DVRSF: A Deep Reinforcement Learning Method Based on Dynamic Velocity Reward & Stagnation Fine

Qiyang Zhang
Army Engineering University of PLA

Xiliang Chen (✉️ 383618393@qq.com)
Army Engineering University of PLA

Qingwei Li
The 28th Research Institute of China Electronics Technology Group

Article

Keywords: Deep Reinforcement Learning, Knowledge Transfer, Curiosity, Reward Shaping, General RL

Posted Date: October 26th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2184666/v1

License: ☺️/licenses This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License
DVRSF: A Deep Reinforcement Learning Method Based on Dynamic Velocity Reward & Stagnation Fine

Qiyang Zhang¹, Xiliang Chen¹*, Qingwei Li²
1. Army Engineering University of PLA, China, Nanjing, 210007
2. The 28th Research Institute of China Electronics Technology Group, China, Nanjing, 210000

Correspondence should be addressed to Xiliang Chen: 383618393@qq.com

Abstract: When dealing with the difficult exploration problems caused by sparse rewards, pre-designed environment reward leads to poor performance of most RL algorithms. The existing algorithms fail to resolve the contradiction between the difficulty exploration and exploration efficiency. In this situation, it is necessary to design rewards more carefully, make more accurate judgments and feedback on the exploration status of agents. Therefore, we designed a RL exploration method based on trajectory entropy, and extracted two features during the training process to participate in the construction of internal rewards. On this basis, we add a monitoring method based on exploration speed to improve the over divergence problem in the agent exploration process. This method constructs a potential feature based on exploration speed according to the information sampling obtained in real time during training, gives real-time feedback to the agent, and dynamically adds it to the reward of each MDP. These two methods are called stagnation fine and velocity reward. Stagnation fine is intended to encourage the agent to get out of the dilemma as soon as possible when the agent is falling in. Velocity reward is used to improve the agent's exploration speed in the right direction and improve the over divergence of exploration. The algorithm has good generality, can avoid the influence of network structure in different algorithms, and can be easily added to existing RL algorithms, such as A3C and PPO. We evaluated our approach in different types of environments in Super Mario Bros. The experimental results show that the learning speed and accuracy of agents using DVRSF are significantly better than those of baseline such as PPO and A3C in most test tasks.

Key Words: Deep Reinforcement Learning, Knowledge Transfer, Curiosity, Reward Shaping, General RL

1 Introduction

Deep Reinforcement Learning (DRL) is one of the most important branches in the field of machine learning. It has achieved good results in many cutting-edge scientific and technological fields, including computer vision, turn-based game, robot control, NLP and clinical care[1-6], is a hot research direction at present. In related research, hard exploration is the biggest constraint to the
effect of RL algorithm. The main reasons are the lack of rewards in the environment and the imperfect design of agents. In order to solve this problem, researchers have proposed a variety of extended exploration methods, but in the process of expanding the exploration boundary, there is often a lack of supervision of agents, resulting in the lack of guidance in the exploration of agents.

A recent study believes that reward design is the decisive factor to determine the intelligence of RL agents \[7\]. Inspired by this, we designed a dynamic reward shaping method based on potential function to alleviate the problem of over divergence in extended exploration. We have designed two features, one of which is used to represent the agent's exploration situation. By calculating the agent's trajectory entropy during training, we transfer the exploration situation during training to the agent for subsequent training. Another feature is used to constrain the exploration direction. It calculates the exploration speed of the agent, constructs a potential energy system based on the speed, and dynamically shapes the overall reward in real time at each time step.

The main contributions of this paper are as follows:

1. We design a RL exploration method based on trajectory entropy for the problem of hard exploration in the sparse reward environment. This is a novel algorithm, which makes a more accurate judgment on the exploration situation by constructing the sampling area.
2. We propose a dynamic reward shaping method based on velocity potential function, which has a good effect on improving the over divergence problem and can effectively guide the exploration direction of agents.
3. By combining the above (1) and (2), we have obtained an effective algorithm for learning control tasks with hard exploration. The research results expand the application of DRL in solving hard exploration.

2 Background

Our method is based on RL and improved in the Markov decision process, aiming to solve the sparse reward problem in DRL. This method can be combined with the policy gradient algorithm and has a wide range of applications. This section mainly introduces the basic theoretical knowledge and related research background.

