sensors

MDPI

Article

Adaptive Discount Factor for Deep Reinforcement Learning in Continuing Tasks with Uncertainty

MyeongSeop Kim [1,2], Jung-Su Kim [1,*], Myoung-Su Choi [2] and Jae-Han Park [2]

[1] Research Center for Electrical and Information Technology, Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, Korea
[2] Applied Robot R&D Department, Korea Institute of Industrial Technology (KITECH), Ansan 15588, Korea
* Correspondence: jungsu@seoultech.ac.kr; Tel.: +82-2-970-6547

Abstract: Reinforcement learning (RL) trains an agent by maximizing the sum of a discounted reward. Since the discount factor has a critical effect on the learning performance of the RL agent, it is important to choose the discount factor properly. When uncertainties are involved in the training, the learning performance with a constant discount factor can be limited. For the purpose of obtaining acceptable learning performance consistently, this paper proposes an adaptive rule for the discount factor based on the advantage function. Additionally, how to use the advantage function in both on-policy and off-policy algorithms is presented. To demonstrate the performance of the proposed adaptive rule, it is applied to PPO (Proximal Policy Optimization) for Tetris in order to validate the on-policy case, and to SAC (Soft Actor-Critic) for the motion planning of a robot manipulator to validate the off-policy case. In both cases, the proposed method results in a better or similar performance compared with cases using the best constant discount factors found by exhaustive search. Hence, the proposed adaptive discount factor automatically finds a discount factor that leads to comparable training performance, and that can be applied to representative deep reinforcement learning problems.

Keywords: reinforcement learning; discount factor; uncertainty; path planning; Tetris

Citation: Kim, M.; Kim, J.-S.; Choi, M.-S.; Park, J.-H. Adaptive Discount Factor for Deep Reinforcement Learning in Continuing Tasks with Uncertainty. *Sensors* **2022**, *22*, 7266. https://doi.org/10.3390/s22197266

Academic Editors: Gianni D'Angelo and Arcangelo Castiglione

Received: 27 July 2022
Accepted: 21 September 2022
Published: 25 September 2022

1. Introduction

Reinforcement learning (RL) is a branch of machine learning in which an agent interacts with a given environment and learns the optimal policy to achieve the predefined goal. Deep RL is a method that uses neural networks to estimate the value function or the policy in reinforcement learning. In RL, an agent makes a decision, called action, at a state according to the policy, and the action is applied to the environment. Then, the agent obtains the next state and reward from the environment. In order to compute the next action, the agent maximizes the expectation of the sum of the discounted reward signals over a finite or infinite horizon. Since the future reward is unknown, the deep RL employs neural networks to estimate the expectation, called the value function, for the future [1,2].

In real-world problems, it may be difficult to predict the value function by receiving unseen reward signals or state information, due to unexpected situations [3,4]. These problems make it difficult to estimate the value in reinforcement learning. Since the value function is computed by the reward and discount factor, it is of utmost importance to decide them properly. In particular, the focus of this paper is placed on the discount factor, which is mostly constant in the existing literature.

If the discount factor is large in an environment with high uncertainty, it may be risky to estimate the value function by considering the rewards in the distant future. Estimating the sum of rewards over a longer period of time can lead to unexpected situations due to the high uncertainty. For instance, agents can overestimate rewards in the distant future, or the convergence of the value function may not be guaranteed [5–8]. On the other hand,

if the discount factor is too small, it can lead to a better generalization performance [9–11] but hinder the convergence speed of learning [12]. Furthermore, with the small discount factor, the reward only in the near future is taken into account in evaluating the value function, which can make short-sighted or aggressive actions. This observation motivates us to devise an adaptive rule to update the discount factor, rather than a constant.

This paper especially focuses on developing an adaptive rule for the discount factor in the policy gradient algorithms that use advantage functions [13,14]. To this end, the initial low and upper bounds of the discount factor are defined in the proposed method, and the bounds are shrunk towards a higher advantage function as the training goes on. Since it is different to computing the advantage function in on-policy and off-policy algorithms, how to apply the proposed method to both on-policy and off-policy algorithms is presented.

The discount factor by the proposed method converges to the optimal constant discount factor. Note that it is a nontrivial task to find the optimal constant discount factor when the MDP is high dimensional and the environment contains uncertainties. Moreover, the resulting training performance using the proposed method is comparable with that via the optimal constant discount factor. This means that the proposed adaptive method enables the agent to counteract the overestimation of an uncertain reward sum. To verify the performance of the proposed method, two environments are used: Tetris for the on-policy and robot motion planning for the off-policy algorithms. It is shown that the proposed adaptive discount factor outperforms the cases with constant discount factors when the environment has uncertainty. Furthermore, another adjustable algorithm in [15] gradually increases the discount factor, which means that the algorithm can get into trouble when the optimal discount factor is small. On the contrary, compared with the adjustable algorithm in [15], the proposed method can adjust the discount factor adaptively for any case, i.e., either a small or large optimal discount factor.

2. Related Work

Various methods are known to prevent an overestimation in reinforcement learning. Among them, the method of taking the minimum value using two or more value estimators is known to work well in various environments, and to produce good performance [5,6,16,17]. In this paper, similarly, two value estimators are used to deal with overestimation. In an environment where uncertainty exists, an agent can be trained insensitive to the uncertainty if the experience of obtaining a high reward sum is evaluated without considering uncertainty. Therefore, distributional reinforcement learning methods for estimating the distribution of the reward sum have been introduced [18–20]. When evaluating the value function, these methods estimate a distribution rather than a scalar value, and suppress the variance of the estimated distribution. Therefore, they do not overestimate the sum of rewards with a high-risk distribution. However, sometimes these methods may perform poorly compared to the methods for estimating the value function of a scalar value [10,21]. In this paper, we present a method to cope with uncertainty by using the discount factor so that it can be applied to the method that does not estimate the distribution.

The discount factor is a constant to reflect the value of the reward signal over time in reinforcement learning, and is generally fixed to a high value [2,21]. However, sometimes a lower discount factor leads to better learning outcomes. A high discount factor is useful if the given environment has a sparse reward problem. However, if the environment does not have the problem, a low discount factor may be considered [9]. It is also known that a low discount factor can increase the generalization performance [11,12]. However, the appropriate value for the discount factor is different depending on the environment, and several trials and errors are required to find the optimal value. To compensate for this, the discount factor can be tuned gradually from a low value to a high value [15,22]. In addition, the method of increasing the discount factor may obtain a higher performance than when a fixed discount factor is used. In this paper, similar to the above method, the discount factor is increased from a low value to a high value. At the same time,

the proposed method can find an appropriate discount factor according to the given environment during the agent's training. Additionally, we confirm that the discount factor found during training can achieve higher performance than the commonly used value.

3. Discount Factor in Reinforcement Learning

In this section, for the purpose of presenting the main results clearly, the reinforcement learning is reviewed and the role of the discount factor is investigated for the different environments.

3.1. Reinforcement Learning

The Markov Decision Process (MDP) in reinforcement learning is described by the state $s_t \in \mathcal{S}$ at time step t, the reward $r_t \in \mathcal{R}$, and the action $a_t \in \mathcal{A}$. When the state s_t is given, the agent computes the action a_t and the computed action is applied to the environment. Afterward, the environment generates the next state s_{t+1} and reward r_{t+1}. Then, this procedure is repeated. One iteration of the procedure is called an episode.

When the MDP is stochastic, the next state s_{t+1} is determined through the state transition probability model. When the current state is s_t and the action $a_t = a$ is performed, the probability of $s_{t+1} = s'$ is defined as $T^{s'}_{s,a} = Pr[s_{t+1} = s' | s_t = s,\ a_t = a]$, where $Pr[\cdot]$ means the probability of a given event. The reward r_{t+1} is determined by the predefined reward function $r(s_t, a_t)$, i.e., $r_{t+1} = r(s_t, a_t) \in \mathbb{R}$. The action a_t is determined by the policy $\pi(a_t|s_t)$ learned by the agent. The policy $\pi(a|s)$ is the probability of the event that the action becomes a when the state s is given, i.e., $\pi(a|s) = Pr[a_t = a|s_t = s]$.

The optimal policy π^* is the policy that maximizes the expected reward sum up to time t_{end}, and is defined as (1).

$$\pi^* = argmax_\pi \mathbb{E}_{s_t,a_t}[\sum_{t=0}^{t_{end}} \gamma^{t-1} \cdot r(s_t, a_t)] \tag{1}$$

where $\gamma \in [0, 1]$ means the discount factor, and $\mathbb{E}[\cdot]$ means the expectation. The sum of rewards $\sum_{k=t}^{t_{end}} \gamma^{k-1} \cdot r(s_k, a_k)$ is the cumulative sum of the reward signals obtained up to the end of the last episode, and is called a return G_t. The actual return can be calculated at the end of the agent's episode. Therefore, a value function is used to estimate the sum of the discounted rewards that the agent can obtain at the present time. In RL, there are two kinds of value functions: the state value function and the action value function. The state value function $V(s_t)$ at the state s_t represents the expected return value obtained from the current state, i.e., $V(s_t) = \mathbb{E}[G_t|s_t]$. The action value function or Q-function denoted by $Q(s_t, a_t)$ represents the expected return when the action a_t is taken at the current state s_t. In other words, it is defined as $Q(s_t, a_t) = r(s_t, a_t) + \gamma V(s_{t+1})$. The optimal state value function is denoted by $V^*(s_t)$, and the optimal action value function by Q-function $Q^*(s_t, a_t)$, respectively. Rigorously, the two optimal value functions are defined as (2) and (3), respectively, according to the Bellman optimal equation.

$$V^*(s) = \max_a \mathbb{E}[r(s_t, a_t) + \gamma V^*(s_{t+1})|s_t = s, a_t = a] \tag{2}$$

$$Q^*(s, a) = \mathbb{E}[r(s_t, a_t) + \gamma \max_{a'} Q^*(s_{t+1}, a')|s_t = s, a_t = a] \tag{3}$$

A method of performing a greedy policy based on the optimal value function is called value-based reinforcement learning. Greedy policy deterministically selects the action that leads to the maximum value function. In other words, the optimal policy is found from $\pi^*(s) = \text{argmax}_a Q^*(s, a)$. The value function is generally learned by temporal difference learning, and its value is approximated through a neural network. When the state value

function is modeled through the neural network with parameter ψ and is denoted by V_ψ, the objective function $J_V(\psi)$ for training the neural network is given by

$$J_V(\psi) = \mathbb{E}_{s_t}[\frac{1}{2}(r_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t))^2]. \quad (4)$$

Temporal difference learning eliminates the need to find an actual return in learning the value function. In other words, learning can proceed without completing the episode. Most of the deep reinforcement learning methods using temporal difference learning implement an experience replay memory to store the agent's experiences, and uses them during training through random sampling [2,16,21].

