Reinforcement Learning with Potential Functions Trained to Discriminate Good and Bad States

Yifei Chen, Hamidreza Kasaei, Lambert Schomaker and Marco Wiering
Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence
University of Groningen, The Netherlands
Email: \{yifei.chen, hamidreza.kasaei, l.r.b.schomaker, m.a.wiering\}@rug.nl

Abstract—Reward shaping is an efficient way to incorporate domain knowledge into a reinforcement learning agent. Nevertheless, it is unpractical and inconvenient to require prior knowledge for designing shaping rewards. Therefore, learning the shaping reward function by the agent during training could be more effective. In this paper, based on the potential-based reward shaping framework, which guarantees policy invariance, we propose to learn a potential function concurrently with training an agent using a reinforcement learning algorithm.

In the proposed method, the potential function is trained by examining states that occur in good and in bad episodes. We apply the proposed adaptive potential function while training an agent with Q-learning and develop two novel algorithms. One is APF-QMLP, which applies the good/bad state potential function combined with Q-learning and multi-layer perceptrons (MLPs) to estimate the Q-function. The other is APF-Dueling-DQN, which combines the novel potential function with Dueling DQN. In particular, an autoencoder is adopted in APF-Dueling-DQN to map image states from Atari games to hash codes. We evaluated the created algorithms empirically in four environments: a six-room maze, CartPole, Acrobot, and Ms-Pacman, involving low-dimensional or high-dimensional state spaces. The experimental results showed that the proposed adaptive potential function improved the performances of the selected reinforcement learning algorithms.

I. INTRODUCTION

Reinforcement learning (RL) has achieved great success in various domains, such as game playing [1]–[3], robotics [4]–[6], and optimal control [7]–[10]. Designing an appropriate reward function is a critical step in RL because it implies the ultimate goal of the learning process. Rewards are sometimes very sparse in the environment, especially in real-world settings. This can make learning a good behavior very slow and therefore examining ways to change the reward function is important.

In the RL domain, many approaches, such as self-play and gradually increasing the task complexity [11]–[13], were proposed by getting inspiration from the behavioral psychology domain, e.g., a rat can learn to press a lever overtime when its behavior is reinforced with food [14], [15]. Reward shaping is an efficient way to incorporate domain knowledge into an agent. It can be formulated as an additive reward that is added to the environmental reward to speed up the learning process. However, it may also lead to unintended behaviors. One famous example is that the agent learned to drive a bicycle in circles instead of reaching the destination to get small but stable rewards [16].

Contributions of this paper. This paper proposes an adaptive potential function, termed APF, to serve in the potential-based reward shaping framework [17]. The proposed APF learns to shape rewards concurrently with learning the policy and state-action value function without the need for expert knowledge about the given task. The novel method is based on keeping track of good episodes and bad episodes and training the APF based on this distinction.

We use the APF in Q-learning [18] combined with multi-layer perceptrons (QMLP) and formed the APF-QMLP algorithm. To examine the performance of the proposed algorithm, we designed a $30 \times 30$ six-room maze environment. In the experiment, we compared the following four agents: (i) a QMLP agent, (ii) a QMLP agent with a count-based exploration strategy, (iii) an APF-QMLP agent, and (iv) an APF-QMLP agent with the count-based exploration strategy. Furthermore, we also compared the performance of the APF-QMLP agent with the QMLP agent in CartPole and Acrobat, i.e., two classic Open-AI gym environments, to show the effect of using the APF in well-known tasks with low-dimensional state spaces. The obtained results show the usefulness of adding the APF to QMLP to increase the learning speed or final performance for the maze problem and CartPole.

Additionally, we extend the APF method to be used in high-dimensional environments. In particular, an autoencoder network is employed to map a given state to a low-dimensional code, and then the obtained code is used for developing the APF. We combined the APF with the Dueling Deep Q-network (DQN) [19] and created the APF-Dueling-DQN algorithm. To evaluate the performance of the proposed algorithm, we con-
ducted a set of experiments in the Atari game Ms-Pacman. The results show that the APF-Dueling-DQN agent outperformed the Dueling DQN agent.

Paper outline. The remainder of this paper is organized as follows: first, related work will be discussed. Section III describes the fundamental concepts that are used in the paper. In Section IV, the details of the proposed APF method and the two developed algorithms are explained. The experimental results are presented in Section V. Finally, Section VI concludes the paper and discusses potential future works.