2.1 Markov decision process

Markov Decision Process (MDP) is the mathematical basis of Reinforcement Learning (RL) and the theoretical framework of RL. The complete MDP is a five tuple, which contains finite state set \( S \), action set \( A \), state transition probability \( p \), reward function \( R \) and discount factor. \( S, A \) and \( R \) are the basic elements of RL. When these elements are finite, they are called finite MDP. The formal sequence is shown as follows \((s_0, a_0, r_0, ..., s_t, a_t, r_t, ...)\), the state transition probability is expressed as
\[ p(s', r \mid s, a) = P(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a) \] (2-1)

In MDP, the value of \( s, r \) only depends on the previous state \( s_{t-1} \) and action \( a_{t-1} \), and has nothing to do with earlier states and actions, this property is called Markov property.\[8\] Return is the income obtained after discounting the reward. Due to the uncertainty of the model, we need to use the discount factor \( \gamma \) to discount the further reward for continuous tasks. For each continuous task track, we can calculate its total return by reward:

\[ G(t) = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^k R_{t+k} = \sum_{k=0}^{\infty} \gamma^k R_{t+k} \] (2-2)

where the value of discount factor \( \gamma \) is between 0 and 1. The closer to the current state, the more important is the reward to calculate the return. When \( \gamma = 0 \), the return calculation only considers the current reward. When the future reward can be obtained in advance, the discount calculation is not required, when \( \gamma = 1 \). The state value function \( V(S) \) represents the long-term value of state \( s \) and is defined as the expected value of return \( V(S) = E[G_t \mid S_t = S] \). The above calculation process is called Markov reward process. Since no action is introduced, it does not involve decision-making steps. In the complete MDP, the reward function is no longer only related to state, but also related to action. The reward function is the immediate reward for taking a specific action in a certain state. What action to take in a certain state is called a policy, which is defined as

\[ \pi(a \mid s) = P(A_t = a \mid S_t = s) \] (2-3)

The input of value function is divided into state \( s \) and state-action pairs. When the input is state, it is called state value function

\[ V^\pi(S) = E_q[G_t \mid S_t = S] \] (2-4)

When the input is a state-action pair \((s, a)\), it is called an action value function

\[ q^\pi(s, a) = E_q[G_t \mid S_t = s, A_t = a] \] (2-5)

In the above formula

\[ q^\pi(s, a) = E_q[G_t \mid S_t = s, A_t = a] \] (2-6)

MDP is a framework for describing sequential decision-making problems under uncertainty. The proposal of this framework provides basic assumptions for the proposal of Q-learning \[9\], SARSA \[10\] and other basic RL algorithms.

### 2.2 Policy gradient algorithm

DRL algorithms can be roughly divided into two categories: value based and policy based. Value based algorithms calculate the value of each state action, and then select the action with the greatest value to execute. This process is realized by using the state action value function. Input the state and action input, output the Q value of each action, and select the largest action to execute. In the policy-based method, a similar approach is adopted to approximate the policy. At this time, the policy \( \pi \) can be described as a function containing parameters \( \theta \), input state \( s \) can obtain the
output action $a = \pi(s, \theta)$, and the output action probability is $P(a | s, \theta)$. After the policy is expressed as a continuous function, the policy gradient method is used to find the optimal policy.

In the above method, the network parameters can only be updated at the end of each episode of update, which is inefficient. If the distribution of sampled actions is extreme, many states will not be explored and fall into local optimum. We can introduce the value-based idea to solve this problem, Actor-Critic (AC) algorithm. The AC algorithm is divided into two parts, actor and critic. Actor is responsible for using the action selection function to generate actions and interact with the environment, Critic uses the method of value function to evaluate the performance of actor and guide the next actions of actor. The proposed AC algorithm solves the problem of continuous reading action space in RL.

In DRL, the expected rewards we get can be described by the actor, environment and reward function. (Fig.1) The actor is responsible for controlling the agent, and the environment is responsible for controlling other elements in the scene. For example, after the agent acts, other components in the environment react accordingly, and the reward function gives score feedback after the agent acts, it is a reward obtained by taking a certain action in a certain state. What RL needs to do is to adjust the internal parameters of the actor so that the $R(\tau)$ larger the better. In fact, $R(\tau)$ is a random variable. What we can calculate is it’s expectation

$$\bar{R}_\theta = \sum_{\tau} R(\tau) p_\theta(\tau) = E_{\tau: \ p_\theta(\tau)}[R(\tau)]$$  \hspace{1cm} (2-7)

the gradient calculation formula used to update the policy is

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{\tau} \sum_{n=0}^{T} R(\tau^n) \nabla \log p_\theta(a^n | s^n)$$  \hspace{1cm} (2-8)

Fig.1 Model representation of RL

Advantage Actor-Critic (A2C) optimizes the AC algorithm by introducing a baseline function $B(s)$ into the AC algorithm to construct an advantage function $A(s, a) = Q(s, a) - B(s)$, so as to adjust the parameters of the actor through the positive or negative feedback received by $A(s, a)$. 
This baseline is usually represented by the state value function, $B(s) = V(s)$, and the gradient calculation formula is

$$\nabla \tilde{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n} (Q(s_t, a_t) - V(s_t)) \nabla \log p(a_t | s_t^{n_t})$$

(2-9)

