Shannon entropy of arbitrary policy distribution.
$H^\alpha$ Shannon entropy of action preference distribution.

Deep reinforcement learning (DRL) algorithms have been applied to address complex control tasks from a wide range of domains with a series of remarkable successes, including but not limited to games [1], [2], vision problems [3], [4], robotics locomotion [5], [6], and autonomous driving [7], [8]. With the powerful representation capabilities of data-driven approaches [9], [10], [11], DRL can make appropriate decisions by automatically reasoning and extracting structural knowledge from various raw features, such as images [12] and electronic signals [13].

Deep $Q$ network (DQN) is perhaps the most popular among various RL algorithms [14], [15] for solving discrete decision-making problems. The rationale of DQN is to select the action that maximizes the expected state–action return, i.e., the $Q$ value, which is represented by a deep neural network. DQN has been successfully applied to the Atari games and presented human-level performance [12]. However, the original DQN algorithm possesses many defects. In particular, it suffers the overestimation issue where the value of the action maximizing $Q$ value is consistently overestimated. To address this issue, double DQN [16] and dueling DQN [17] have been proposed. Other technical amelioration includes combining the dueling
DQN and double DQN with taking advantage of both methods [18] and learning the return distribution to capture the intrinsic randomness of returns [19] for better return modeling.

Despite the performance improvement, aforementioned canonical DQN variants still cannot sample data effectively, mainly attributed to their naive exploration strategy: $\epsilon$-greedy policy. The situation even gets worse when DQN algorithms are employed to address diverse and complicated problems, such as robot manipulation and autonomous driving. Although there is already a considerable amount of research devoted to addressing this exploration problem, most existing stochastic policy approaches, however, have their individual limitations. For example, NoisyNet [20] explores the action space by introducing zero-mean noise into the network architecture. However, such noisy-based exploration has a fatal limitation where it can only learn a unimodal distribution regardless of landscape of $Q$ values and, thus, is not suitable for complex tasks, which have multiple optimal decisions [21]. To explore actions from a distribution closely coupled with corresponding $Q$ values, a framework named energy-based Boltzmann exploration has been proposed [21], [22], which maximizes a so-called soft $Q$ value that combines the original $Q$ value with the entropy of its Boltzmann distribution. Although it works well, the hybrid objective fundamentally changes the agent’s learning outcome. Specifically, the actions that maximize the sum of $Q$ value and the entropy of Boltzmann distribution are preferred, therefore biasing the agent from its original objective function: maximizing overall return [23]. In addition, balancing the entropy and $Q$ value can be tricky in practice [24]. In summary, an effective and nonbiased stochastic exploration method that can learn action distribution closely coupled with corresponding $Q$ values is still missing from the current deep $Q$-learning framework.

This article proposes a novel improved DQN algorithm that realizes an efficient and nonbiased exploration through a dual-branch architecture. Specifically, one branch is the original DQN, which we call the $Q$ branch in the rest of this article, while the other infers the action preference distribution of DQN, which we call the preference branch. These two branches share a common embedding network. During the training, the actions are sampled guided by their preference rather than following the naive $\epsilon$-greedy strategy [12], [16], [17], [18], [19], [25], [26], [27]. More precisely, with $1 - \epsilon$ probability, the actions are greedily selected according to the $Q$ value, or with $\epsilon$ probability to be sampled proportional to the preferences. Intuitively, the preference of the action is proved to be proportional to its $Q$ value, which means an action with a higher $Q$ value is preferable to be selected. Therefore, if one of the $Q$ values significantly dominates the others, the distribution of action preference tends to be a unimodal landscape (unimodality). On the contrary, if the $Q$ value distribution possesses multiple peaks (i.e., multimodality), they will be reflected on the distribution of the action preference accordingly. We update the preference branch by maximizing an entropy regularized objective function to prevent premature convergence to suboptimal policy due to false-positive $Q$ values (inaccurately estimated large $Q$ values) during the early stage. In specific, when $Q$ values are inaccurate, all actions are sampled from a smooth distribution that maximizes the entropy (i.e., random exploration). Moreover, we theoretically prove (and experimentally show) that maximizing such an objective function is equivalent to minimizing the statistical distance between the action preference and the corresponding $Q$ values. Therefore, actions with larger $Q$ values can be sampled more frequently, while those with smaller $Q$ values still have a chance to be selected (i.e., preference-guided exploration), thus encouraging the exploration when estimated $Q$ values are accurate. The $Q$ branch is updated against the original DQN objective. Since we do not invoke any bias in the $Q$ network nor change its objective, our method introduces no bias to the original DQN algorithm. We summarize the main contributions of this article as follows.

1) We propose a novel DRL algorithm to realize an efficient and nonbiased exploration for the DQN algorithm by introducing a preference-guided $\epsilon$-greedy policy. We formally prove that the proposed policy has an appealing property: it preserves the policy improvement guarantee of the $Q$-learning framework.

2) We show that the preference of actions inferred by the proposed network explicitly reflects the favor of the corresponding $Q$ value, i.e., actions with larger $Q$ values will be sampled more frequently. Therefore, we can encourage DQN to sample actions from the action preference distribution that is in line with the landscape of $Q$ values to achieve better exploration.

3) As for the practical aspect of contribution, we instantiate the proposed framework as a concrete algorithm that efficiently trains the $Q$ branch and the preference branch. Extensive experiments confirm that our method delivers superior performance (improved up to 136%) against strong DQN variants with a faster convergence speed (enhanced up to 85%) in a wide range of challenging environments.

II. RELATED WORK

As one of the prestigious DRL algorithms, the DQN [12] has brought a series of breakthroughs in many areas since its emergence. By leveraging the powerful representation learning ability of deep neural networks [28], DQN learns from scratch without human intervention to achieve human-level performance with only screenshots as input in Atari games [12]. The success of DQN has prompted people to further improve its performance by identifying and fixing its intrinsic defects. Two of the notable improvements are double DQN [16] and dueling DQN [17]. Specifically, the double-DQN algorithm resolves the overestimation of action values by evaluating the actions with an independent neural network. As for dueling DQN, it generalizes the learning across different actions by estimating the action advantage function with a separate stream. Predictably, combining the advantages of double DQN and dueling DQN can yield even better performance [18], [29].