II. RELATED WORK

Reward shaping has been under investigation for a long time in various research fields, such as behavioral psychology, reinforcement learning, and robotics [20]–[22]. Although an exhaustive survey of reward shaping is beyond the scope of this paper, we will review the main efforts.

Ng et al. [17] showed that utilizing the difference in subsequent values of a potential function (PF) for reward shaping guarantees policy invariance. This means that using the PF in this way will not change the optimal policy and is therefore always safe to use. Wiewiora et al. [23] extended the PF over states to over both actions and states. The authors offered the look-ahead advice and look-back advice methods. For the look-ahead advice (further described in [24]), Wiewiora has proved that using a static PF is equivalent to simply initializing the Q-values based on the PF in a discrete domain. In [25], the authors presented the first demonstration that effectively combined human feedback with reward shaping, benefiting from applying the potential-based shaping reward. Moreover, the PF can also be applied in multi-agent systems and reserves the Nash Equilibrium of multiple agents [26]–[29].

In practice, learning a PF during training rather than specifying a static PF beforehand would be more useful. As reported in [29], Devlin and Kudenko demonstrated that potential-based reward shaping warrants the learning to converge despite the PF being dynamic or even misleading. This laid the groundwork for learning a dynamic PF.

Related works include learning the reward shaping function directly. In [26], Marthi approximated the distance function, which worked as a reward shaping function, by converting the problem to a simpler abstract one. The author used an approximate dynamic programming algorithm to approximate the distance function. The dynamics \( P(s' | s, o) \) were estimated by performing an option \( o \) in state \( s \) that terminates in some state \( s' \).

Some other works were focusing on learning the state-value function as the PF. In [30], Grzes and Kudenko used multigrid discretization to form two levels of discrete states and learned an abstract state-value function on the high level used as the potential for the ground level. However, the PF learned was tabular, which means the scalability to more complex problems is limited. Moreover, necessary knowledge was required for defining discretization. Later in [31], Grzes and Kudenko developed two algorithms for constructing the PF. One is for model-free RL, which was based on multigrid discretization, analogically to [30]. The other is for model-based RL, which learned the PF based on a free space assumption. Both algorithms learned the state-value function at an abstract level as the PF. Likewise, necessary knowledge of the free space assumption is required.

There are also some methods that learn the action-value function as the PF. Based on the work of Wiewiora et al. [23] and Devlin and Kudenko [29], Harutyunyan et al. proposed a method that learns a secondary action-value function as the dynamic advice PF based on the negation of an expert-provided reward function [32]. In [33], the authors extracted expert-defined rewards by using inverse reinforcement learning. Based on this, a secondary action-value function was learned as the PF. However, both learned PFs are based on human input, and therefore require designers with certain background knowledge. In all, learning a PF concurrently with the RL algorithm without the environmental model or expert knowledge remains a challenge.

All these previous methods work very differently from the proposed adaptive potential function in this paper. The proposed technique is based on keeping track of episodic rewards and training the APF to discriminate between states occurring in good and bad episodes.

Aiming at a different scientific challenge, researchers proposed count-based exploration algorithms to solve hard-exploration tasks. The idea was initially used in bandit problems for measuring the uncertainty of estimating the action’s value, using how many times the action was selected [34]. Similarly, in the RL domain, the number of visits of a state can be transformed into a reward bonus so that the agent is pushed to explore less-visited states [35], [36].

The recent development of deep RL promotes further development of count-based exploration algorithms. Bellemare et al. successfully generated pseudo-counts from raw pixels utilizing a density model [37]. The exploration bonuses were then calculated based on the pseudo-counts to tackle challenging exploration problems. Later in [38], Ostrovski et al. further researched density models and achieved better performances by using PixelCNN than the CTS density model [37]. On the other hand, Tang et al. [36] employed hash functions to map high-dimensional states to hash codes to count states using a hash table.

III. BACKGROUND

A. Markov Decision Processes

In this paper, we consider Markov decision processes (MDPs). A finite-state MDP can be modeled as a tuple \( M = \langle S, A, P, \gamma, R \rangle \), where \( S \) represents the set of all states of the environment. \( A \) is the set of actions that can be taken by the agent. \( P(s', a, s') \) represents the transition probability of transitioning from state \( s \) to \( s' \) when the agent takes action \( a \). The parameter \( \gamma \in [0, 1] \) is the discount factor. Moreover, \( R : S \times A \times S \mapsto \mathbb{R} \), is the environmental reward function.