The rich representation of neural network makes RL more effective. When neural network is used to approximate action value function, the strong correlation of data will make the training of neural network very unstable. Therefore, many studies use various methods to break the correlation between data. For example, DQN uses the method of experience playback to break the association between data. It stores sampled data, and then reduces the association between data through small batch random extraction to improve the stability of the training process. In addition, it also uses asynchronous methods to solve this problem, such as n-step Q-learning. In order to solve the problem that AC is difficult to converge, the Asynchronous Advantage Actor-Critic (A3C) is purposed by Google Deepmind, which adopts the asynchronous and concurrent learning model. The A3C algorithm creates multiple parallel environments, and puts multiple local models into multiple threads for synchronous training to make them interact with the local models respectively. After the local models complete the calculation of their respective update amount, the global models update their parameters. (Fig.2) At the same time, they will also get updated information from the global models to guide their interaction with the environment.

![Fig.2 A3C algorithm training framework](image)

PPO algorithm is another RL method based on AC architecture. PPO also has two networks, actor and critic, of which the actor network is divided into old Actor and new Actor. The problem
with the original AC algorithm is that the policy optimization cannot be guaranteed. If the advantage of a certain state is negative, we can only know that the possibility of taking action in this state should be reduced, but we do not know how much. If the gradient descent is performed in multiple stages, the gradient update is easy to become too large, resulting in the policy being moved to other unknown areas in the action space. Schulman et al. Proposed the concept of Trust Region Policy Optimization (TRPO), which solves this problem through the difference between the old policy and the new policy. TRPO limits the scale of policy update based on the difference between the old policy and the current policy, so as to ensure the smooth improvement of the policy. However, TRPO is not easy to implement and has poor compatibility with the network structure. PPO algorithm is improved by updating policy parameters. Its purpose is to combine the simplicity of common policy gradient with the stability and efficiency of TRPO.

2.3 Sparse reward problem

In the process of using RL to solve problems, the rewards of many environments are very sparse. Most of the time, agents are exploring in the distant future and cannot get rewards. It is useless for an agent to learn without a reward for a long time, and it is difficult to achieve the desired effect in the end. Therefore, in addition to the rewards that an agent can directly obtain from the environment, we need to construct some additional reward functions to make the rewards denser and play a positive role in guiding the agent's exploration process. How to solve the reward sparsity problem is one of the hottest research hotspots in RL. Researchers have also given many different solutions, including curiosity mechanism \([16-18]\), reward shaping \([19,20]\), state initialization \([21-23]\), related work will be introduced in Section 3.

3 Related work

Yuri Burda et al. proposed the Random Network Distillation (RND) \(^{18}\) method to design the internal reward and predict the output of the randomly initialized neural network in the current state, which uses the prediction error in the feature space as the measure of state importance. Because it is a high-dimensional continuous space, this count can be regarded more as a density estimation. If the number of visits to a similar state is less, it indicates that the state is novel, and a higher internal reward will be given. This method requires the construction of two neural networks, in which the target network \(f(x)\) is a deterministic and randomly initialized network, through which the prediction problem is set to observe the inserted \(x\) value, and the prediction network \(\hat{f}(x;\theta)\) is trained through gradient descent according to the data collected by the agent. The goal is to obtain the parameters that minimize the expected mean square error

\[
\left\| \hat{f}(x;\theta) - f(x) \right\|^2
\]

(3-1)

Reward shaping guides agent training by artificially designing additional rewards, but in some
problems, artificially designed reward functions often lead to agents' opportunism and unable to learn the optimal policy. To solve this problem, Andrew Ng et al.\cite{19} Proposed potential based reward shaping (PBRS). PBRS takes the difference between the two potential functions as the reward shaping function

\[ F(s,a,s') = \gamma \Phi(s') - \Phi(s) \]  \hspace{1cm} (3-2)

which is equivalent to defining a potential energy for each state, giving a positive reward from the state with low potential energy to the state with high potential energy, and a negative reward on the contrary. This solves the problem that the agent circulates and accumulates rewards in situ in the original reward shaping method. PBRS significantly improves the time required to learn the optimal policy. This method does not change the optimal policy of a single agent, and can ensure the consistency of the policy.

Bellemare et al. purposed the method of Intrinsic Motivation (IM)\cite{16}, which constructs internal rewards based on the idea of counting, that is, in addition to trying to obtain larger external rewards, agents also use internal rewards to motivate agents to explore the state space they have not encountered. However, there are also some problems with this kind of method\cite{24}. Although the internal reward can encourage the agent to explore a larger unknown state space, there are many states between the current state and the explored state space boundary. IM method cannot encourage the agent to go beyond these states with very small internal reward to explore. For continuous action space, some researches use density instead of counting\cite{25}, and map states to hash codes\cite{26} to improve it. Another problem is that the noise or artificial random disturbance also limits the intelligent physical ability to perfectly reproduce the previous trajectory and directly go to the exploration boundary, and the path usually shifts. It seems to be full of great difficulties for the agent to reach the boundary state of exploration. Ecoffet A and others consider directly initializing the state,\cite{23} archiving the state when the agent explores a long distance, and then continuing to learn with the initial state of the archive initialization environment, so as to explore a larger state space. This method is expensive to implement because it needs to save every detail of the environment state in each episode, and it is not suitable for environments that cannot be loaded from the interactive process.