On the other hand, a method of directly learning a stochastic optimal policy is called policy-based reinforcement learning. Policy-based reinforcement learning can also model a policy through a neural network, and a policy modeled by the network parameter with parameter ϕ is denoted by π_ϕ. The method of finding the optimal policy by tuning the parameter ϕ is called the policy gradient. The objective function $J_\pi(\phi)$ for training the neural network modeling the policy gradient method is given by

$$J_\pi(\phi) = \mathbb{E}_{\pi_\phi}[G_t] = \mathbb{E}_{s_t}[\sum_{a_t} \pi_\phi(s_t, a_t) G_t]. \quad (5)$$

In Equation (5), the return is obtained only after the episode is finished, and the expected return is estimated through the Monte Carlo method.

Therefore, there is a disadvantage that the variance of the sampling is high and the policy cannot be learned in the middle of an episode. To alleviate this problem, the return can be estimated through a value function, and this method is called the Actor-Critic algorithm [14,23]. An actor-critic's actor refers to the model of the policy, and the critic performs the model of the value function. The critic can be approximated by the objective function (4).

3.2. Role of the Discount Factor in Reinforcement Learning

In computing the return G_t, the discount factor γ plays a key role in obtaining the value function, and also the optimal policy. In order to describe the motivation of the paper in detail, this section explains how the discount factor affects the optimal policy according to the environment using numerical examples.

If the discount factor is set to 1, the effect of the reward signal over time on the return is constant. Figure 1 shows how the value function is computed when a reward signal is obtained deterministically. The value inside the circle indicates the reward r_t at the time t. When the reward signal $r_{T_1} = 1$ is received at the end of the episode at time T_1, in the case of $\gamma = 1$, the resulting Q function is 1, which is the same for any time step. That is, $Q(s_t, a_t) = 1$ $(t = 1, 2, \cdots, T_1 - 1)$. However, depending on the task goal, it may be better for the agent to obtain the reward within a faster time. Let the action value function be Q_π when the policy is π. Assume that the time step T_1 is greater than T_2. If the agent follows the policy π_1 as in Figure 1, the agent obtains a reward at the time T_1. If the agent follows the policy π_2, a reward is given at T_2. Hence, if the discount factor is set to $0 < \gamma < 1$, then $Q_{\pi_1}(s_t, a_t) < Q_{\pi_2}(s'_t, a'_t)$. Therefore, with a discount factor of less than 1, the closer the time point the reward signal is, the higher it is evaluated. However, if the end of the episode exists (i.e., episodic task), the optimal policy does not change according to changes of the the discount factor. As long as the discount factor is between 0 and 1, π_2 is always the optimal policy because $\gamma^{T_1-t-1} \cdot 1 < \gamma^{T_2-t-1} \cdot 1$.

On the other hand, in the case of a continuing task where the end of the episode does not exist, the optimal policy may change depending on the discount factor [11,24]. Unlike the episodic task, a continuing task is not limited to the endpoint of the episode. A continuing task does not finish the episode until the agent makes a big mistake, and the agent can receive numerous reward signals within an episode. Figure 2 illustrates an example of obtaining a reward signal from a continuing task. When the policy is π_1,

the agent receives 1 for the reward signal every two steps. If the policy π_2 is followed, the agent receives 2 for the reward signal every three steps. With these policies, if $\gamma = 0.5$, it follows that $Q_{\pi_1} > Q_{\pi_2}$. Hence, π_1 becomes the optimal policy. On the other hand, if $\gamma = 0.9$, π_2 is the optimal policy. If the discount factor is set to 1, the maximum value function cannot be distinguished. This is because the value functions Q_{π_1} or Q_{π_2} are close to infinity when an agent continuously obtains reward signals. In view of these, it is observed that the optimal policy can vary according to the discount factor. In such a situation, we must redesign the reward function according to the importance of the reward signal (reward shaping). Note that rewards in the near future are more important than rewards in the distant future. By increasing the value of the reward signal in accordance with the importance, the agent may learn properly.

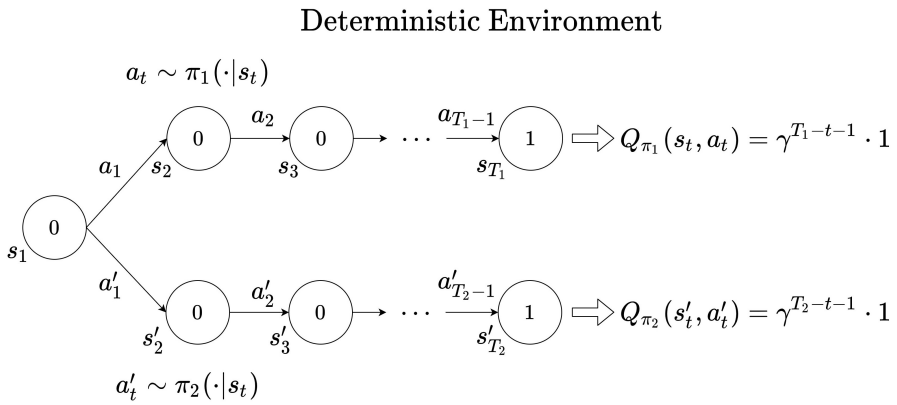

Figure 1. Example of trajectory in a deterministic environment.

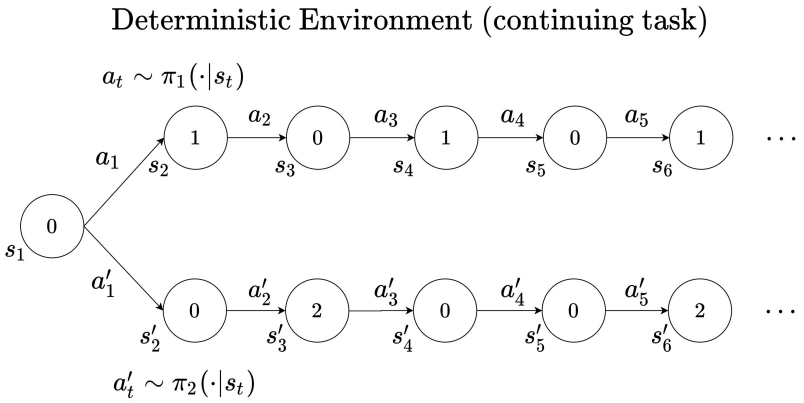

Figure 2. Example of trajectory in a continuing task.

If there exists uncertainty in the continuing task environment, it may be difficult to learn the optimal policy with reward shaping. Figure 3 is an example of an environment with inherent uncertainty due to stochasticity. When the agent follows the policy π_1, it is probabilistically rewarded with $r_6 \in \{-1, 4\}$. If the agent follow the policy π_2, then the agent obtains the deterministic reward $r_4 = 2$. Suppose that G_1 denotes the return that the agent obtains when following the policy π_1 at the state s_1. Then, it can be expressed as (6) or (7).

$$G_1 = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot 0 + \gamma^4 \cdot 4 + \gamma^5 G_6 \tag{6}$$

$$G_1 = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot 0 + \gamma^4 \cdot (-1) + \gamma^5 G_6 \tag{7}$$

Furthermore, at the state s_1, the state value function resulting from following the policy π_1 can be estimated as follows:

$$V_{\pi_1}(s_1) \simeq 0 + \gamma^4 \mathbb{E}[r_6] + \gamma^5 V_{\pi_1}(s_6) \tag{8}$$

In Equation (8), $\mathbb{E}[r_6]$ is the average after sampling according to the distribution of the reward signal $r(s_5, a_5)$. The reward sum estimated by the value function V_{π_1} is always different from the actual return (6) or (7) received by the agent. Accordingly, when the value function is approximated by the Equation (4), error $(r(s_5, a_5) - (V(s_5) - \gamma V(s_6)))^2$ occurs whenever the agent visits the state s_1. If such a value function is used in the policy learning, it may be difficult for the policy network to converge because of the variation in the value function induced by the error. However, if the discount factor is lowered, this error can be reduced, which means that the magnitude of $\gamma^4 \mathbb{E}[r_6]$ in the value function (8) also decreases. As a result, the overestimation of the probabilistically determined reward signal $r(s_5, a_5)$ is reduced as well.

