Reinforcement Learning using Reward Expectations in Scenarios with Aleatoric Uncertainties

Yubin Wang
PLA SSF Information Engineering University
Zhengzhou, China
bin2129670034@163.com

Yifeng Sun*
PLA SSF Information Engineering University
Zhengzhou, China
yfsun001@163.com

Jiang Wu
PLA SSF Information Engineering University
Zhengzhou, China
liam181113@163.com

Hao Hu
PLA SSF Information Engineering University
Zhengzhou, China
wjjhh_908@163.com

Zhiqiang Wu
PLA SSF Information Engineering University
Zhengzhou, China
herowzq@126.com

Weigui Huang
PLA SSF Information Engineering University
Zhengzhou, China
weiguiwong@aliyun.com

***Abstract*—In scenarios with aleatoric uncertainties, the reward got by an agent when executing the same action in the same state is random, which can reduce the stability and convergence speed of the reinforcement algorithms. However, in most scenarios, reward functions have regularity, and their expectations are determined, which can be got through models or sample statistics. This paper discusses the distribution relationship between reward functions and value functions in scenarios with aleatoric uncertainties and proves the feasibility of using reward expectations for reinforcement learning. Finally, experiments show that algorithms have better stability and convergence speed when using reward expectations than random rewards.**

Keywords—reinforcement learning, aleatoric uncertainty, reward expectations, algorithm stability

I. INTRODUCTION

Deep Reinforcement Learning (DRL) has achieved significant success in the domains of unmanned driving [1], robot control [2], go [3]–[5], video games [6], but it also faces challenges such as large state-action space, sparse and delayed rewards, noisy distractions, multi-agent exploration [7]. Go, video games and other tasks usually have the following three characteristics: first, the environmental states are completely observable; second, most of the state transition processes are clear and repeatable; third, the rewards are determined. However, affected by environmental noise and other factors, the state transition processes of many environments in the real world have aleatoric uncertainty. The aleatoric uncertainty will lead to the randomness of action rewards in reinforcement learning and make it more difficult for agents to learn effective policies.

To solve the two types of uncertainty exploration problems, methods based on the UCB and the Thompson Sampling method are mainly used for exploration [7]. The aleatoric uncertainty cannot be eliminated compared with the epistemic uncertainty [8]. The Quantile Regression DQN (QR-DQN) [9] improves the performance of DQN by explicitly modeling the return distribution. Reference [10] proposed an algorithm to estimate the aleatoric uncertainty and the epistemic uncertainty using two neural networks. Decaying Left Truncated Variance (DLTV) [11] uses a decaying schedule to suppress the intrinsic uncertainty and uses the upper quantiles to calculate the exploration rewards. These algorithms not only improve the exploration efficiency but also cause more computation. Reward shaping [12] is a widely used method for setting reward functions in complex environments and multi-agent collaboration scenarios. Reference [13] improves the convergence speed of DRL algorithms with the Reward-Randomized Policy Gradient (RPG) algorithm, and it also shows that different results can be explored under different reward functions. Therefore, reward shaping can not only improve the convergence speeds of reinforcement learning algorithms but also lead to the deviation of strategy convergence results. So using scores as reward function values is a common practice when there are clear scoring rules. However, the uncertainty of scores got directly in scenarios with aleatoric uncertainties will lead to the difficulty for convergences of the algorithms.

The uncertain rewards in scenarios with aleatoric uncertainties are usually regular, and their expectations can be calculated from the samples. Especially in computer simulation systems, the reward expectations can be got directly from the models. The values of reward expectations are certain and unchanged compared with the uncertain rewards. If the reward expectations are used as the reward function values, the corresponding optimal policy will be certain. This paper discusses and proves the feasibility of using the reward expectations as the reward function values for the reinforcement learning algorithms in scenarios with aleatoric uncertainties. Experiments show that compared with the uncertain rewards got in the process of exploration, using the reward expectations as the reward function values can increase the stabilities and the convergence speeds of reinforcement learning algorithms.

The main contributions of this work are listed as follows:

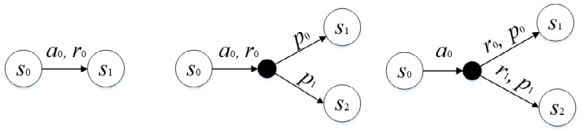
- This paper analyzes the influence of the uncertain rewards on reinforcement learning in scenarios with aleatoric uncertainties, and puts forward the idea of using the reward expectations as the reward function values.
- The feasibility of using the reward expectations in different cases is discussed. It is proved that the stabilities of the algorithms using the reward expectations are better than using the uncertain rewards got directly.
- The effectiveness of the method is verified based on the DQN algorithm and the AC algorithm in a scenario with aleatoric uncertainties.