A tricky question for DRL algorithms is building the just-right exploration to find promising but infrequent actions to discover potentially better policies. The original DQN algorithm seeks to balance exploration and exploitation
Similarly, the value of action \( a \) for the value function instances immediate (bounded) payoff \( r \) substantially changes the learning outcome of the agent, objective function for DQN. However, such a combination fundamentally changes the learning process. As a result, an effective and nonbiased exploration strategy to learn an action distribution in line with the landscape of corresponding \( Q \) values for DQN is still an open problem. To the best of our knowledge, this work presents the first attempt to solve this challenging problem.

III. PRELIMINARIES

A. Markov Decision Process

We consider a standard Markov decision process (MDP) with a controlled agent. In particular, a standard MDP is formulated as a tuple \( \langle S, A, P, R \rangle \). \( S \) is a set of states describing the possible configuration of the agent and the external environment. Given the current state \( s_t \in S \), the agent selects an action \( a_t \in A \) from its action space \( A \) at each time step \( t \). \( P(s_{t+1}|s_t, a_t); S \times A \rightarrow [0, 1] \) models the environment transition probability from the current state \( s_t \) at the next state \( s_{t+1} \in S \) after executing an action \( a_t \in A \) at \( s_t \). \( R(s_t, a_t); S \times A \rightarrow \mathbb{R} \) is the reward function evaluating the consequent immediate (bounded) payoff \( r_t = R(s_t, a_t) \) the agent received after taking the action \( a_t \) at \( s_t \).

The agent is usually controlled by a policy \( \pi_t \sim \pi_{\phi}(\cdot|s_t) : S \rightarrow A \) with parameters \( \phi \), which is a probability distribution modeling the confidence the agent has about its decision at each state. The objective of the agent is to maximize the expected total return starting from an initial state \( s \), i.e., \( V^\pi(s) \), defined as follows:

\[
V^\pi(s) = \mathbb{E}_{s_t \sim P} \left[ \sum_{t=0}^T \gamma^t \cdot r_t \right]
\]

where \( 0 < \gamma \leq 1 \) is the discounting factor. To reflect that the value function \( V \) is related to \( \pi \), we superscript \( V \) by \( \pi \). Similarly, the value of action \( a_t \) at state \( s_t \) for some time step \( t \) can be calculated as follows:

\[
Q^\pi(s_t, a_t) = r_t + \gamma \cdot \mathbb{E}_{s_{t+1} \sim P, a_{t+1} \sim \pi} \left[ V^\pi(s_{t+1}) \right] \\
= r_t + \gamma \cdot \mathbb{E}_{s_{t+1} \sim P, a_{t+1} \sim \pi} \left[ Q^\pi(s_{t+1}, a_{t+1}) \right] \\
V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi} \left[ Q^\pi(s_t, a_t) \right] \\
= Q^\pi(s_t, \pi(a_t|s_t)).
\]

B. Policy Gradient Theorem

The basic idea behind policy gradient algorithms is to adjust the parameters \( \phi \) of policy \( \pi \) in the direction of the performance gradient. The fundamental result underlying these algorithms is the policy gradient theorem [31]

\[
\nabla_{\phi} \mathcal{L}(\pi) = \mathbb{E}_{s_t \sim P, a_t \sim \pi} \left[ Q^\pi(s_t, a_t) \nabla_{\phi} \log \pi(a_t|s_t) \right].
\]

To mitigate the variance of the gradient, a simple technique is to subtract a baseline (a state-dependent variant) from the state–action value \( Q \) in (3) [32], [33]. One of the common choices of such baseline is the state value \( V: A^T(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \), where \( A^T \) is the resulted new performance measure called the advantage function. In addition, to prevent the policy from trapping in the local optima, it usually introduces the entropy of the policy into the objective. Hence, the final gradient takes the following form:

\[
\nabla_{\phi} \mathcal{L}(\pi) = \mathbb{E}_{s_t \sim P, a_t \sim \pi} \left[ A^T(s_t, a_t) \nabla_{\phi} \log \pi(a_t|s_t) \right] \\
+ \alpha \mathbb{E}_{s_t \sim P} \left[ \nabla_{\phi} H^T(s_t) \right]
\]

where \( H^T(s_t) = -\sum_{a} \pi(a_t|s_t) \log \pi(a_t|s_t) \) represents the Shannon entropy of \( \pi \) and \( \alpha > 0 \) is a temperature parameter that controls the level of stochasticity of the policy, which can be either fixed as a constant or optimized with objective function [22].

C. DQN Algorithm

DQN is perhaps the most popular RL algorithm for discrete action. It estimates the \( Q \) value \( (Q_\theta) \) using a deep neural network with trainable parameters \( \theta \). The DQN algorithm interleaves policy-evaluation and policy-improvement processes to search for the optimal \( Q \) value iteratively by following the direction of the gradient of \( \theta \):

\[
\theta \leftarrow \theta - \lambda Q \left( y_{\text{target}} - Q_\theta(s_t, a_t) \right) \nabla_\theta Q_\theta(s_t, a_t)
\]

where \( \lambda Q \) is the learning rate for parameters \( \theta \) and \( y_{\text{target}} \) is the target regression value computed by one-step bootstrapping

\[
y_{\text{target}} \equiv r_t + \gamma \max_a Q_{\text{target}}(s_{t+1}, a_{t+1}).
\]

Once finding the optimal \( Q^* \), the corresponding (optimal) policy \( \pi^* \) can be read out as follows:

\[
\pi^*(a_t|s_t) = \text{argmax}_a Q^*(s_t, a_t).
\]

In order to stabilize the training procedure, DQN employs two techniques. First, it maintains a target \( Q \)-network \( Q_{\text{target}} \), which duplicates the architecture of the behavior network \( Q_\theta \).
but with the independent parameters \( \theta_{\text{target}} \). \( \theta_{\text{target}} \) gets updated periodically by copying from \( \theta \). When computing \( Q_{\text{target}} \) in (6), DQN substitutes \( Q_{\text{target}} \) rather than \( Q \) to get a stable target value. Second, the training data are randomly sampled from an experience replay [34] regardless of their time dependence. The experience reply can be interpreted as a buffer storing and dynamically refreshing the data. These two tricks are critical for the performance of DQN.

IV. METHODOLOGY

This section first introduces the overall framework of the proposed method, whose architecture consists of two major components, i.e., a \( Q \) branch and a preference branch. Next, we derive a novel policy by combining the action preference with \( \epsilon \) greedy, which we call the preference-guided \( \epsilon \)-greedy policy and show that it preserves the policy improvement property of \( Q \) learning. Most importantly, this property offers strong theoretical grounding to the subsequent instantiation of the concrete algorithm. Then, we present theoretical insights showing that the action preference is updated along the direction of minimizing the statistical distance from the corresponding \( Q \) values, which is the key to achieving action preference learning and adequate exploration. Finally, we discuss the intuitive benefits of preference-guided \( \epsilon \)-greedy policy and present a concrete algorithm to train the proposed architecture efficiently.