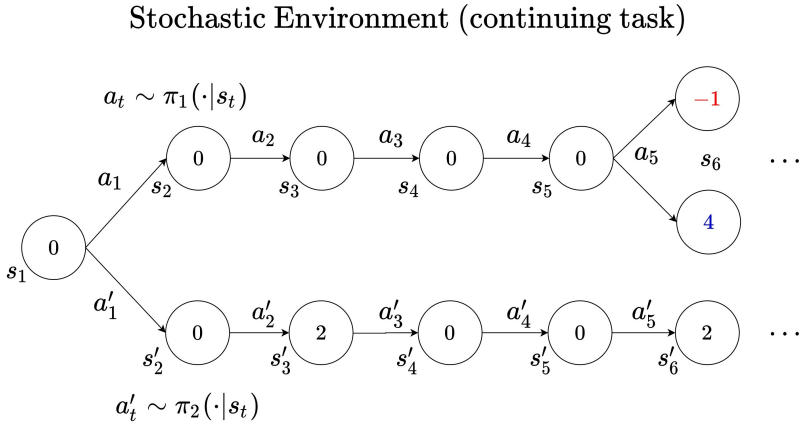

Figure 3. Example of trajectory in the environment with inherent uncertainty.

There is also a possible problem when the discount factor is too low. Figure 4 describes an example of a sparse reward environment. A sparse reward environment refers to an environment in which the agent obtains meaningful reward signals once in a while. As shown in Figure 4, the reward signal is 0 in most cases. Only when the agent reaches the state s_T does it obtains a meaningful reward signal $r_T = 1$. It is not easy for a less trained agent to find these reward signals while interacting with the environment. Even if the reward signal is found by the agent, if the reward sum is estimated through the value function, it may be difficult to predict the value at a state far away from the reward signal. The Q function $Q(s_1, a_1)$ at state s_1 can be estimated as $Q(s_1, a_1) = \gamma^{T-1} \cdot 1$ after obtaining reward $r_T = 1$. This estimated reward sum can be evaluated to be lower when the time step T is large or the discount factor is low. Therefore, since this makes only very little difference between $Q(s_1, a_1)$ and $Q(s_1, a'_1)$, the agent can not be sure that a_1 is indeed a rewarding action.

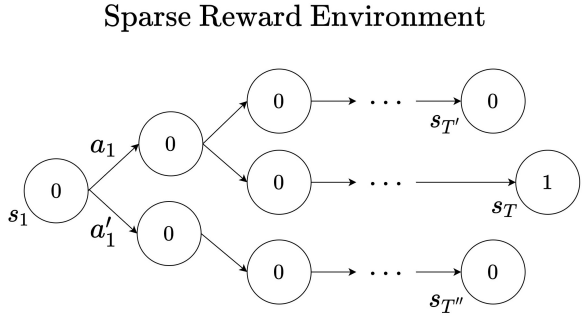

Figure 4. Example of trajectory in the environment with sparse reward.

As explained earlier, the discount factor has a great influence on learning the optimal policy according to the given environment. A high discount factor can be advantageous

for learning when the reward signals are sparse. However, if there is uncertainty in the environment, a high discount factor may rather hinder the convergence of learning. Therefore, depending on the given environment, there must be the most appropriate discount factor for learning. With this observation in mind, in this paper, an adaptive discount factor method is proposed, such that it can find an appropriate value for the discount factor during learning.

4. Adaptive Discount Factor in Reinforcement Learning

4.1. Adaptive Discount Factor

The discount factor has to be adjusted in consideration of environmental uncertainty and the sparsity of the reward [9,11,12]. Without these considerations, the agent can show lower performance. To this end, we want to define an evaluation function of an agent's performance according to the current discount factor. Additionally, based on the evaluation function, we want to find an update rule for the discount factor that can improve the agent's performance.

The definition of the return with a specific discount factor γ_1 is defined by the Equation (9). The return $G_t(\gamma_1)$ is the sum of rewards obtained by the current policy learned using γ_1. However, if the future reward signal is obtained probabilistically or if there is a risk (probabilistically negative reward) in the future reward signal, then the return can be overestimated.

$$G_t(\gamma_1) = r_{t+1} + \gamma_1 \cdot r_{t+2} + \gamma_1^2 \cdot r_{t+3} + \cdots + \gamma_1^{T-1-t} \cdot r_T \tag{9}$$

A high discount factor can lead to an overestimation for the reward in the future when there is uncertainty or stochasticity. By considering uncertainty or risk in the environment and reward overestimation due to an inappropriate discount factor, the agent has to be able to measure confidence for the reward or return. If the expectation of the return is known, it is possible to estimate the sum of the return that is probabilistically determined. For instance, suppose that the agent obtains the sum of the reward $G(\gamma_1)$ generated by the current policy and the discount factor γ_1. If the sum of the reward (i.e., $G_t(\gamma_1)$) is larger than its expectation (i.e., $\mathbb{E}[G_t(\gamma_1)]$), then the positive future rewards for the current policy is guaranteed. In this case, the high discount factor does not make any overestimation and it can be increased. On the other hand, if the return yielded by the current policy is less than or equal to the expected return, the large reward sum in the future is not guaranteed or a lower reward sum can be made. For this case, since the high discount factor can result in overestimation, the discount factor needs to be decreased. To this end, the return confidence corresponding to the current policy is measured by $\frac{1}{M}\sum^{M}(G_t - \mathbb{E}[G_t])$, where M denotes the number of data points. However, the expected return does not know its exact value during training. Hence, the expected return can be replaced by the corresponding value function as follows:

$$\frac{1}{M}\sum^{M}(G_t - \mathbb{E}[G_t]) \simeq \frac{1}{M}\sum^{M}(G_t - V(s_t)), \tag{10}$$

where the right hand side of this equation is called an advantage function and is denoted by A_t. The advantage function can be used to evaluate the trained policy using the discount factor. Similarly, the advantage function can also be expressed as

$$A_t = \frac{1}{M}\sum^{M}(Q(s_t, a_t) - V(s_t)). \tag{11}$$

The proposed adaptive discount factor algorithm is devised by evaluating the performance of the current discount factor using the advantage function. When the advantage function is low, it is interpreted that the discount factor can lead to overestimation. Hence, the discount factor needs to be decreased. Conversely, if the advantage function is high, it means that the current policy is learning via a high reward sum, which suggests that the discount factor should be increased.

The proposed adaptive discount factor is assumed to be used during only the transient time in training. This is because it is also observed that a constant discount factor shows acceptable performance after the transient period of the training.

The proposed adaptive algorithm has mainly two tuning parameters: γ_1 and γ_2. γ_1 is used as the discount factor for the current policy, its initial condition is 0.5, and the algorithm increases γ_1 to obtain a higher advantage function. γ_2 is used as the baseline and the upper bound of γ_1. The initial condition of γ_2 is 0.99. Depending on the advantage function, the proposed adaptive algorithm decreases γ_2.

Let $A_t(\gamma)$ denote the advantage function with the discount factor γ and time t. The proposed adaptive algorithm adjusts the discount factor γ_1 and γ_2, such that it results in the highest advantage function between the upper and lower limits, and is summarized as

$$(\gamma_1, \gamma_2) = \begin{cases} \gamma_1 \leftarrow \gamma_1 + c \cdot (1/\sigma_t) & \text{if } A_t(\gamma_1) < A_t(\gamma_2) \\ \gamma_2 \leftarrow \gamma_2 - c \cdot (1/\sigma_t) & \text{else if } A_t(\gamma_1) > A_t(\gamma_2) \\ \text{do nothing} & \text{else if } \gamma_1 = \gamma_2 \end{cases} \tag{12}$$

where σ_t is the correction function defined as

$$\sigma_t = 1 + \frac{|\delta_{t-1} - \delta_t|}{\delta_t}, \tag{13}$$

$\delta_t = (r_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t))^2$, and the parameter c is the stepsize. The first line of (12) means that γ_1 is increased when $A_t(\gamma_1)$ is smaller than the baseline advantage function $A_t(\gamma_2)$. On the other hand, the second line of (12) implies that γ_2 is decreased when the baseline advantage function $A_t(\gamma_2)$ is smaller than $A_t(\gamma_1)$. Note that $A_t(\gamma_2) < A_t(\gamma_1)$ can imply that the baseline is not good enough, and hence, the baseline needs to be decreased. By doing this, the baseline can be improved, which can lead to a better discount factor γ_1 afterward. In view of the definition of δ_t, naturally, it is large at the beginning of training and its variation becomes smaller as the training goes on. The rationale behind the definition of σ_t is that the adaptation rule changes the discount factor during the transient period of the training since the value function is not reliable in the beginning.

According to the rule of the adaptive discount factor described in (12), the following holds.

$$A_t(\gamma_{1,k}) \leq A_t(\gamma_{1,N}), \tag{14}$$

where k is the number of training epochs and N is the terminal epoch. Hence, $\gamma_{1,k+1} = \gamma_{1,k} + c \cdot (1/\sigma_t)$ when $A_t(\gamma_1) < A_t(\gamma_2)$. The discount factor γ_1 for training the current policy is updated according to γ_2 and its advantage function. This is because $\gamma_{1,k} \leq \gamma_{1,k+1} \leq \gamma_2$, $\gamma_{1,k}$ is always less than or equal to $\gamma_{1,k+1}$ because of their update rule (12). Note that $0 < \gamma < 1$, γ_1 is always non-decreasing, and γ_2 is always non-increasing. According to the definition of return (9), (14) can be written as

$$\begin{aligned} &\sum_{l=t}^{t_{end}} \gamma_{1,k}^{l-1} \cdot r_l - \mathbb{E}[G_t(\gamma_{1,k})] \\ &\quad \leq \sum_{l=t}^{t_{end}} \gamma_{1,N}^{l-1} \cdot r_l - \mathbb{E}[G_t(\gamma_{1,N})]. \end{aligned} \tag{15}$$

In light of (15), the advantage function $A_t(\gamma_1)$ can be viewed as an $(t_{end} - 1)$th-order polynomial in γ_1. Figure 5 shows an example of a graph for the advantage function. This means that γ_1 and γ_2 have to meet at a point where the maximum of the advantage function is achieved like the red circle in Figure 5.