This work was supported by the National Natural Science Foundation of China under Grant No. 61902427.

II. BACKGROUND

An agent sequential decision-making problem can usually be modeled by Markov Decision Process (MDP) which can be described by the tuple $\langle \boldsymbol{S}, \boldsymbol{A}, R, P \rangle$. $\boldsymbol{S}$ is the collection of all states of the agent. $\boldsymbol{A}$ is the collection of optional actions of the agent. R is the reward function, and $R(s,a,s')$ is the reward got by the agent when the state changes to s' after taking action a in the state s. P is the state transition probability, and $P(s'|s,a)$ is the probability that the agent will reach the state s' after taking action a in the state s. The state transition meets the equation $P(s_{t+1}|s_0,s_1,\cdots,s_t,a)=P(s_{t+1}|s_t,a)$, that is, the next state is only related to the current state and action. Every time an agent makes a decision, it needs to choose actions according to the current state, and the goal of its strategy $\pi: a \sim \pi(\cdot|s)$ is to maximize the cumulative reward.

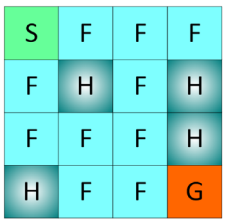
According to the state transition processes and rewards, MDP can be divided into three categories as shown in Fig. 1, in which s, a, r and p represent the corresponding values of $\langle \boldsymbol{S}, \boldsymbol{A}, R, P \rangle$. In (a) and (b), the reward got by the agent after taking the same action in the same state is certain, while in (c), it is uncertain. The rewards got by the agent through interaction with the environment are the basis for its strategy optimization. When the state reached after one action is uncertain (similar to the k-arm slot machine), it will usually lead to uncertainty of the reward after the same state and action. In this case, the agent must explore to obtain the cognitive law in the rewards $R(s,a,s')$, that is, to master the probability distribution. If we do not grasp this law, it will lead the difficulty for convergences of the algorithms.



(a) State transitions and rewards are certain (b) State transitions are uncertain and rewards are certain (c) State transitions and rewards are uncertain

Fig. 1. The state transition processes and rewards in MDP.

Agents must execute actions in the same states many times to explore the above laws before obtaining them. Bayesian posterior probability and distributional value functions are usually used to model the epistemic uncertainty and the aleatoric uncertainty. However, in scenarios with aleatoric uncertainties, agents can only obtain fragmented cognition in the process of finite exploration. For example, in the war game, the attack result of the same equipment against the same opponent is not a certain value. It is usually highly random due to the influence of terrain characteristics and some other factors, but generally follows a certain probability distribution, which is a general cognition. When exploring in massive state-action space, the agent may get poor reward feedback under a good strategy due to the aleatoric uncertainty. In this case, the samples will optimize the strategy in the wrong direction, resulting in the failure of this exploration. For another example, in the Frozen Lake [14] game show in Fig. 2, there may be a gust of wind to blow the agent onto any square after it chooses to move up, down, left or right. In this scenario, even if the agent chooses the correct moving direction, it may fall into the ice cave due to uncertainty, In this case, the feedback got by the agent is unfavorable to the strategy optimization. There are also similar situations in StarCraft II [15], [16], Dota2 [17] and some other games.



S: starting point, safe
F: frozen surface, safe
H: hole, fall to your doom
G: goal, where the frisbee is located

Fig. 2. The Frozen lake games.

In conclusion, the rewards agents got in scenarios with aleatoric uncertainties have high randomness. In this case, the dilemma of reinforcement learning is that the cognition obtained by finite exploration is often unreliable. To solve the problem, the reward expectations got from a priori model knowledge or samples are used to assist the exploration process.

III. THE METHOD

A. Difficulties in Scenarios with Aleatoric Uncertainties

The action-value function $Q_\pi(s,a)$ of an MDP is the expected return starting from the state s, taking action a, and then following policy π, which can be defined as (1):

$$Q_\pi(s,a)=\mathbb{E}_{a_t\sim\pi(\cdot|s_t),s_{t+1}\sim P(\cdot|s_t,a_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1})|s_0=s,a_0=a\right] \quad (1)$$

The state-value function $V_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π, which can be defined as (2):

$$V_\pi(s)=\mathbb{E}_{a_t\sim\pi(\cdot|s_t),s_{t+1}\sim P(\cdot|s_t,a_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1})|s_0=s\right] \quad (2)$$