A. Framework

Realizing the nonbiased exploration for DQN requires the candidate approach not to modify DQN’s fundamental logic, for example, its underlying objective. Therefore, one possible approach is to encourage the exploration from the external of DQN, which we show can be achieved by explicitly learning its action preference. Specifically, we learn the action preference via a preference branch, which shares a common embedding network with \( Q \) branch. Thus, we term our method the (action) preference-guided DQN (PGDQN). Once we infer the action preference, to encourage exploration, we motivate DQN to explore actions according to action preference distribution rather than simply taking greedy actions. Furthermore, we can learn adaptive action preference distribution, either unipeak or multipeak, depending on the landscape of \( Q \) values. As a result, without modifying DQN, we achieve an efficient sampling method that motivates the agent to sample actions from the distribution that closely coupled with corresponding \( Q \) values.

The overall framework of PGDQN is depicted in Fig. 1. It consists of two major components, i.e., a dual architecture with the \( Q \) branch and the preference branch, and a preference-guided \( \epsilon \)-greedy policy on the top. In particular, the \( Q \) branch is a copy of DQN with its original architectures and objectives. Meanwhile, the preference branch learns the action preference of the \( Q \) branch, whose architectures and objectives will be introduced later. Precisely, given a state, the shared embedding network first extracts a latent representation of the state from its basic features and passes to both \( Q \) branch and preference branch. Then, the \( Q \) branch and preference branch output the \( Q \) values and the action preference, respectively. Next, the preference-guided \( \epsilon \)-greedy policy either samples a greedy action according to the \( Q \) values with probability \( 1 - \epsilon \) or samples an action proportional to the action preference with probability \( \epsilon \). Finally, the selected action is executed in the environment to trigger state transition for the next round of interaction.

In our case, the shared embedding network is critical for learning diverse knowledge. The weights of the embedding network are updated via backpropagation from both objectives of the \( Q \) branch and the preference branch simultaneously. This weight sharing is a widely used technique in multitask learning [35]. Intuitively speaking, the shared embedding network learns the latent representation from two different perspectives: maximizing the \( Q \) values and inferring action preference. The knowledge acquired by sharing weights is more helpful to search for optimal policies than optimizing a single objective, i.e., maximizing the \( Q \) values. We provide empirical results in Section V-F to show this. Next, we give the details of the preference-guided \( \epsilon \)-greedy policy and the theoretical insights to support action preference learning and nonbiased exploration for DQN. For simplicity, we omit time step \( t \) in all formulas in the rest of this article.

B. Preference-Guided \( \epsilon \)-Greedy Policy

We construct the preference-guided \( \epsilon \)-greedy policy, denoted as \( \pi_{\text{PG}}(a|s) \), by incorporating action preference with \( \epsilon \) greedy. Specifically, any action \( a \) has a probability \( \epsilon \) to be sampled proportional to its preference, or the greedy action \( a^* = \text{argmax}_a Q(a,s) \) will be selected at each state \( s \)

\[
\pi_{\text{PG}}(a|s) = \begin{cases} 
1 - \epsilon + \epsilon \eta(a|s), & \text{if } a = a^* \\
\epsilon \eta(a|s), & a \neq a^*
\end{cases}
\]

(8)

where \( a \sim \eta(\cdot|s) \) is an action distribution representing action preference. The more preference of \( a \), the higher probability...
it has. \( \epsilon \in (0, 1) \) is a small coefficient. Note that we have no restrictions on the action preference \( \eta \) as long as it is an action distribution. Therefore, we can interpret the original \( \epsilon \)-greedy policy as a particular case of \( \pi_{PG}(a|s) \) with a uniform action preference \( \eta(a|s) = U[0, 1] \), where \( U \) represents uniform distribution. We now show that \( \pi_{PG}(a|s) \) possesses an appealing property: the policy \( \pi_{PG}(a|s) \) after one step of update with action preference \( \eta(a|s) \) is at least as good as before, i.e., \( \pi_{PG}(a|s) \) possesses a policy improvement guarantee.

**Theorem 1 (Policy Improvement Theorem):** For any \( \pi_{PG}^i (a|s) \) with action preference \( \eta^i (a|s) \) and state–action value \( Q^i \), the policy \( \pi_{PG}^{i+1} (a|s) \) one step greedily updated by maximizing expected overall return, with corresponding action preference \( \eta^{i+1}(a|s) \), is a monotonic improvement, i.e., \( Q^{i+1} \) \( \geq \) \( Q^i \), for all \( (s, a) \in S \times A \) and \( \epsilon \in (0, 1) \).

**Proof:** Following the definition of preference-guided \( \epsilon \)-greedy policy, we can expand the expectation of state-action value in (2) with respect to \( Q^i \):

\[
Q^i(s, \pi_{PG}^{i+1}(a|s)) = \mathbb{E}_{a \sim \pi_{PG}^{i+1}}\left[ Q^i(s, a) \right] \\
= \sum_a \pi_{PG}^{i+1}(a|s) Q^i(s, a) \\
= \epsilon \sum_a \eta^{i+1}(a|s) Q^i(s, a) \\
+ (1 - \epsilon) \max_a Q^i(s, a). \tag{9}
\]

The second term in (9) can be further expanded as follows:

\[
\max_a Q^i(s, a) = \max_a Q^0(s, a) \frac{1 - \epsilon}{1 - \epsilon} \\
= \max_a Q^0(s, a) \sum_a \pi_{PG}(a|s) - \epsilon \eta^i(a|s) \frac{1 - \epsilon}{1 - \epsilon} \\
\geq \sum_a \pi_{PG}(a|s) - \epsilon \eta^i(a|s) Q^i(s, a) \\
= V^i(s) - \epsilon \sum_a \eta^i(a|s) Q^i(s, a). \tag{10}
\]