Thanks to this, the proposed method can automatically find the maximum advantage function between the discount factor 0.5 and 0.99. A discount factor of less than 0.5 is ignored because it is too small to take a long-term reward sum into account. Since the return

$\sum_{l=t}^{t_{end}} \gamma_{1,N}^{l-1}$ in (15) is proportional to the objective function of policy (5), the update size of the policy gradient can be larger when $\gamma_{1,k}$ is updated to $\gamma_{1,N}$. In summary, the return and discount factor are evaluated by the advantage function, the discount factor is determined according to that evaluation, and the return with a higher advantage function value is considered further in the training.

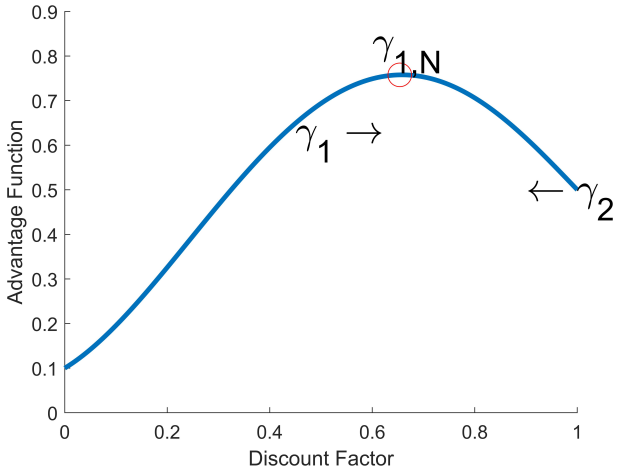

Figure 5. Example of discount factor adjustment using the adaptive discount factor.

In the next sections, it is shown how the proposed adaptive rule can be implemented in both on-policy and off-policy reinforcement learning algorithms. This is necessary because the advantage function has to be computed differently in on-policy and off-policy reinforcement learning algorithms.

4.2. Adaptive Discount Factor in On-Policy RL

For the purpose of presenting how to apply the proposed method to an on-policy algorithm, PPO (Proximal Policy Optimization) is employed in this paper. For other on-policy algorithms, a similar procedure can be applied. PPO is a representative on-policy deep reinforcement learning algorithm that is a simplified method of the trust region method of TRPO (Trust Region Policy Optimization) [13,25].

The objective function for policy approximation in PPO is defined as (16), and the network parameter ϕ is tuned to maximize it.

$$J^{CLIP}(\phi) = \mathbb{E}[\min(\rho_t(\phi) \cdot A_t,\ clip(\rho_t(\phi), 1-\epsilon, 1+\epsilon) \cdot A_t)], \tag{16}$$

where A_t is the advantage function and is defined after defining the value function approximation, and $\rho_t(\phi)$ denotes the policy conservation ratio and is defined as $\rho_t(\phi) = \frac{\pi_\phi(s_t|a_t)}{\pi_{\phi_{old}}(s_t|a_t)}$. In this way, when the policy $\pi_\phi(s_t|a_t)$ is learned, it is guaranteed not to degrade its performance compared with $\pi_{\phi_{old}}(s_t|a_t)$.

The adaptive discount factor algorithm (12) needs to compute two advantage (or value) functions, unlike the original PPO. To this end, two neural networks parameterized with ψ_1 and ψ_2 are trained in order to approximate each value function. The objective function for learning the state value functions V_{ψ_1} and V_{ψ_2} is given by

$$J^V(\psi_{i=1,2}) = \mathbb{E}[\frac{1}{2}(r_t + \gamma_i V_{\psi_{i=1,2}}(s_{t+1}) - V_{\psi_{i=1,2}}(s_t))^2]. \tag{17}$$

Based on the state value functions V_{ψ_1} and V_{ψ_2}, the advantage function is computed as follows.

$$A_t(\gamma_i) = r_{t+1} + \cdots + \gamma_i^{T-2-t} \cdot r_{T-1} + \gamma_i^{T-1-t} \cdot V_{\psi_i}(s_T) - V_{\psi_i}(s_t) \tag{18}$$

These advantage functions $A_t(\gamma_i)$, $i = 1, 2$ are used in the proposed adaptive discount factor. During policy learning, $A_t(\gamma_1)$ is applied to the objective function (16) and is used in policy learning. PPO with the proposed adaptive discount factor is summarized in Algorithm A1.

4.3. Adaptive Discount Factor in Off-Policy RL

This section presents how to implement the proposed algorithm for off-policy deep reinforcement learning using SAC (Soft Actor-Critic). SAC is a model-free reinforcement learning algorithm suitable for continuous action tasks [16,26].

Similar to the previous section, two value functions parameterized by ψ_1 and ψ_2 are needed for the adaptive algorithm. The objective function for approximating the state value function of SAC is defined as follows:

$$J^V(\psi_{i=1,2}) = \frac{1}{2}(V_{\psi_i}(s_t) - \mathbb{E}_{a_t}[Q_{\theta_i}(s_t, a_t) - \alpha \log \pi_\phi(a_t|s_t)])^2, \tag{19}$$

where $-\alpha \log \pi_\phi(a_t|s_t)$ is the entropy of the policy distribution, the constant α is a tuning parameter and determines the ratio of entropy, and Q_{θ_i} is the action value function parameterized by θ_1 and θ_2. The following objective function is used to approximate the action value function.

$$J^Q(\theta_{i=1,2}) = \frac{1}{2}(Q_{\theta_i}(s_t, a_t) - (r(s_t, a_t) + \gamma_i V_{\psi_i}(s_{t+1})))^2 \tag{20}$$

The neural network to approximate the policy is trained using the following objective function:

$$J^\pi(\phi) = \log \pi_\phi(a_t|s_t) - Q_{\theta_1}(s_t, a_t) \tag{21}$$

where the right hand side is nothing but the difference between the distribution of the policy and the distribution of the action value function. Hence, the objective function (21) is the form of a Kullback–Leibler divergence between $\pi_\phi(a_t|s_t)$ and $Q_{\theta_1}(s_t, a_t)$.

With these definitions in mind, in SAC, the advantage function for the adaptive algorithm can be defined as

$$A_t(\gamma_{i=1,2}) = Q_{\theta_i}(s_t, a_t) - \alpha \log \pi_\phi(a_t|s_t) - V_{\psi_i}(s_t). \tag{22}$$

Note that entropy is added to the definition of the advantage function (11). Since the state value function $V_{\psi_i}(s_t)$ approximates the reward sum using the entropy-augmented reward in SAC, it is also considered in the advantage function. The SAC with the proposed adaptive algorithm is summarized in Algorithm A2.

5. Experiment in the Environments with Uncertainty

In this section, applications of the proposed adaptive algorithm are shown to validate the performance. To test the performance of the on-policy and off-policy algorithms, Tetris game agent and motion planning for a robot manipulator are presented.

The adaptive discount factor algorithm is compared with fixed discount factors and the progressively increasing discount factor. The method of the progressively increasing discount factor is adopted from [15]. This increasing rule is described by

$$\gamma_{1,k+1} = 1 - 0.98(1 - \gamma_{1,k}). \tag{23}$$

The discount factor $\gamma_{1,k}$ denotes a value of γ_1 at the kth training epoch. The initial and final values of $\gamma_{1,k}$ are set to 0.5 and 0.99, respectively. This increasing discount factor method shows better policy improvements compared with a constant discount factor [15].

5.1. Tetris

Tetris has been used a lot as a challenge in the field of artificial intelligence, and there have been efforts to solve it with various machine learning algorithms [27–29]. However, Tetris is known as an NP-hard problem, and it is a challenging problem with deep reinforcement learning [28]. Additionally, as Tetris is an environment with inherent uncertainty, it is not easy to estimate the maximum reward sum by approximating the value function [3]. Tetris is a game in which randomly given blocks, called tetrominoes, are stacked on the bottom, and if the blocks are filled without empty spaces, a score proportional to the number of filled rows is given. Tetris basically amounts to a continuing task, but if the player stacks up blocks without filling until the top, the game ends there. Thus, an overestimation of a reward for the time far away from the current time can mislead the game without realizing the uncertainty. For this, the proposed adaptive discount factor can be a suitable method for developing an RL-based game agent for Tetris.

Tetris is a discrete action task, and at every step you can move blocks left, right, down, or rotate 90°. There are two kinds of drop actions: soft and hard drops. The soft drop moves the block down by one space, and the hard drop, to the bottom.

In this paper, we redefine the agent's action space as a compound action, like [30]. Compound action space $\mathcal{A} = \{0, 1, 2, \cdots, 35\}$ is defined, which consists of move left/right (9), rotate a block (4), and a hard drop (1). The compound actions can speed up learning by limiting the agent's meaningless actions. The Tetris game screen is 10×20 in this paper.

The state of MDP is composed of two images of the screen, i.e., $10 \times 20 \times 2$. One is the image of the currently stacked blocks without the currently controllable block, and the other is the images of the controllable blocks without the currently stacked blocks. See Figure 6. The reward is determined based mainly on the number of lines cleared by the agent. Table 1 defines the reward.

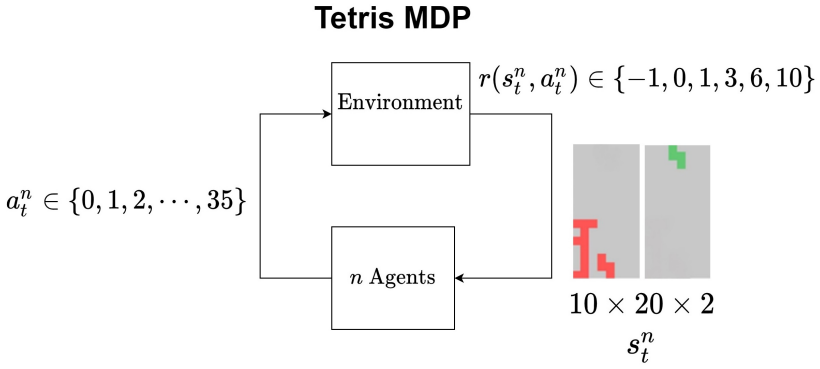

Figure 6. Schematic of MDP in the game of Tetris.