In (1) and (2), $\gamma\in[0,1]$ is the weight of long-term rewards when calculating the cumulative return. The greater its value, the more attention it pays to long-term rewards. Based on (1) and (2), the Bellman equation can be got by recursion. The corresponding expressions are as follows:

$$Q_\pi(s,a)=\mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a,s')+\gamma\mathbb{E}_{a'\sim\pi(\cdot|s')}\left[Q_\pi(s',a')\right]\right] \quad (3)$$

$$V_\pi(s)=\mathbb{E}_{a\sim\pi(\cdot|s),s'\sim P(\cdot|s,a)}\left[R(s,a,s')+\gamma V_\pi(s')\right] \quad (4)$$

It can be deduced from (3) and (4) that when the rewards got by the agent are certain as (a) and (b) in Fig. 1, the convergence speeds of algorithms using the action-value function or the state-value function in reinforcement learning are mainly affected by the probability distribution $s'\sim P(\cdot|s,a)$. But when the rewards got by the agent are uncertain as (c) in Fig. 1, it is also affected by the uncertainty.

If $R(s,a)$ is used to represent the expectation of $R(s,a,s')$, (3) can be converted to (5).

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a,s') + \gamma \mathbb{E}_{a'\sim\pi(\cdot|s')}\left[Q_\pi(s',a')\right]\right] \\ &= \mathbb{E}_{s'\sim P(\cdot|s,a)} R(s,a,s') + \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[\gamma \mathbb{E}_{a'\sim\pi(\cdot|s')}\left[Q_\pi(s',a')\right]\right] \\ &= R(s,a,) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a),a'\sim\pi(\cdot|s')}\left[Q_\pi(s',a')\right] \end{aligned} \quad (5)$$

Taking reinforcement learning based on the action-value function as an example, in the actual calculation process, the optimal action-value function $Q_\pi(s',a')=\max_{a'} Q_\pi(s',a')$ is usually used for value function iteration. If $Q_\pi^*(s,a)$ is used to represent the target value of $Q_\pi(s,a)$ at each iteration, it can be calculated by (6).

$$Q_\pi^*(s,a) = R(s,a,s') + \gamma \max_{a'} Q_\pi(s',a') \quad (6)$$

In deep reinforcement learning, $\boldsymbol{\theta}$ is usually used to represent the parameters of a neural network. The size of the sample set used to optimize the neural network is B. State s_i and action a_i represent the state and action of the i-th sample respectively. As the error of $\boldsymbol{\theta}$, $\mathcal{L}(\boldsymbol{\theta})$ is calculated using mean square deviation as shown in (7).

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i,s_i') + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') - Q_{\boldsymbol{\theta}}(s_i,a_i)\right)^2\right] \\ &\approx \mathbb{E}_{i\in[1,B]}\left[\begin{pmatrix} R(s_i,a_i,s_i') + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') \\ -\left(R(s_i,a_i) + \gamma \mathbb{E}_{s'\sim P(\cdot|s_i,a_i)}\left[\max_{a'} Q_{\boldsymbol{\theta}}(s',a')\right]\right)\end{pmatrix}^2\right] \\ &= \mathbb{E}_{i\in[1,B]}\left[\begin{pmatrix} R(s_i,a_i,s_i') - R(s_i,a_i) + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') \\ -\gamma \mathbb{E}_{s'\sim P(\cdot|s_i,a_i)}\left[\max_{a'} Q_{\boldsymbol{\theta}}(s',a')\right]\end{pmatrix}^2\right] \end{aligned} \quad (7)$$

When B is constant, $\mathcal{L}(\boldsymbol{\theta})$ is mainly affected by the uncertainty of the reward $R(s_i,a_i,s_i')$ and the action-value $Q_{\boldsymbol{\theta}}(s_i',a')$. The greater the uncertainty, the greater the uncertainty of $\mathcal{L}(\boldsymbol{\theta})$, the greater the disturbance to the strategy optimization, and the more difficult the strategy optimization is. In deep reinforcement learning, it will cause the uncertainty of the strategy gradient of the neural network and affect the convergence speed of the parameters of the neural networks.

B. Convergences of Algorithms using Reward Expectations

In scenarios with aleatoric uncertainties, although the same action in a specific state may reach different result states, the overall trend is clear, that is, the magnitude of the expectations is clear. If the expectations are used as the reward function values, will it improve the convergences of the reinforcement learning algorithms?