By greedily maximizing the expected total return (expectation of \( Q \) value) with a one-step look ahead on \( \pi_{PG}(a|s) \), we can obtain \( \eta^{i+1} \) from \( \eta^i \):

\[
\mathbb{E}_{a \sim \pi_{PG}^i} \left[ Q^i(s, a) \right] \\
\leq \mathbb{E}_{a \sim \pi_{PG}^i} \left[ Q^{i+1}(s, a) \right] \\
\epsilon \sum_a \eta^i(a|s) Q^i(s, a) + (1 - \epsilon) \max_a Q^i(s, a) \\
\leq \epsilon \sum_a \eta^{i+1}(a|s) Q^{i+1}(s, a) + (1 - \epsilon) \max_a Q^i(s, a) \tag{11}
\]

i.e., we have

\[
\sum_a \eta^i(a|s) Q^i(s, a) \leq \sum_a \eta^{i+1}(a|s) Q^{i+1}(s, a). \tag{12}
\]

Now, substitute (12) into (10)

\[
\max_a Q^{i+1}(s, a) \geq \frac{V^{i+1}(s)}{1 - \epsilon} - \frac{\epsilon}{1 - \epsilon} \sum_a \eta^{i+1}(a|s) Q^{i+1}(s, a) \\
\geq \frac{V^{i+1}(s)}{1 - \epsilon} - \frac{\epsilon}{1 - \epsilon} \sum_a \eta^{i+1}(a|s) Q^{i+1}(s, a). \tag{13}
\]

Then, (9) can be rewritten as follows:

\[
Q^{i+1}(s, \pi_{PG}^{i+1}(a|s)) \\
= \mathbb{E}_{a \sim \pi_{PG}^i} \left[ Q^i(s, a) \right] \\
\geq \epsilon \sum_a \eta^{i+1}(a|s) Q^{i+1}(s, a) \\
+ (1 - \epsilon) \left[ V^{i+1}(s) - \frac{\epsilon}{1 - \epsilon} \sum_a \eta^{i+1}(a|s) Q^{i+1}(s, a) \right] \\
= V^{i+1}(s). \tag{14}
\]

Therefore, we can prove Theorem 1 by expanding the state–action value as follows:

\[
Q^{i+1}(s, a) = r_0 + \mathbb{E}_{s \sim p} \left[ \gamma V^{i+1}(s_1) \right] \\
\leq r_0 + \mathbb{E}_{s \sim p} \left[ \gamma \mathbb{E}_{a \sim \pi_{PG}^i} \left[ Q^{i+1}(s_1, a_1) \right] \right] \\
= r_0 + \mathbb{E}_{s \sim p} \left[ \gamma r_1 + \mathbb{E}_{s \sim p} \left[ Q^{i+1}(s_2, a_2) \right] \right] \\
\leq r_0 + \mathbb{E}_{s \sim p} \left[ \gamma r_1 + \gamma^2 \mathbb{E}_{s \sim p} \left[ Q^{i+1}(s_2, a_2) \right] \right] \\
\vdots \\
\leq r_0 + \mathbb{E}_{s \sim p} \left[ \sum_{t=1}^T \gamma^t r_t \right] \\
= Q^{i+1}(s, a)
\]

where \( \tau \) represents the trajectory.

No restrictions on action preference distribution \( \eta \) is a significant benefit, since the policy improvement theorem will hold for any learned preferences. Therefore, we can learn a specific preference \( \eta \) whose shape aligns with that of the \( Q \) values, i.e., they are close in terms of a statistical measure. Then, we can achieve action preference learning and an efficient unbiased exploration for DQN.

**C. Learning Action Preference**

Exploration means trying out actions that are never selected by the greedy policy. Since the original DQN explores new actions with equal probability, increasing the sampling probability of all the valuable actions (actions with high \( Q \) values) is the key to enhance sampling efficiency. We show that this idea can be realized by learning an action preference \( \eta \) that explicitly models the distribution in line with the landscape.
of corresponding \( Q \) values. First, we present the objective function of learning such action preference \( \eta \)
\[
L(\phi) = \mathbb{E}_{s \sim P, \ a \sim \eta} \left[ (Q^\eta(s, a) - V^\eta(s))\eta_\phi(a|s) + \alpha \mathcal{H}^\eta(s) \right] 
\]
\[
= \mathbb{E}_{s \sim P, \ a \sim \eta} \left[ A^\eta(s, a)\eta_\phi(a|s) + \alpha \mathcal{H}^\eta(s) \right] 
\]  
(15)
where the values of \( \phi \) are the parameters of action preference and \( \mathcal{H}^\eta(s) = -\sum_a \eta_\phi(a|s) \log \eta_\phi(a|s) \) is the Shannon entropy of \( \eta_\phi(a|s) \) at state \( s \). There are three significant reasons that we decided to optimize the action preference along the direction of maximizing the expected overall return rather than directly getting the distribution by taking softmax operation over \( Q \) values. First, action preference optimized by maximizing overall return can guarantee the policy improvement theorem (Theorem 1), while the latter method cannot. Second, the latter method results in premature convergence to suboptimal solutions due to softmax operation based on false-positive \( Q \) values during the early stage when \( Q \) values are inaccurately estimated. On the contrary, exploration based on action preference can avoid this issue by sampling actions from a smooth distribution that maximizes the entropy. Last but not least, optimizing the action preference \( \eta_\phi(a|s) \) by maximizing the above objective function (15) is equivalent to minimizing the statistical distance between action preference and corresponding \( Q \) values.

**Proposition 1:** Optimizing the action preference through maximizing entropy regularized objective function is equivalent to minimizing \( KL \) divergence between action preference and corresponding \( Q \) values.

According to (15), one can formulate the optimization problem as follows:

argmax
\[
\phi \mathcal{L}(\phi) 
= -\arg\min \mathcal{L}(\phi) 
= -\arg\min \phi \sum_a \eta_\phi(a|s)(Q^\eta(s, a) - V^\eta(s) - \alpha \log \eta_\phi(a|s)). 
\]  
(16)

Scaling the objective function by temperature parameter

argmax
\[
\phi \frac{1}{\alpha} \mathcal{L}(\phi) 
= \arg\min \phi \sum_a \eta_\phi(a|s)\left[ \log \eta_\phi(a|s) - 1/\alpha(Q^\eta(s, a) - V^\eta(s)) \right] 
\]
\[
= \arg\min \phi \left[ \sum_a \eta_\phi(a|s) \log \eta(a|s) 
- \sum_a \eta_\phi(a|s) \log \left( \frac{e^{\frac{1}{\alpha}Q^\eta(s,a)}}{e^{\frac{1}{\alpha}V^\eta(s)}} \right) \right] 
\]
\[
= \arg\min D_{KL}(\eta_\phi(a|s) \| \frac{e^{\frac{1}{\alpha}Q^\eta(s,a)}}{e^{\frac{1}{\alpha}V^\eta(s)}}) 
\]  
(17)
where \( C^\eta(s) = \exp((1/\alpha)V^\eta(s)) \) is a constant term with respect to action preference parameter \( \phi \) and \( D_{KL} \) is the KL divergence [36] that measures the statistical distance of two objects. Therefore, when estimated \( Q \) values become more accurate, actions with larger \( Q \) values can be sampled more frequently, while actions with smaller \( Q \) values still have a chance to be explored (i.e., preference-guided exploration). We also empirically show action preference learning through visualization method in Section V.