Table 1. Number of cleared lines and reward.

n_r	$-\infty$	0	1	2	3	4
r	-1	0	1	3	6	10

Here, n_r denotes the number of cleared lines by the current action, and r, the corresponding reward. $-\infty$ in the second column in Table 1 describes that the blocks reach the top. In other words, the agent loses the game. Figure 6 depicts the structure of MDP of Tetris.

The PPO algorithm is implemented using a parallel agent method such as A3C (Asynchronous Advantage Actor-Critic) for efficient learning, and is applied to develop a Tetris game agent for fast performance improvement. The A3C algorithm samples data from multiple agents in parallel, and uses them for training. When the number of a specific agent is n, the data obtained from the nth agent in the time step t is $(s_t^n, a_t^n, r_t^n, s_{t+1}^n)$. Each agent decides a policy through the same neural network with parameters ϕ, and synchronizes

all neural networks whenever policy learning is performed. Figure 7 describes how the proposed adaptive discount factor is used to devise a PPO-based game agent for Tetris.

The performance of Tetris can be evaluated by the number of lines cleared by the agent during one episode. Learning performance with the adaptive discount factor is compared with the performance with fixed discount factors.

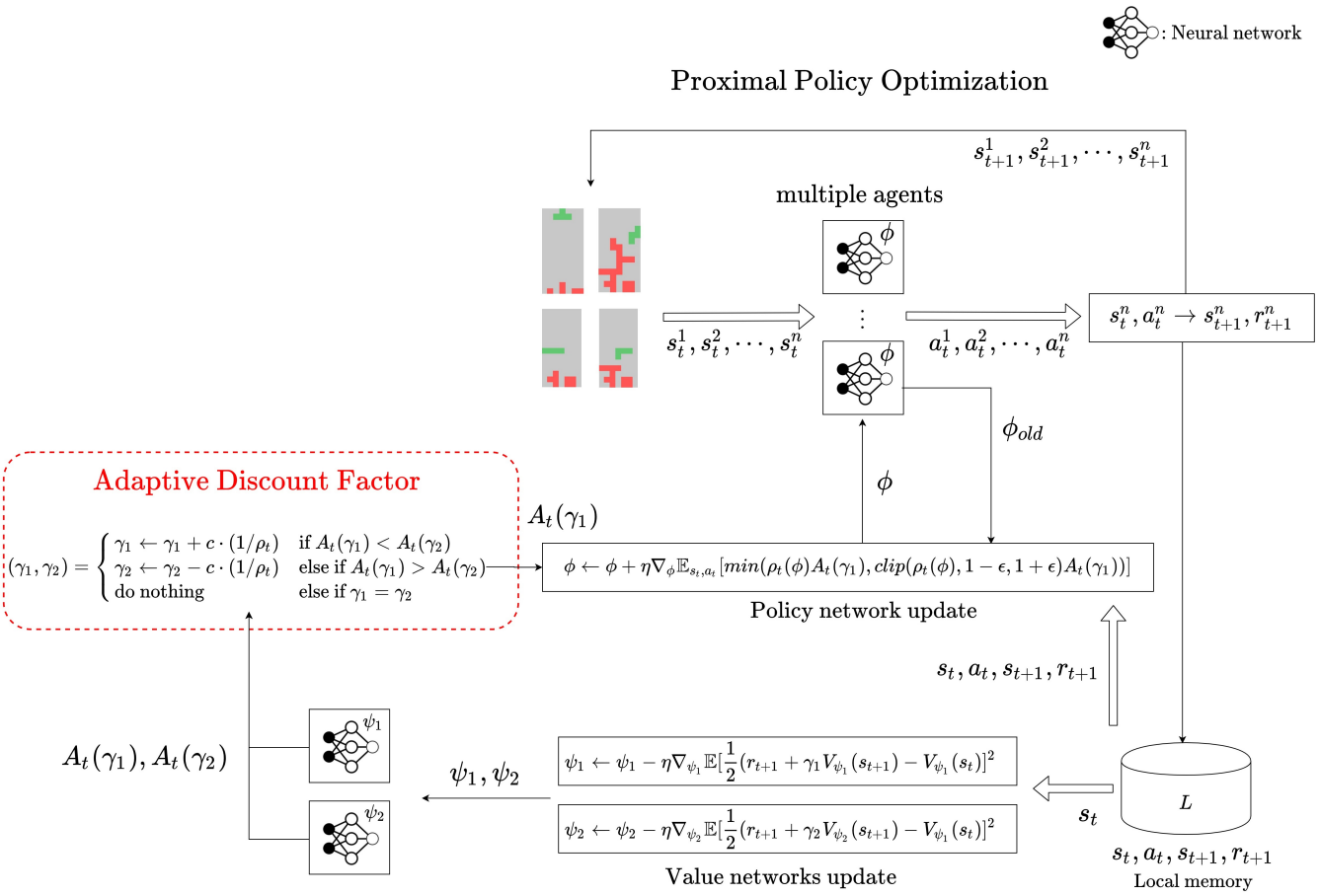

Figure 7. PPO with adaptive discount factor for the Tetris environment.

Figure 8 shows training performances by four different fixed discount factors, the proposed adaptive discount factor, and the progressively increasing discount factor. The discount factor of 0.99 is a commonly used value in deep reinforcement learning, and 0.5 is the low limit of the adaptive discount factor. The horizon axis of the graph indicates the number of episodes learned by the agent, and the vertical axis indicates the number of lines cleared by the agent for each episode. The highest discount factor of 0.99 shows a fast performance improvement at the beginning of learning, but the performance converges to a certain level as the episode progresses. The lowest discount factor of 0.5 shows the lowest performance compared to other discount factors. The discount factor of 0.9 shows a faster performance improvement than 0.99 at the beginning of training. It is confirmed that the discount factor of 0.7 can reach the highest performance, although the performance improvement is slow at the beginning of training. The performance of the progressively increasing discount factor was improved quickly at the beginning, but in the end, it converged to a similar performance by the fixed discount factor of 0.99. The performance of the agent trained using the adaptive discount factor is lower than those by the fixed discount factors at the beginning of training, but the final performance exceeds that of the agent using a discount factor of 0.7.

Figure 9 shows the used fixed discount factors and the resulting adaptive discount factor and the increasing discount factor. In the case of the adaptive discount factor, adjustment is stopped when the discount factor reaches $\gamma_1 = \gamma_2$. The adaptive discount factor is stopped at $\gamma_1 = 0.7214$. At the beginning of the adjustment, γ_1 does not change because $A_t(\gamma_1 = 0.5) > A_t(\gamma_2)$. However, when $A_t(\gamma_1 = 0.5) < A_t(\gamma_2 = 0.7214)$, γ_1 is increased, as is shown in Figure 9. Note that the adaptive discount factor stops adjusting at the value near 0.7, which results in a high performance among the fixed discount factors in Figure 8. Table 2 shows the comparison of the final performance of the fixed discount factors and the adaptive discount factor.

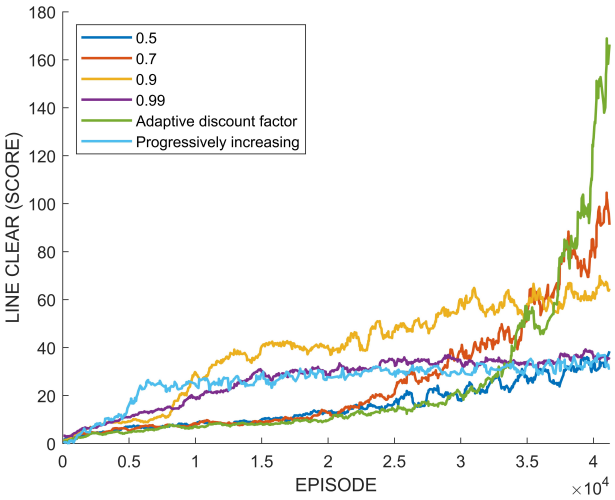

Figure 8. Performance comparison between fixed and adaptive discount factor in Tetris.

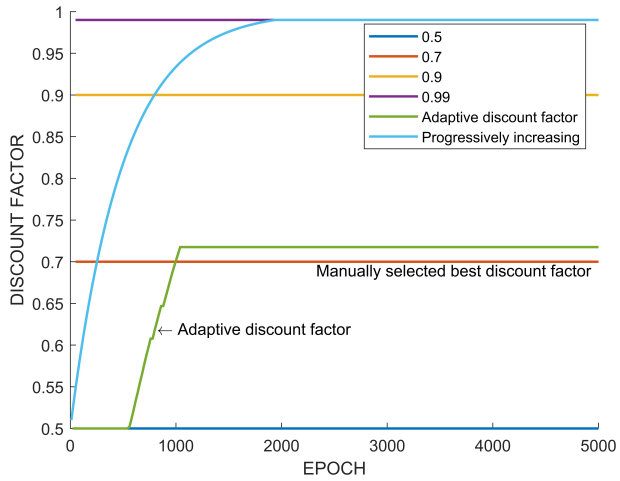

Figure 9. Adjustment of discount factor in Tetris.

Table 2. Score comparison between fixed and adaptive discount factor in Tetris.