The goal of an RL problem is to maximize the expected return, where the return \( G_t \) of a sequence of rewards after the time step \( t \) is defined as follows:
\[ G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \]  

where \( T \) is the terminal time step. The expected return from a state \( s \) is defined as the state-value function \( V^*_M \):  

\[ V^*_M(s) = \mathbb{E}[G_t | S_t = s] \]  

where \( \pi \) is the policy the agent follows. Then the action-value function \( Q^*_M \) at a state \( s \) when taking action \( a \) is defined as:  

\[ Q^*_M(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a] \]  

Accordingly, the optimal state-value function is \( V^*_M(s) = \sup_a V^*_M(s, a) \), and the optimal action-value function is \( Q^*_M(s, a) = \sup_a Q^*_M(s, a) \). The optimal policy \( \pi^*_M \) is obtained by selecting optimal actions using \( \pi^*_M(s) = \arg \max_a Q^*_M(s, a) \).

### B. Q-learning

Many RL algorithms were proposed to learn the optimal action-value function to obtain the optimal policy. Q-learning [18], which is an off-policy temporal difference algorithm, is a workhorse among RL algorithms. It learns from experiences \((S_t, A_t, R_{t+1}, S_{t+1})\) with \( S_t \) and \( A_t \) the state and action at time step \( t \), to update the Q-function with the following learning rule:

\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \]  

where \( \alpha \) is the learning rate. Q-learning with a proper learning rate schedule is guaranteed to converge if all state-action pairs are explored for infinite times [39]–[41].

### C. Changing the Reward Function

Our work is primarily based on the potential-based reward shaping framework [17] where a PF is used to shape rewards. This framework guarantees policy invariance. Additionally, inspired by the count-based exploration method [35], [36], we propose the novel APF method based on visitations of good and bad states in low- and high-dimensional state spaces. This subsection explains these two methods and the new APF method is explained in the next section.

1) Potential-based Shaping Function: In this work, the shaping reward function is denoted as \( F(s, a, s') : S \times A \times S \rightarrow \mathbb{R} \), which provides an additive reward to the environmental reward function \( R \). Hence, the shaped reward function is \( R'(s, a, s') = R(s, a, s') + F(s, a, s') \) and will be used during the learning process.

With a shaped reward function, the original MDP \( M \) is turned into a transformed MDP \( M' = \langle S, A, P, \gamma, R' \rangle \). However, the optimal policy learned for \( M' \) is not ensured to be optimal for \( M \) as well. The necessary and sufficient condition of policy invariance is that \( F \) is a potential-based function [17], which is defined in the form of the difference of a PF \( \Phi \):

\[ F(s, a, s') = \gamma \Phi(s') - \Phi(s) \]  

where \( \gamma \) is the same discount factor as in \( M \). This means any potential function \( \Phi \) can be used in this way. The PF can increase the learning speed, but some bad PFs will make it harder to learn the optimal policy. Therefore, creating a good PF is important, but may be complex.

2) Count-based Exploration: The count-based exploration strategy was proposed to tackle hard-exploration problems where the \( \epsilon \)-greedy strategy is incapacitated. The basic idea is to explore more on less-visited states by assigning bonuses accordingly. As is shown in Eq. (6), the exploration bonus at the state \( s \) is:

\[ r^+(s) = \frac{\beta}{\sqrt{N(s)}} \]  

where \( \beta \geq 0 \) is the bonus coefficient, and \( N(s) \) represents the number of times state \( s \) has been visited. Then an RL agent can be trained on rewards \( \{r + r^+\} \), where \( r \) is the environmental reward. Our work adopts the idea of using visitations of states but uses it differently. The difference is explained in Section IV-A. Furthermore, we also combine count-based exploration with the new APF method in the first experiment.

### IV. Methodology

#### A. Adaptive Potential Function

In this paper, we propose a simple and effective adaptive PF, termed APF, based on discriminating good and bad states. Specifically, the proposed APF is updated by analysing how often an agent visits a state in good and bad trajectories.