When the reward function values are uncertain, the reward of each sample actually corresponds to different optimal strategies. On the contrary, if the certain reward expectations are used as the reward function values for reinforcement learning, the corresponding optimal strategy should be determined. If the reward expectations $R(s,a)$ is used as the reward function value instead of the uncertain rewards $R(s,a,s')$, $Q_\pi^*(s,a)$ can be calculated by (8).

$$Q_\pi^*(s,a) = R(s,a) + \gamma \max_{a'} Q_\pi(s',a') \quad (8)$$

Therefore, the action-value function $Q_\pi^*(s,a)$ calculated by using $R(s,a)$ as the reward function value in (8) is an unbiased estimation of the one in (6). $R(s,a)$ is used to represent the expectation of $R(s,a,s')$. Similarly, uses $R(s,a)$ instead of $R(s,a,s')$ as the reward function, (9) can be derived in the same way as to (7). In (9), $\mathcal{L}(\boldsymbol{\theta})$ is only related to the uncertainty of the next action-value $Q_{\boldsymbol{\theta}}(s_i',a')$.

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &\approx \mathbb{E}_{i\in[1,B]}\left[\begin{pmatrix} R(s_i,a_i) - R(s_i,a_i) + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') \\ -\gamma \mathbb{E}_{s'\sim P(\cdot|s_i,a_i)}\left[\max_{a'} Q_{\boldsymbol{\theta}}(s',a')\right]\end{pmatrix}^2\right] \\ &= \mathbb{E}_{i\in[1,B]}\left[\gamma^2\begin{pmatrix} \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') \\ -\mathbb{E}_{s'\sim P(\cdot|s_i,a_i)}\left[\max_{a'} Q_{\boldsymbol{\theta}}(s',a')\right]\end{pmatrix}^2\right] \end{aligned} \quad (9)$$

In the actual calculation processes, the uncertainty magnitudes of $R(s_i,a_i,s_i')$ and $Q_{\boldsymbol{\theta}}(s_i',a')$ in (7) may be different, and there may be a certain correlation between them. The convergences of the algorithms using the uncertain rewards got directly and the reward expectations as the reward function values are discussed as follows.

1) If the variance caused by reward uncertainty is much greater than that caused by state transition uncertainty: Inequality (10) holds.

$$\begin{aligned} &\mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i,s_i') - R(s_i,a_i)\right)^2\right] \\ &\gg \mathbb{E}_{i\in[1,B]}\left[\gamma^2\left(\max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') - \mathbb{E}_{s'\sim P(\cdot|s_i,a_i)}\left[\max_{a'} Q_{\boldsymbol{\theta}}(s',a')\right]\right)^2\right] \end{aligned} \quad (10)$$

Substituting (7) and (9) into (10), inequality (11) holds.

$$\begin{aligned} &\mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i,s_i') + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') - Q_{\boldsymbol{\theta}}(s_i,a_i)\right)^2\right] \\ &\gg \mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i) + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') - Q_{\boldsymbol{\theta}}(s_i,a_i)\right)^2\right] \end{aligned} \quad (11)$$

In this case, the variance of iteration using reward expectations is smaller, which can improve the stability of the algorithm.

2) If the variance caused by reward uncertainty is much smaller than that caused by state transition uncertainty: Inequality (12) holds.

$$\begin{aligned} &\mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i,s_i') - R(s_i,a_i)\right)^2\right] \\ &\ll \mathbb{E}_{i\in[1,B]}\left[\gamma^2\left(\max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') - \mathbb{E}_{s'\sim P(\cdot|s_i,a_i)}\left[\max_{a'} Q_{\boldsymbol{\theta}}(s',a')\right]\right)^2\right] \end{aligned} \quad (12)$$

Substituting (7) and (9) into (12), approximately equation (13) holds.

$$\begin{aligned} &\mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i,s_i') + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') - Q_{\boldsymbol{\theta}}(s_i,a_i)\right)^2\right] \\ &\approx \mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i) + \gamma \max_{a'} Q_{\boldsymbol{\theta}}(s_i',a') - Q_{\boldsymbol{\theta}}(s_i,a_i)\right)^2\right] \end{aligned} \quad (13)$$

In this case, the variances of iteration using the rewards got directly and using the reward expectations are approximately equal, and the stabilities of the corresponding algorithms are approximately equal.

3) If the variance caused by reward uncertainty and that caused by state transition uncertainty are approximately

equal: The variance and covariance are represented by $\mathbb{D}$ and $\mathbb{COV}$ respectively.

a) If $R(s_i,a_i,s_i')$ *and* $Q_\theta(s_i',a')$ *are positively correlated:* Inequality (14) holds.

$$\mathbb{COV}\left(R(s_i,a_i,s_i')+\gamma\max_{a'}Q_\theta(s_i',a')\right)>0 \tag{14}$$

Inequality (15) can be derived from (14).

$$\mathbb{D}\left(R(s_i,a_i,s_i')+\gamma\max_{a'}Q_\theta(s_i',a')\right)>\mathbb{D}\left(\gamma\max_{a'}Q_\theta(s_i',a')\right) \tag{15}$$