	Maximum Score	**Average Score**	γ_1
Adaptive discount factor	**318**	**139.28**	**0.7214**
Increasing discount factor	37	29.94	0.99
Fixed discount factor 1	49	25.98	0.99
Fixed discount factor 2	115	65.03	0.9
Fixed discount factor 3	**179**	**97.94**	**0.7**
Fixed discount factor 4	61	35.87	0.5

In Table 2, the scores in bold mean the highest maximum score or highest average score along the discount factors and algorithms. Also, the discount factors in bold are the values corresponding to the maximum score.

5.2. Motion Planning

This section presents an application of the proposed adaptive discount factor to the path planning of the robot manipulator for the purpose of validating the proposed method for off-policy RL [31,32]. The joint value of the robot arm is expressed in the configuration space, and the current state representing the joint value of the robot arm at the current time step t is denoted by $q_t \in \mathcal{Q}$ [33,34]. The action a_t is the amount of change in the joint value, and the action space is given by $\mathcal{A} = (0, 1)$. Since the action of the agent is continuous, the SAC algorithm is employed for motion planning. In general, although path planning problems are achieved via simulation, which means that there are no uncertainties in the problem, in order to consider real environments in this paper, two uncertainties are added to the path planning problem in the simulation. First, noise ϵ_t is added to the state evolution. In other words, the next state is determined by $q_{t+1} = q_t + \beta a_t + \epsilon_t$, where β is a constant, a_t is the amount of change in the joint value, and the noise $\epsilon_t \sim \mathcal{N}(0, 1)$. Second, a reward signal is transmitted probabilistically. In general, in the path planning task, the reward signal is given a positive value when the agent arrives at a goal point, and a negative value when it collides [35]. In this paper, when the agent arrives at the goal point, one of $\{1, 2, 3\}$ is randomly transmitted. Therefore, the agent cannot easily approximate the reward sum at the current point. Additionally, since there is noise in the environment, the agent cannot easily determine whether to prioritize collision avoidance or to prioritize goal arrival, even if collision is considered. In addition, the problem of uncertainty is more pronounced in continuing the task. In the path planning problem, an episodic task ends when the goal is reached, but for a continuing task, a new goal is given to the agent [36]. This aspect makes it difficult for the path planning environment to predict the distant future rewards, and an overestimation has to be prevented. Therefore, the proposed adaptive discount factor can handle this difficulty appropriately. Figure 10 describes the MDP for the path planning based on the reinforcement learning.

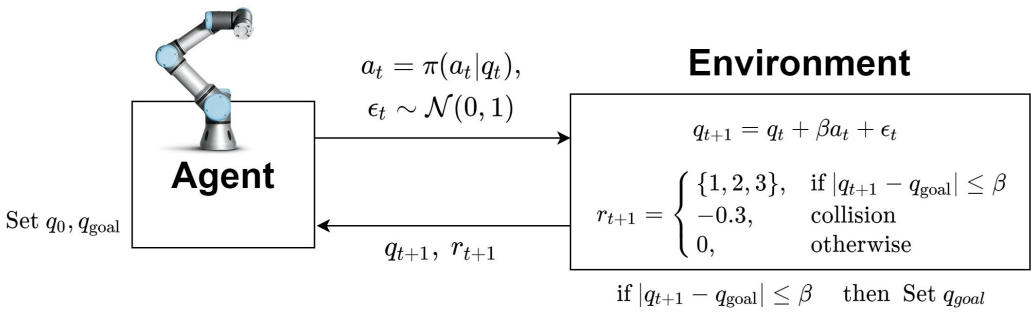

Figure 10. Schematic of MDP in robot manipulation task.

In Figure 10, q_{goal} represents the goal position given to the agent, and the inequality $|q_{t+1} - q_{goal}| \le \beta$ is understood as the goal position is reached. Compared with PPO from the view point of applying the adaptive discount factor, the difference is in how to calculate the advantage function. The parameters of all neural networks are needed because SAC computes the advantage function based on Equation (22). Since SAC is an off-policy algorithm, it implements the experience replay memory D, randomly samples data during training, and applies it to each neural network training. Figure 11 describes how SAC with the adaptive discount factor is applied to the path planning of the robot manipulator.

For the validation of the performance of the path planning by the proposed method, two environments are considered. One is path planning in the form of an episodic task in which an episode terminates when the agent reaches the goal. The other is in the form of a continuing task in which a goal is given again when the agent arrives at the previous

goal, and this is repeated until the agent collides. As explained before, the effect of the environmental uncertainty is more critical in the continuing task. In the episodic task, the episode ends when the agent reaches the goal, but in the continuing task, the next goal is given at a random location when the agent reaches the goal point. Therefore, in the case of the continuing task, it is not easy to estimate the expected reward sum. In this section, we test the learning performance of both the adaptive and fixed discount factors. In the path planning, the performance can be evaluated as the success ratio of the agent's arrival at the goal point, that is, the success ratio of path creation. This success ratio is defined as the ratio of reaching the goal point in the last 10 episodes during training.

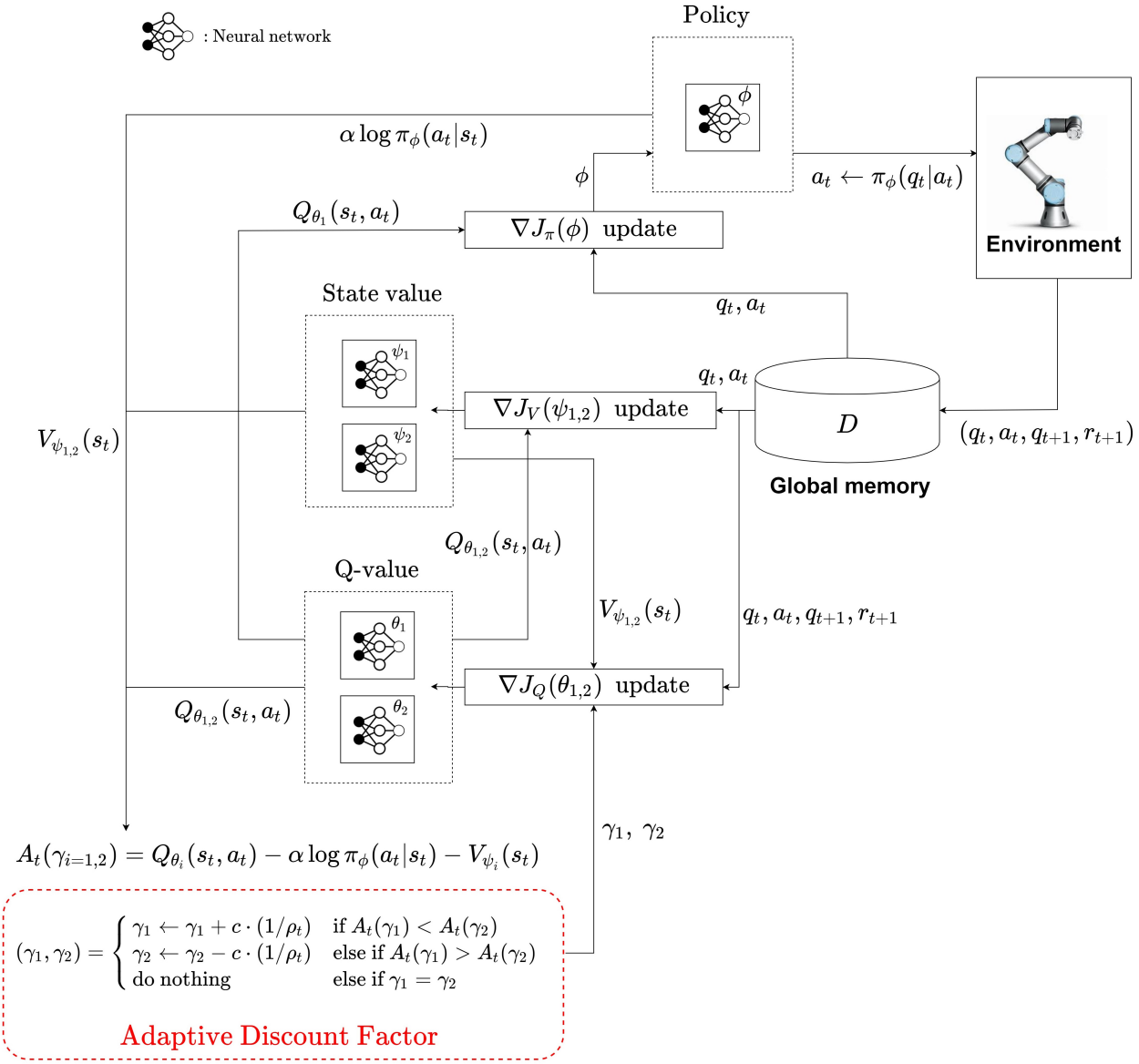

Figure 11. SAC with adaptive discount factor for robot manipulation environment.

Figure 12 shows the learning performance of the fixed discount factors 0.5, 0.7, 0.8, and 0.98, the adaptive discount factor, and the increasing discount factor in the episodic task. The horizontal axis is the number of episodes, and the vertical axis is the success ratio of path generation. The lowest discount factor of 0.5 results in the lowest success rate. As the discount factor is increased, the success ratio is also increased. When the adaptive discount factor is applied, it shows a somewhat low performance at the beginning, but as learning progresses, it can be seen that the performance is closest to the fixed discount factor of 0.98.

The performance of the increasing discount factor method was also similar and reached a high performance. Since, in the episodic task, the uncertainty is low and the reward sum is easy to predict, the fixed discount factor of 0.98 shows a higher performance.