Now, we present a discussion about the intuitive benefits of our proposed exploration method. Following Theorem 1 and Proposition 1, the preference-guided \( \epsilon \)-greedy policy is monotonically improved and samples the action based on the action preference instead of random exploration over uniform distribution. Intuitively, this exploration method is more in line with ethology. According to the Law of Effect mentioned in The Principles of Learning and Behavior [37], the behavior is more likely to occur with a favorable consequence and less likely to repeat with an unsatisfying response. Notice that the preference branch updates its parameters exactly in this way (regarding positive advantage value as a reward and negative advantage value as a punishment). Furthermore, the action preference asymptotically approaches the distribution in line with the landscape of \( Q \) values, since it is optimized in the sense of minimizing the statistical distance from its corresponding \( Q \) values. Consequently, given an arbitrary state, the landscape of the overall action distribution is adaptive to that of the corresponding \( Q \) values. In other words, the action preference is either unipick (unimodal) when one \( Q \) value dominates the others or multipeak (multimodal) in case there exist several high-\( Q \) values. Therefore, by exploring stochastic actions over the action preference distribution closely coupled with \( Q \) values, PGDQN is potentially more data-efficient than vanilla \( \epsilon \) greedy (uniform distribution) and NoisyNet (unimodality).

**D. Training Algorithm for PGDQN**

This section presents a tangible algorithm to train the dual architecture effectively. According to the objective function in (15), the preference branch follows a one-step online update, and the rule for calculating new parameters of the preference branch can be written as follows:

\[
\Delta \phi \propto \mathbb{E}_{s \sim P, \ a \sim \eta} \left[ A^\eta(s, a)\log \eta_\phi(a|s) + \alpha \mathcal{H}^\eta(s) \right]. 
\]  
(18)

Also, we automatically tune the temperature parameter \( \alpha \) by solving a minimax optimization problem [24]

\[
\min_{\phi} \max_{\alpha \geq 0} \left( \mathbb{E}_{s \sim P, \ a \sim \eta} \left[ A^\eta(s, a)\eta_\phi(a|s) + \alpha \mathcal{H}^\eta(s) \right] \right) - \alpha \xi_{\text{entropy}} 
\]  
(19)
where \( \xi_{\text{entropy}} \) is minimum expected entropy.

As for the \( Q \) branch, the parameter update follows the offline batch update, which is exactly the same as that of original DQN except for the sampling approach:

\[
\Delta \theta \propto \left( \mathbb{E}_{\text{target}} \nabla_{\theta} Q_{\text{target}}^\eta(s, a) - Q^\eta(s, a) \right) \nabla_{\theta} Q_\theta(s, a) 
\]  
(20)
where \( Q_{\text{target}} \) is preference-guided \( \epsilon \)-greedy policy defined in (8) and subjects to \( \sum_a \pi_\phi(a|s) = 1 \).

Lumping all procedures, the final algorithm is provided in Algorithm 1.
Algorithm 1: PGDQNs

Initialize behavior network parameters: $\phi, \theta$.
Initialize entropy parameters: $\alpha, \xi_{\text{entropy}}$.
Initialize learning rates: $\lambda_Q, \lambda_n, \lambda_\alpha$.
Initialize batch size $N$ and replay buffer $D \leftarrow \emptyset$.
Assign target parameters: $\phi_{\text{target}} \leftarrow \phi, \theta_{\text{target}} \leftarrow \theta$.

for episode $= 1$ to $E$ do
    Initialize the environment: $s_i \sim \text{Env}$
    for step $= 1$ to $S$ do
        Sample an action: $a_t \leftarrow \pi_{PG}(a_t|s_t)$
        Interact with the environment: $r_t, s_{t+1} \sim \text{Env}$
        Store the transition: $D \leftarrow D \cup (s_t, a_t, r_t, s_{t+1})$
        if step mod $\tau_\phi$ then
            Sample the current online data: $(s'_t, a'_t, r'_t, s'_{t+1}) \sim \text{Env}$
            Compute loss function of preference branch $\mathcal{L}_\phi(\phi)$.
            Update preference branch: $\phi \leftarrow \phi - \lambda_n \nabla_\phi \mathcal{L}_\phi(\phi)$
            Update temperature parameter: $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha$
        end if
        if step mod $\tau_Q$ then
            Sample a batch of the data: $(s'_i, a'_i, r'_i, s'_{i+1})_{i=1}^N \sim D$
            Compute loss function of Q branch $\mathcal{L}_Q(\theta)$.
            Update Q branch: $\theta \leftarrow \theta - \lambda_Q \nabla_\theta \mathcal{L}_Q(\theta)$
        end if
        if epoch mod $\tau_{\text{target}}$ then
            Update target network: $\theta_{\text{target}} \leftarrow \theta, \phi_{\text{target}} \leftarrow \phi$
        end if
    end for
end for

V. EXPERIMENT

To thoroughly assess our PGDQN algorithm, we test it in two different classes of environments, respectively, which include three popular Classical Control tasks and six Atari games [38]. In the following, we provide the environment setup and the configuration of the PGDQN algorithm and introduce the baselines we use.

A. Environment Setup and PGDQN Configuration

1) Classic Control Tasks: The three Classic Control tasks are CartPole, MountainCar, and Arcrobot, which follow the default settings of that of the OpenAi-Gym suite [38].

2) Atari Games: The six Atari games are Alien, Pong, Berzerk, CrazyClimber, FishingDerby, and Amidar, which follow the default settings of that of the OpenAi-Gym suite [38]. Among these six games, three are low-action-dimension games with action dimensionality less or equal to 10, and they are Pong (six actions), CrazyClimber (nine actions), and Amidar (ten actions). In contrast, the other three games are considered more challenging high-action-dimension games with more than ten actions (18 actions for Alien, Berzerk, and FishingDerby).