Substituting (7) and (9) into (15), inequality (16) holds.

$$\begin{aligned}&\mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i,s_i')+\gamma\max_{a'}Q_\theta(s_i',a')-Q_\theta(s_i,a_i)\right)^2\right]\\&>\mathbb{E}_{i\in[1,B]}\left[\left(R(s_i,a_i)+\gamma\max_{a'}Q_\theta(s_i',a')-Q_\theta(s_i,a_i)\right)^2\right]\end{aligned} \tag{16}$$

In this case, the variance of iteration using reward expectations is smaller, which can improve the stability of the algorithm.

b) If $R(s_i,a_i,s_i')$ *and* $Q_\theta(s_i',a')$ *are irrelevant:* Inequality (17) holds.

$$\mathbb{COV}\left(R(s_i,a_i,s_i')+\gamma\max_{a'}Q_\theta(s_i',a')\right)=0 \tag{17}$$

In this case, the same result can be got as that in *a)*.

c) If $R(s_i,a_i,s_i')$ *and* $Q_\theta(s_i',a')$ *are negatively correlated:* The variances of iteration using the rewards got directly and using the reward expectations are is uncertain. This situation rarely occurs.

To sum up, in most cases, by using the reward expectation instead of the uncertain reward got directly from the environment, the uncertainty of reward function values can be decoupled from the iterative process of action-value function, and the stability of reinforcement learning algorithm in scenarios with aleatoric uncertainties can be improved.

C. Algorithms using Reward Expectations

In real scenarios, especially in complex decision-making problems, $Q_\pi(s,a)$ and $V_\pi(s)$ usually needs a lot of calculation, while the reward expectation functions are usually only related to some variables in the states. In this case, x is used to represent the variables related to the reward expectation functions in s, that is, there is a function R^* to make (18) true.

$$R^*(x,a)=R(s,a),x\subset s \tag{18}$$

The difficulty of the solving function R^* is generally much less than that of strategy optimization. Taking the reinforcement learning algorithm to solve the war game problem described in part II as an example, this problem is a complex Multi-Agent Reinforcement Learning problem, and its strategy learning is very difficult. However, the result of an attack action is only related to a small number of factors such as elevation and terrain. This functional relationship between the attack results and related factors R^* is easy to obtain through the simulation system model or the statistics of the sample data got from the environment. In this case, it is feasible to solve the reward expectation function R^* first and then use it as the reward function for reinforcement learning.

In the actual calculation process, if the system model is known, R^* can be obtained through the model directly. If the system model is unknown, the function R^* needs to be solved first. When the system has no model, if R^* is simple, it can be solved through sample statistics using (19).

$$R^*(x,a)=\mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a,s')\right],x\subset s \tag{19}$$

When the system has no model, if R^* is complex, it can be modeled by using a neural network (called R^*-net here). The inputs of R^*-net are the status factors and actions related to rewards, and the outputs are the reward expectations. Taking the representative the DQN algorithm and the AC algorithm as examples, the algorithms using reward expectation function R^* as the reward function are as follows.

1) DQN algorithm using reward expectations: The parameter of action-value neural network Q is represented by $\boldsymbol{\theta}$. The relationship between x and s is $x=\phi(s)$. The exploration rate is ε. The algorithm is shown as Algorithm 1.

2) AC algorithm using reward expectations: The parameters of policy neural network π and state-value neural network V are represented by $\boldsymbol{\theta}$ and $\boldsymbol{w}$ respectively. The relationship between x and s is $x=\phi(s)$. The algorithm is shown as Algorithm 2

Algorithm 1 DQN using reward expectations

Input: a differentiable action-value function parameterization $Q_\theta(s,a)$, the reward expectation function $R^*(x,a)$
Parameters: discount factor γ, replay memory capacity N, network parameter update step sizes $\alpha_\theta>0$, delay steps for action-value target network updates C
Initialize replay memory D to capacity N
Initialize action-value function Q with weights $\boldsymbol{\theta}$
Initialize target action-value function $\hat{Q}$ with weights $\boldsymbol{\theta}^-=\boldsymbol{\theta}$
for episode $=1,2,3\cdots$ **do**
  Initialize sequence $s_0=s$
  for $t=1,2,3\cdots$ **do**
    With probability ε select a random action a_t otherwise select $a_t=\max_a Q_{\theta^*}(s_t,a)$
    Execute action a_t in emulator and observe reward r_t and the next state s_{t+1}
    Preprocess $x_t=\phi(s_t)$
    If using (19) or R^*-net as the reward expectation function R^*, update R^* with sample $R^*(x_t,a_t)=r_t$