4 Dynamic Velocity Reward & Stagnation Fine

The goal of DVRSF is to solve the problem of the difficulty of agent exploration in DRL. The difficulty of exploration is mainly divided into two situations: one is that the agent is trapped in a sparse reward environment, and the agent has no clear exploration direction, and loses the motivation to explore; the other is that the agent is trapped in a local reward trap, and continues to obtain environmental rewards at a certain location, and gives up on further exploration when it is trapped in a local optimum. In this section, according to these two difficult exploration problems,
the reward function of the agent is redesigned for more advanced exploration, and a method of
dynamic shaping the reward based on the potential function of speed is designed to solve the
problem of possible exploration divergence during the exploration process. (Fig.3) The form of the
reward function we construct is as follows:

\[ R = r_E(s, a) + \phi_R(s, a) f(r_{SF}) \]  

(3-3)

where \( r_E \) is the original reward in the environment, \( f(r_{SF}) \) is stagnation fine, \( \phi_R(s, a) \) is the
velocity potential energy function dynamically calculated based on the velocity potential function.

This internal reward construction method can improve the agent learning efficiency and the model
stability of the learning model compared with the baseline without changing the optimal policy of
the original problem.\[19\]

![Fig.3 DVRSF reward composition diagram](image)

4.1 Extracting knowledge from state transitions

Our method originates from the idea of state knowledge transfer, which is a process of
transforming the state transfer in RL into some form of knowledge to participate in the follow-up
training. State is one of the most important data in RL model. Each exploration process starts from
one state to another. State knowledge transfer transfers state as knowledge to a new learning task or
learning process to accelerate the efficiency of agent exploration. The state knowledge transfer is
suitable for solving the problem of expanding the exploration boundary, and solving the problem of
difficult exploration in the sparse reward environment. The method of transferring state knowledge
is effective.

State information is one of the most important components in MDP. In the interaction process
of most RL models, the state information of the agent can be obtained at each time step. By building
a global recorder, the parameters of the agent state in each training iteration can be recorded, such
as the location of the agent in the environment, the time it takes to reach the farthest location, the
maximum score and other important state information. After obtaining the state information, we
should process the state information and make use of it. Generally, we preprocess the data through
linear weighted processing, and then dynamically shape the reward function to generate a new
reward function. After using the newly constructed reward function, the hyperparameters are
adjusted through the specific performance in the experiment until better results are achieved. There
are many ways to construct intrinsic rewards. The common construction modes are as follows:
\[ r_i = f(N) \ast \mu \] (4-1)
where \( N \) is the state information obtained after preprocessing, \( f(N) \) is the linear function of \( N \),
and \( \mu \) is the hyperparameter that controls its influence factor.

4.2 Exploration depth measurement method

In the RL scenario, exploration is the process of interacting with the environment from the
starting point to finally achieve the goal. In each training, it will go through various paths to different
positions. We define two indicators of exploration depth to evaluate the effectiveness of agent
exploration.

4.2.1 Distance exploration depth

We define the Distance Explore Depth (DED) as the number of state nodes that the agent's
MDP path passes through during the exploration process. Due to the complexity and randomness of
the RL environment, the distance explore depth in most environments can be expressed by the
projection of the agent's mobile path in the effective direction, which represents the correct direction
of agent exploration and the direction of learning the optimal policy. DED can accurately reflect the
scope and boundary of agent exploration, and further reflect the situation encountered during agent
exploration. At the end of each episode, we record the DED and store it in the cache. During training,
it is sampled and added to the corresponding structure of the algorithm after calculation, Fig 4 shows
the recording and sampling process of DED during training.

![Fig.4 The recording and sampling process of DED](image)

\[ r_i = f(N) \ast \mu \] (4-1)
4.2.2 Timestep Exploration Depth

The number of time steps that the agent interacts with the environment is also one of the important indicators to measure the exploration status. In some specific complex scenes, because there are many MDP loops, the DED indicator alone cannot accurately represent the exploration of the agent. In the exploration trajectory of an agent, the time step can represent the exploration degree of the agent to a large extent. We define the time step used for each episode to the agent as the Timestep Exploration Depth (TED). Due to the difficult exploration of agents in complex environments, TED is usually used in combination with DED, such as calculating the exploration speed and constructing the potential function, which can more accurately describe the exploration status of agents, and then use it to calculate the internal reward and construct the reward function after further processing.