3) PGDQN Configuration: We adjust configurations of PGDQN regarding the different classes of environments, and the details can be found in Tables I and II. Specifically, we keep the $Q$ branch hyperparameters settings the same as those from the original DQN paper, while the unique hyperparameters of the preference branch are determined empirically through experiments over Classic Control task, which are discussed in Section V-G. During training, we let PGDQN learn for a maximum of 30 million (30M) frames for Atari games, whereas learning till convergence in Classic Control tasks, since they are fast in computation and consume relatively fewer resources.

B. Baselines

In this article, we consider four baseline algorithms, including the original DQN algorithm [12] and its two well-known variants the double DQN (D2QN) [16] and the variant of double DQN with dueling architecture (V-D D3QN) [18], as well as a state-of-the-art noisy-based exploration algorithm for DQN, namely, the NoisyNet-DQN. Here, we do not consider soft $Q$-learning algorithm [21] as our baseline, since it targets the tasks with continuous actions, thus not applicable.

| Parameter                           | Value    |
|-------------------------------------|----------|
| minibatch size                      | 32       |
| reply buffer size                   | 1000000  |
| learning start size                 | 50000    |
| discount factor                     | 0.99     |
| agent history length                | 4        |
| action repeat                       | 4        |
| $Q$ branch update frequency         | 4        |
| preference branch update frequency  | 4        |
| target network update frequency     | 100000   |
| RMSProp learning rate for preference-branch | 0.00025 |
| RMSProp learning rate for Q-branch  | 0.00025  |
| RMSProp learning rate for baselines | 0.00025  |
| Adam learning rate for temperature parameter | 0.00025 |
| initial exploration                | 1        |
| final exploration                  | 0.1      |
| final exploration frame            | 1000000  |

TABLE II

| Parameter                              | Value     |
|----------------------------------------|-----------|
| input dimension                        | [84, 84, 4] |
| first hidden layer                     | [4, 32]   |
| kernel size                            | 8 $\times$ 8 |
| stride size                            | 4        |
| second hidden layer                    | [32, 64]  |
| kernel size                            | 4 $\times$ 4 |
| stride size                            | 2        |
| third hidden layer                     | [64, 64]  |
| kernel size                            | 3 $\times$ 3 |
| stride size                            | 1        |
| latent state space dimension           | [7, 7, 64] |
| fully connected layer for Q-branch     | 256       |
| fully connected layer for preference branch | 256    |
| fully connected layer for baselines    | 512       |
| output dimension                       | dim($\mathcal{A}$) |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
for discrete-action tasks in this article. All these baselines are trained for a maximum of 30 million (30M) frames for Atari games and till convergence for Classic Control tasks. The network architectures and all other hyperparameters follow those in their original paper. The detailed configurations for baseline methods are provided in Tables I and II. All experiments are carried out in a machine with an AMD Ryzen 3900X CPU and a single NVIDIA GeForce 1660S GPU.

C. Learning Results

The learning curves of PGDQN and all the baseline methods for the nine environments are given in Fig. 2. The red dotted line and solid lines represent the average episode returns of PGDQN and baseline methods, respectively, which are collected from five different random seeds. The shaded areas indicate the variance of each method during training. From Fig. 2, we can visualize that PGDQN consistently outperforms all baselines with a faster convergence speed in all benchmark environments while possessing a lower variance, which basically confirms that our PGDQN algorithm is an effective exploration method for DQN. To have a better quantitative result, we compute the amount of improvement of PGDQN against baselines according to three different metrics and rank all the methods based on their overall performance and data efficiency. Specifically, the three selected metrics are the average percentage of performance improvement, the average percentage of efficiency improvement, and the average performance rank. In particular, the average percentage of performance improvement has the form

$$\frac{sc(PGDQN) - sc(Baseline)}{sc(Baseline)}$$

where $sc(PGDQN)$ and $sc(Baseline)$ are the average highest scores (across five random seeds) achieved by PGDQN and the baseline method being compared, respectively. Therefore, the higher this metric, the better the performance against the baseline.
improvement measures the percentage of frames reduced when PGDQN reaches the average highest score (across five random seeds) of the baseline:

\[
\text{frm}(\text{Baseline, } \text{sc}^*(\text{Baseline})) - \text{frm}(\text{PGDQN, } \text{sc}^*(\text{Baseline}))
\]

\[
\text{frm}(\text{Baseline, } \text{sc}^*(\text{Baseline}))
\]

(22)

where \( \text{frm}(\text{Baseline, } \text{sc}^*(\text{Baseline})) \) is the total amount of frames the baseline required to achieve its highest episode reward, while \( \text{frm}(\text{PGDQN, } \text{sc}^*(\text{Baseline})) \) is the total amount of frames needed for PGDQN to achieve the baselines’ highest score. Intuitively speaking, the fewer frames PGDQN needs to reach baseline performance, the faster PGDQN learns. Finally, the average performance rank provides the overall ranking of all methods according to their average best performance and average data efficiency. Specifically, we rank any pair of algorithms first according to their best performance. However, if some methods possess equally good performance, we rank based on their average data efficiency percentage. Specifically, we rank any pair of algorithms first according to their best performance. However, if some methods possess equally good performance, we rank based on their average data efficiency percentage.

As shown in Tables III and IV, PGDQN demonstrates an improvement by a large margin against baselines in terms of both performance and data efficiency in the Classic Control tasks and almost all Atari games. For example, PGDQN outperforms D2QN by up to 136% in Amidar environment and consumes 85% fewer data as compared with DQN in Berzerk environment. However, the overall improvement on Classic Control tasks is relatively less significant than that of Atari games. One possible reason could be the simplicity of the task, as Classic Control tasks are considered less challenging than Atari games due to smaller action space, simpler state representation, and easily attainable goals. The only exception is the FishingDerby environment, where, however, PGDQN shows a degradation down to 10% of efficiency against DQN and 4% of performance against V-D D3QN algorithm. Notice that all the methods reach the possible maximum episode rewards for the environments CartPole, MountainCar, and Pong; thus, performance improvements of these environments are not evaluated in Table IV. Nonetheless, our PGDQN possesses a clear faster convergence speed and better performance than the other methods. Moreover, we observe that PGDQN ranks first in almost all environments (except for FishingDerby) for either the performance or the data efficiency in Fig. 3.