Algorithm 1 DQN using reward expectations

Predict reward expectation $R_t = R^*(x_t, a_t)$

Store transition (s_t, a_t, R_t, s_{t+1}) in D

Sample random minibatch of transitions (s_j, a_j, R_t, s_{j+1}) from D

Set $y_j = \begin{cases} R_j & \text{if episode terminates at step } j+1 \\ R_j + \gamma \max_{a'} \hat{Q}_{\theta^-}(s_{j+1}, a') & \text{otherwith} \end{cases}$

Perform a gradient descent step on $(y_j - Q_\theta(s_j, a_j))^2$ with respect to the network parameter $\boldsymbol{\theta}$ according to $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_\theta (y_j - Q_\theta(s_j, a_j)) \nabla_\theta Q_\theta(s_j, a_j)$

Every C steps reset $\hat{Q} = Q$

end for

end for

Algorithm 2 AC using reward expectations

Input: a differentiable policy parameterization $\pi_\theta(a|s)$, a differentiable state-value function parameterization $V_w(s)$

Parameters: discount factor γ, network parameter update step sizes $\alpha_\theta > 0$, $\alpha_w > 0$

Initialize policy function π with weights $\boldsymbol{\theta}$

Initialize state-value function V with weights $\boldsymbol{w}$

for episode $= 1, 2, 3 \cdots$ **do**

Initialize sequence $s_0 = s$

for $t = 1, 2, 3 \cdots$ **do**

Select an action a_t according to $\pi_\theta(a|s)$

Execute action a_t in emulator and observe reward r_t and the next state s_{t+1}

Preprocess $x_t = \phi(s_t)$

If using (19) or $R^* - net$ as the reward expectation function R^*, update R^* with sample $R^*(x_t, a_t) = r_t$

Predict reward expectation $R_t = R^*(x_t, a_t)$

Store transition (s_t, a_t, R_t, s_{t+1})

Valuate the advantage function $A_t = \begin{cases} R_t - V_w(s_t) & \text{if episode terminates at step } j+1 \\ R_t + \gamma V_w(s_{t+1}) - V_w(s_t) & \text{otherwith} \end{cases}$

Perform a gradient descent step on A_t^2 with respect to the network parameter $\boldsymbol{w}$ according to $\boldsymbol{w} = \boldsymbol{w} + \alpha_w A_t \nabla_\theta V_w(s_t)$

Perform a gradient descent step on $A_t \log \pi_\theta(a_t|s_t)$ with respect to the network parameter $\boldsymbol{\theta}$ according to $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_\theta A_t \nabla_\theta \log \pi_\theta(a_t|s_t)$

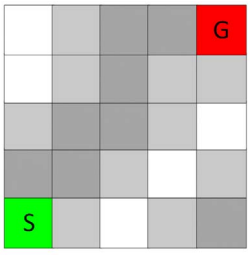
end for

end for

IV. Experiments and Analysis

A. Introduction to the Experimental Environment

To verify the efficiency of the method described in this paper, the experiment shown in Fig. 3 are carried out on the background of the war game described in part II. It simulates the path planning problem in the war game. Starting from grid S, the initial life value of the agent is 10. At each step, the agent can move one grid to one of the neighbor grids, and will be attacked by the opponent at the same time. When the agent reaches the target point G or its life value is 0, the game ends, and the remaining life value of the agent is the final score.



(a) The topographic map

1.5	1	0.5	0.5	G
1.5	1	0.5	1	1
1	0.5	0.5	1	1.5
0.5	0.5	1	1.5	1
S	1	1.5	1	0.5

(b) Attack result correction coefficien

Fig. 3. The experiment based on war games.

Each attack by the opponent is an uncertain value. The experiment simulates the uncertain attacks in two ways: normal distribution and uniform distribution. The probability distributions are shown in Table I.

TABLE I. Attack Value Probability Distributions

Attack Value	Probability Distributions	
	Normal distribution	*Uniform distribution*
0	0.0071	1/11
1	0.0021	1/11
2	0.0588	1/11
3	0.1192	1/11
4	0.1875	1/11
5	0.2128	1/11
6	0.1875	1/11
7	0.1192	1/11
8	0.0588	1/11
9	0.0021	1/11
10	0.0071	1/11

In Fig. 3 (a), the grayscale of different grids indicates that the grids have different masking effects. The deeper the grayscale, the better the masking effect. The correction

coefficient to the attack result is shown in Fig. 3 (b). At step t, if the agent's life value is l_t, the masking effect correction coefficient of the grid where the agent is after moving is m_t, and the random attack value by the opponent is k_t (k_t follows the normal distribution or uniform distribution in Table I), the life of the agent after moving l_{t+1} can be calculated by (20).