The storage mode of TED is similar to that of DED, which records the time steps used in each episode. The read and write operations of the buffer are only carried out at the end of each game. The time complexity is $O(n)$ and the time cost is small, which will hardly affect the running speed of the basic algorithm. When sampling, sampling multiple groups of game data can avoid the error caused by the randomness of the environment and improve the stability of the model.

4.3 Dynamic weighted reward shaping

We use the two kinds of exploration depth DED and TED introduced in Section 4.2 to construct the dynamic weighted shaping reward. In the process of reward shaping, we construct two kinds of internal rewards, namely, velocity reward and stagnation fine, to improve the problems of slow exploration and difficult exploration of agents.

We conduct reward shaping based on the extracted two features. The stagnation fine is a global reward based on the results of recent rounds of training. The training purpose has not changed, and its difference is reflected in different training episodes. The velocity reward is the reward calculated in real time during each MDP conversion in the training according to the speed potential function defined by us. It is divided into positive and negative to indicate whether the agent is in the right exploration direction. When it is in the right direction, it will generate a positive reward, otherwise it will generate a negative reward, which is the same as the concept of potential energy in physics. The addition of velocity reward will introduce a reward for real-time control agent for training, but it will not change the consistency between our overall training goal and the optimal policy. In the research of Andrew Ng et al., the sufficiency and necessity of reward shaping based on potential energy to ensure the consistency of the optimal policy are proved, Algorithm 1 provides an outline of the basic training loop.
Algorithm 1: Dynamic Velocity Reward & Stagnation Fine (DVRSF)

1: for $i = 1, \ldots, N$ do
2:   if $i > N_{SF}$ do
3:     Trajectory entropy $r'_{SF} = D(DED) + D(TED) \cdot \mu$ with the latest $N_{SF}$ data
4:   end if
5:   Calculate Stagnation Fine $f(r'_{SF}) = f(s, a)$
6: if $i > N_{VR}$ do
7:   Calculate Base Velocity $V^i_B = \sum_{t=1}^{N} (DED[i] - TED[i]) / N$ with the latest $N_{VR}$ data
8: end if
9: Generating velocity based potential functions $\phi_{VR}$
10: for timestep $= 1, \ldots, T$ do
11:   use policy $\pi_\theta$ to interact with the environment and get $\{s_t, a_t, r^t\}$
12:   Calculate Current Velocity $v^t = p^t - p^0 / t$
13:   Calculate Velocity Reward $z(\phi_{VR}) = z_q(s, a) = v^t / V^i_B$
14: end for
15: $DED[i] = x \_ pos$
16: $TED[i] = end \_ timestep$
17: Advantage estimation $A^{GAE}(t, l) = \sum_{l=0}^{\infty} (\gamma^l)^l \delta^{l+1} = \sum_{l=0}^{\infty} (\gamma^l)^l \big( R_l + \gamma V(s_{l+1}) - V(s_l) \big)$
18: update policy $\pi_{\theta \_ old} \leftarrow \pi_{\theta \_ new}$
19: for $j = 1, \ldots, M$ do
20:   $r_j(\theta) = \frac{\text{prob}_\text{new}(a_t | s_t)}{\text{prob}_\text{old}(a_t | s_t)}$
21:   $r^{clip}_j(\theta) = \text{clip}(r_j(\theta), 1 - \varepsilon, 1 + \varepsilon)$
22:   $L^{\text{CLIP}}(\theta) = \hat{E}_t \left[ \min \left\{ r_j(\theta) \hat{A}_t, r^{clip}_j(\theta) \hat{A}_t \right\} \right]$
23: update $\theta$ with $L^{\text{CLIP}}(\theta)$
24: end for
25: end for

4.3.1 Stagnation Fine

In some specific scenarios, the intelligence will fall into local optimum, continue to get rewards at a fixed position, or stop exploring in order to avoid the next risk. This opportunistic approach limits the scope of exploration of the intelligence. We design a penalty mechanism to guide the agent to avoid falling into the local reward trap. We use the environment information, DED and TED to construct the reward function. When the trajectory stagnates due to the local optimum, we add the negative reward. For the judgment of local optimum, we consider sampling the DED and TED
recorded at the end of the recent training to calculate the entropy of the exploration trajectory. When the exploration trajectory tends to be constant, the trajectory entropy is small, while when the agent exploration is in a more active state, the trajectory entropy is large. We use the following formula to construct the trajectory entropy

\[ r_{SF} = D(DED) \mu_{DED} + D(TED) \mu_{TED} \]  

(4-2)

where \( D(DED) \) represents the variance of the sampled DED set. Further, \( E_{SF} \) can be simplified to

\[ E_{SF} = D(DED) + D(TED) \mu \]  

(4-3)

by adjusting external hyperparameters. The range and stability of \( E_{SF} \) values in different environments are usually somewhat different, so it is necessary to further process the \( E_{SF} \) values according to the specific experimental environment, such as normalization and clipping operations, to keep the values within a reasonable range. When the DED and TED of the sampling area tend to be constant and the agent does not converge to the target, that is, the \( r_{SF} \) is small, we think that the agent is in the exploration dilemma. At this time, we need to add the stagnation fine \( f(r_{SF}) \), in order to make the agent get out of the difficulty as soon as possible and divergent exploration. Corresponding to the punitive stagnation fine, when the agent's exploration state is more active, we add appropriate rewards to the agent to encourage the agent's exploration.