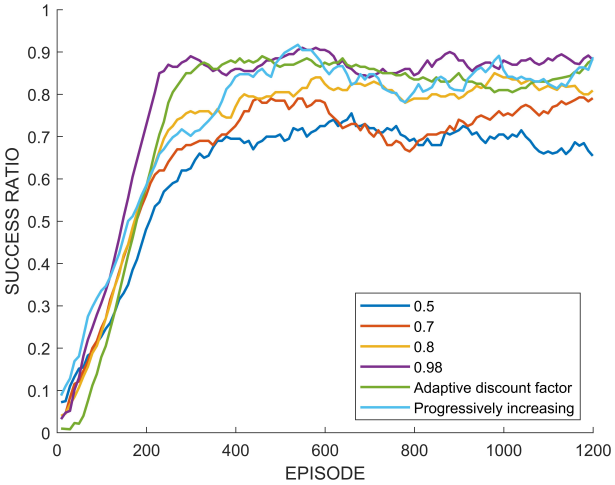

Figure 12. Success rate of path planning in the episodic task.

Figure 13 compares the evolution of the adaptive discount factor with other fixed discount factors in the episodic work. It can be seen that the adaptive discount factor converges close to the highest discount factor of 0.98 as the training goes on. A high discount factor is advantageous for learning for the path planning in episodic tasks, since a reward signal is given and the episode ends when the final goal is reached. Furthermore, since the path planning problem is generally regarded as a sparse reward problem, a high discount factor is helpful. Table 3 shows the success ratio of path creation according to each discount factor. In Table 3, the success ratio is the average of the past 10 test episodes, and if the path generation succeeds in all 10 times, it is 1.0. The values of the success rate in bold are the highest value among them and the discount factors in bold are the optimal values in the episodic environment.

In the following, the learning performance of the adaptive discount factor, increasing discount factor, and the fixed discount factor in the path planning is compared in a continuing task environment. When the agent reaches the current goal point, a new goal point is given, and this is repeated. In other words, in a continuing task, the episode continues until the agent collides. For this case, the learning performance is evaluated by the number of paths generated by the agent before collision.

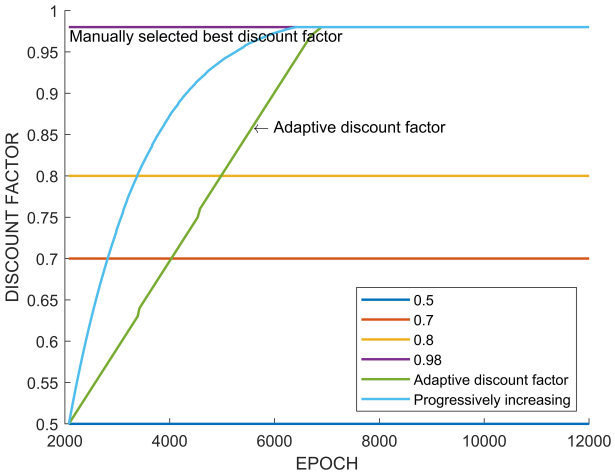

Figure 13. Adjustment of discount factor in episodic path planning task.

Table 3. Comparison of success rate in episodic path planning task.

	Success Rate	γ_1
Adaptive discount factor	0.8846	**0.98**
Increasing discount factor	**0.8915**	**0.98**
Fixed discount factor 1	**0.8884**	**0.98**
Fixed discount factor 2	0.8222	0.8
Fixed discount factor 3	0.7833	0.7
Fixed discount factor 4	0.6588	0.5

Figure 14 shows the learning performance of the fixed discount factors of 0.5, 0.75, 0.85, and 0.99, increasing the discount factor and the adaptive discount factor for the path planning in the continuous episode setting. Three observations can be made. First, among the fixed discount factors, the resulting performance is not monotone. Namely, it does not hold that the higher the discount factor is, the better the performance that is generated. Second, except for a discount factor of 0.99, all of the other cases including the adaptive discount factor and the increasing discount factor show similar performances during the transient time, and a performance of 0.99 was not high, even at the end of training. Third, the adaptive discount factor shows the best performance in the end.

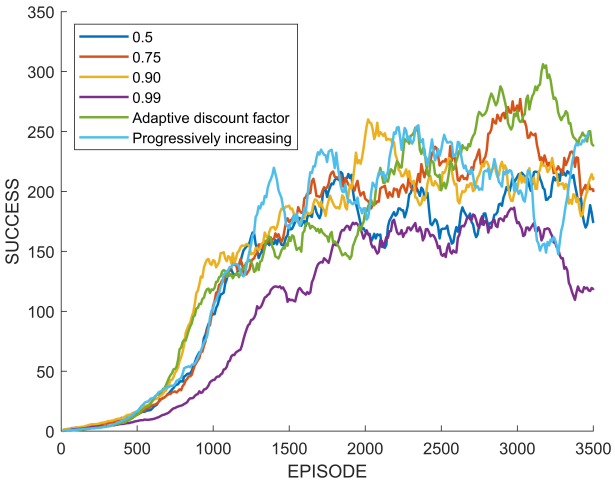

Figure 14. Perforamance of path planning in continuing task.

Figure 15 shows the evolution of the adaptive discount factor and the fixed discount factors. Note that a fixed discount factor of 0.75 results in the best performance and the adaptive discount factor converges to a value of near 0.75. Table 4 summarizes the performance according to each discount factor. The numbers of created paths in bold mean the highest maximum or the highest average number of paths according to algorithms and discount factors. Also, the discount factors in bold correspond to the highest performance.

Hence, it is verified that the proposed adaptive discount factor leads to a good performance, although there are inherent uncertainties in the environment, and the reward is sparse.

In view of these case studies, it is confirmed that the deep RL with the proposed adaptive discount factor results in a comparable learning performance with that by the best and fixed discount factor that has to be determined by exhaustive simulations.

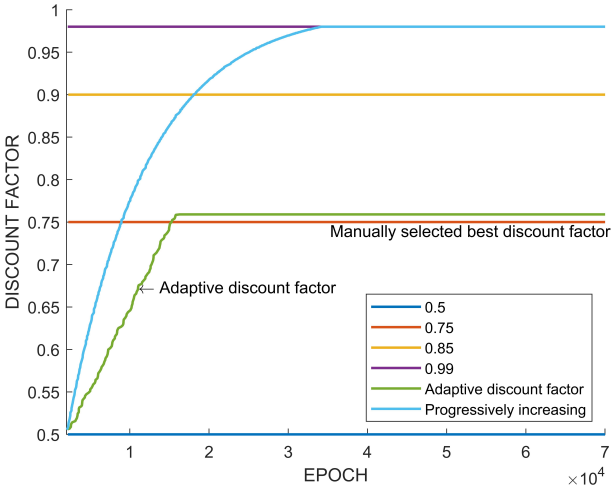

Figure 15. Adjustment of discount factor in continuing path planning task.

Table 4. Comparison of performance in continuing path planning task.

	Maximum	Averages	γ_1
Adaptive discount factor	**464**	**287.77**	**0.7591**
Increasing discount factor	409	194.74	0.99
Fixed discount factor 1	256	155.76	0.99
Fixed discount factor 2	351	212.87	0.90
Fixed discount factor 3	**421**	**231.86**	**0.75**
Fixed discount factor 4	388	189.09	0.5

5.3. Analysis and Discussion

In view of the previous case study, it is confirmed that the adaptive discount factor is indeed effective. In many existing results on reinforcement learning, a large discount factor is usually used, such as 0.99. However, as seen from the case study, the high discount factor does not necessarily always guarantee a high performance. In the previous learning experiment, when a discount factor was randomly selected among values between 0.5 and 0.99 and training was performed, there was a case in which a higher performance was obtained with a value of lower than 0.99, as shown in Figures 8 and 14. On the other hand, Figure 12 showed the highest performance, with 0.99. The difference between these tasks is the existence of an uncertain risk or an unpredictable negative reward among distant future rewards. In the case of Tetris, the agent might obtain negative rewards according to future mistakes, but it is impossible to predict this because blocks are randomly determined. In the case of path planning, it is impossible to predict the failure and collision of path generation when generating a path to a new randomly generated goal point. These correspond to the example in Figure 3. On the other hand, in the case of the path planning problem that is an episodic task, it is relatively easy to predict the success or failure of path creation for a fixed goal point. This case only matches Figure 4, which is a sparse reward environment. Therefore, the discount factor of 0.99 showed the highest performance in this task. The adaptive discount factor algorithm closely finds the discount factor that can give the best performance in all cases. This saves the effort in finding a suitable discount factor. Additionally, as shown in Figures 9, 13 and 15, the adjustment of the discount factor is quickly terminated at the beginning of training. Since the adjusted discount factor is fixed and learning proceeds with the fixed value, it is not computational expensive. Another discount factor adjustment algorithm, called progressively increasing discount factor [15], gradually increases the discount rate to 0.99. It has been suggested that this method can achieve a higher performance than a fixed discount factor. However, the proposed adaptive discount factor outperforms in an environment requiring a low discount factor.

This also suggests that a somewhat lower discount factor may be appropriate, depending on the environment.

6. Conclusions

This paper proposed an adaptive discount factor for the environment with uncertainty. The discount factor has to be decided differently, according to the distribution of rewards given in the environment. A commonly used high discount factor places a high value on future rewards. However, depending on the environment, it can be risky to consider the distant future rewards too much. In this case, a high discount factor can impair the performance of the agent. In such an environment, a low discount factor can be an effective countermeasure. Generally, it is difficult to know what the appropriate discount factor is before training. Therefore, in this paper, we proposed an adaptive discount factor algorithm that can find a proper value of the discount factor automatically at the beginning of learning. The algorithm adjusts the discount factor according to the advantage function during training, such that the reinforcement learning agent shows good performance in the end. It is shown that the adaptive discount factor converges to the discount factor that is manually found via exhaustive search, and that leads to the best performance. Using Tetris and the path planning problem, we found that, depending on the environment, a slightly lower discount factor could yield a higher performance than a higher discount factor. Tetris has difficulty in predicting distant future rewards because of the next block that appears randomly. For the path planning problem, a high discount factor was suitable for an episodic task, but a low discount factor showed a high performance in a continuing task in which the goal point was randomly given. It is very cumbersome to find this from multiple test training. The proposed algorithm finds the discount factor by comparing the advantage functions created by different neural networks. In addition, in order to minimize the computational cost, it is performed quickly to search for the optimal discount factor at the beginning of learning. The proposed adaptive discount factor results consistently in a good performance for various environments.