1) Updating rules of the APF: During training, the sequence of all states in an episode is stored as a trajectory \( \text{traj} \). Each trajectory is stored in a trajectory replay buffer \( D_{\text{traj}} \), which is implemented by a priority queue. Different trajectories are sorted according to their corresponding episodic reward \( R_e \), which is defined by the (undiscounted) sum of rewards from the initial state.

In the trajectory replay buffer \( D_{\text{traj}} \), the best 20% trajectories are assumed to be good trajectories while the rest are considered as bad trajectories. States in good trajectories are seen as good states. The bad states are defined analogously. If episodic rewards of all trajectories are the same, the terminal signal will be used to distinguish good and bad trajectories.

During training, a batch size of trajectories is randomly sampled in each APF update, with half good trajectories and half bad trajectories. By counting occurrences of each state \( s \) in good trajectories, \( (N_{\text{good}}[s]) \) and in bad trajectories \( (N_{\text{bad}}[s]) \), the target potential \( P \) of state \( s \) is then calculated as:

\[ P(s) = \frac{N_{\text{good}}[s] - N_{\text{bad}}[s]}{N_{\text{good}}[s] + N_{\text{bad}}[s]} \]  

Note that if a state was not visited, then it will also not be used for updating the APF. Furthermore, the target value \( P \)
is between -1 and 1. Then the loss function for updating the APF neural network $\Phi$ with parameters $\phi$ is as follows:

$$L_\phi = \mathbb{E}[(P(s) - \Phi(s; \phi))^2]$$ (8)

Our proposed APF method is inspired by but very distinct from the count-based exploration method. In the count-based exploration method, states are globally counted, and the counts are directly transformed into exploration bonuses to encourage exploration. In our proposed method, we count the sampled good and bad states in each APF update. The counts are used in constructing shaping potentials and consequently guide the agent to more desirable states. Therefore, the APF method does not encourage exploration but guides the agent to follow the most successful past trajectories.

2) State Representation Methods: In our experiments, computing the target potentials of states is done using lookup tables for tasks with a low-dimensional state space. However, computing these targets in this way will fail in high-dimensional state spaces for two reasons. First, storing and counting a huge amount of states is often infeasible. Second, most states in high-dimensional environments only occur once. Therefore, inspired by [36], we utilized an autoencoder (AE) to compress each state to a low-dimensional code. Details of our experimental AE are illustrated in Section IV-C.

B. Adaptive Potential Function with QMLP

DQN achieved striking successes by estimating Q-functions with convolutional neural networks [1]. Similarly, we use multilayer perceptrons (MLPs) as function approximation methods with the Q-learning algorithm, denoted as QMLP [42]. Specifically, the Q-function is approximated by an MLP with parameters $\theta$. A target network with parameter $\theta^-$ is used by copying from parameters $\theta$ periodically. The loss function for updating QMLP on each step after experience $<s, a, r, s'>$ is shown in Eq. (9):

$$L_{\theta} = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$$ (9)

We propose a new algorithm by combining the APF with QMLP, termed APF-QMLP, for solving tasks with low-dimensional state spaces. In this case, the AE method is not applied, which means states are stored explicitly. Compared to QMLP, APF-QMLP shapes the environmental rewards according to a periodically updated APF. The pseudo-code of APF-QMLP is shown in Algorithm 1. Lines 18-22 update the APF, and in lines 19 and 20 $I(\cdot)$ is the indicator function.

C. Adaptive Potential Function with Dueling DQN

To extend the APF to be used in high-dimensional environments, we propose another new algorithm based on Dueling DQN, refer to as APF-Dueling-DQN. To make the high-dimensional states available for counting, a vanilla AE is applied. Therefore, four networks are constructed in total: the behavior network ($Q$) and the target network ($\hat{Q}$) as in the Dueling DQN algorithm, plus the vanilla AE and the APF network $\Phi$ for implementing the APF. As illustrated in Fig. 2, the APF-Dueling-DQN algorithm updates the target policy with Dueling DQN using rewards shaped by the APF network. Meanwhile, a vanilla AE learns to map each state to a 512-element code. Then the APF network is trained to shape the environmental rewards using the coded states. Note that the behavior network ($Q$) is trained on the stack of the four most recent frames of the game, while the vanilla AE only takes the latest frame as input. The pseudo-code of the APF-Dueling-DQN algorithm is briefly summarized in Algorithm 2.