4.3.2 Velocity Reward

In the process of stagnation fine participating in training, it can help the agent to expand the exploration boundary as much as possible. However, when the agent focuses on expanding exploration, it may also have some side effects, that is, the excessive divergence of exploration may lead to the slowdown of effective exploration. The so-called effective exploration speed is the agent's exploration speed in the direction of the optimal policy. Maintaining an appropriate effective exploration speed is very necessary to improve the convergence speed of RL. Fig 5 (a) shows the MDP state transition diagram without velocity reward, and Fig 5 (b) adds velocity reward, in which the green nodes (from light to dark) are the optimal path explored by the agent. After adding velocity reward, due to the supervision of speed potential function, the agent reduces the invalid exploration time and is more directional. The green arrow in Fig 5 (b) is the direction of velocity potential.
In order to solve the above problems, we abstractly apply the definition of velocity in Physics: \( v = \frac{s}{t} \) to the process of reinforcement learning, and propose a calculation method of velocity potential energy. First of all, we sample the DED and TED data generated by the recent exploration. DED represents the exploration state distance at the end of the exploration, and TED represents the timestep at the end of the exploration. Based on this, we calculate the benchmark exploration speed \( \frac{DED}{TED} \), and then calculate the average value of \( \frac{DED}{TED} \) in the sampling area to

\[
\overline{v} = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{DED}{TED} \right)
\]

(4-4)

as the prediction speed compared with the real-time exploration speed in this episode. The real-time exploration speed is calculated in real time at each time step, and its value is equal to the ratio of the state distance from the starting point of the agent to the time steps

\[
v_c^i = \frac{p^i - p^0}{t}
\]

(4-5)

Then it is compared with the predicted speed \( \overline{v} \) to calculate the speed potential energy \( \phi_R = \frac{v}{\overline{v}} \). In practice, we found that an inspirit constant \( C_{\text{inspirit}} \) slightly greater than 1 was added to promote the agent to improve the exploration speed. This constant can be regarded as the base point of potential energy, and the final potential energy of speed became \( \phi_R = \phi_R^i - C_{\text{inspirit}} \). We combined the meaning of potential energy of speed, multiplied it outside the overall reward, and as an independent potential energy parameter, we can better express the impact of exploration speed. The reward value in each MDP is dynamically shaped into

\[
r_e^i = r_e^i - z_e(\phi_R) f(r_{SF})
\]

according to the velocity potential energy \( \phi_R^i \), where \( z_e(\phi_R) \) is the potential function calculated according to the current velocity potential energy \( \phi_R^i \) of the agent. Since there will be large fluctuations in the real-time exploration speed at the beginning of agent exploration, in order to obtain better stability, it is usually necessary to further clip the \( r_e^i \).

5 Experiment

We used A3C and PPO algorithms as the baseline, and carried out comparative experimental
analysis with our improved algorithm in Super Mario Bros series games. The experimental environment is shown in the Fig 6, which has 32 game environments of 4 different styles. The learning speed of our method is faster than baseline in more than half of the environments. The combination of A3C and PPO algorithm has achieved significant improvement, which proves that the algorithm has good robustness.

5.1 Ablation study

We performed ablation experiments in a test environment in Super Mario Bros to study the impact of different stagnation fine and velocity reward decomposition that make up the dynamic weighted reward and the influence of exploration depth sampling area size on learning effect. Tab.1 shows the results of ablation experiments conducted in two different environments, the red font is the best effect in each column, almost all of them are created by our algorithm. We truncated the data in three time steps. The table shows the success rate of the agent reaching the end point when reaching the corresponding time step. The number in brackets uses the variance of the exploration distance to represent the stability. The data is scaled in equal proportion to ensure the consistency of the data. The comparative experiment is mainly carried out on the basis of A3C algorithm. In addition, we also carry out a simple test on PPO algorithm to verify the effectiveness and universality of the algorithm.