Future research includes how to enhance the performance of the value estimation when there is a nontrivial gap between the model and the real environment. In this paper, the algorithm is applied in response to the simulation problem in which uncertainty exists. However, in real environments, more unpredictable situations may arise. Therefore, we would like to consider ways to adjust the trade-off between uncertainty and the value of rewards in real environments.

Author Contributions: M.K. and M.-S.C. surveyed the backgrounds of this research, designed the preprocessing data, designed the deep learning network, and performed the simulations to show the merits of the proposed method. J.-S.K. and J.-H.P. supervised and supported this study. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Industrial Fundamental Technology Development Program (20014786, Development of AI-based Concrete Slab Finishing Automation System) funded by the Ministry of Trade, Industry & Energy (MOTIE) of Korea, and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2019R1A6A1A03032119.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Algorithms A1 and A2 show the adaptive discount factor algorithms applied to PPO and SAC. In the algorithms, the advantage function is calculated using two neural networks, respectively, and the two discount factors γ_1 and γ_2 are evaluated according

to the advantage functions. γ_1 is adjusted in the direction where the advantage function becomes larger, and when the two discount factors are equal, the adjustment is terminated.

Algorithm A1: Adaptive Discount Factor in PPO

Initialize the discount factors : $\gamma_1 = 0.5,\ \gamma_2 = 0.99$
Initialize the network parameters : $\phi,\ \psi_1,\ \psi_2$
while Training N epoch **do**
  Samples M data from MDP
  $G_t(\gamma_i) = r_{t+1} + \cdots + \gamma_i^{T-2-t} \cdot r_{T-1} + \gamma_i^{T-1-t} \cdot V_{\psi_i}(s_T)$
  $A_t(\gamma_i) = G_t(\gamma_i) - V_{\psi_i}(s_t)$
  if $\frac{1}{M}\sum^{M} A_t(\gamma_1) < \frac{1}{M}\sum^{M} A_t(\gamma_2)$ **then**
    $\gamma_1 \leftarrow \gamma_1 + c \cdot (1/\sigma_t)$
  else if $\frac{1}{M}\sum^{M} A_t(\gamma_1) > \frac{1}{M}\sum^{M} A_t(\gamma_2)$ **then**
    $\gamma_2 \leftarrow \gamma_2 - c \cdot (1/\sigma_t)$
  else if $\gamma_1 = \gamma_2$ **then**
    do nothing
  end if
  train policy parameter ϕ with γ_1
  $\phi \leftarrow argmax_\phi \mathbb{E}[\min(\rho_t(\phi) \cdot A_t(\gamma_1),\ clip(\rho_t(\phi), 1-\epsilon, 1+\epsilon) \cdot A_t(\gamma_1))]$
end while

Algorithm A2: Adaptive Discount Factor in SAC

Initialize the discount factors : $\gamma_1 = 0.5,\ \gamma_2 = 0.99$
Initialize the network parameters : $\phi,\ \psi_1,\ \psi_2,\ \theta$
while Training N epoch **do**
  Samples M data from Experience Replay Memory
  $A_t(\gamma_{i=1,2}) = Q_{\theta_i}(s_t, a_t) - \alpha \log \pi_\phi(a_t|s_t) - V_{\psi_i}(s_t)$
  if $\frac{1}{M}\sum^{M} A_t(\gamma_1) < \frac{1}{M}\sum^{M} A_t(\gamma_2)$ **then**
    $\gamma_1 \leftarrow \gamma_1 + c \cdot (1/\sigma_t)$
  else if $\frac{1}{M}\sum^{M} A_t(\gamma_1) > \frac{1}{M}\sum^{M} A_t(\gamma_2)$ **then**
    $\gamma_2 \leftarrow \gamma_2 - c \cdot (1/\sigma_t)$
  else if $\gamma_1 = \gamma_2$ **then**
    do nothing
  end if
  train policy parameter ϕ with γ_1
  $\theta_{i=1,2} \leftarrow argmin_{\theta_i} \mathbb{E}[\frac{1}{2}(Q_{\theta_i}(s_t, a_t) - (r(s_t, a_t) + \gamma_i V_{\psi_i}(s_{t+1})))^2]$
  $\phi \leftarrow argmin_\phi \mathbb{E}[\log \pi_\phi(a_t|s_t) - Q_{\theta_1}(s_t, a_t)]$
end while

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
2. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
3. Berseth, G.; Geng, D.; Devin, C.M.; Rhinehart, N.; Finn, C.; Jayaraman, D.; Levine, S. SMiRL: Surprise Minimizing Reinforcement Learning in Unstable Environments. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
4. Garcıa, J.; Fernández, F. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* **2015**, *16*, 1437–1480.
5. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
6. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 1587–1596.
7. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
8. Thrun, S.; Schwartz, A. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*; Lawrence Erlbaum: Hillsdale, NJ, USA, 1993; Volume 6, pp. 1–9.

9. Badia, A.P.; Sprechmann, P.; Vitvitskyi, A.; Guo, D.; Piot, B.; Kapturowski, S.; Tieleman, O.; Arjovsky, M.; Pritzel, A.; Bolt, A.; et al. Never Give Up: Learning Directed Exploration Strategies. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
10. Badia, A.P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, Z.D.; Blundell, C. Agent57: Outperforming the atari human benchmark. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 13–18 July 2020; pp. 507–517.
11. Amit, R.; Meir, R.; Ciosek, K. Discount factor as a regularizer in reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 13–18 July 2020; pp. 269–278.
12. Van Seijen, H.; Fatemi, M.; Tavakoli, A. Using a logarithmic mapping to enable lower discount factors in reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2019**, *33*, 14134–14144.
13. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
14. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; pp. 1928–1937.
15. François-Lavet, V.; Fonteneau, R.; Ernst, D. How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies. In Proceedings of the NIPS 2015 Workshop on Deep Reinforcement Learning, Montréal, QC, Canada, 11 December 2015.
16. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1861–1870.
17. Chen, X.; Wang, C.; Zhou, Z.; Ross, K.W. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
18. Dabney, W.; Ostrovski, G.; Silver, D.; Munos, R. Implicit quantile networks for distributional reinforcement learning. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1096–1105.
19. Mavrin, B.; Yao, H.; Kong, L.; Wu, K.; Yu, Y. Distributional reinforcement learning for efficient exploration. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 4424–4434.
20. Yang, D.; Zhao, L.; Lin, Z.; Qin, T.; Bian, J.; Liu, T.Y. Fully parameterized quantile function for distributional reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2019**, *33*, 6193-6202.
21. Kapturowski, S.; Ostrovski, G.; Quan, J.; Munos, R.; Dabney, W. Recurrent experience replay in distributed reinforcement learning. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
22. Fedus, W.; Gelada, C.; Bengio, Y.; Bellemare, M.G.; Larochelle, H. Hyperbolic discounting and learning over multiple horizons. *arXiv* **2019**, arXiv:1902.06865.
23. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst.* **1999**, *12*, 1057-1063.
24. Naik, A.; Shariff, R.; Yasui, N.; Yao, H.; Sutton, R.S. Discounted reinforcement learning is not an optimization problem. *arXiv* **2019**, arXiv:1910.02140.
25. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning, Lille, France, 7–9 July 2015; pp. 1889–1897.
26. Haarnoja, T.; Tang, H.; Abbeel, P.; Levine, S. Reinforcement learning with deep energy-based policies. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 1352–1361.
27. Algorta, S.; Şimşek, Ö. The game of tetris in machine learning. *arXiv* **2019**, arXiv:1905.01652.
28. Demaine, E.D.; Hohenberger, S.; Liben-Nowell, D. Tetris is hard, even to approximate. In Proceedings of the International Computing and Combinatorics Conference, Big Sky, MT, USA, 25–28 July 2003; Springer: New York, NY, USA, 2003; pp. 351–363.
29. Gabillon, V.; Ghavamzadeh, M.; Scherrer, B. Approximate dynamic programming finally performs well in the game of Tetris. *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 1754-1762.
30. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **2019**, *575*, 350–354. [CrossRef] [PubMed]
31. Gu, S.; Holly, E.; Lillicrap, T.; Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; IEEE: Providence, RI, USA, 2017; pp. 3389–3396.
32. Kalashnikov, D.; Irpan, A.; Pastor, P.; Ibarz, J.; Herzog, A.; Jang, E.; Quillen, D.; Holly, E.; Kalakrishnan, M.; Vanhoucke, V.; et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In Proceedings of the Conference on Robot Learning, Zurich, Switzerland, 29–31 October 2018 ; pp. 651–673.
33. Spong, M.W.; Hutchinson, S.; Vidyasagar, M. *Robot Modeling and Control*; Wiley: New York, NY, USA, 2006; Volume 3.
34. Frank, M.; Leitner, J.; Stollenga, M.; Förster, A.; Schmidhuber, J. Curiosity driven reinforcement learning for motion planning on humanoids. *Front. Neurorobot.* **2014**, *7*, 25. [CrossRef] [PubMed]

35. Prianto, E.; Kim, M.; Park, J.H.; Bae, J.H.; Kim, J.S. Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay. *Sensors* **2020**, *20*, 5911. [CrossRef] [PubMed]
36. Zhu, H.; Yu, J.; Gupta, A.; Shah, D.; Hartikainen, K.; Singh, A.; Kumar, V.; Levine, S. The Ingredients of Real World Robotic Reinforcement Learning. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.