**D. Action Preference Visualization**

Here, we visualize the action preference and state–action values in Fig. 4 to empirically verify that the learned action preference is indeed coupled with its corresponding \( Q \) values, and they have similar landscapes. To demonstrate, we use the Pong environment from Atari Games as an example. In specific, the left pair of heat maps shows the action preference (top diagram) and normalized state–action values (bottom diagram) of PGDQN, whereas on the right, we have two pairs of bar charts zooming in two given time steps (indicated with red dots between the heat maps on the left). From these results, we observe that the action preference is proportional to the amplitude of the \( Q \) values. Specifically, for the given first-time step, it shows clearly that the action preference does form...
a multipeaks distribution (i.e., multimodality) following the multimodal landscape of the $Q$ values. Similarly, for the given second-time step, the action preference does possess a single peak (i.e., unimodality) where the value of the corresponding $Q$ value is significantly higher than the others. Therefore, we verify that the preference branch has successfully learned the action preference distribution in line with the landscape of corresponding $Q$ values by maximizing the objective function in (15).

**E. Comparing With Counter-Based Exploration**

Besides the stochastic exploration approaches we have thoroughly studied in Section V-C, we would like to also investigate how superior is our method compared with other types of exploration strategy, such as counter-based exploration. In light of this, we compare our algorithm with additional baseline called upper-confidence bounds with $Q$ ensembles (UCB $Q$ ensemble) [39] on two Atari games, and they are Berzerk and Pong environment. Fig. 5 illustrates the learning effort of two algorithms. As for the Berzerk environment, we can observe that PGDQN shows superior learning behavior in terms of both efficiency and performance compared with UCB $Q$ ensemble. Similarly, the converge curve of Pong demonstrates that our algorithm clearly consumes less data than the baseline to achieve maximum rewards, indicating the potential superiority of the PGDQN is not limited to stochastic exploration-based DQN algorithms.

**F. Ablation Study for Shared Embedding Network**

To verify that the knowledge learned via sharing embedding network indeed improves the performance of DQN, we compare the original DQN with a modified PGDQN. Specifically, the modified PGDQN employs the dual architecture with the shared embedding network but adopts the DQN’s $\epsilon$-greedy policy to sample actions. In other words, the modified PGDQN is the same as DQN except that the weights of the embedding network also get updated by the objective of the preference branch, i.e., it learns the knowledge from two objectives concurrently. We use two Atari games, namely, the Alien environment and the Amidar environment, as the test beds. The results are shown in Fig. 6. We can observe that the PGDQN with $\epsilon$ greedy outperforms DQN by a clear
The algorithm is robust to these two hyperparameters. Considering results are presented in Fig. 7, which shows that our PGDQN for Action Preference Branch

\begin{equation}
\eta \sim \exp(-\lambda \cdot \text{loss})
\end{equation}

is simple, we omit subscript \( \eta \). We have decided to fix these two hyperparameters as \( \{ \tau = 5, \lambda = 2.5e-4 \} \) for all the environments in this work.

\section{VI. Conclusion}

This article proposes a novel DRL algorithm realizing an efficient and nonbiased exploration for DQN by introducing an action preference branch, namely, the PGDQN algorithm. Specifically, we combine the preference branch with the \( \epsilon \)-greedy strategy to sample actions. We theoretically prove that the preference-guided \( \epsilon \)-greedy policy preserves the policy improvement property and empirically show the inferred preferences, obtained by maximizing entropy regularized objective function, explicitly expressing the favor of corresponding \( Q \) values. Consequently, during the early stage when \( Q \) values are inaccurate, all actions are explored from a smooth distribution that maximizes the entropy (i.e., random exploration), and as the \( Q \) values become more accurate, actions with larger \( Q \) values can be sampled more frequently, while those with smaller \( Q \) values still have a chance to be explored (i.e., preference-guided exploration), thus encouraging the overall exploration.

Extensive experiments in nine challenging environments confirm the superiority of our proposed method in terms of performance and convergence speed. In the future, we can extend our PGDQN algorithm to the human-in-the-loop (HiL) RL framework to address real-world complex tasks. One possible extension of our work will be combining human policy with action preference to solve high-level decision-making problems of autonomous driving with the personalized driving style.