<table>
<thead>
<tr>
<th>Method/Map</th>
<th>W1S1</th>
<th>W1S2</th>
<th>W1S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVRSF</td>
<td>0.0(±0.17)</td>
<td>0.47(±0.66)</td>
<td>0.97(±0.06)</td>
</tr>
<tr>
<td>SF(SC10)</td>
<td>0.0(±0.13)</td>
<td>0.41(±0.57)</td>
<td>0.97(±0.10)</td>
</tr>
<tr>
<td>SF(SC100)</td>
<td>0.0(±0.15)</td>
<td>0.68(±0.28)</td>
<td>0.97(±0.10)</td>
</tr>
<tr>
<td>DVR(SC10)</td>
<td>0.0(±0.15)</td>
<td>0.32(±0.71)</td>
<td>0.95(±0.16)</td>
</tr>
<tr>
<td>DVR(SC100)</td>
<td>0.0(±0.22)</td>
<td>0.53(±0.50)</td>
<td>0.85(±0.35)</td>
</tr>
<tr>
<td>A3C</td>
<td>0.0(±0.21)</td>
<td>0.41(±0.45)</td>
<td>0.93(±0.09)</td>
</tr>
<tr>
<td>DVRSF(PPO)</td>
<td>0.06(±0.51)</td>
<td>0.62(±0.45)</td>
<td>0.87(±0.24)</td>
</tr>
<tr>
<td>PPO</td>
<td>0.01(±0.29)</td>
<td>0.23(±0.54)</td>
<td>0.39(±0.73)</td>
</tr>
</tbody>
</table>
5.1.1 Dynamic weighted reward ablation

As described in Section 4.3, we divide the intrinsic reward into two modules: stagnation fine and velocity reward for dynamic modeling. In order to determine the impact of each reward on the training effect, we only use velocity reward, stagnation fine and two intrinsic rewards to train different types of maps in Super Mario Bros. It can be seen in the table that the algorithm using only stagnation fine has a relatively fast convergence speed, The algorithm using only dynamic velocity reward has no obvious improvement effect, which is in line with our expectations. The original intention of designing dynamic velocity reward is to supplement the problem of exploration dispersion caused by stagnation fine method. When the addition degree of stagnation fine has a side effect on the agent, dynamic velocity reward can effectively make the agent out of the state of random collision or stagnation, the experimental demonstration on this point will be described in section 5.5.

5.1.2 Exploration depth sampling ablation

In the process of constructing two different rewards, we need to sample the data, so we have a reasonable guess: changing the size of the sampling interval may lead to different experimental results. For this, we designed two different sampling area sizes to compare in the ablation experiment. The comparison results show that, on the whole, the effect of larger sampling area is better and the experimental results are stable, but it is not inevitable. We analyze that the optimal size of sampling area depends on the specific characteristics of the environment, and there is no universal optimal value, we leave it to future work to explore this direction further.

5.2 Advantages of dynamic reward shaping

DVRSF carries out reward shaping dynamically. Before that, we also studied that for different VR, SF feature values stipulate some fixed reward values to participate in training. We call this method VRSF. In the experiment, the reward settings of VRSF are shown in the Tab.2.

<table>
<thead>
<tr>
<th>Reward type</th>
<th>Characteristic value</th>
<th>Range</th>
<th>Reward settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stagnation Fine</td>
<td>$D(DED) + D(TED)$</td>
<td>[0,1e2)</td>
<td>-0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1e2,1e3)</td>
<td>-0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1e3,1e4)</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1e6, +∞)</td>
<td>+0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0,0.5)</td>
<td>-0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[0.5,0.7)</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1.4, 2]</td>
<td>+0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2, +∞)</td>
<td>+0.2</td>
</tr>
</tbody>
</table>

We compared the change of the success rate of reaching the destination and the change of the exploration boundary in the experimental process. The experimental results show that DVRSF has
faster learning speed (Fig.7a) and better exploration ability (Fig.7b), can be more widely adapted to the changing complex environment.

![Fig.7a](image1.png) ![Fig.7b](image2.png)

Fig.7 Comparison of learning speed between VRSF and DVRSF

5.3 Learning efficiency

In order to prove that our algorithm has an accelerating effect on exploration, we counted the five indicators of the time step and the time step when the time step and the success rate reached 0.1, 0.2, 0.5 and 0.9 respectively when we first reached the destination (among which the success rate of PPO algorithm has not reached 0.9 within 50M steps). Through comparison (Tab.3), we found that DVRSF improved the PPO algorithm significantly, and the effect of A3C based DVRSF algorithm in some environments was not good, The learning efficiency of the agent successfully reaching the end point for the first time increased by an average of 3.54% (among which the PPO group increased by 38.44%), and the learning efficiency from the first reaching the end point to the success rate of 0.5 increased by 76.17% (among which the PPO group increased by 87.47%).