$$l_{t+1} = \max(l_t - r_t \cdot m_t, 0) \tag{20}$$

For example, if the agent move right at step 0, and the random attack value by the opponent is K, then the masking effect correction coefficient m_0 will be 1, and the life of the agent after moving l_1 will be $10-K$. Compared with the traditional maze game, the agent in the experiment also needs to explore and learn the optimal path to the target grid. The difference is that in the maze experiment, when the agent acts in the same state, the feedback got by exploration at each step is certain, while in this experiment, the feedback is uncertain. For example, in this experiment, the moving up action in step 0 is better than moving right. However, due to the uncertainty of the opponent's attack value, sometimes the feedback got by the agent when moving right will be better than that of moving up. Feedback from the environment is often used as reward directly. If the value got in this case is used as the reward function directly, the agent strategy will be optimized in the wrong direction, resulting in the difficulty of the strategy learning.

B. *Analysis of the Results of Different Algorithms*

To test the convergences of the DQN algorithm and the AC algorithm using reward expectations described in 3.2, the performances of the following four methods with different reward functions are compared in the experiment:

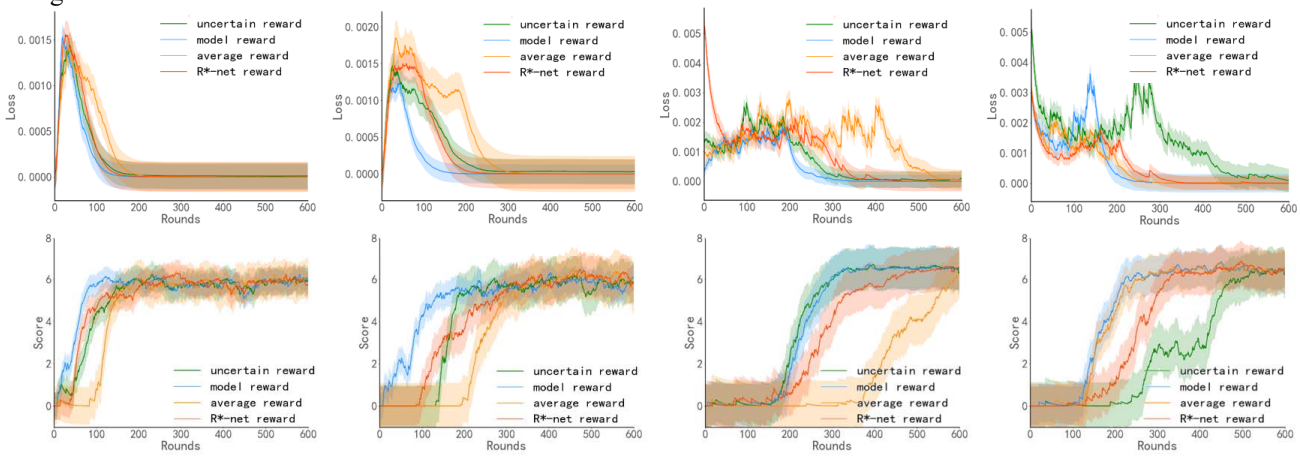
1) Uncertain reward: The uncertain reward values got by exploration are directly used as the reward function values.

2) Model reward: Assuming that the system model is known, the reward expectations calculated through Fig. 3 (b) and Table I are used as the reward function values.

3) Average reward: Assuming that the system model is known, The statistical mean of historical samples got by exploration is used as the approximate value of the reward function values, which are used as the reward function values.

4) R^*-net *reward:* Assuming that the system model is unknown, a neural network is trained with the explored historical samples, and the neural network is used to predict reward expectations, which are used as the reward function values. This method is consistent with the algorithm given in part C of part III.

The stability and convergence speed of the algorithms are measured by the change of loss values $\mathcal{L}(\boldsymbol{\theta})$ and scores got by the agents. The score curves of the DQN algorithm and the AC algorithm using the above four reward functions are shown in Fig. 4.



(a) DQN when attacks follow the normal distribution

(b) DQN when attacks follow the uniform distribution

(c) AC when attacks follow the normal distribution

(d) AC when attacks follow the uniform distribution

Fig. 4. Comparison of the DQN algorithm and the AC algorithm using different reward functions.

Comparing the score curves and loss curves in the results, it is proved that the algorithms using uncertain rewards as the reward functions are more vulnerable to the influence of reward uncertainty, especially algorithms using model rewards as the reward functions are most stable the scores of which increase faster and the loss values of which are smaller.