\begin{thebibliography}{10}
\bibitem{1} Z. Cao and C.-T. Lin, “Reinforcement learning from hierarchical critics,” \textit{IEEE Trans. Neural Netw. Learn. Syst.}, vol. 34, no. 2, pp. 1066–1073, Feb. 2023.
\bibitem{2} P. R. Wurman et al., “Outracing champion gran turismo drivers with deep reinforcement learning,” \textit{Nature}, vol. 602, no. 7896, pp. 223–228, Feb. 2022.
\bibitem{3} J. Han, L. Yang, D. Zhang, X. Chang, and X. Liang, “Reinforcement-cutting-branch learning for video object segmentation,” in \textit{Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.}, Jun. 2018, pp. 9080–9089.
\bibitem{4} D. Zhang, J. Han, L. Zhao, and T. Zhao, “From discriminant to complete: Reinforcement searching-agent learning for weakly supervised object detection,” \textit{IEEE Trans. Neural Netw. Learn. Syst.}, vol. 31, no. 12, pp. 5549–5560, Dec. 2020.
\bibitem{5} L. Xie et al., “Learning with stochastic guidance for robot navigation,” \textit{IEEE Trans. Neural Netw. Learn. Syst.}, vol. 32, no. 1, pp. 166–176, Jan. 2021.
\bibitem{6} W. Huang, Y. Zhou, X. He, and C. Lv, “Goal-guided transformer-enabled reinforcement learning for efficient autonomous navigation,” 2023, arXiv:2301.00362.
\bibitem{7} Z. Huang, J. Wu, and C. Lv, “Efficient deep reinforcement learning with imitative expert priors for autonomous driving,” \textit{IEEE Trans. Neural Netw. Learn. Syst.}, early access, Jan. 26, 2022, doi: 10.1109/TNNLS.2022.3142822.
\bibitem{8} J. Wu, Z. Huang, W. Huang, and C. Lv, “Prioritized experience-based reinforcement learning with human guidance for autonomous driving,” \textit{IEEE Trans. Neural Netw. Learn. Syst.}, early access, Jun. 10, 2022, doi: 10.1109/TNNLS.2022.317685.
\bibitem{9} W. Liu, J. Sun, G. Wang, F. Bullo, and J. Chen, “Data-driven resilient predictive control under denial-of-service,” \textit{IEEE Trans. Autom. Control}, vol. 68, no. 8, pp. 4722–4737, Aug. 2023.
\bibitem{10} X. Wang, J. Sun, G. Wang, F. Allgöwer, and J. Chen, “Data-driven control of distributed event-triggered network systems,” \textit{IEEE/CAA J. Autom. Sinica}, vol. 10, no. 2, pp. 351–364, Feb. 2023.
\bibitem{11} Y. Li, X. Wang, J. Sun, G. Wang, and J. Chen, “Data-driven consensus control of fully distributed event-triggered multi-agent systems,” \textit{Sci. China Inf. Sci.}, vol. 66, no. 5, May 2023, Art. no. 152202.
\bibitem{12} V. Mnih et al., “Human-level control through deep reinforcement learning,” \textit{Nature}, vol. 518, no. 7540, pp. 529–533, 2015.
\bibitem{13} H. Kim, M. Jordan, S. Sastry, and A. Ng, “Autonomous helicopter flight via reinforcement learning,” in \textit{Proc. Adv. Neural Inf. Process. Syst.}, vol. 16, 2003, pp. 799–806.
\bibitem{14} Y. Yang, B. Kiumarsi, H. Modares, and C. Xu, “Model-free \( \lambda \)-policy iteration for discrete-time linear quadratic regulation,” \textit{IEEE Trans. Neural Netw. Learn. Syst.}, vol. 34, no. 2, pp. 635–649, Feb. 2023.
\bibitem{15} Y. Yang, H. Modares, K. G. vanVoudakis, W. He, C.-Z. Xu, and D. C. Wunsch, “Hamiltonian-driven adaptive dynamic programming with approximation errors,” \textit{IEEE Trans. Cybern.}, vol. 52, no. 12, pp. 13762–13773, Dec. 2022.
\bibitem{16} H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in \textit{Proc. AAAI Conf. Artif. Intell.}, vol. 30, no. 1, 2016, pp. 2094–2100.
\end{thebibliography}
[17] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in Proc. Int. Conf. Mach. Learn., 2016, pp. 1995–2003.
[18] Y. Huang, G. Wei, and Y. Wang, “V-D D3QN: The variant of double deep Q-learning network with dueling architecture,” in Proc. 37th Chin. Control Conf. (CCC), Jul. 2018, pp. 9130–9135.
[19] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, “Distributional reinforcement learning with quantile regression,” in Proc. AAAI Conf. Artif. Intell., vol. 32, no. 1, 2018, pp. 2892–2901.
[20] M. Fortunato et al., “Noisy networks for exploration,” in Proc. Int. Conf. Learn. Represent., 2018.
[21] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, “Reinforcement learning and autonomous navigation, reinforcement learning, and human-in-the-loop reinforcement learning,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 1352–1361.
[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 1861–1870.
[23] B. R. Kiran et al., “Deep reinforcement learning for autonomous driving: A survey,” IEEE Trans. Intell. Transp. Syst., vol. 23, no. 6, pp. 4909–4926, Jun. 2022.
[24] T. Haarnoja et al., “Soft actor-critic algorithms and applications,” 2018, arXiv:1812.05905.
[25] M. G. Bellemare, W. Dabney, and R. Munos, “A distributional perspective on reinforcement learning,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 449–458.
[26] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, “Implicit quantile networks for distributional reinforcement learning,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 1096–1105.
[27] Q. Lan, Y. Pan, A. Fyshe, and M. White, “Maxmin Q-learning: Controlling the estimation bias of Q-learning,” in Proc. Int. Conf. Learn. Represent., 2020.
[28] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning.” Nature, vol. 521, no. 7553, pp. 436–444, 2015.
[29] M. Hessel et al., “Rainbow: Combining improvements in deep reinforcement learning,” in Proc. 32nd AAAI Conf. Artif. Intell., 2018, pp. 3215–3222.
[30] M. Plappert et al., “Parameter space noise for exploration,” in Proc. Int. Conf. Learn. Represent., 2018.
[31] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in Proc. Adv. Neural Inf. Process. Syst., vol. 12, 1999, pp. 1057–1063.
[32] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, “Natural actor-critic algorithms,” Automatica, vol. 45, no. 11, pp. 2471–2482, Nov. 2009.
[33] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 2018.
[34] L.-J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” Mach. Learn., vol. 8, nos. 3–4, pp. 293–321, May 1992.
[35] O. Sener and V. Koltun, “Multi-task learning as multi-objective optimization,” in Proc. Adv. Neural Inf. Process. Syst., vol. 31, 2018, pp. 525–536.
[36] S. Kullback and R. A. Leibler, “On information and sufficiency,” Ann. Math. Statist., vol. 22, no. 1, pp. 79–86, 1951.
[37] M. P. Donjran and J. W. Grau, The Principles of Learning and Behavior. USA: Cengage Learning, 2015. [Online]. Available: https://books.google.com.sg/books?id=vxLAAgAAQBAJ.
[38] G. Brockman et al., “OpenAI gym,” 2016, arXiv:1606.01540.
[39] R. Y. Chen, S. Sidor, P. Abbeel, and J. Schulman, “UCB exploration via Q-ensembles,” 2017, arXiv:1706.01502.

Wenhui Huang (Graduate Student Member, IEEE) received the B.S. degree in vehicle engineering from the Wuhan University of Technology, Wuhan, China, and the M.Sc. degree in mechanical engineering from the Polytechnic University of Milan, Milan, Italy, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree with the School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore. He served as a Research Engineer at HoloMatic Technology (Beijing) Company Ltd., Beijing, China, from 2019 to 2020. His research mainly focuses on autonomous driving, autonomous navigation, reinforcement learning, and human-in-the-loop reinforcement learning.

Chen Lv (Senior Member, IEEE) received the Ph.D. degree from the Department of Automotive Engineering, Tsinghua University, Beijing, China, in January 2016. He was a joint Ph.D. Researcher at UC Berkeley, Berkeley, CA, USA, from 2014 to 2015, and worked as a Research Fellow at Cranfield University, Cranfield, U.K., from 2016 to 2018. He is currently a Nanyang Assistant Professor at the School of Mechanical and Aerospace Engineering, Nanyang Technological University (NTU), Singapore, where he is also the Cluster Director of Future Mobility Solutions. He joined NTU, where he has been a Founder of the Automated Driving and Human–Machine System (AutoMan) Research Laboratory, since June 2018. His research focuses on intelligent vehicles, automated driving, and human–machine systems.