<table>
<thead>
<tr>
<th>Method/Map</th>
<th>W1S1</th>
<th>W1S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success rate</td>
<td>1st get</td>
<td>0.1</td>
</tr>
<tr>
<td>DVRSF</td>
<td>354K</td>
<td>1.35M</td>
</tr>
<tr>
<td>A3C</td>
<td>533K</td>
<td>1.39M</td>
</tr>
<tr>
<td>DVRSF(PPO)</td>
<td>255K</td>
<td>384K</td>
</tr>
<tr>
<td>PPO</td>
<td>263K</td>
<td>699K</td>
</tr>
</tbody>
</table>

Because the environment is too complex, in some game levels, the existing algorithms and improved algorithms cannot successfully reach the end point. However, when training in such a difficult environment, DVRSF still plays a positive role in expanding the exploration boundary.

5.4 Improvement of sparse reward

The design purpose of DVRSF is to improve the sparse reward problem of the original environment. In order to verify whether the reward has been improved, we recorded the reward acquisition of the agent at each timestep, as shown in the Fig 8, we recorded the original
environment reward (external reward), SF reward, VR reward, and the final synthesized DVRSF reward. The experimental results show that the proportion of zero reward in the original environmental reward is 23.75% (11879/50000). After the improvement of DVRSF, the proportion of zero reward is only 2.6% (1314/50000), and almost all of them are at the beginning of training. The main reason is that the exploration depth sampling area at this stage has not been established, so there is no internal reward. The experimental results can prove that DVRSF method greatly improves the distribution of rewards and can effectively solve the problem of sparse rewards.

Fig.8 Decomposition reward for the first 50000 steps

5.5 Exploration over divergent

For the difficult problem of exploration, the appropriate divergent exploration direction is an effective solution. However, when the agent focuses on the divergent exploration direction, it is easy to lose the correct exploration direction. We call this phenomenon ‘exploration over divergent’ problem. As mentioned above, the purpose of adding velocity reward is to reduce the over divergence of exploration caused by SF. We expect SF to maintain a high trajectory entropy during exploration, that is, keep it in a more active state, but this also brings another problem, that is, the agent may not directly "bump" into a correct exploration direction every time. At this time, the addition of velocity reward can help the agent grasp the exploration direction. The reward guides the agent to explore in the right direction with the exploration speed in the right exploration direction. In the exploration process shown in the Fig 9, when using the DVR algorithm, the growth of success rate stagnated for a period of time due to the loss of exploration direction, while the growth curve of the DVRSF algorithm with velocity reward was significantly smoother, which proved the inhibition effect on the over divergent phenomenon of exploration.
6 Discussion

In this study, we propose a DRL method based on dynamic velocity reward and stagnation fine. The main contribution of this method is to propose a set of features and their sampling methods for calculating the exploration state of agents. It improves the subsequent training process by extracting features in real time during the training process. Experiments show that it can effectively represent the problems encountered by agents during exploration. Adding the intrinsic rewards constructed by the above features has a positive effect on the training. On this basis, we use the velocity potential function to constrain the exploration direction of the agent, so as to avoid the problem of exploration divergence caused by expanding the exploration boundary. In the Super Mario Bros environment, our algorithm can make the agent reach the destination earlier than the baseline, and the success rate is at a higher level, so as to train a more stable model with better accuracy.

In addition, the algorithm is a more general DRL exploration method. It almost does not need to consider the network structure and the process architecture of the algorithm. It can be easily improved in different algorithms, and has good universality. In the experiment, we have implemented the DVRSF algorithm based on A3C and PPO algorithm. In addition, more algorithms will be adapted.

During the experiment, we also found some problems that have not been improved. DVRSF algorithm is easy to make wrong calculation of trajectory entropy in some complex scenarios, resulting in the failure of stagnation fine. This usually happens when the critical path is closely surrounded by dense penalty.

7 Conclusion

We propose a RL exploration method based on the dynamic shaping of intrinsic rewards by the speed potential function. This method relies on the construction of two features we designed. It is generated in real time during the exploration of agents, and can better represent the exploration progress and obstacles. Our method can be effectively combined with almost all RL algorithms.
without adding complex neural networks or carefully optimizing parameters. DVRSF algorithm can enable agents to obtain more accurate real-time rewards, so that agents can quickly leave the exploration dilemma and find the optimal policy faster. We have compared the DVRSF algorithm with several baselines through detailed comparative experiments. The experimental results show that DVRSF improves the original algorithm from many aspects, and increases the learning efficiency and model effect without increasing computing resources. In future work, we intend to design a more accurate calculation method of trajectory entropy to more accurately represent the trajectory feature while taking into account the computing resources. Another important work in the future will be to find a more suitable sampling method for DED and TED.

**Funding**

This research was funded by the National Natural Science Foundation of China, grant number 62806221.

**Data Availability Statement**

All data included in this study are available upon request by contact with the corresponding author. The data used to support the findings of this study are available from the website https://github.com/uviper/Super-mario-bros-PPO-pytorch.

**Conflicts of Interest**

The authors declare no conflict of interest.

**Reference**