<table>
<thead>
<tr>
<th>Algorithm 1: APF-QMLP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialize:</strong></td>
</tr>
<tr>
<td>Behavior network $Q$ with weights $\theta$ randomly;</td>
</tr>
<tr>
<td>Target network $\hat{Q}$ with weights $\hat{\theta}$ randomly;</td>
</tr>
<tr>
<td>APF network $\Phi$ with weights $\phi$ randomly;</td>
</tr>
<tr>
<td>Experience replay buffer $D$ with capacity $10^6$;</td>
</tr>
<tr>
<td>Trajectory replay buffer $D_{traj}$ with capacity $10^3$;</td>
</tr>
<tr>
<td>$total_steps &lt; 0$;</td>
</tr>
<tr>
<td><strong>for all</strong> episode do</td>
</tr>
<tr>
<td>Reset $s$; $R_e &lt; 0$; Initialize an empty list $T$;</td>
</tr>
<tr>
<td><strong>for all</strong> step in episode do</td>
</tr>
<tr>
<td>Choose $a$ using $\varepsilon$-greedy derived from $Q$;</td>
</tr>
<tr>
<td>Take action $a$, observe $r$, $s'$, done;</td>
</tr>
<tr>
<td>$P_{s'}$ to $T$; $total_steps += 1$;</td>
</tr>
<tr>
<td>$R_e += r$; $r' = r + \gamma \Phi(s') - \Phi(s)$;</td>
</tr>
<tr>
<td>Store transition $(s, a, r', done, s')$ in $D$;</td>
</tr>
<tr>
<td><strong>if</strong> $</td>
</tr>
<tr>
<td>Sample 512 experiences from $D$;</td>
</tr>
<tr>
<td>$y := r + \gamma \max_{a} \hat{Q}(s', a; \hat{\theta})(1 - done)$;</td>
</tr>
<tr>
<td>$\theta := \arg\min_{\theta} \mathbb{E}[(y - Q(s, a; \theta))^2]$;</td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
<tr>
<td><strong>if</strong> $total_steps$ mod $10^4 = 0$ <strong>then</strong></td>
</tr>
<tr>
<td>$\theta := \hat{\theta}$;</td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
<tr>
<td><strong>if</strong> $total_steps$ mod $10^4 = 0$ and $</td>
</tr>
<tr>
<td>Sample 48 good trajectories $T_{good}$ and 48 bad trajectories $T_{bad}$ from $D_{traj}$;</td>
</tr>
<tr>
<td>$N_{good}[s_i] = \sum_{T_i \in T_{good}} I(s_i)$;</td>
</tr>
<tr>
<td>$N_{bad}[s_i] = \sum_{T_i \in T_{bad}} I(s_i)$;</td>
</tr>
<tr>
<td>$P(s_i) = N_{good}[s_i] - N_{bad}[s_i]; \quad \forall s_i \in {T_{good}, T_{bad}}$;</td>
</tr>
<tr>
<td>$\phi := \arg\min_{\phi} \mathbb{E}[(P(s) - \Phi(s; \phi))^2]$;</td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
<tr>
<td>$s \leftarrow s'$;</td>
</tr>
<tr>
<td><strong>end for</strong></td>
</tr>
<tr>
<td>Store $[T, R_e, done]$ in $D_{traj}$;</td>
</tr>
<tr>
<td><strong>end for</strong></td>
</tr>
</tbody>
</table>

V. EXPERIMENTAL RESULTS

We conducted four sets of experiments to compare the APF-QMLP (and APF-Dueling-DQN) algorithm with the QMLP (and Dueling DQN) algorithm. The only difference is that APF-QMLP (and APF-Dueling-DQN) is an augmented QMLP.
(and Dueling DQN) by shaping environmental rewards based on the APF. Therefore, we will only describe the hyperparameters of the augmented algorithms. The hyperparameters of the standard RL algorithms are consistent with them.

A. Six-Room Maze

We first examined the APF-QMLP algorithm in a 30 × 30 six-room maze, as shown in Fig. 1. The environmental states are represented by the agent’s coordinates, i.e., the state space $S = \{(x, y) | x, y \in [0, 29]\}$. The goal of the agent is to navigate from the upper left corner (the starting state [0, 0]) to the lower right corner (the terminal state [29, 29]). There are four actions: west, north, east, south. The environmental reward is sparse: the agent receives $-1$ at each nonterminal step and gets $0$ once it reaches the goal (a terminal state). The state transition function of the environment is deterministic.