When using the R^*-net reward as the reward functions, because the evaluation of the reward functions are not accurate at the beginning, the early convergence speeds and stabilities of the algorithms are generally lower than that of using model rewards. With the increase of the number of samples, the estimated values gradually approach the reward expectations, and the convergences of the algorithms using R^*-net reward are gradually close to that of the algorithms using model rewards. Because the R^* function in the experiment is simple, using the average reward can also get the approximate value of reward expectation easily. So the convergence speeds and stabilities of the algorithms using the average reward are approximately equal to those using the R^*-net reward.

Because the uncertainty of the normal distribution is weaker than that of the uniform distribution, the convergences of different algorithms are more different when the uncertainty follows the uniform distribution. This shows that the greater the uncertainty of the scenario, the greater the advantage of using R^* function than using random reward.

Because the DQN algorithm uses the maximum values of the action-value to iterate, and always maintains the exploration probability ε, its variances are larger than that of the AC algorithm, so the stability and convergence speed of the algorithms based on the DQN algorithm are worse than those based on the AC algorithm.

To sum up, both the DQN algorithm and the AC algorithm using reward expectations as reward function values have better convergence speeds and stabilities than that using uncertain reward got directly from the environment.

V. CONCLUSION

To solve the difficulty of high randomness of reward in scenarios with aleatoric uncertainties, this paper analyzes the influence of the uncertainty of reward function on the speed and result of strategy optimization. The method using the reward expectations which are certain instead of uncertain rewards got directly from the environments is proposed to decouple the solution process of reward and value function solution. The feasibility of the method is discussed and proved. Finally, the effectiveness of the method is verified by experiments. Experiments show that the reinforcement learning algorithms using reward expectations as reward function values can improve the stability and convergence speed of the algorithms in scenarios with aleatoric uncertainties. The method to solve the problem of reward uncertainty in scenarios with aleatoric uncertainties proposed in this paper focuses on the case that the reward function is positively correlated or irrelevant to the latter state value. At the same time, it is assumed that the reward value is only related to some factors in the state, and the function is known or its solution is less difficult. When the reward function is complex, how to solve it and how to combine it with the existing methods to solve difficulties in scenarios with aleatoric uncertainties are still problems that need to be further studied.

REFERENCES

[1] A. R. Fayjie, S. Hossain, D. Oualid, and D. J. Lee, "Driverless Car: Autonomous Driving Using Deep Reinforcement Learning in Urban Environment,", 2018, pp. 896-901.

[2] A. H. Tan, F. P. Bejarano, and G. Nejat, "Deep Reinforcement Learning for Decentralized Multi-Robot Exploration with Macro Actions,", 2021.

[3] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484-489, 2016.

[4] D. Silver et al., "Mastering the game of Go without human knowledge," Nature, vol. 550, no. 7676, pp. 354-359, 2017.

[5] J. Schrittwieser et al., "Mastering Atari, Go, chess and shogi by planning with a learned model," Nature, vol. 588, no. 7839, pp. 604-609, 2020.

[6] S. Risi and M. Preuss, "From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI,". vol. 34 Heidelberg: Springer Nature B.V, 2020, pp. 7-17.

[7] T. Yang et al., "Exploration in Deep Reinforcement Learning: A Comprehensive Survey,", 2021.

[8] A. D. Kiureghian and O. Ditlevsen, "Aleatory or epistemic? Does it matter?" Struct. Saf., vol. 31, no. 2, pp. 105-112, 2009.

[9] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, "Distributional Reinforcement Learning with Quantile Regression,", 2017.

[10] W. R. Clements, B. Van Delft, B. Robaglia, R. B. Slaoui, and S. Toth, "Estimating Risk and Uncertainty in Deep Reinforcement Learning,", 2019.

[11] B. Mavrin, S. Zhang, H. Yao, L. Kong, K. Wu, and Y. Yu, "Distributional Reinforcement Learning for Efficient Exploration,", 2019.

[12] T. Aotani, T. Kobayashi, and K. Sugimoto, "Bottom-up multi-agent reinforcement learning by reward shaping for cooperative-competitive tasks," Applied Intelligence, vol. 51, no. 7, pp. 4434-4452, 2021.

[13] Z. Tang et al., "Discovering Diverse Multi-Agent Strategic Behavior via Reward Randomization,", 2021.

[14] http://gym.openai.com/envs/FrozenLake-v0/.

[15] O. Vinyals et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," Nature, vol. 575, no. 7782, pp. 350-354, 2019.

[16] O. Vinyals et al., "StarCraft II: A New Challenge for Reinforcement Learning,", 2017.

[17] C. Berner et al., "Dota 2 with Large Scale Deep Reinforcement Learning,", 2019.