In the experiment, values of both inputs were normalized to $[0, 1]$. The $\epsilon$-greedy exploration strategy was used and $\epsilon$ decays from 1.0 to 0.0 in the first 2000 episodes. Each episode ends if the agent reaches the goal or exceeds the maximal episodic length of 1800 steps. The discount factor was 0.95.

Three MLPs, the behavior network $Q$, the target network $\hat{Q}$, and the APF network $\Phi$, were initialized with the same structure, which used two hidden layers: the first with 256 and the second with 512 hidden units. Each hidden unit was activated via the rectified linear unit function [43]. Adam [44] with the standard values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ was used as the optimizer. The learning rate for training each MLP was $10^{-4}$, with $x$ linearly decaying from $-3$ to $-6$ during training. Other hyperparameters are the same as in Algorithm 1.

We additionally implemented two more agents for the comparison by using the count-based exploration strategy instead of $\epsilon$-greedy. One is QMLP with the count-based exploration strategy (state-count). The other is APF-QMLP with the count-based exploration strategy (APF+state-count).

Fig. 3 showcases the results of cumulative environmental rewards (i.e., without shaping) over episodes for the four agents. Each curve was smoothed by a Savgol filter for the interpretability of the plot. The shaded regions show the standard deviation. Specifically, the last-episode cumulative reward ($R$) and the average cumulative reward of all episodes ($\bar{R}$) for all agents in the maze environment are shown in Table I. The standard deviation is represented as $\sigma$.

As we can see, the APF-QMLP agent outperformed the QMLP agent in learning speed, although the final results for all algorithms are similar. The results also show that the APF can be combined with exploration strategies other than $\epsilon$-greedy and this boosts the performance of the selected RL algorithm.

<table>
<thead>
<tr>
<th>Agent</th>
<th>$\bar{R} \pm \sigma$</th>
<th>$R \pm \sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMLP</td>
<td>$-73.2 \pm 0.6$</td>
<td>$-395 \pm 147$</td>
</tr>
<tr>
<td>APF-QMLP</td>
<td>$-73.0 \pm 0.0$</td>
<td>$-309 \pm 73$</td>
</tr>
<tr>
<td>state-count</td>
<td>$-75.6 \pm 0.9$</td>
<td>$-401 \pm 105$</td>
</tr>
<tr>
<td>APF+state-count</td>
<td>$-74.2 \pm 1.0$</td>
<td>$-272 \pm 117$</td>
</tr>
</tbody>
</table>

B. CartPole and Acrobat

The effect of the APF-QMLP algorithm was further examined on two classic Open-AI gym control tasks: CartPole

![Algorithm 2: APF-Dueling-DQN](image)

**Algorithm 2: APF-Dueling-DQN**

*Initialize:*

- APF network $\Phi$ randomly;
- Vanilla AE $\Psi$ randomly;

1. **for all episode**
2. Take action $a$ at state $s$ using $\epsilon$-greedy, and observe $r, s', done$ on each step;
3. Collect $\{s'\}$ for learning $\Psi$;
4. Store the trajectory of $\{\Psi(s_n)\}_{n=1}^{\text{episode}}$ in $T$;
5. Store $\{T, R_e, done\}$ for updating $\Phi$;
6. **if** $\Phi$ has learned for 10 episodes **then**
7. $r = r + \gamma \Phi(s') - \Phi(s)$;
8. **end if**
9. Collect transition $(s, a, r, done, s')$ for updating the target policy with Dueling DQN;
10. **end for**

![Six-Room Maze](image)
Fig. 4. The learning curves of the QMLP and APF-QMLP algorithms over 10 runs in CartPole-v0 and Acrobot-v1.

TABLE II
RESULTS IN CARTPOLE AND ACROBOT.

<table>
<thead>
<tr>
<th></th>
<th>CartPole</th>
<th>Acrobot</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMLP</td>
<td>$R \pm \sigma$</td>
<td>$R \pm \sigma$</td>
</tr>
<tr>
<td>APF-QMLP</td>
<td>175 ± 26</td>
<td>97 ± 31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$R \pm \sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMLP</td>
<td>191 ± 18</td>
</tr>
<tr>
<td>APF-QMLP</td>
<td>107 ± 31</td>
</tr>
</tbody>
</table>

and Acrobot. The state-space of both environments is low-dimensional, where the state of CartPole is a four-element array and of Acrobot is a six-element array.

Hyper-parameters were slightly changed to adapt to the gym environments. The target network $\hat{Q}$ was updated every 64 steps. The update frequency of the APF network $\Phi$ was lowered to 100 steps. Each agent was trained for 500 episodes, with $\epsilon$ decaying from 1.0 to 0.0 in the first 400 episodes.

The results are shown in Fig. 4. Each curve was averaged over 10 runs. We can see that the APF-QMLP agent outperformed the QMLP agent in the CartPole environment, and both agents performed well in the Acrobot environment. Detailed results are shown in Table II. The results exhibit that the APF improves the performance of the RL agent.

C. MsPacman

The APF-Dueling-DQN algorithm was evaluated on an Atari game using both deterministic and stochastic versions: MsPacmanDeterministic-v4 and MsPacman-v0. The maximum length of each episode was 18000 frames. The discount factor was 0.99. The capacity of the experience replay buffer is 1 million. Maximal 1000 trajectories were stored for updating the APF network. Details of implemented networks in APF-Dueling-DQN are shown in Table III.

As shown in Fig. 5, by mapping states from Atari games into hash codes via an AE, the performances of an RL agent in a high-dimensional state space can be improved by learning an APF using the codes. The detailed experimental results are listed in Table IV. These results demonstrate the effectiveness of the APF method to also improve RL in a complex environment.

TABLE III
DESCRIPTIONS OF NETWORKS IN APF-DUELING-DQN

<table>
<thead>
<tr>
<th>Networks</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavior Network (Q)</td>
<td>Updated every 4 frames. The batch size is 32. The dueling architecture is shown in Fig. 2.</td>
</tr>
<tr>
<td>Target Network (Q)</td>
<td>Copy from Q every 10000 steps.</td>
</tr>
<tr>
<td>APF Network $\Phi$</td>
<td>Updated every 1000 frames. The batch size is 64. A 3-layer MLP with 1024 hidden units.</td>
</tr>
<tr>
<td>Vanilla AE</td>
<td>Updated every 1000 frames. The batch size is 256. A 3-layer MLP with 512 hidden units (i.e. code_size=512).</td>
</tr>
</tbody>
</table>

TABLE IV
RESULTS IN THE DETERMINISTIC AND THE STOCHASTIC MS-PACMAN ENVIRONMENTS.

<table>
<thead>
<tr>
<th></th>
<th>Deterministic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dueling DQN</td>
<td>$R \pm \sigma$</td>
<td>$R \pm \sigma$</td>
</tr>
<tr>
<td>APF-Dueling-DQN</td>
<td>1695 ± 158</td>
<td>1225 ± 96</td>
</tr>
<tr>
<td>APF-Dueling-DQN</td>
<td>2083 ± 201</td>
<td>1373 ± 171</td>
</tr>
<tr>
<td>Dueling DQN</td>
<td>1465 ± 73</td>
<td>1139 ± 78</td>
</tr>
<tr>
<td>APF-Dueling-DQN</td>
<td>1609 ± 125</td>
<td>1155 ± 85</td>
</tr>
</tbody>
</table>

VI. CONCLUSION

In this paper, our main contributions are three-fold. First, we propose a novel adaptive potential function within the potential-based reward shaping framework. The APF is updated by counting state occurrences in good/bad episodes. Second, we applied the APF in Q-learning algorithms and formed two new RL algorithms: APF-QMLP and APF-Dueling-DQN. Particularly, an AE was utilized to enable the APF in the high-dimensional domain. Finally, we evaluated the two algorithms in various environments, including both low-dimensional and high-dimensional state spaces. Results demonstrated the power of the proposed APF to improve the performances of the selected RL algorithms.

In continuation of this work, we want to investigate improving the proposed algorithm by directly training the APF network without need for discrete states. Other future work includes researching the APF method in environments with a high-dimensional state space. Moreover, we want to study other APF updating rules that are more precisely connected with the exact cumulative rewards for different episodes.
ACKNOWLEDGMENT
We would like to thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Peregrine high performance computing cluster. Yifei Chen acknowledges the China Scholarship Council (Grant number: 201806320353) for financial support.

REFERENCES